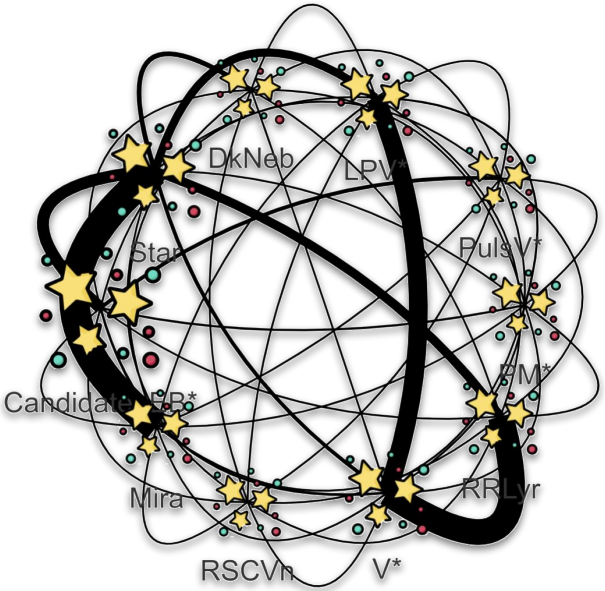


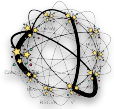
Classification Graphs to organise alerts

Julius Hrivnac, IJCLab

FINK@World

8-12/6/26

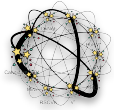
Orsay



Project

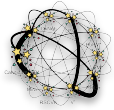
- To organize access to alerts allowing **similarity** search and navigation
  - **Vector databases** functionality
- To be flexible about what **similarity** means (its defining object properties) and how to evaluate it (which **metric** to use)
- Organizing **classifications** in **graphs**
- Support for multiple classifications schemas:
  - **survey**: ZTF, LSST, ANY
  - **classifier**: FINK, XMATCH (for ZTF), …, TAG (special)
  - **flavour**: describing model, training data, hyperparameters,...
- Objects are assigned to classes of classifiers
- Can compare objects within one classifier or between classifiers
  - With defined distance **specification** and **metric**
- Special classifier: TAG
  - Assigned individually (by users), but interoperable with other classifiers
  - Each tag has:
    - **Class**: assigned by admin
    - **Name**: assigned by user
    - **Weight**
    - **User**: email (?)
    - **Annotation**: free text
    - **Reference**: url pointing to external source (may be database url)

CLI Example - LSST (1)



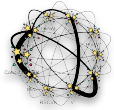
```
// just take one object
oid = '313985349745377418'

// how it is classified (there is just one classifier so far: 'FINK')
gr.classification(oid)
==>[weight:0.5714285714285714,classifier:FINK,flavor:,class:rubin.tag_early_snia_candidate]
==>[weight:0.42857142857142855,classifier:FINK,flavor:,class:rubin.tag_sn_near_galaxy_candidate]

// what are the most similar objects
gr.objectNeighborhood(oid, 'FINK', 10, 'JensenShannon')
==>313888627059851362=0.0={rubin.tag_early_snia_candidate=0.5714285714285714,
rubin.tag_sn_near_galaxy_candidate=0.42857142857142855}
==>313998539864670258=0.0={rubin.tag_early_snia_candidate=0.5714285714285714,
rubin.tag_sn_near_galaxy_candidate=0.42857142857142855}
==>313972153277481152=0.020508231493818984={rubin.tag_early_snia_candidate=0.6, rubin.tag_sn_near_galaxy_candidate=0.4}
==>313980945208705140=0.05065909441930129={rubin.tag_early_snia_candidate=0.5, rubin.tag_sn_near_galaxy_candidate=0.5}
==>313897383836516483=0.05065909441930129={rubin.tag_early_snia_candidate=0.5, rubin.tag_sn_near_galaxy_candidate=0.5}
==>313871014596444283=0.05065909441930129={rubin.tag_early_snia_candidate=0.5, rubin.tag_sn_near_galaxy_candidate=0.5}
==>313972183142498363=0.05065909441930129={rubin.tag_early_snia_candidate=0.5, rubin.tag_sn_near_galaxy_candidate=0.5}
==>313928194479095810=0.05065909441930129={rubin.tag_early_snia_candidate=0.5, rubin.tag_sn_near_galaxy_candidate=0.5}
==>313664312954060801=0.05065909441930129={rubin.tag_early_snia_candidate=0.5, rubin.tag_sn_near_galaxy_candidate=0.5}
==>170028486134595648=0.05065909441930129={rubin.tag_early_snia_candidate=0.5, rubin.tag_sn_near_galaxy_candidate=0.5}
```

CLI Example - LSST (2)



