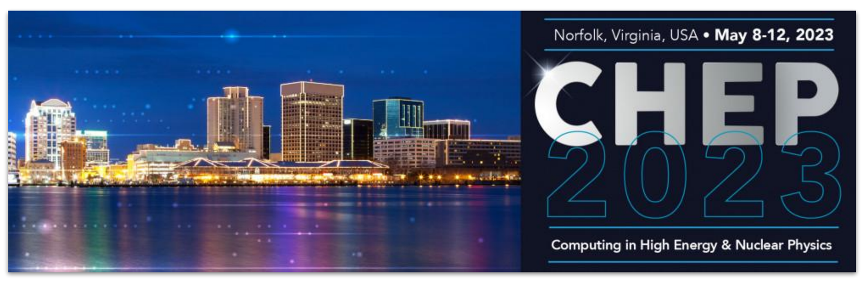
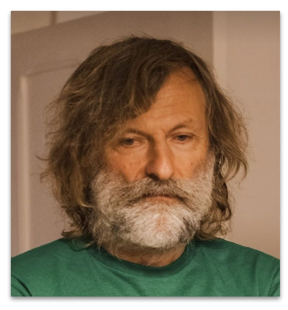


Multidatabase

Julius Hrivnac, Julien Peloton (IJClab Orsay)



Particle Physics experiments and Astronomy telescopes store a large amount of data, most of those data are usually stored in simple files in various format. Databases are used only to a limited extend, while database technology has made a tremendous progress in recent years, offering a large spectrum of applications in all possible domains. Those possibilities are still largely underused.

A lot of ongoing HEP effort to make execution more structured and parallel (Parallel programming, Functional programming).

Less effort (so far) to structure the data: More structured data => simpler and faster access.

A wide spectrum of database technologies exist. A pragmatic approach is needed to use each technology in its domain of strength, combining them in a hybrid architecture.

Hybrid Architecture

- Store unstructured (raw) data in table-like storage (SQL, NoSQL)
  - Suitable for intensive, parallel processing (Spark,...)
  - Interpretable as **Datagrams**-like apis
- Express persistent **structure** as a **Graph**
- Allow for ad-hoc (a'priory volatile) Graph relations
  - Possibly in separed (but connected) graphs
    - Playgrounds, Whiteboards,...
- Connect everything behind the **common API**

