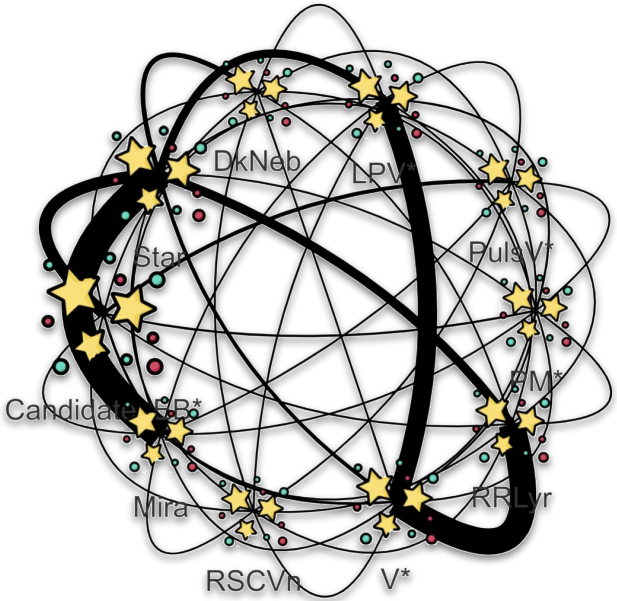
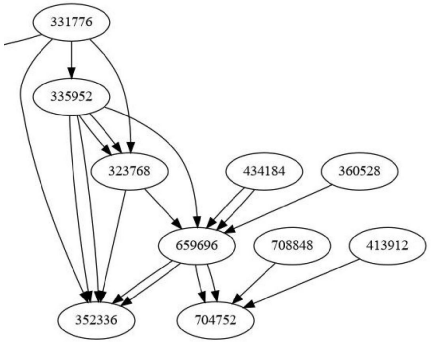
Classification Graph

tools to organize and investigate (multiple) classifications



- **Multi-Classification** *framework*
- **Classification Schemas**
  - Fink Portal
  - Features
  - Light Curves

  concrete classifiers (protos, tests)
- **Exploring Classification Graphs**
  - Neighborhood
  - Re-classification
  - Overlaps
- **Tags**
- **CLI**
- **Status**



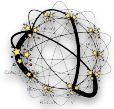
Julius Hrivnac, IJCLab

Fink WS
16-18/7/25
Clermont-Ferrand

Aim

- We are using classification to know what an object nature is
  - We can use it also to organise objects
- **Organizing classifications** of sources and alerts in structures (graphs) to allow (fast):
  - Searching for sources classified in a certain way
  - Searching for ‘similar’ sources and alerts
  - Comparing different classifications schemas
    - Evaluating how close they are
    - Interpreting classification with one schema by classes of another schema
      - E.g. Interpreting AI based classification
- Intuitive and interactive visualisation of those features relations
- Simple CLI
- Integration in Fink Portal

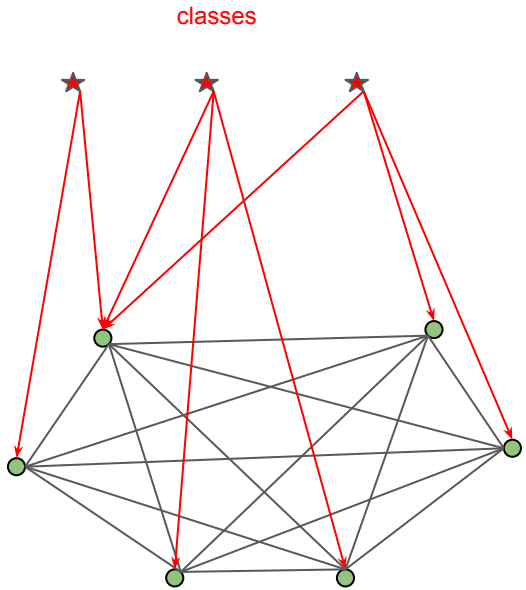
Note: source = object



Multi-Classification

Simple Classification

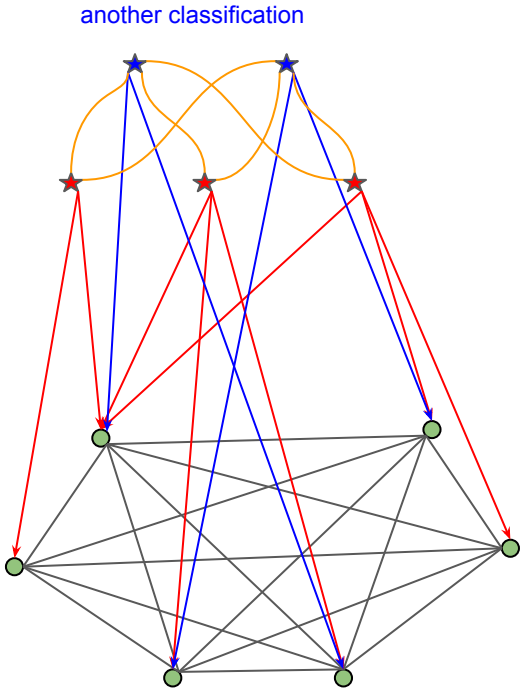
- Want to investigate relations (similarities) between objects *(alerts, sources)*
- We should define a **measure** to quantify objects similarity
  - Can have several different measures
- The measure allows objects classification
  - Without discrete classification, the problem is `n*n (n = number of sources/alerts)`, which is undoable for large number of objects
  - With a discrete classification scheme, the problem is `n*m + m*m + m (m = number of classes, m << n)`
    - `n*m` - classification of new objects, only done when they arrive
    - `m*m` - overlaps, when needed
    - `m` - to get information for a particular object
  - Some classifications should be trained, but that can be done on the subset of objects and re-done only when something changes
- Several objects types (having relations) can be classified in parallel *(alerts, sources)*
  - Objects *Xyz* are classified via *XyzOfInterest* collection
    - All *Xyz* of a class belongs to that class *XyzOfInterest*
  - One object can belong to several classes (with respective **weights**)