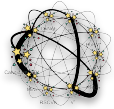
```
// tag oid 'TAG' classifier
// ========================

// get 'TAG' classifier for 'LSST' survey and no flavour
tagClassifier = Classifier.instance('TAG', 'LSST', '')
// create 'MyTag' tag (class) - protected operation (done by admin)
tagClassifier.createTag(gr, 'MyTag')

// tag oid with name='some tag' and weight=1.0
// you could sign it, include annotation and url reference to additional resource
tagClassifier.classify(gr, oid, 'some tag', 1.0, null, null, null)

// how would be that oid classified in 'FINK'
// (it uses information about its classification in 'TAG'
//  and the correlations between classification in 'FINK' and 'TAG' schemas,
//  it doesn't use information how oid is actually classified in 'FINK')
gr.reclassification(oid, 'TAG', 'FINK', 10, true)
==>[classifier:FINK,flavor:,weight:0.5358983848622454,class:rubin.tag_early_snia_candidate]
==>[classifier:FINK,flavor:,weight:0.4641016151377546,class:rubin.tag_sn_near_galaxy_candidate]
```

CLI Example - LSST (3)



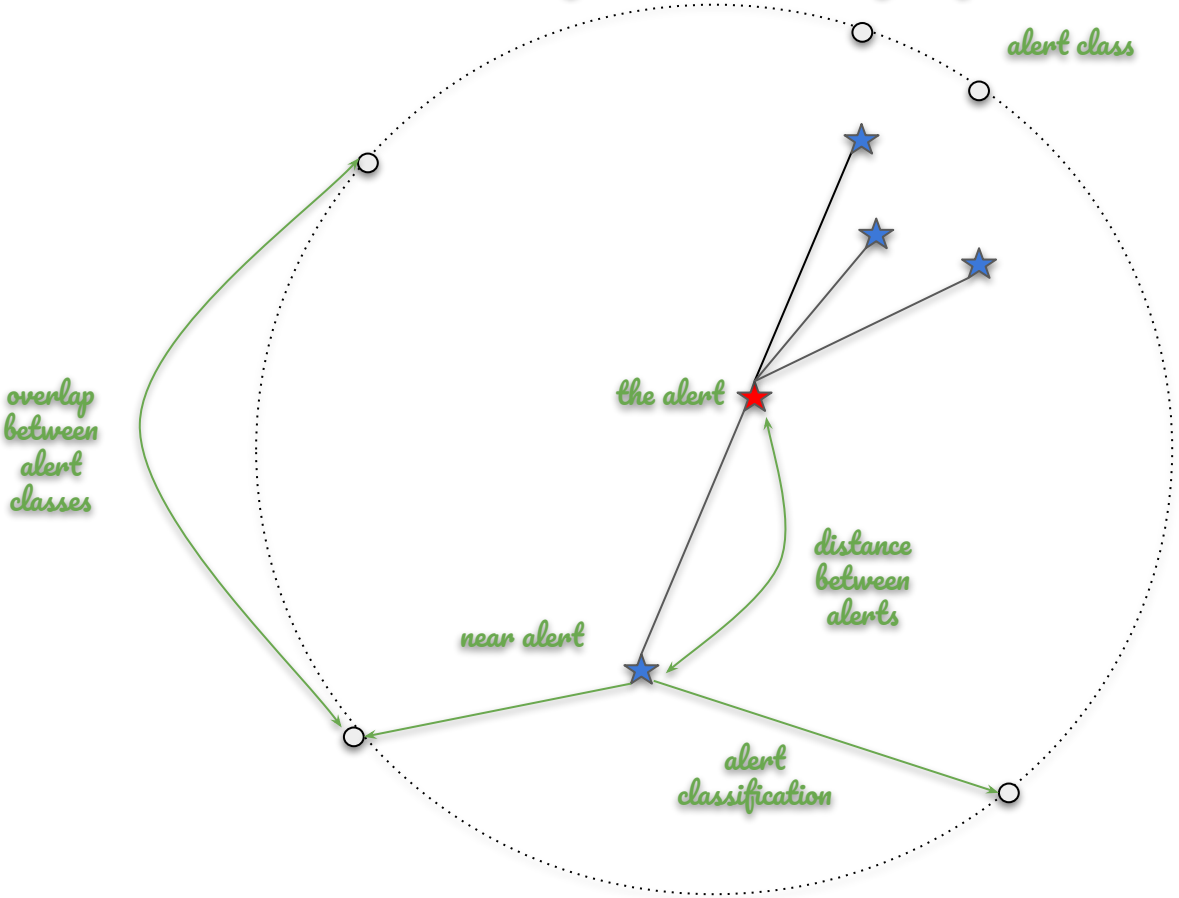
```
// give weighted number of overlaps between classes
gr.overlaps('FINK')
==>OCol:FINK::rubin.tag_extragalactic_new_candidate * OCol:FINK::rubin.tag_extragalactic_new_candidate=180139.12876163903
==>OCol:FINK::rubin.tag_sn_near_galaxy_candidate * OCol:FINK::rubin.tag_sn_near_galaxy_candidate=30896.90397209318
==>OCol:FINK::rubin.tag_sn_near_galaxy_candidate * OCol:FINK::rubin.tag_extragalactic_new_candidate=9200.999084314393
==>OCol:FINK::rubin.tag_extragalactic_new_candidate * OCol:FINK::rubin.tag_sn_near_galaxy_candidate=9200.999084314393
==>OCol:FINK::rubin.tag_uniform_sample * OCol:FINK::rubin.tag_uniform_sample=3750.4540335858887
==>OCol:FINK::rubin.tag_hostless_candidate * OCol:FINK::rubin.tag_hostless_candidate=772.9054000418471
==>OCol:FINK::rubin.tag_hostless_candidate * OCol:FINK::rubin.tag_extragalactic_new_candidate=754.5038281607216
==>OCol:FINK::rubin.tag_extragalactic_new_candidate * OCol:FINK::rubin.tag_hostless_candidate=754.5038281607216
==>OCol:FINK::rubin.tag_in_tns * OCol:FINK::rubin.tag_in_tns=649.690364338746
==>OCol:FINK::rubin.tag_early_snia_candidate * OCol:FINK::rubin.tag_early_snia_candidate=277.4629522470143
...
```