=

SQL

- Efficient for well structured homogeneous data
- Limited flexibility

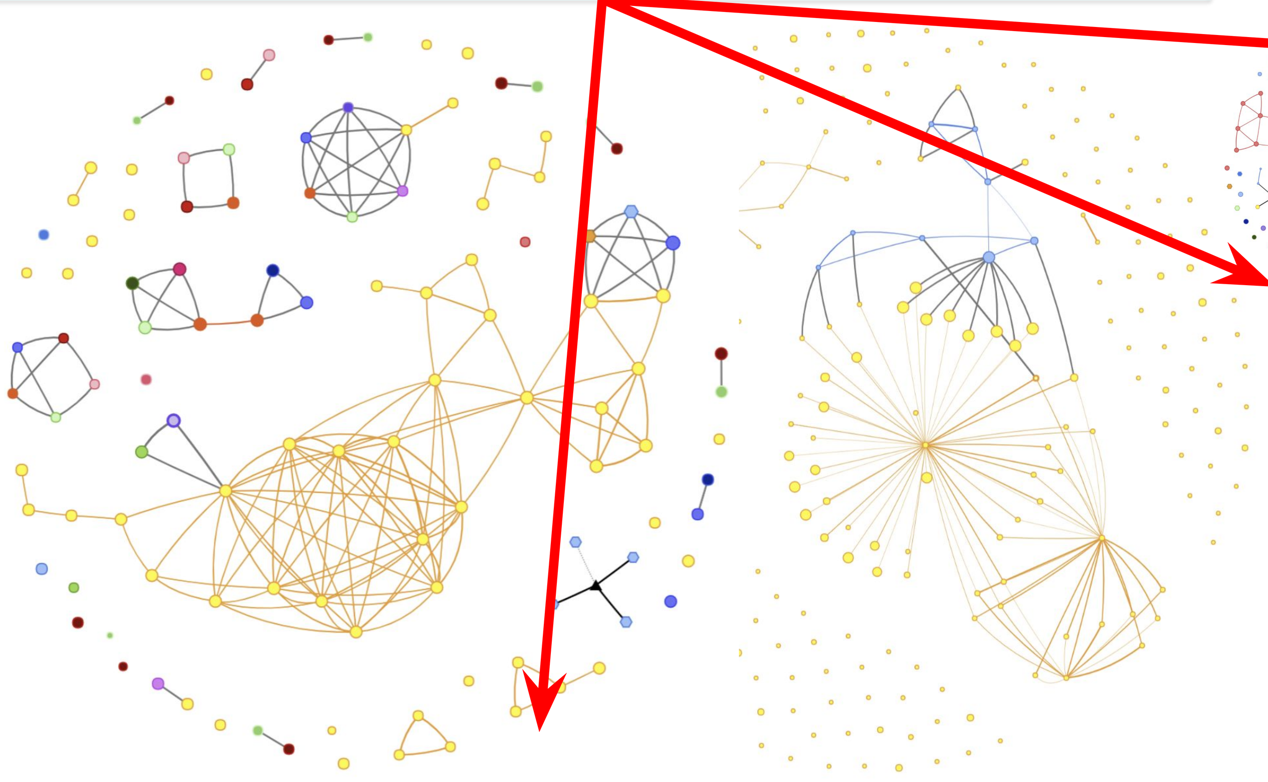
NoSQL

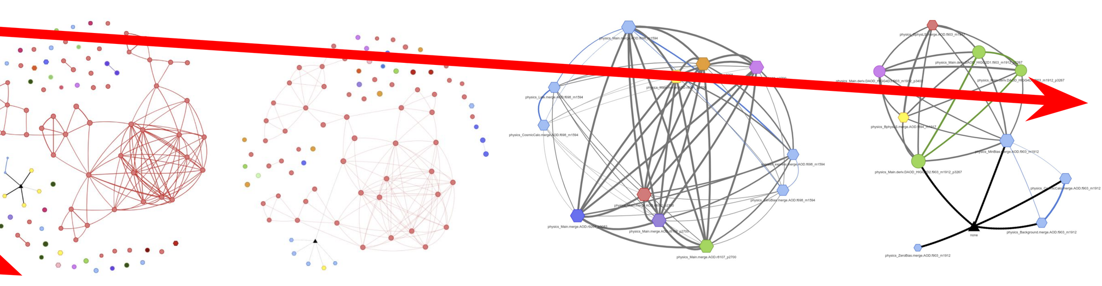
- Flexible
- Scalable
- Often lacks higher level API (query language)

\+

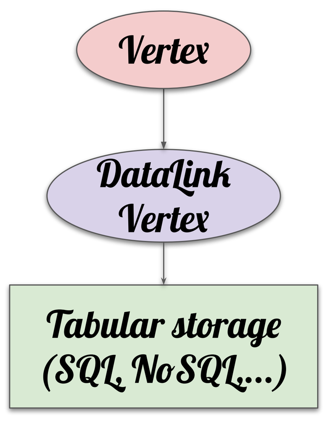
Graph

- Captures essential relations
- Flexible
- Fast access
- Slower injection
- Unnecessary overhead for simple data