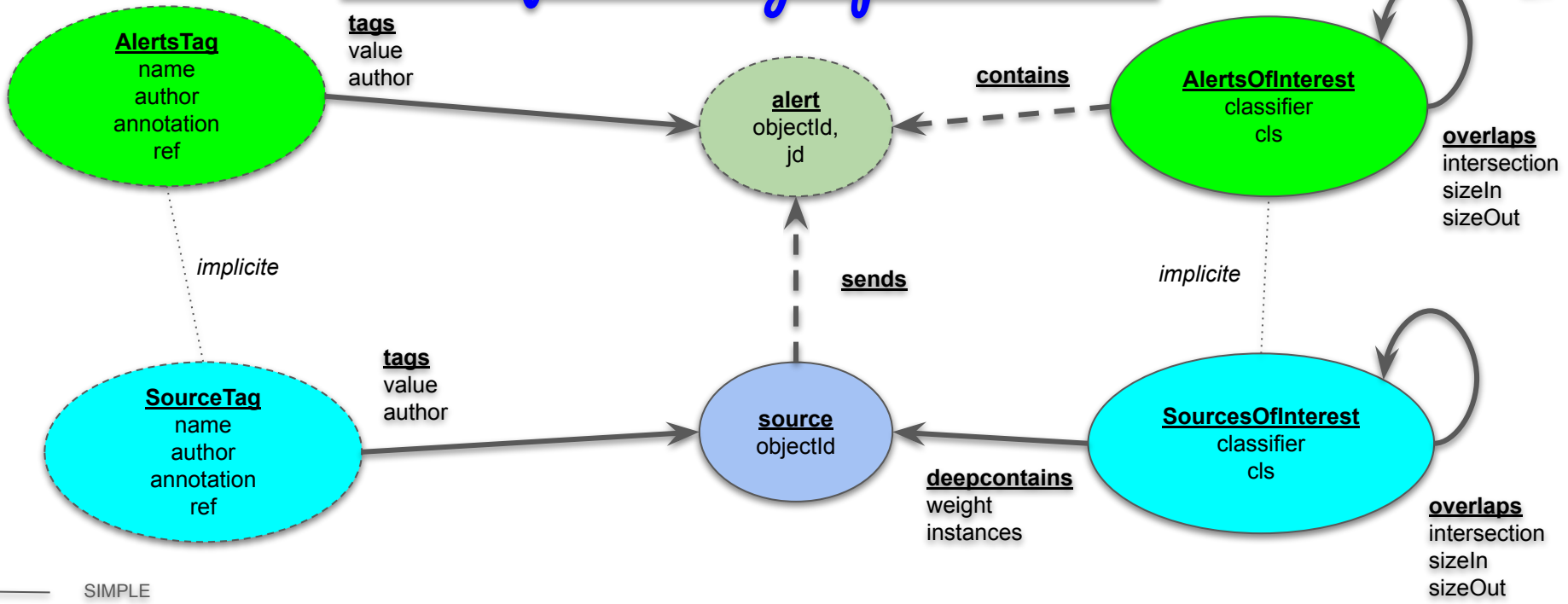
objects with their relations

Multiple Classification

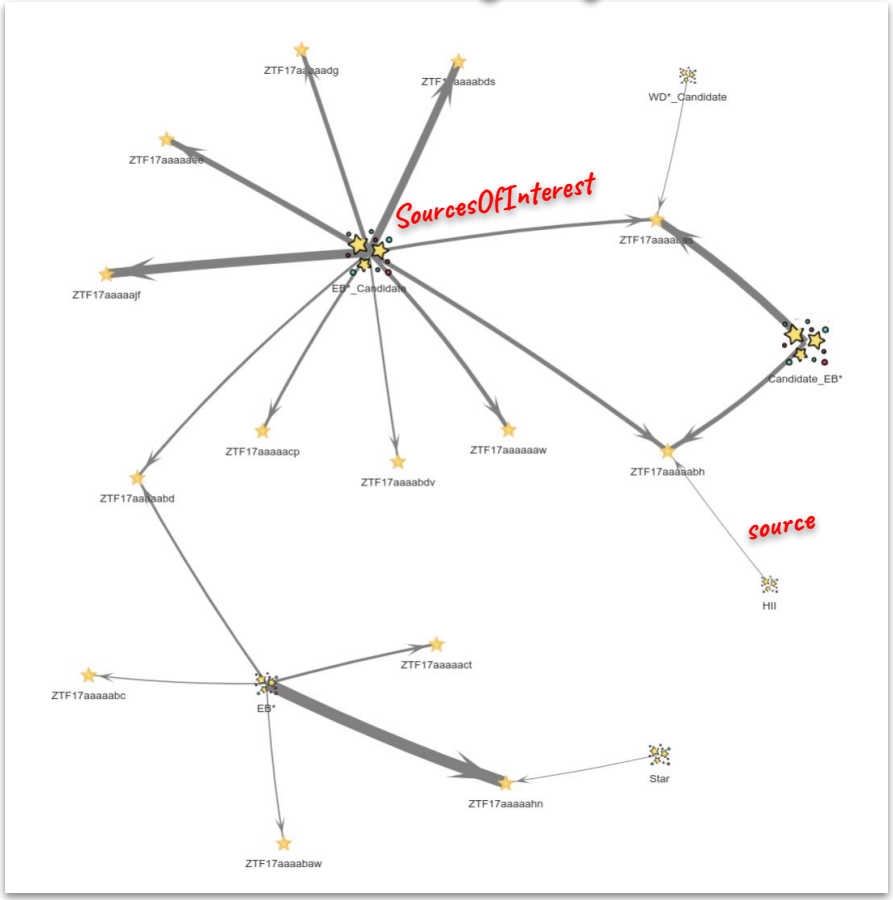
- Objects can be classified with several schemas, exploring different kind of similarities:
  - Analytical: calculated for each object individually by an algorithm, or taken from third services
    - Fink Portal
  - AI: calculated by trained network
    - Features
    - Light Curves
    - Cutouts ?
  - Tags: supplied directly by users
- Classification schemas can have several ***flavours***
  - From different classification sources or hyperparams,...
    - Fink Portal itself contains several classification schemas
- Classification schemas can be **combined** into *superschemas*
- Classification schemas can be **divided** into *subschemas*
- Once existing objects are classified, we can easily (and quickly):
  - Classify new objects
  - Find overlaps between classes in one schema or between schemas
  - Comparing truthfulness of schemas
  - Interpreting AI classes with Analytical classes



Classification Graph Schema

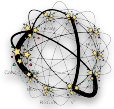


Classification Graph



Multiple Classification Schemas

- FINK_PORTAL: The standard classification defined by Fink Portal
  - Just imported into Graph
- FEATURES: The classification derived from alert ‘features’
  - PCA + Clustering by Spark
- LIGHT_CURVES: The classification derived from similarities of alerts ‘light curves’
- CUTOUTS ?
- + several *flavours* for each schema

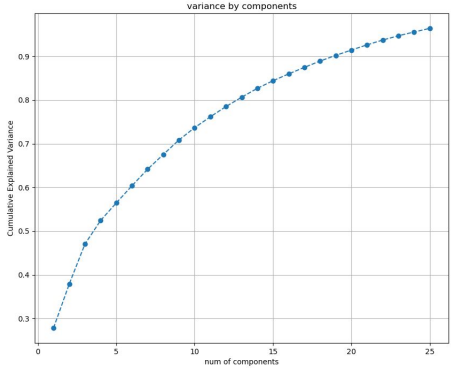


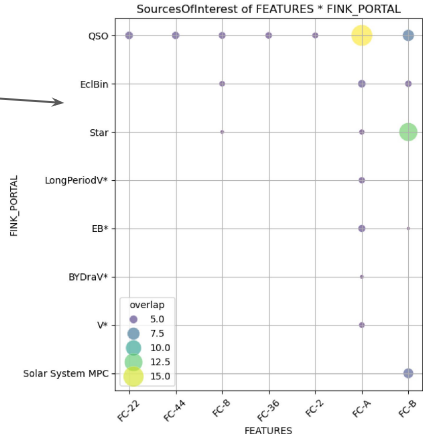
Classification Schemas

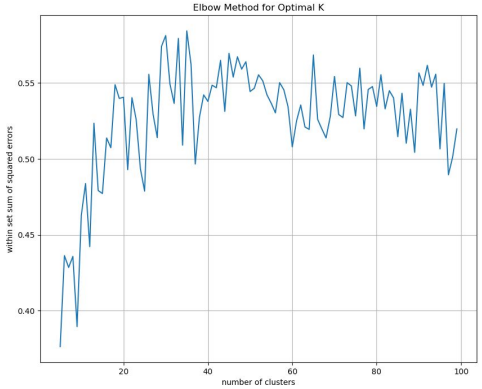
FEATURES

PCA Clustering

- Preparing training data
  - Well classified alerts and classes with enough statistics
- PCA
  - choose number of pca parameters @ 80% variance = 13
- KMeans clustering
  - choose number of clusters @ maximum silhouette = 30
- Output clustering model
- Creating FEATURES graph
- Generating overlaps
  - Merging similar classes
- All new alerts are (re)classified as they are arriving
- Reasonable results for some classes
- So far very naive & brute force
  - Just taking all features and existing classes
  - Various selections of training sources



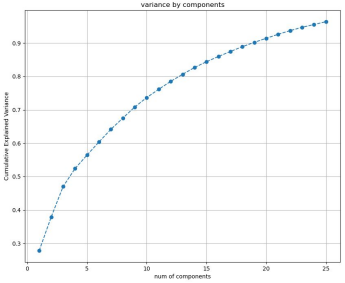




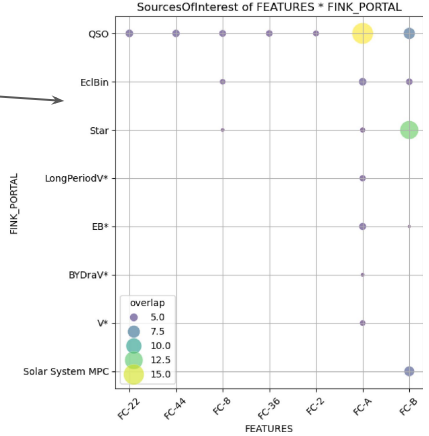
FEATURES

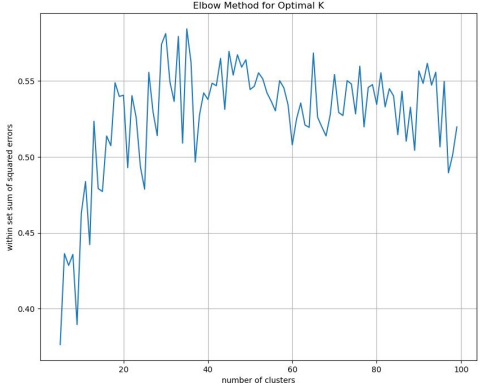
PCA Clustering

- Preparing training data
  - Well classified alerts and classes with enough statistics
- PCA
  - choose number of pca parameters @ 80% variance = 13
- KMeans clustering
  - choose number of clusters @ maximum silhouette = 30
- Output clustering model
- Creating FEATURES graph
- Generating overlaps
  - Merging similar classes
- All new alerts are (re)classified as they are arriving
- Reasonable results for some classes
- So far very naive & brute force
  - Just taking all features and existing classes
  - Various selections of training sources

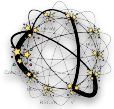



```
"mean",
"weighted_mean",
"standard_deviation",
"median",
"amplitude",
"beyond_1_std",
"cusum",
"inter_percentile_range_10",
"kurtosis",
"linear_trend",
"linear_trend_sigma",
"linear_trend_noise",
"linear_fit_slope",
"linear_fit_slope_sigma",
"linear_fit_reduced_chi2",
"magnitude_percentage_ratio_40_5",
"magnitude_percentage_ratio_20_10",
"maximum_slope",
"median_absolute_deviation",
"median_buffer_range_percentage_10",
"percent_amplitude",
"mean_variance",
"anderson_darling_normal",
"chi2",
"skew",
"stetson_K"
```





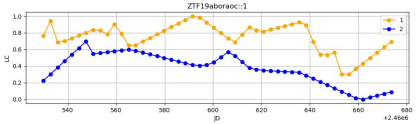
LIGHT CURVES

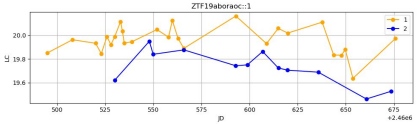


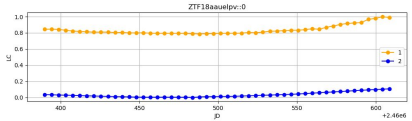
- Clean the curves
  - using naive brute force
- Combine composite curves
  - can use AI to find useful compositions
- Randomise and split data: 75% for training, 25% for testing
- Configure long short-term memory recurrent NN (LSTM-RNN)
- Train & test
- So far very naive & brute force
  - Just selecting non-trivial curves
  - Or their combinations