CLI Example - ZTF



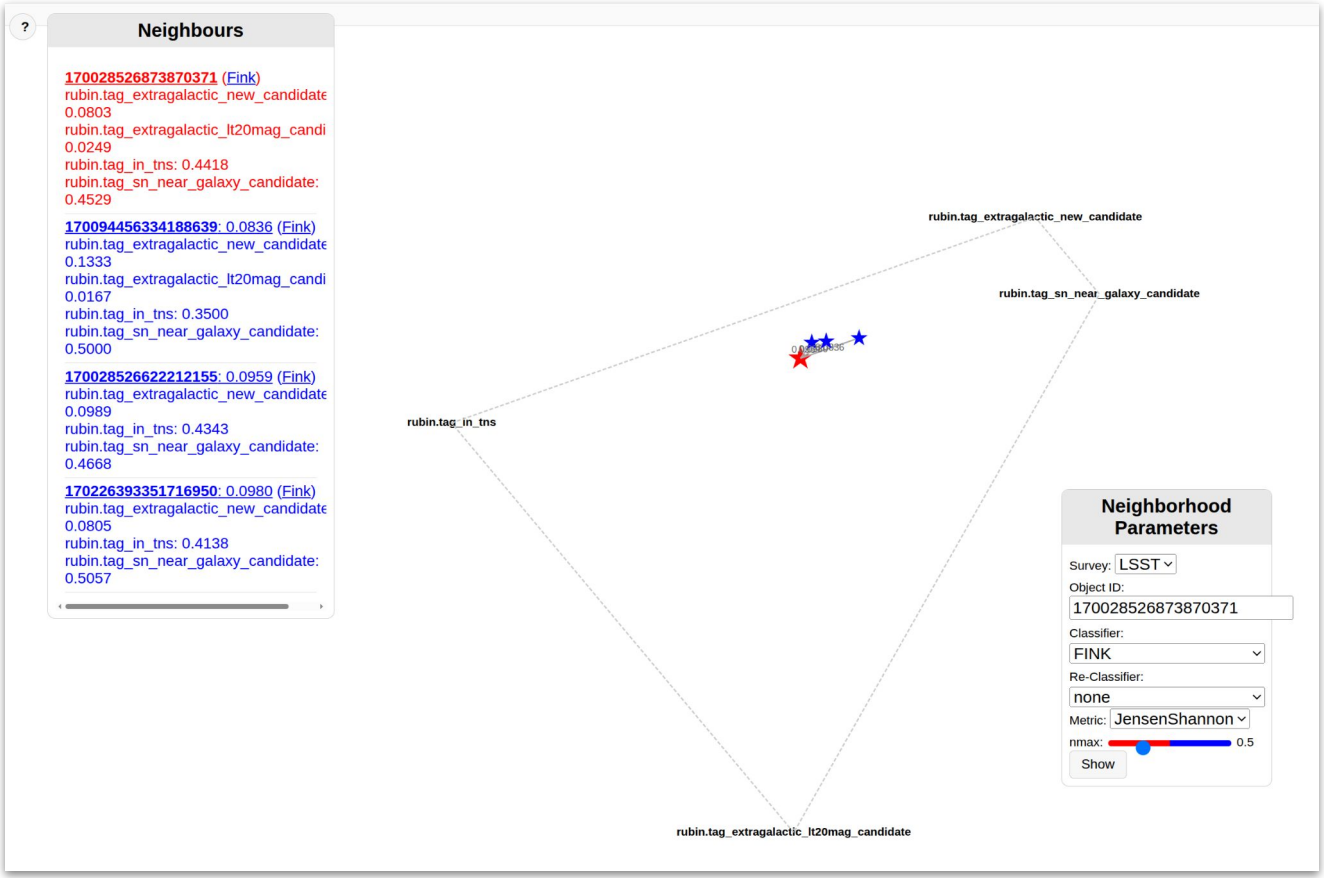
```
// classification from 'FINK_PORTAL' classifier
gr.classification("ZTF25aaksfzy", 'FINK_PORTAL')
==>[weight:1.0,classifier:FINK_PORTAL,class:Solar System candidate]

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

// classification from 'FEATURES' classifier
//   interpreted by 'FINK_PORTAL' classifier
//   using correlations between two classifiers
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
```