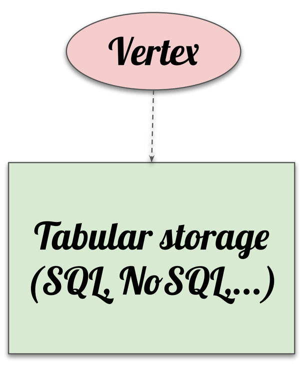
Bridge



- **Make special kind of DataLink Vertex representing relations to external data (in any storage)**
- Those Vertexes can be attached to any Vertex
- Advantage:
  - Easy to implement
  - Transparent logic
  - Works between any pair of databases with any technology
    - We can even connect to Graphs like that

```
// Create DataLink Vertexes with associated data in another database
// (Phoenix/SQL, Graph, HBase,...)
w1 = g.addV().property('lbl', 'datalink').
              property('technology', 'Phoenix').
              property('url', 'jdbc:phoenix:ithdp2101.cern.ch:2181').
              property('query', "...")
w2 = g.addV().property('lbl', 'datalink').
              property('technology', 'Graph').
              property('url', 'hbase:188.184.87.217:8182:janusgraph').
              property('query', "...")
w3 = g.addV().property('lbl', 'datalink').
              property('technology', 'HBase').
              property('url', '134.158.74.54:2183:ztf:schema').
              property('query', "...")
// Connect DataLink to any Vertex
theVertex.addEdge('externalData').to(w)
// Get associated data
externalData = Lomikel.getDataLink(w)
```

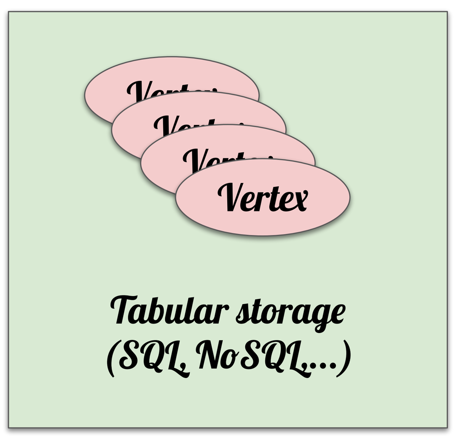
Graph Envelope



- **Make an enhanced version of Vertex with additional methods to fill it from external tabular storage**
- Feasible, has been implemented
- Logical problems:
  - Consistency between (already copied) Vertex and original data
  - Search semantics
- Unpredictable performance
  - Don't know where actually are data and whether will be copied or accessed remotely

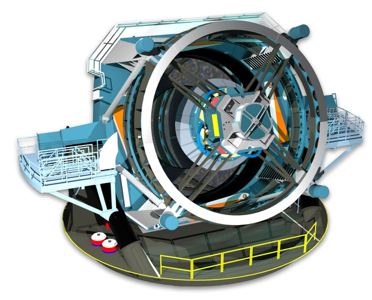
```
// Create an alert vertex
v = g.addV().property('lbl', 'alert')
// Dress it as (a subtype of) Hertex (= HBase backed Vertex)
// which _is_a_ Vertex so it has all Vertex properties
h = Hertex.enhance(v)

// Create a new alert vertex (connect to HBase data later)
a = Alert.getOrCreate('ZTF19acmbwur_2458789.0311458', g, false);
// Create a new alert vertex (and connect to HBase data)
a = Alert.getOrCreate('ZTF19acmbwur_2458789.0311458', g, true);
```

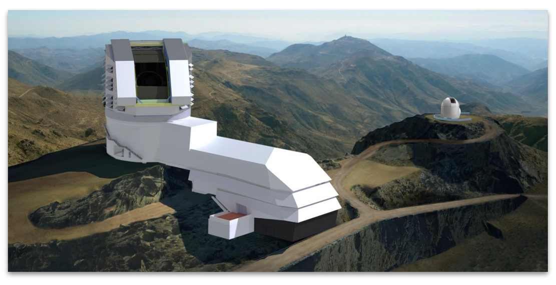
Graph View



- **Interpret existing tabular data as Vertexes in a Graph**
- Add additional Edges to express structures
- Requires full-featured rather generic implementation of the Graph storage
  - Difficult to implement
  - Doesn't exist
  - Most Graph implementation use tabular store as a backend, but impose their own schema