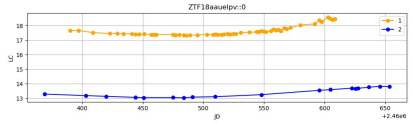
LIGHT_CURVES

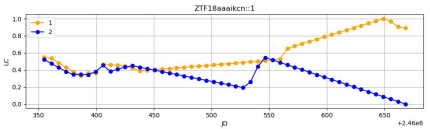
cleaning

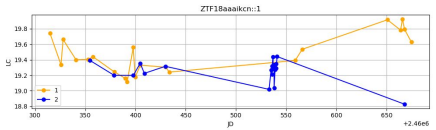


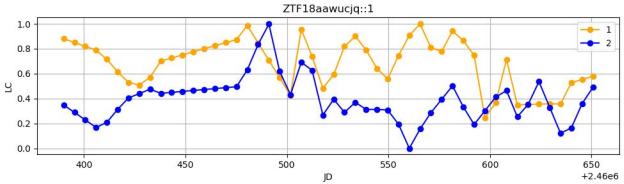


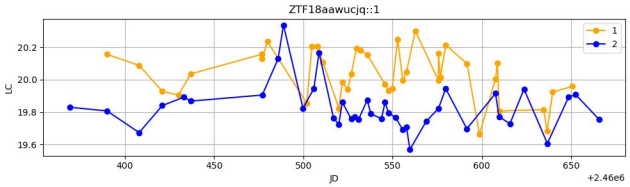


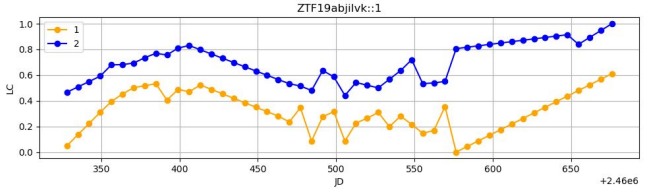


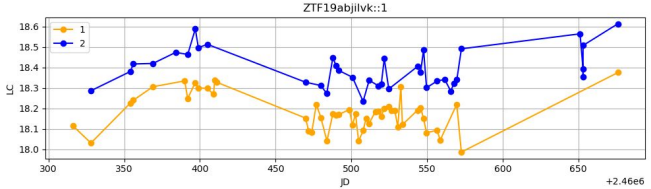


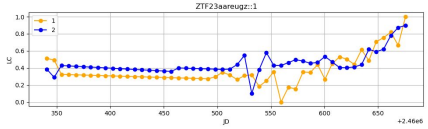


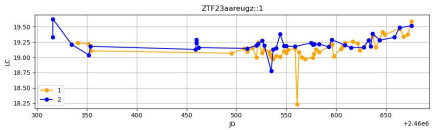


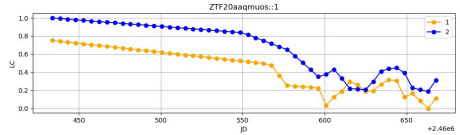


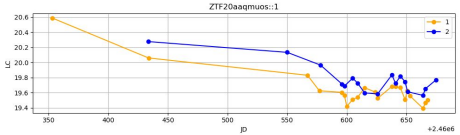


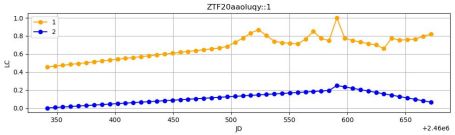


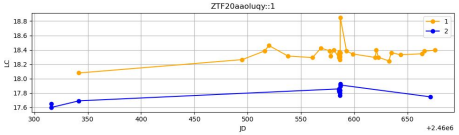






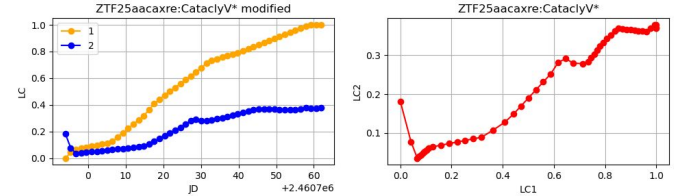


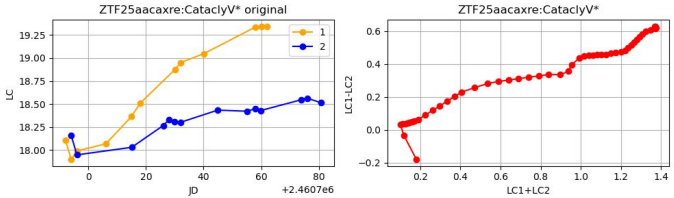


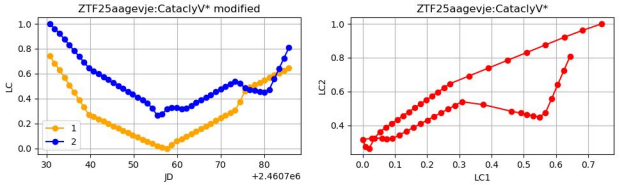


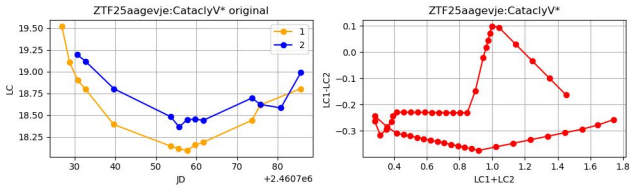
LIGHT_CURVES

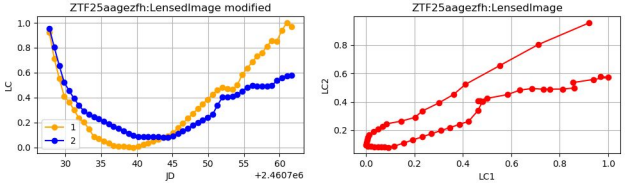
cleaning & composing

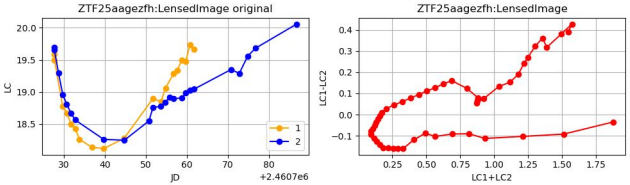


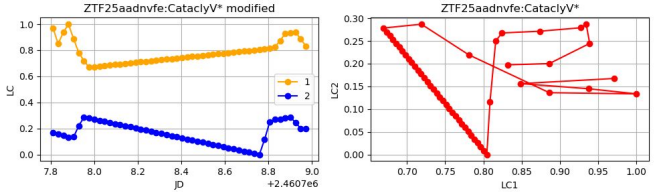


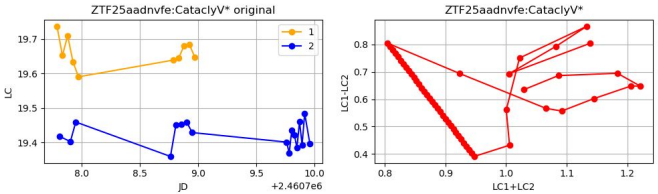










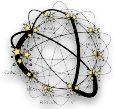


LIGHT_CURVES

training

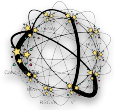
- Clean the curves
- Randomise and split data: 75% for training, 25% for testing
- Configure long short-term memory recurrent NN (LSTM-RNN)
- Train & test
  - Keras LSTM by dl4j
- Preliminary results for some classes

```
conf = new NeuralNetConfiguration.Builder()
                                 .seed(123)
                                 .weightInit(WeightInit.XAVIER)
                                 .updater(new Adam(0.002))
                                 .gradientNormalization(GradientNormalization.ClipElementWiseAbsoluteValue)
                                 .gradientNormalizationThreshold(0.5)
                                 .list()
                                 .layer(new LSTM.Builder()
                                              .activation(Activation.TANH)
                                              .nIn(1)
                                              .nOut(20)
                                              .build())
                                 .layer(new RnnOutputLayer.Builder(LossFunctions.LossFunction.MSE)
                                                          .activation(Activation.SOFTMAX)
                                                          .nIn(20)
                                                          .nOut(numLabelClasses)
                                                          .build())
                                 .build();
```



Exploring Classification Graphs

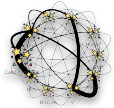
Neighborhood

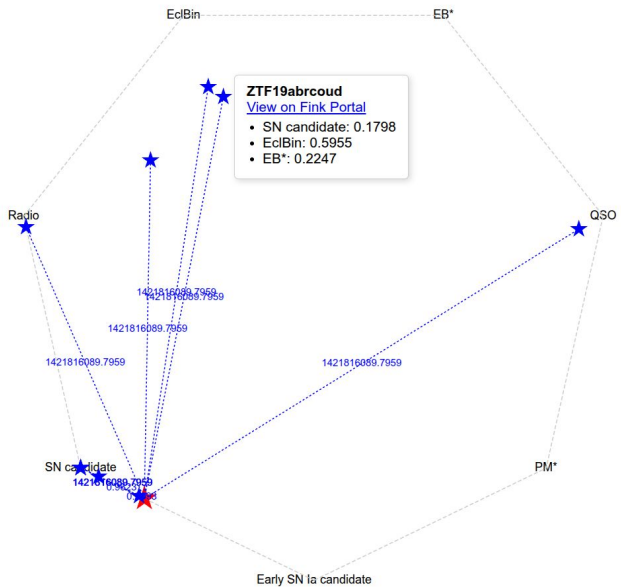


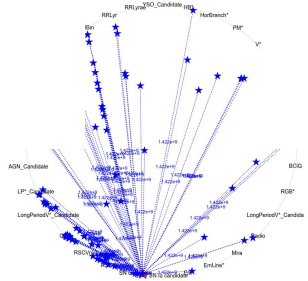
- With respect to a classification schema
- And using certain measure
- Once we get ‘similar’ sources, we can do more detailed comparison

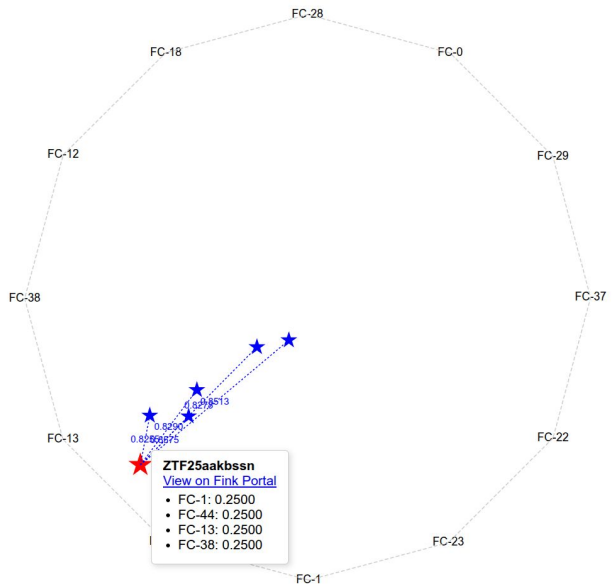
```
gr.sourceNeighborhood('ZTF25aaqyygm', 'FINK_PORTAL', 10, 'JensenShannon', 0.0, false)

  -   calculating source distances
      from ZTF25aaqyygm[SN candidate:0.782608695652174, Early SN Ia candidate:0.21739130434782608]
      using [nmax:10.0, metric:JensenShannon, climit:0.0, allClasses:false]

==>ZTF25aavlimb=0.002668906826373764={SN candidate=0.7857142857142857, Early SN Ia candidate=0.21428571428571427}
==>ZTF25aauhbec=0.015127377117961508={SN candidate=0.7647058823529411, Early SN Ia candidate=0.23529411764705882}
==>ZTF25aavfxlt=0.015132453397871537={SN candidate=0.8, Early SN Ia candidate=0.2}
==>ZTF25aalpqkg=0.027252196630868728={SN candidate=0.75, Early SN Ia candidate=0.25}
==>ZTF25aavozkg=0.027252196630868728={SN candidate=0.75, Early SN Ia candidate=0.25}
==>ZTF25aakbssn=0.04552313886652741={SN candidate=0.7272727272727273, Early SN Ia candidate=0.2727272727272727}
==>ZTF25aayxghj=0.12260324160315225={SN candidate=0.625, Early SN Ia candidate=0.375}
==>ZTF25aascfmn=0.12953628201346198={SN candidate=0.6153846153846154, Early SN Ia candidate=0.38461538461538464}
==>ZTF25aawjnuz=0.17896159576496104={SN candidate=0.5454545454545454, Early SN Ia candidate=0.45454545454545453}
==>ZTF25aatclux=0.17972561983947669={SN candidate=0.95, Early SN Ia candidate=0.05}
```