Neighborhood Graphical View

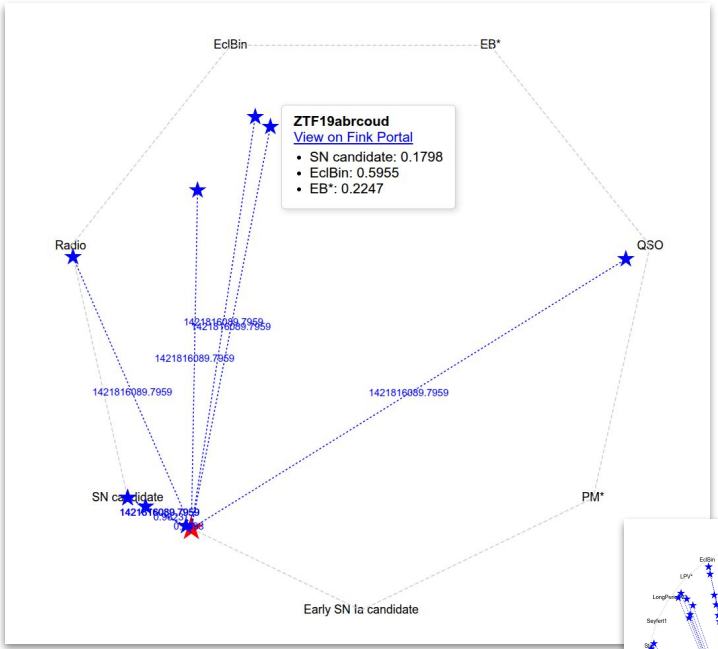


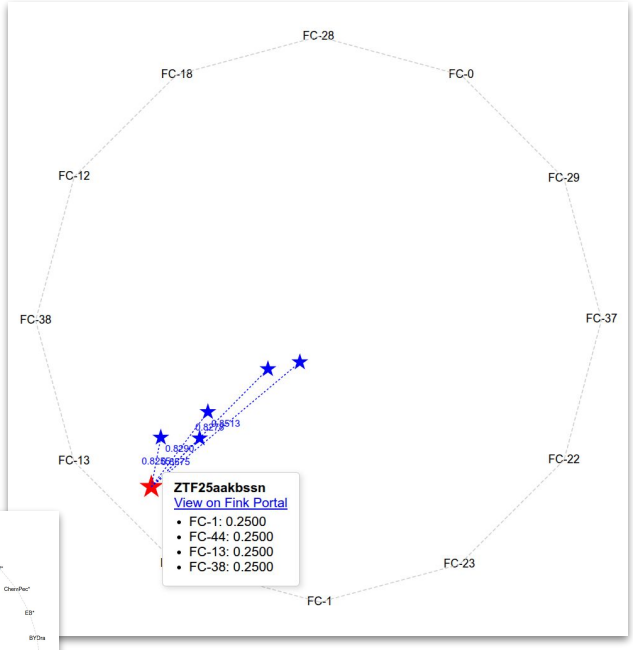
- The distance problem is multi-dimensional
- Its projection into 2-dim flat space is approximative, tuned to be intuitive
  - using AI
- It depends on
  - Chosen classification schema
  - Selected metric
- View is interactive
  - Hovering over an alert will show detailed information, with link to Fink Portal for that alert
  - Clicking on an alert will make this alert central and show its nearests alerts