Real-life examples - Fink/LSST

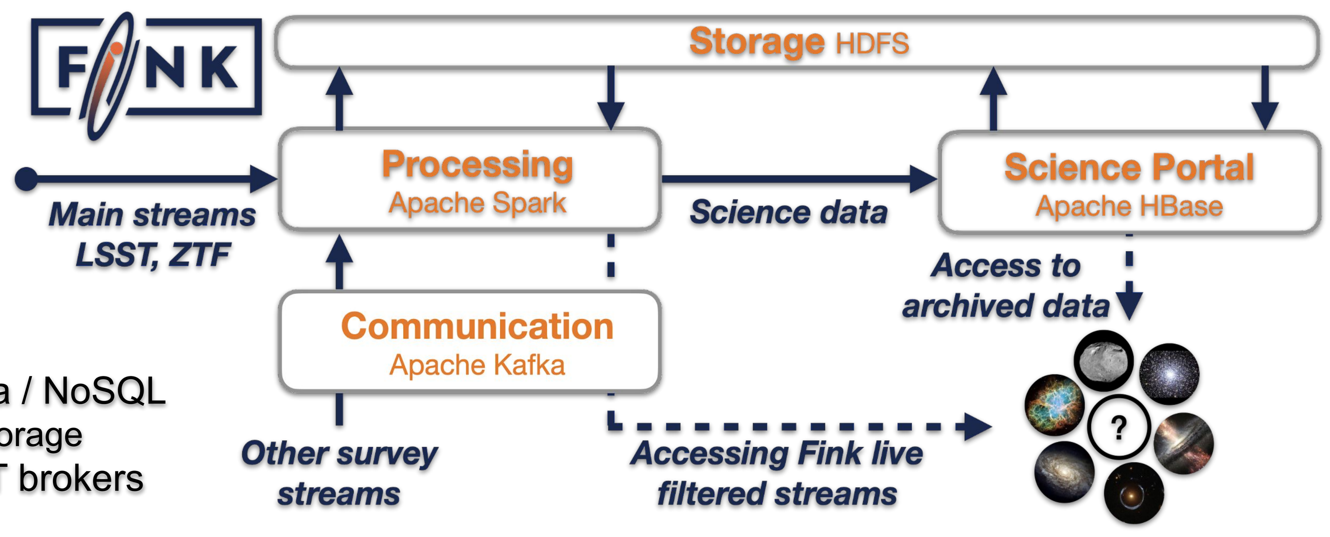


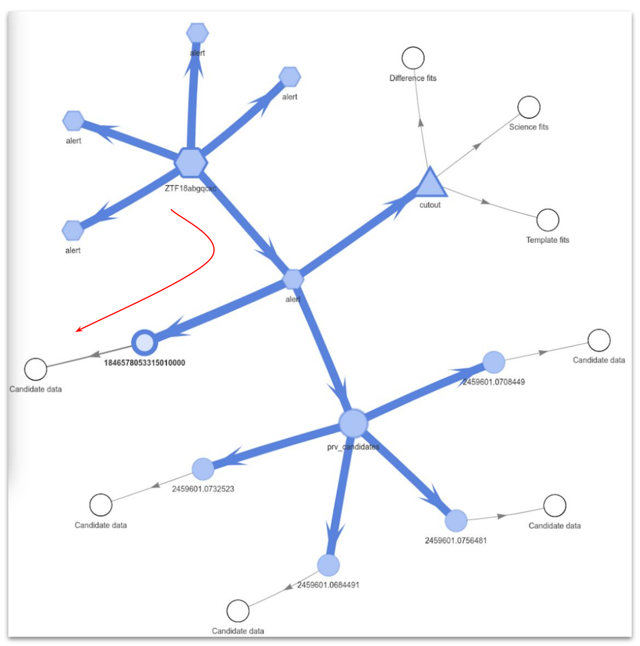


Legacy Survey of Space and Time

- <u>Rubin Observatory for Legacy Survey of Space and Time</u> (LSST)
- Camera 8.4 m, 3.2 Gpixel in Chili
- 10 millions alerts (= 1 TB) nightly
- 20 TB of data (images) nightly
- 500 PB of data in 10 years (3 PB of alerts)
- Alerts send over the world via network of 'brokers'
- Commissioning in 2023, production in 2024