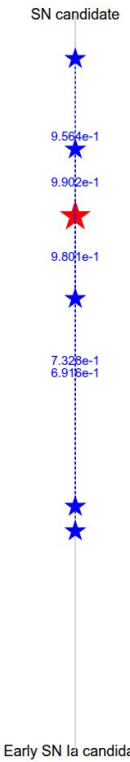
Visual Neighborhood











- Projecting n-dim space into 2-dim plane
- Some classes on the circle should be closer (TBD)
  - Due to overlap

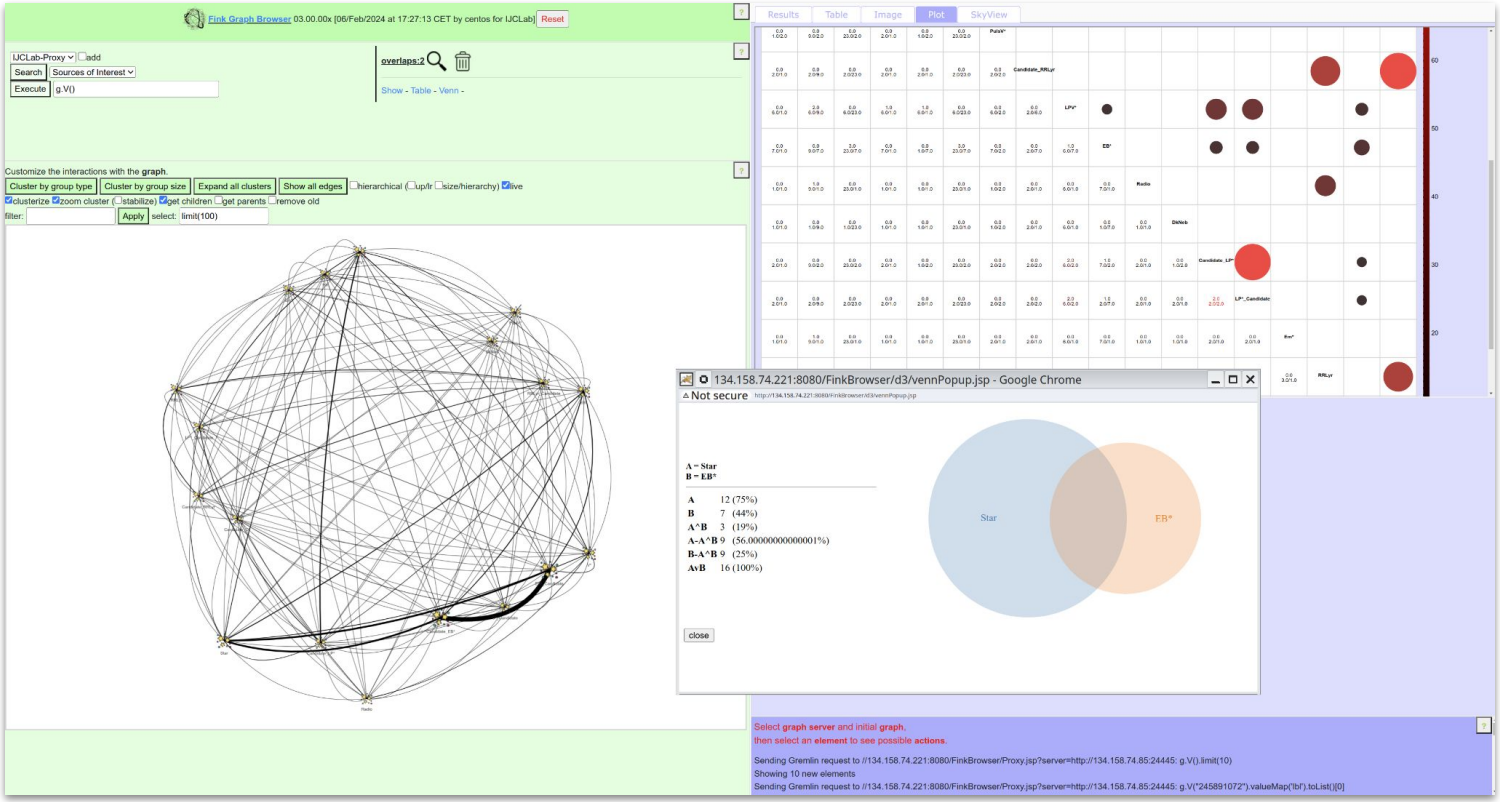
Re-classification

- FEATURES classification is able to reproduce (to some extent) classification of FINK_PORTAL without directly using it
  - for some classes
  - for groups of classes
    - can try to use AI to find good candidates for classes groups

```
// classification from FINK_PORTAL
gr.classification("ZTF25aaksfzy", 'FINK_PORTAL')
==>[weight:1.0,classifier:FINK_PORTAL,class:Solar System candidate]

// classification from FEATURES (PCA+Clustering)
gr.classification("ZTF25aaksfzy", 'FEATURES')
==>[weight:1.0,classifier:FEATURES,class:FC-3]

// classification from FEATURES interpreted by FINK_PORTAL classification
//   using correlations between two classifications
gr.reclassification('ZTF25aaksfzy', 'FEATURES', 'FINK_PORTAL')
==>Solar System candidate=556.0
==>Tracklet=511.0
==>SN candidate=318.0
==>Early SN Ia candidate=11.0
==>Solar System MPC=8.0
==>Ambiguous=2.0
==>Kilonova candidate=2.0
==>Microlensing candidate=1.0
==>BLLac=1.0
```

Overlaps

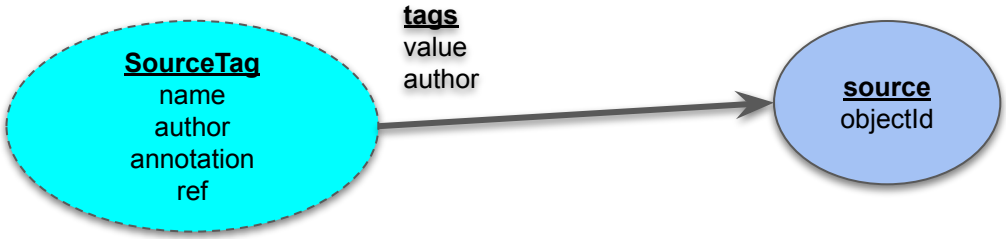
- Within one classification schema
- Or between two classifications



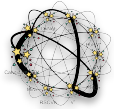
TAGS

- To give user a possibility to tag ‘interesting’ sources
  - To remember them
  - To find similar sources (using multiple classification)
- Creation of Tags (i.e. collection of tagged objects) may require confirmation
  - To prevent multiplication of identical Tags
- We can start from a set of pre-defined Tags.
  - Which ?
- Actual tagging could be open to anyone (it will be signed anyway)
- Tags can contain further annotation
  - With external references ?
- Fink Portal will show tags for each object.
- There will be also a page with overview of existing tags with s possibility to search.
- We may create an alert filter based on Tags + Classification
  - *Send me all new alerts with are classified in the same way as alerts tagged by me as X !*
    How are they tagged by others ?

to discuss



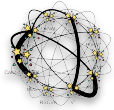
TAGS



to discuss

- Do we need both AlertsTag and SourcesTag ?
- Should be tag values somehow managed (normalised) ?
- Should it be possible to tag one object with several tags at the same time (with weights) ?
- Should we have a possibility to automatically tag certain objects (based on filters,...) ?
- Should more sophisticated searches be exposed via WS or just in a API/CLI ?
  - Show me, how are objects tagged as X classified !
- We may create a filter based on Tags + Classification

CLI



- We can give CLI (python) access to all that functionality to (some) users
  - We can protect data by blocking `commit()`

```
#!/usr/bin/expect
pawn ssh -L 25333:localhost:25333 my_id@server_ip
expect "Enter passphrase for key '/blabla/.ssh/id_rsa':"
send "***\r"
expect "my_id@server_ip"
send "java -jar /blabla/Lomikel-py4j-03.07.00x.exe.jar\r"
interact
```

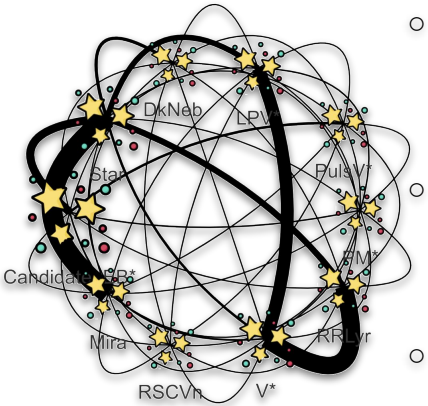
```
from py4j.java_gateway import JavaGateway
gateway = JavaGateway()
jc = gateway.jvm.com.Lomikel.Januser.JanusClient("/blabla/IJCLab.properties");
gr = gateway.jvm.com.astrolabsoftware.FinkBrowser.Januser.FinkGremlinRecipiesG(jc);
results = gr.sourceNeighborhood('ZTF25aaqyygm', 'FINK_PORTAL', 0.5)
print(results)
```

```
$ startServer.expect
```