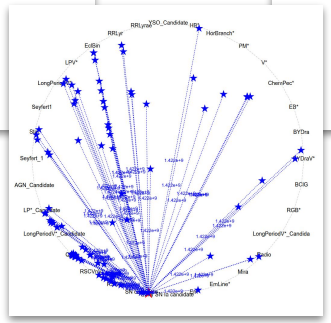
Neighborhood View Example - LSST

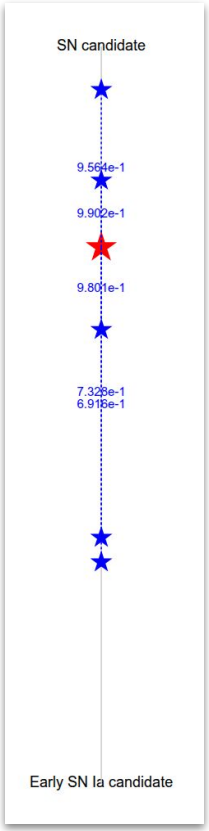


Neighborhood View Example - ZTF





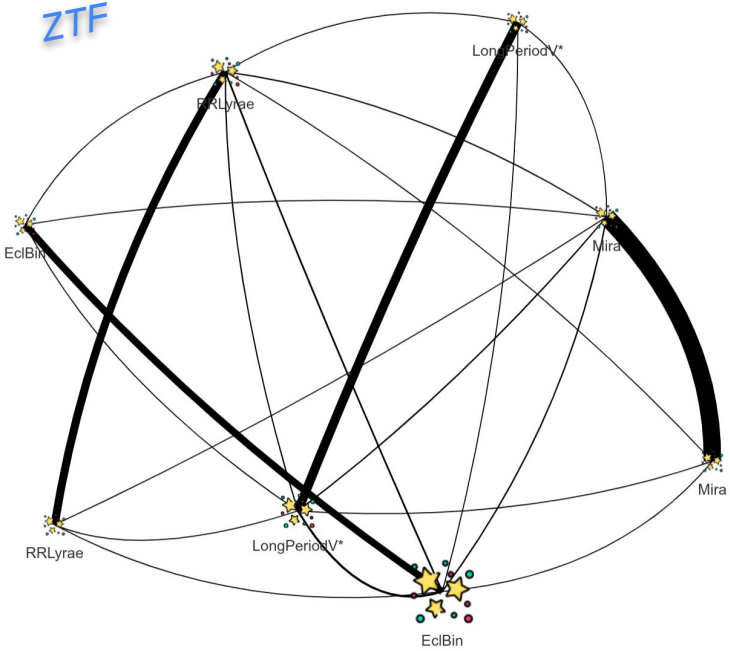




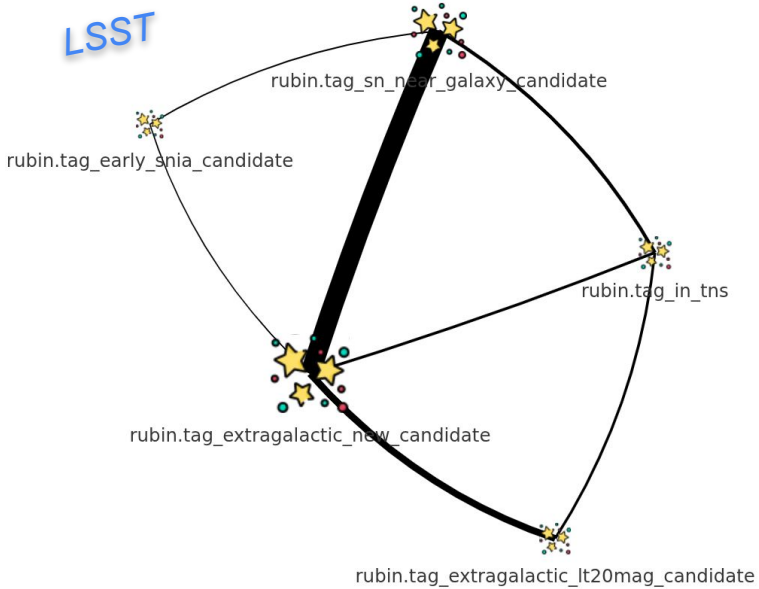
Overlaps View

- Within one classification