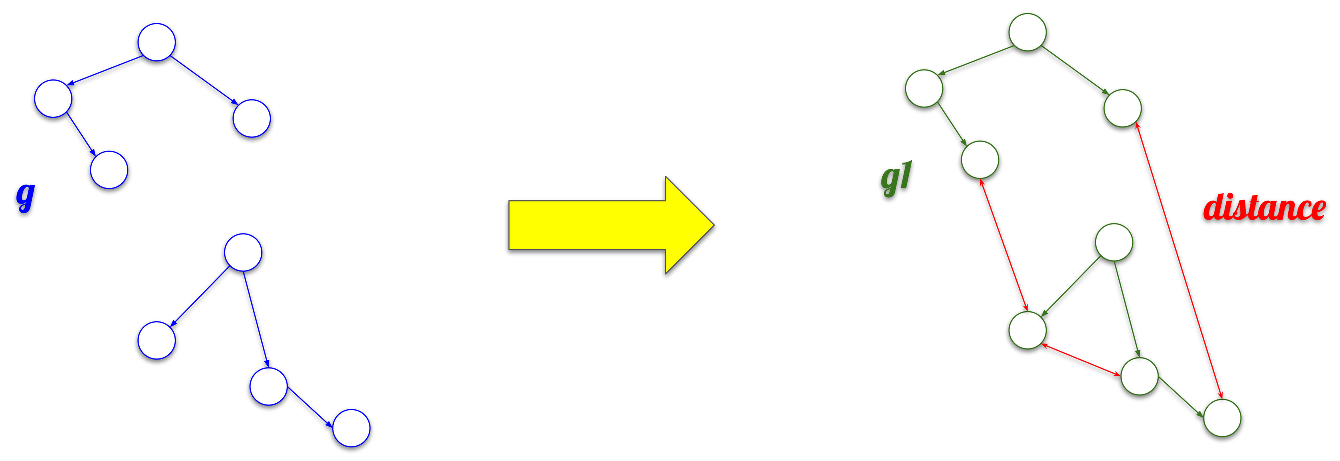
- <u>Fink Broker</u>
- Using Apach/Spark & Big Data / NoSQL
  - Hadoop & JanusGraph for storage
- Fink is one of the official LSST brokers





- 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

```
// Get data (from HBase) attached in 'candidate's
g.V().has('lbl',       'source'      ).
      has('objectId', 'ZTF18abimyys').out().
      has('lbl',       'alert'       ).out().
      has('lbl',       'candidate'   ).out().
      has('lbl',       'datalink'    ).
      each {
        println(FinkBrowser.getDataLink(it))
        }
```



- Search for 'interesting' relations and store them in Graph as Edges for later analyses.
- Do it in your private subgraph.

```
// 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)
  }
// Get GremlinRecipies for the private graph 'g1'.
gr1 = new GremlinRecipies(g1)
// Find all pairs of 'candidate' Vertexes, where difference between their 'rb'
// fields is bigger or equal to 0.01.
// Connect them with the Edge 'distance' having a 'difference' property equal to
// the difference between 'rt' fields.
gr1.structurise(g1.V().has('lbl', 'candidate'), 'rb[0]-rb[1]', 0.01, 'distance', 'difference', …)
// Get some statistics about newly created Edges.
g1.E().hasLabel('distance').values('difference').union(min(), max(), sum(), mean(), count())
```

- <u>Graphical Database advantages</u>
  - More transparent code
    - Stable data structure is handled in the storage layer
  - Suitable for ***Functional Style*** and ***Parallelism***
  - Suitable for ***Deep Learning***
  - Suitable for ***Declarative Analyses***
  - Can help with ***Analysis Preservation***
  - Language & Framework neutral

- <u>Hybrid Storage advantages</u>
  - Expressiveness and flexibility of Graph Databases
  - With performance and simplicity of tabular storage
  - Under transparent interface