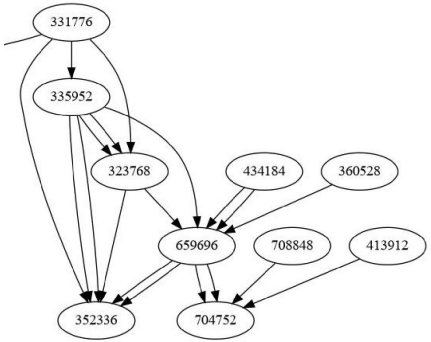
```
$ python n.py
{
JavaObject id=o5: {'SN candidate': 0.7857142857142857, 'Early SN Ia candidate': 0.21428571428571427},
JavaObject id=o10: {'SN candidate': 0.7647058823529411, 'Early SN Ia candidate': 0.23529411764705882},
JavaObject id=o15: {'SN candidate': 0.8, 'Early SN Ia candidate': 0.2},
JavaObject id=o20: {'SN candidate': 0.75, 'Early SN Ia candidate': 0.25},
...}
```

```
// classify not yet classified source
gr.classifySource(Classifiers.FEATURES, oid)
// tag a source
gr.registerSourcesOfInterest(Classifiers.TAG, 'MyTag_1', oid, 1.0)
// get existing classification
gr.classification(oid);
gr.classification(oid, "FINK_PORTAL");
gr.classification(oid, "FEATURES");
// interpret classification as another one
gr.reclassification(oid, "FEATURES", "FINK_PORTAL", 0.5, 'AlertsOfInterest');
// get similar sources
gr.sourceNeighborhood(oid, "FINK_PORTAL", 0.5, 'JensenShannon');
// etc (all those calls are just wrappers over Graph queries)
```

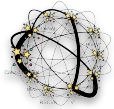
Status



- Multiclassification infrastructure
  - in place, running
- Classification schemas:
  - FINK_PORTAL import
    - nightly import of new alerts (currently about 250 000 sources, 8 000 000 alerts available)
    - if a source is missing (not imported), a user can very simply execute its import and trigger structure creation
  - FEATURES
    - trained with ‘known’ alerts of 2025
    - training further tuned
    - all newly imported sources automatically classified
    - good results for some (groups of) classes
  - LIGHT_CURVES
    - trained with ‘good’ light curves of 2025
    - training tuned
    - not yet included in system
    - similar results as FEATURES
  - CUTOUTS
    - not yet started
  - TAGS
    - acquiring needs/requirements
    - developing CLI, GUI/WS
  - **Any other (existing) schema can be easily included** (push or pull)



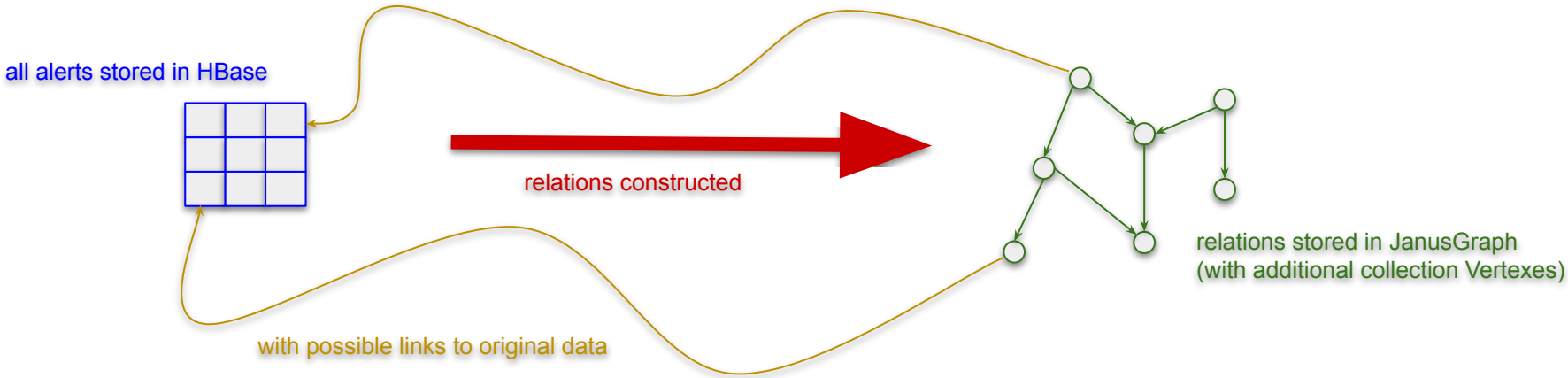
- Access:
  - CLI: Groovy, Python client in machines with access to HBase server; remote access can be easily provided
  - GUI/WS: ready to be integrated in Fink Portal



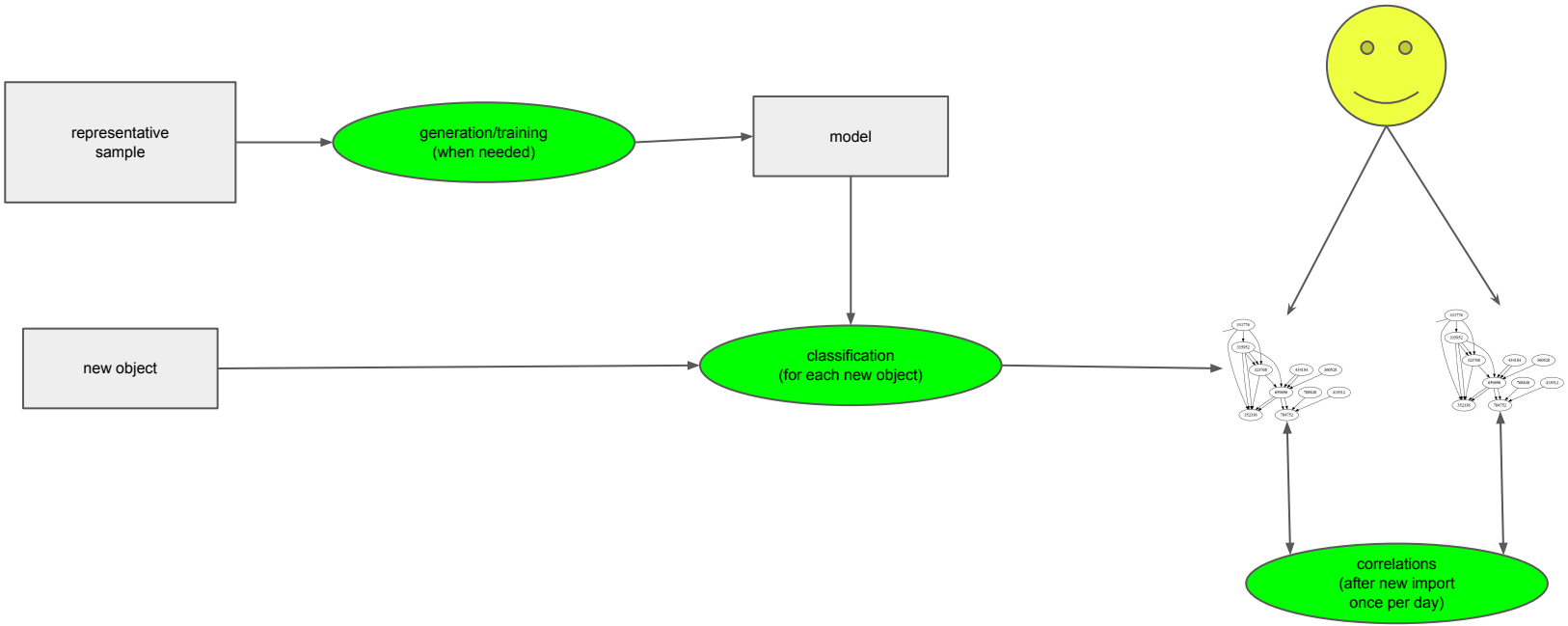
Backup Slides

Constructing Classification Graph

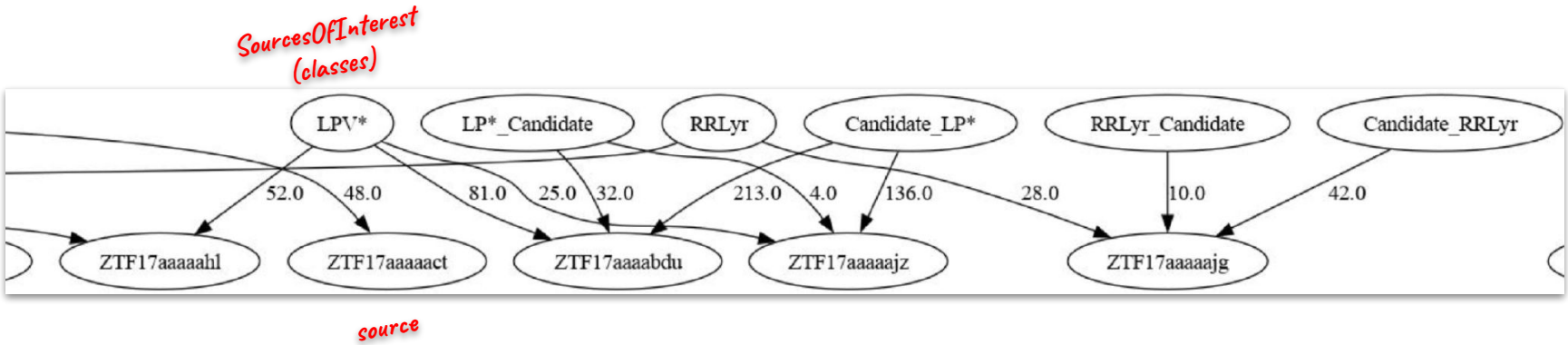
- Import new sources (or update)
  - select interesting (latest, classified as anomaly,...) sources (from HBase or Fink Portal)
  - import them into graph
  - expand to alerts (optional)
  - fill with properties from HBase (optional)
  - construct SoI and AoI collections for alert classes (using classifications)
- Calculate all overlaps (after new imports, cca 1 min)



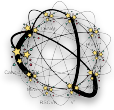
Proces


representative sample
generation/training (when needed)
model
new object
classification (for each new object)
correlations (after new import once per day)

The Classification Graph (exported to GraphViz)



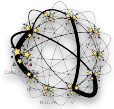
Access to Overlaps



```
jc = JanusClient('IJCLab.properties');
gr = FinkGremlinRecipiesG(jc);
print(gr.overlaps('AlertsOfInterest', 'FINK_PORTAL'));
```

```
AlertsOfInterest:RRLyr * AlertsOfInterest:RRLyr=3666076.0
AlertsOfInterest:V* * AlertsOfInterest:V*=1872160.0
AlertsOfInterest:RRLyr * AlertsOfInterest:Star=1801680.0
AlertsOfInterest:EB* * AlertsOfInterest:EB*=1160366.0
AlertsOfInterest:QSO * AlertsOfInterest:QSO=1082817.0
AlertsOfInterest:LPV* * AlertsOfInterest:LPV*=1079266.0
AlertsOfInterest:Candidate_EB* * AlertsOfInterest:Candidate_EB*=1020285.0
…
AlertsOfInterest:Star * AlertsOfInterest:LPV*=189870.0
AlertsOfInterest:Mira * AlertsOfInterest:LPV*=184988.0
AlertsOfInterest:PulsV* * AlertsOfInterest:PulsV*=184247.0
AlertsOfInterest:RRLyr * AlertsOfInterest:LPV*=179521.0
AlertsOfInterest:LP*_Candidate * AlertsOfInterest:V*=175345.0
AlertsOfInterest:RRLyr * AlertsOfInterest:V*=173963.0
AlertsOfInterest:RRLyr * AlertsOfInterest:GinCl=172219.0
AlertsOfInterest:V* * AlertsOfInterest:RRLyr=162798.0
AlertsOfInterest:LPV* * AlertsOfInterest:Mira=158251.0
…
```