- Or between two classifications

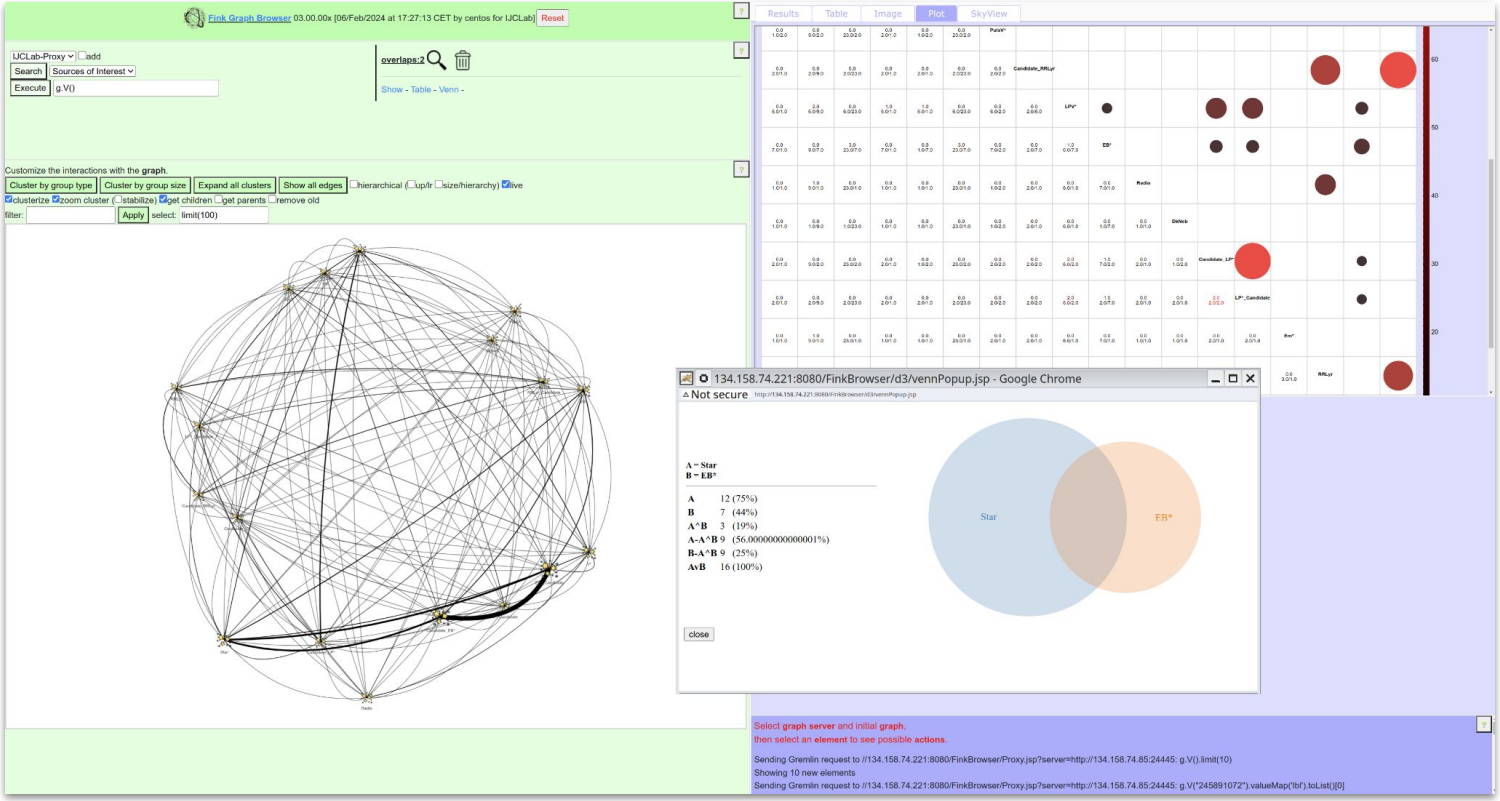
ZTF



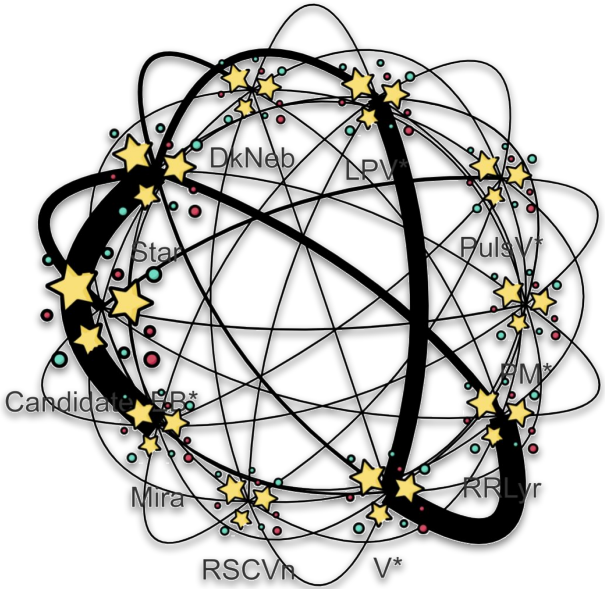
LSST



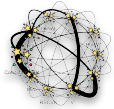
Overlaps View



To be available soon via REST



What kind of functionality to expose ?