Exporting Overlaps (to GraphViz)

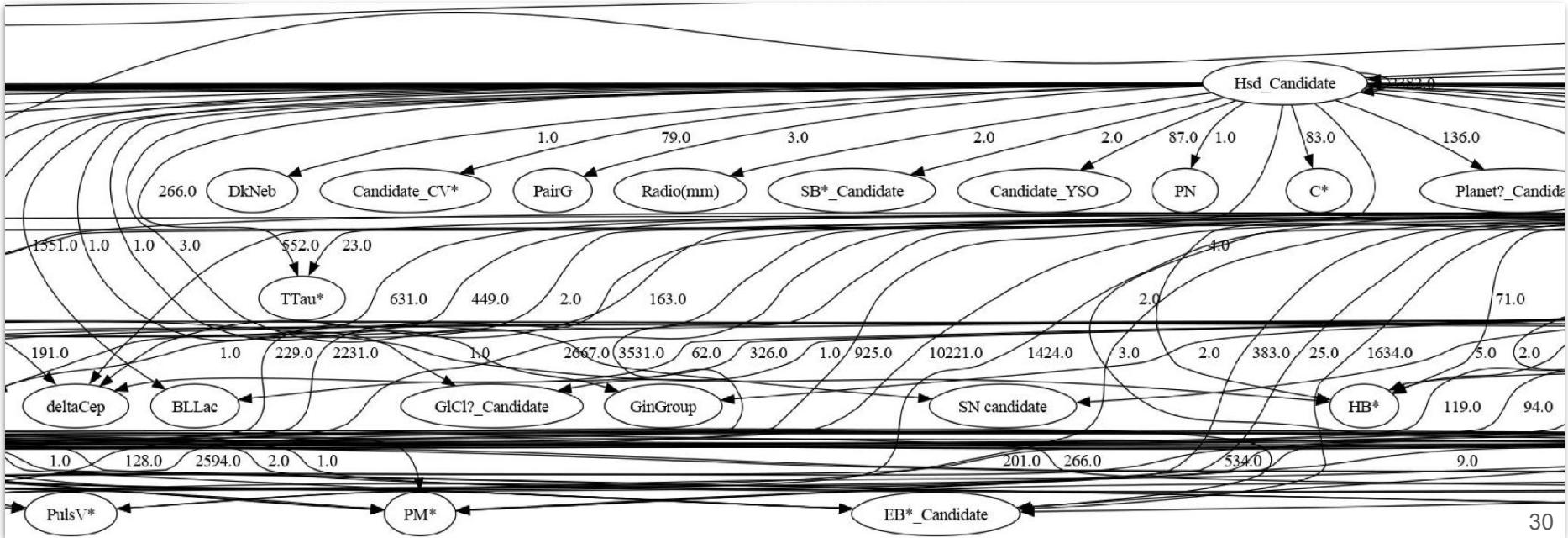


script

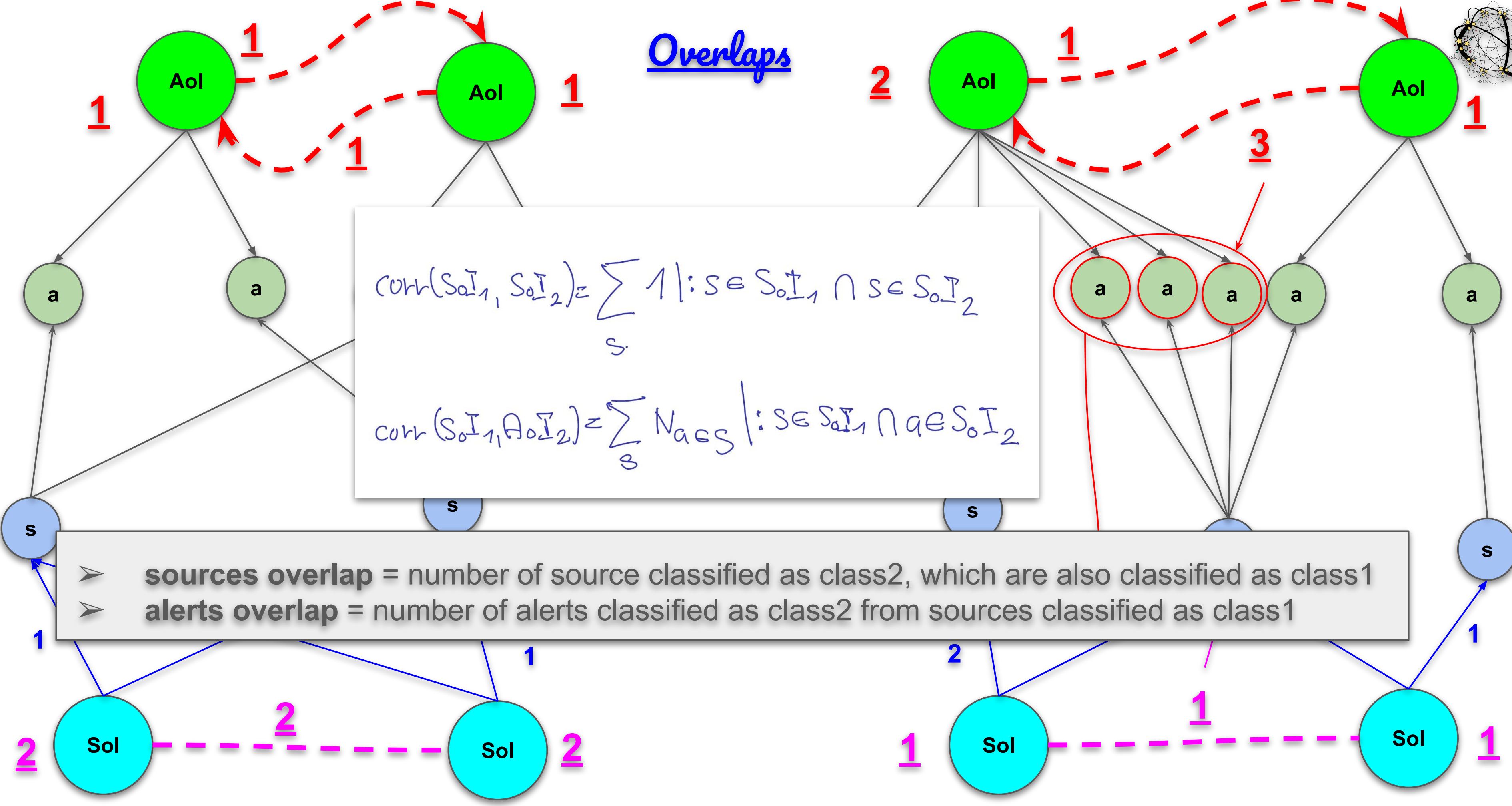
```
jc = JanusClient("IJCLab.properties");
gr = FinkGremlinRecipiesG(jc);
gr.exportAoISoI("/tmp/overlaps.graphml");
```