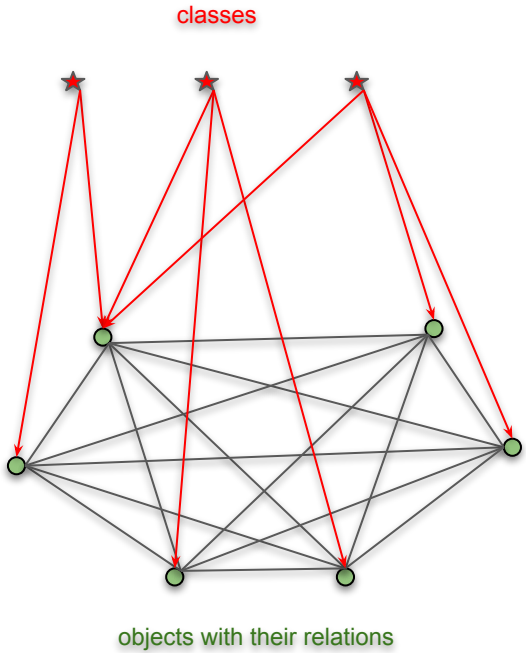


Additional Slides

Simple Classification

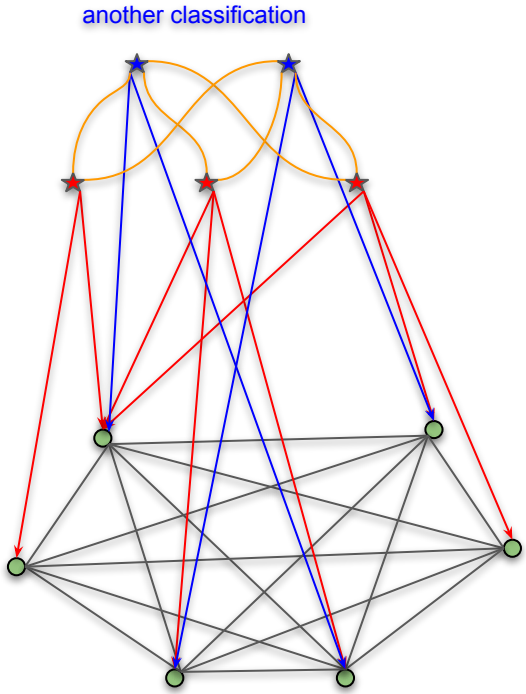
how classification enables similarity search

- How objects classification enables similarity search
  - Without discrete classification, the problem is **`n*n`** `(n = number of objects)`, 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
- Object distance can be calculated by various metrics:
  - **Jensen-Shannon** - the most trustworthy, but slower
  - Euclidean, Cosine,... - faster, less trustworthy

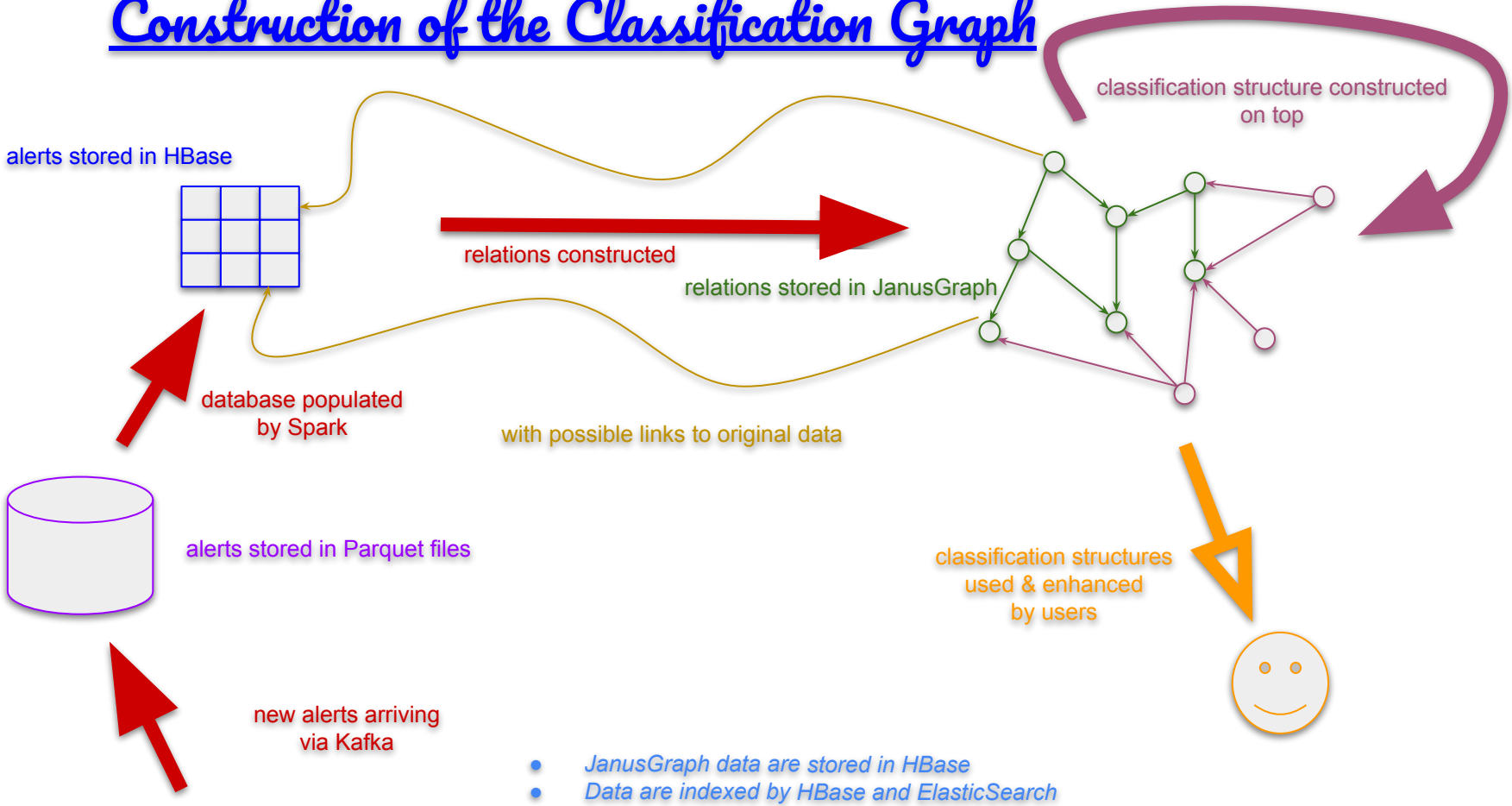


Multiple Classification

- 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 (mostly user-supplied filters)
  - AI: calculated by trained network, PCA,...
  - Tags: supplied individually by users
- Classification schemas can have several ***flavours***
  - from different classification sources or hyperparameters,...
- Classification schemas
  - can be **combined** into *superschemas*
  - can be **divided** into *subschemas*
  - can be organised **hierarchically** (for big amount of data)
- Class Distances
  - Are calculated using modified **Graph Flow** algorithms
    - Often used in recommendation systems
- Once existing objects are classified, we can easily (and quickly):
  - Classify new objects
  - Find overlaps between schemas
  - Comparing truthfulness of schemas
  - Interpreting one schema by another schema

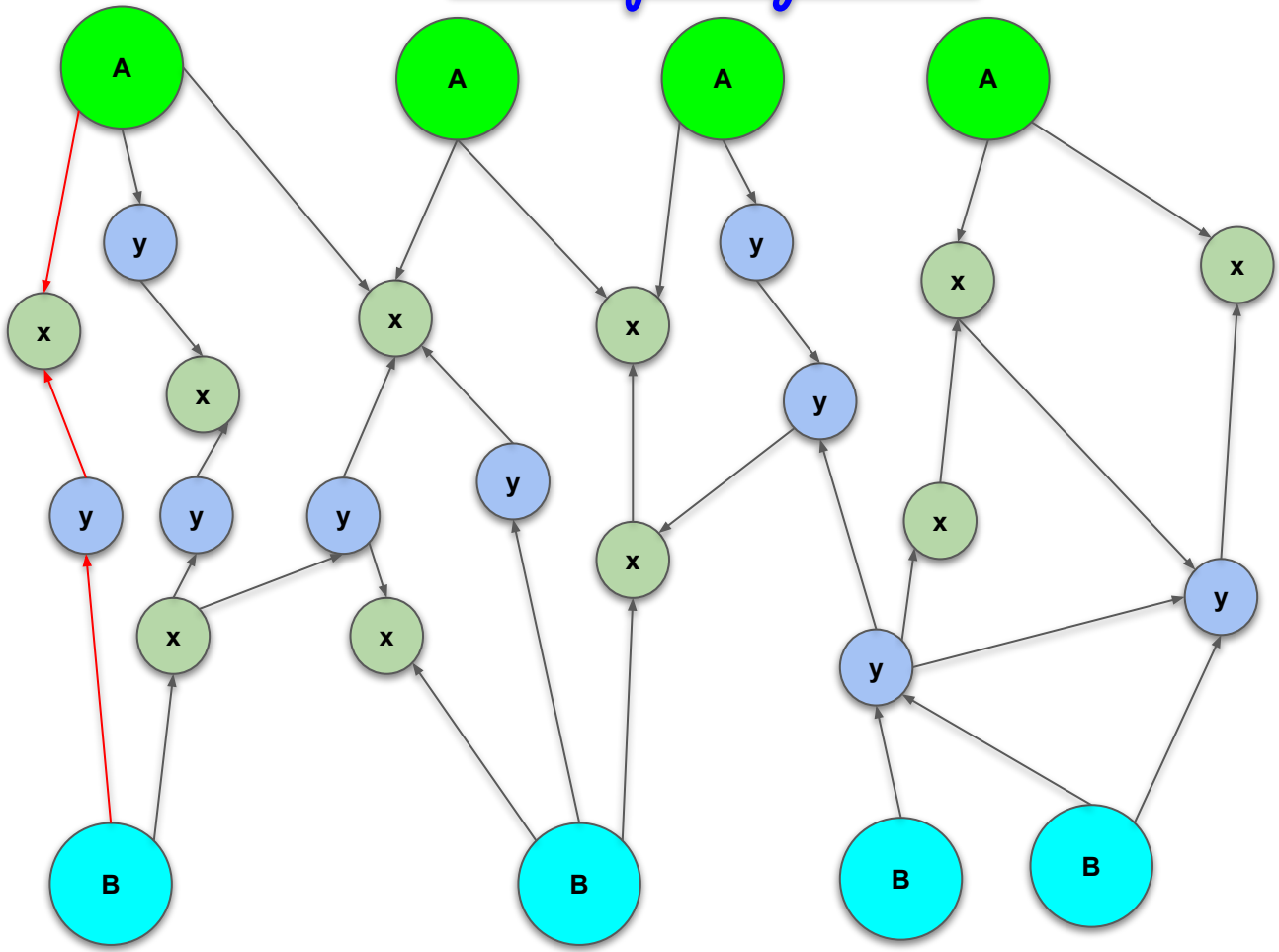


Construction of the Classification Graph



- *JanusGraph data are stored in HBase*
- *Data are indexed by HBase and ElasticSearch reverse indexes*

Overlaps Algorithm



- Finding overlaps between A and B classes with respect to x or y objects
- ‘Similarity’ of classes
- Similar to ‘flow’ algorithm