conversion

```
> grapher -i overlaps.graphml -o overlaps.dot
```



Overlaps

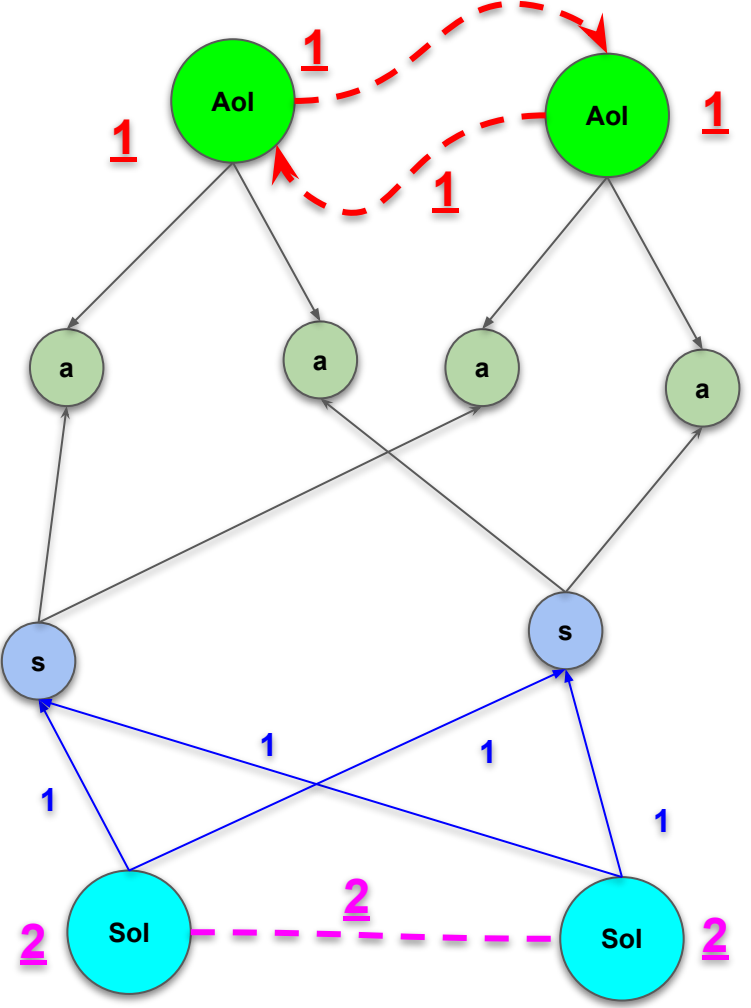


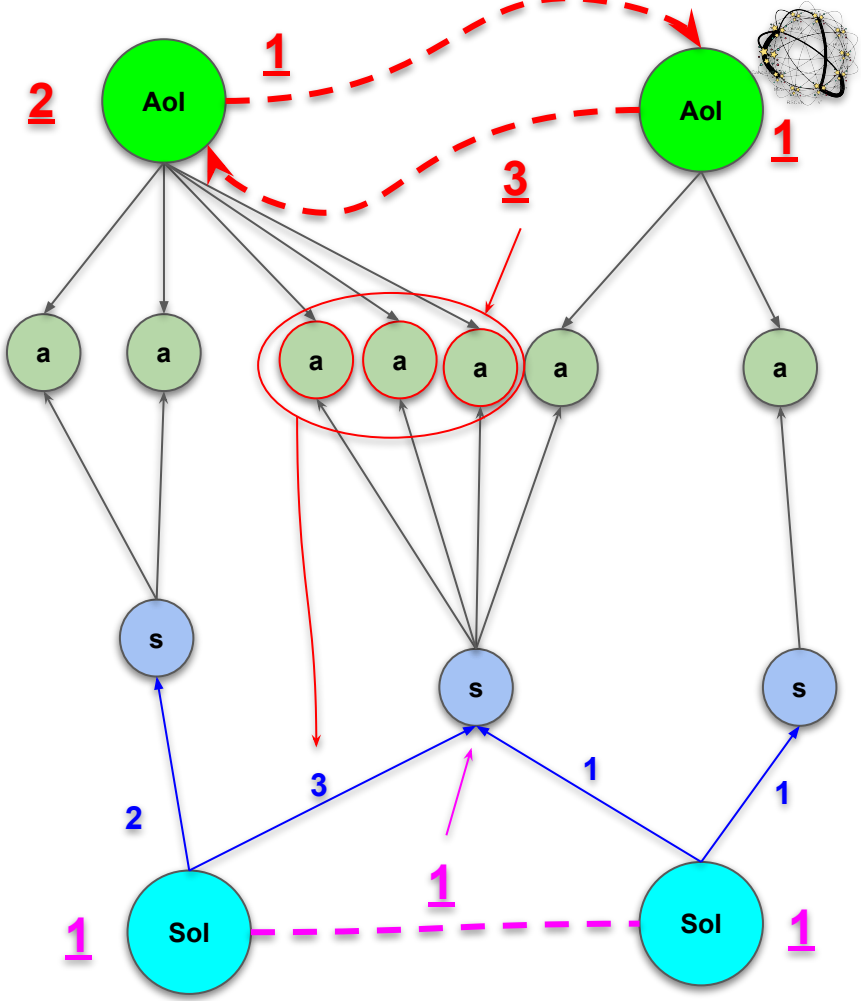
$$corr(SoI_1, SoI_2) = \sum_{s} 1 \,|\, : s \in SoI_1 \cap s \in SoI_2$$

$$corr(SoI_1, AoI_2) = \sum_{s} N_{a \in S} \,|\, : s \in SoI_1 \cap a \in SoI_2$$

- **sources overlap** = number of source classified as class2, which are also classified as class1
- **alerts overlap** = number of alerts classified as class2 from sources classified as class1

Overlaps



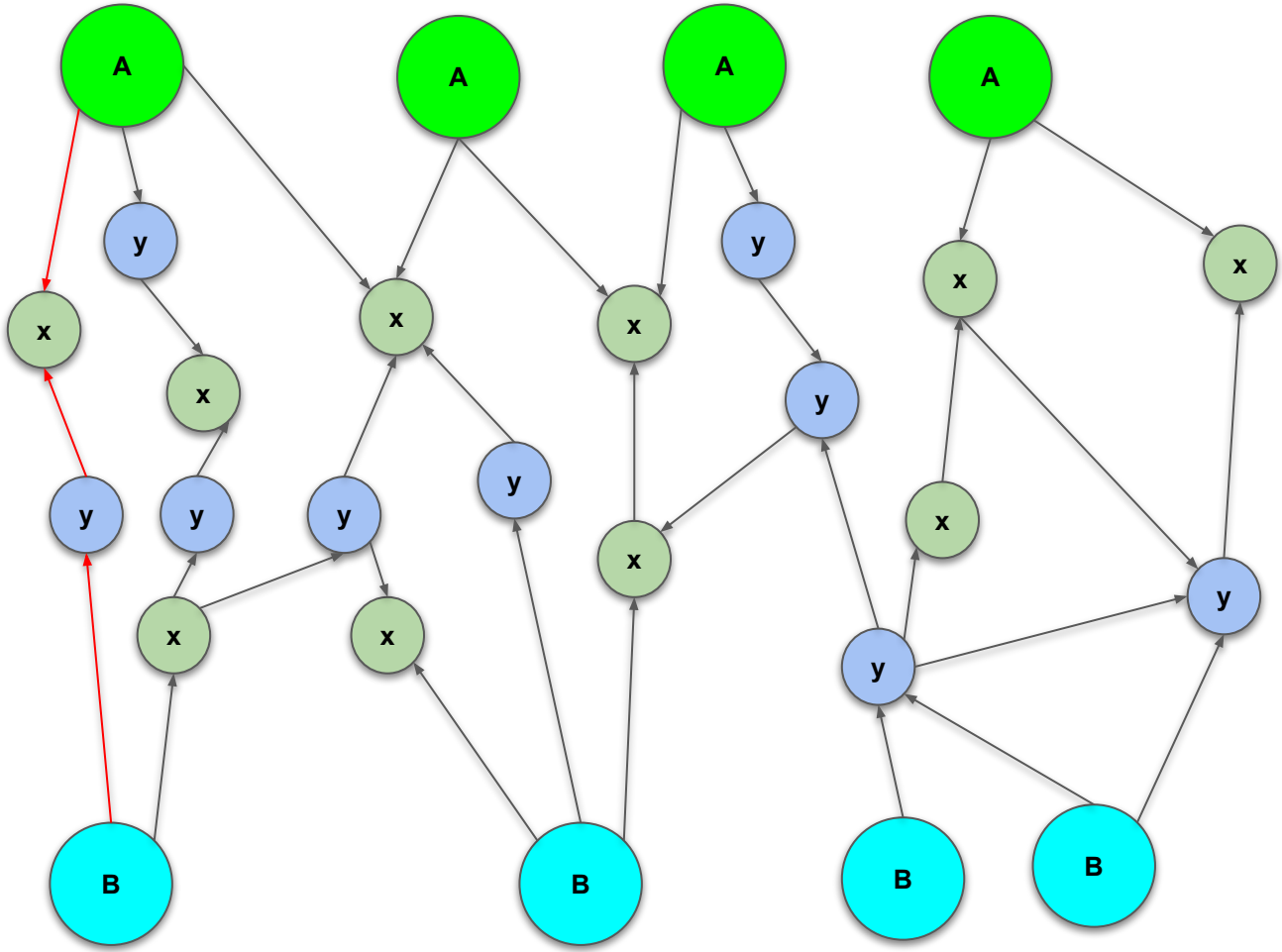


weight = number of **a** connecting **Sol** and **Aol** via **s**

number of **s** common to two **Sol**

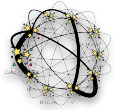
number of **a** from the other **Sol**

Generalisation of Overlaps



- find overlaps between A and B with respect to z
- how to take into account Edge weights ?
- how to include multiple-paths to the same x ?
- should work for A == B
- should work for oriented and unoriented Edges

Analysing Exported Overlaps



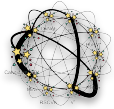
script

```
> grapher -s analyse.groovy
import com.Grapher.Convertors.Convertor;
import com.Grapher.Analysis.Analyser;

cli.setInfile("overlaps.graphml");
convertor = new Convertor(cli);

analyser = new Analyser(cli);
analyser.fill(convertor.read());
analyser.applyConnectivity(10);
```

result

```
Grapher initialised, version: 01.01.00x [23/Apr/2024 at 11:00:43 CEST by hrivnac]
Executing Groovy analyse.groovy ...
Reading overlaps.graphml
Generating weights ...
Imported graph: DefaultGraphType [directed=true, undirected=false, self-loops=true,
multiple-edges=false, weighted=true, allows-cycles=true, modifiable=true][211,
7503] from SoI.graphml
Applying Connectivity Algorithm ...
Most Connected:
    SourcesOfInterest(532525064):Solar System MPC=2.001475751813528E-5
    SourcesOfInterest(122896456):Candidate_EB*=1.9299584561486104E-4
    SourcesOfInterest(246046768):Candidate_RRLyr=2.1192136461512155E-4
    SourcesOfInterest(245780528):EB*_Candidate=2.1717508026100228E-4
    SourcesOfInterest(163864680):RRLyr_Candidate=2.2113304520797968E-4
    SourcesOfInterest(532557832):Candidate_LP*=2.5487725838755677E-4
    SourcesOfInterest(245788784):V*=2.6911282233468034E-4
    SourcesOfInterest(245780592):Mira=2.96411720247869E-4
    SourcesOfInterest(245776432):LP*_Candidate=3.045124013188856E-4
    SourcesOfInterest(532500488):LPV*=3.1365731058502976E-4
Least Connected:
    SourcesOfInterest(670097464):Candidate_post-AGB*=0.16667572451461254
    SourcesOfInterest(332038232):WR*_Candidate=0.187895248349279
    SourcesOfInterest(291020976):Candidate_WR*=0.187895248349279
    SourcesOfInterest(862470184):Possible_GrG=0.19446988682381575
    SourcesOfInterest(179531848):Candidate_SG*=0.1952614379084967
    SourcesOfInterest(453677288):SFregion=0.25003816356905695
    SourcesOfInterest(499388640):Candidate_LMXB=0.3333394893009283
    SourcesOfInterest(167751752):LMXB_Candidate=0.3333394893009283
    SourcesOfInterest(616284304):Candidate_LensSystem=0.34090909090909094
    SourcesOfInterest(177287240):Candidate_RSG*=0.35
```

Analysing Exported Overlaps



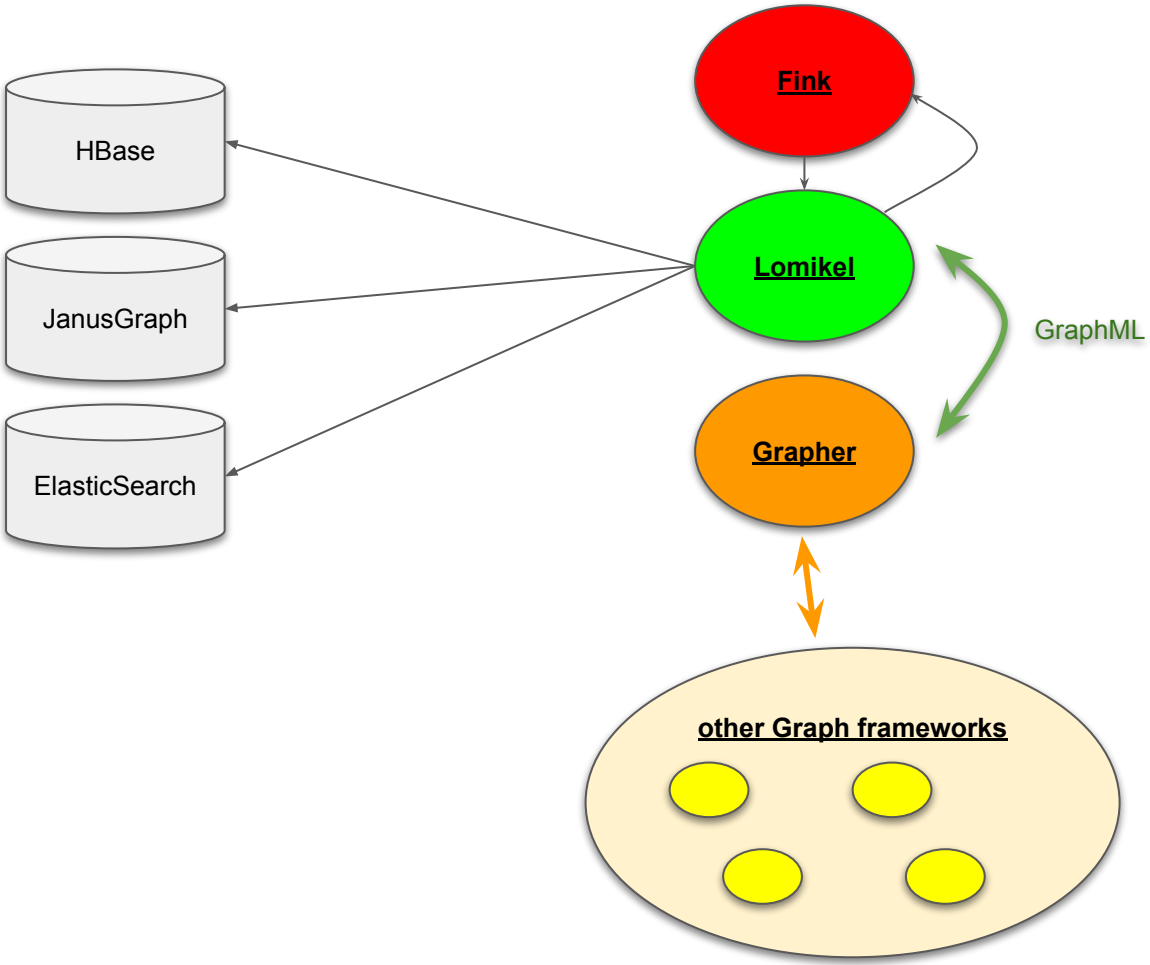
script

```
> grapher -s analyse.groovy
import com.Grapher.Convertors.Convertor;
import com.Grapher.Analysis.Analyser;

cli.setInfile("overlaps.graphml");
convertor = new Convertor(cli);

analyser = new Analyser(cli);
analyser.fill(convertor.read());
analyser.applyClustering("GirvanNewman", 10);
```

result

```
Grapher initialised, version: 01.01.00x [23/Apr/2024 at 11:00:43 CEST by
hrivnac]
Executing Groovy analyse.groovy ...
Reading overlaps.graphml
Generating weights ...
Imported graph: DefaultGraphType [directed=true, undirected=false,
self-loops=true, multiple-edges=false, weighted=true, allows-cycles=true,
modifiable=true][211, 7503] from SoI.graphml
Applying Clustering Algorithm ...
    usingt GirvanNewman algoritm
    searching for 10 clusters
Clusters:
    ...
    [SourcesOfInterest(81973328):Early SN Ia candidate,
SourcesOfInterest(122953800):Solar System candidate,
SourcesOfInterest(163905752):Ambiguous,
SourcesOfInterest(164077672):Kilonova candidate]
    [SourcesOfInterest(499388640):Candidate_LMXB,
SourcesOfInterest(167751752):LMXB_Candidate]
    ...
```

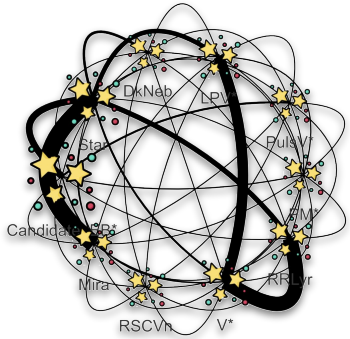
Ecosystem



- Lomikel, Grapher exist as
  - standalone programs, scriptable in Java, Python, Groovy
  - libraries to be included in other Java, Python, Groovy programs
  - interactive Web Service

How To

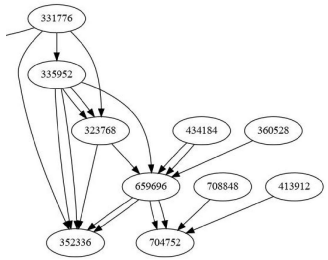
execution




```
# download Lomikel-Janus.exe.jar
# and
# jython.jar (if Python scripting is required)
# if Python scripting is not required:
$ alias lomikel='java -jar Lomikel-Janus.exe.jar'
# if Python scripting is required:
$ alias lomikel='java -cp Lomikel-Janus.exe.jar:jython.jar com.Lomikel.Apps.LUC'
$ lomikel -h
usage: java -jar Lomikel.exe.jar [-a ] [-b] [-g] [-h] [-n] [-q] [-s <file>] [-w]
-a,--api <language> cli language: [bsh|groovy|python] (othewise taken from source extension, defauld is
groovy)
-b,--batch          run in a batch
-g,--gui            run in a graphical window
-h,--help           show help
-n,--notebook       run in an notebook
-q,--quiet          minimal direct feedback
-s,--source <file>  source script file (init. is also sourced)
-w,--web            run a
```

Lomikel

execution

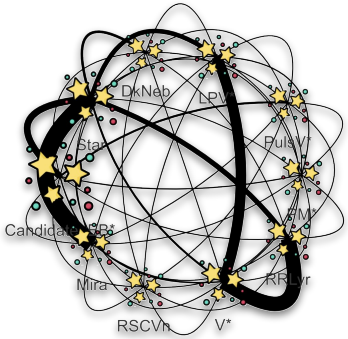



```
# download Grapher.exe.jar
$ alias grapher='java -jar Grapher.jar'
$ grapher -h
usage: java -jar Grapher.exe.jar
 -a,--alg   apply algorithm (instead of just converting)
                      [sc = Strong Connectivity | cl = Clustering | ad = adding distances]]
                      several algorithms can be separated by ;
                      algorithm arguments can be supplied after ,
 -e,--noedge          ignore input edges
 -h,--help            show help
 -i,--in              input file name [.graphml]
 -o,--out             output file name [.dot|.mat|.g6|csv|json|graphml]
 -q,--quiet           minimal direct feedback
 -s,--script          script to run (ignores all other options) [.groovy|.py]
 -v,--novertex        ignore output edge-less vertices
 -w,--show            show in graphical window (instead of converting)
```

Grapher

some Lomikel actions
require direct access to
HBase/JanusGraph/ElasticSearch databases

Lomikel Script

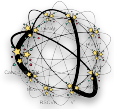


script

```
import com.Lomikel.Januser.JanusClient;
import com.astrolabsoftware.FinkBrowser.Januser.FinkGremlinRecipiesG;

// connect to JanusGraph database
jc = new JanusClient("IJCLab.properties");
// access specific Fink utilities
gr = new FinkGremlinRecipiesG(jc);
// execute Gremlin (graph database API) request
jc.g().V().limit(1).valueMap().next();
// find closest sources
gr.sourceNeighborhood('ZTF17aaawgky', null, null, 10);
// get source classification
gr.classification("ZTF17aaawgky"));
// get all overlaps
gr.overlaps())
// do some statistics
gr.standardDeviationE('deepcontains', ['weight']);
// export overlaps to GraphML file
gr.exportAoISoI('Overlaps.graphml');
```

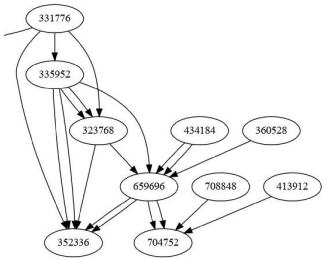
Lomikel

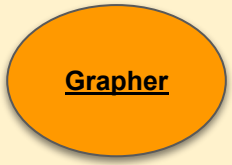
Grapher Script



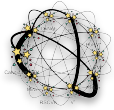
script

```
import com.Grapher.Convertors.Convertor;
Import com.Grapher.Analysis.Analyser;

// convert GraphML file into GraphViz Dot file
cli.setInfile("AoI.graphml");
cli.setOutfile("AoI.dot");
convertor = new Convertor(cli);
convertor.read();
convertor.convert();
// fill data into Analyser
analyser = new Analyser(cli);
analyser.fill(convertor.read());
// apply various Graph algorithms
analyser.applyStrongConnectivity();
analyser.applyConnectivity(10);
analyser.applyClustering("GirvanNewman", 30);
analyser.applyClustering("LabelPropagation", 30);
analyser.applyClustering("KSpanningTree", 30);
```





HowTo



```
// on the machine, which it has access to HBase server:
//   download Lomikel
// start the server
java -jar ../lib/Lomikel-py4j-03.07.00x.exe.jar


com.Grapher.Convertors.Convertor;
Import com.Grapher.Analysis.Analyser;

// convert GraphML file into GraphViz Dot file
cli.setInfile("AoI.graphml");
cli.setOutfile("AoI.dot");
convertor = new Convertor(cli);
convertor.read();
convertor.convert();
// fill data into Analyser
analyser = new Analyser(cli);
analyser.fill(convertor.read());
// apply various Graph algorithms
analyser.applyStrongConnectivity();
analyser.applyConnectivity(10);
analyser.applyClustering("GirvanNewman", 30);
analyser.applyClustering("LabelPropagation", 30);
analyser.applyClustering("KSpanningTree", 30);
```

Other Graph Frameworks