Ongoing investigations around Spark and bringing code to the data

André Schaaff, François-Xavier Pineau, Thomas Boch
Centre de Données astronomiques de Strasbourg

Corentin Sanchez[1], Paul Trehiou[1], Jérôme Desroziers[2]
[1]Université de technologie de Belfort-Montbéliard, [2]Telecom Nancy

IVOA, Santiago, 27-29/10/2017

CENTRE DE DONNÉES ASTRONOMIQUES DE STRASBOURG

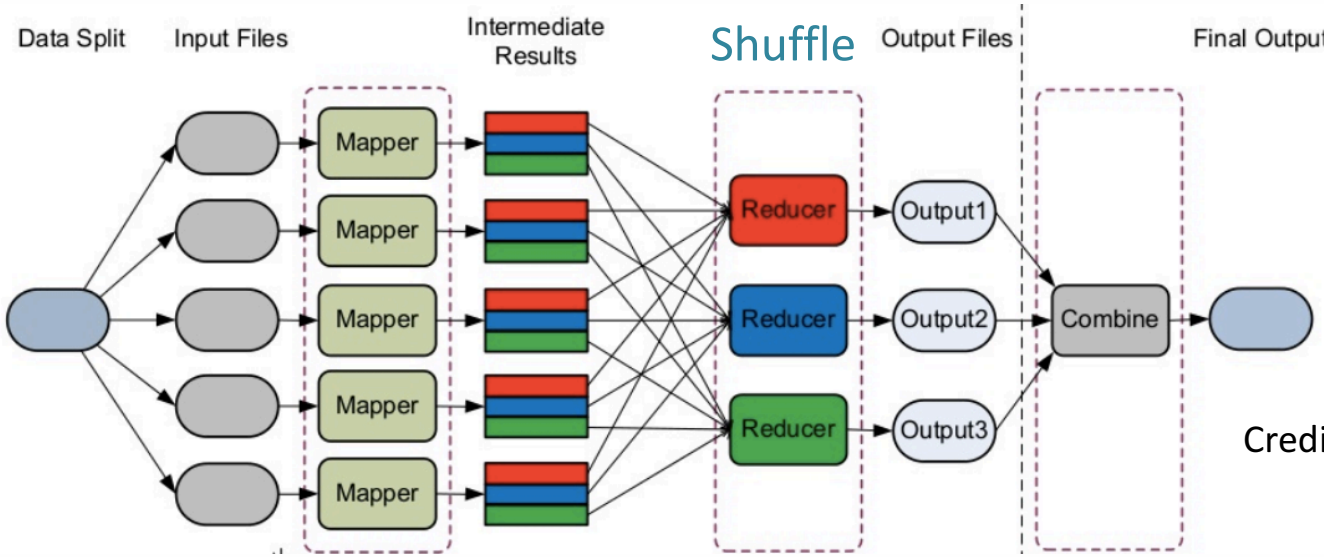
H2020-Astronomy ESFRI and Research Infrastructure Cluster (Grant Agreement number: 653477).

Apache Spark & X-Match

- Reminder, status, next steps

Reminder: Apache Spark

- "Apache Spark is a cluster computing platform designed to be fast and general purpose"
- Extension of the MapReduce model to support more types of computations (interactive queries, stream processing, etc.) offering APIs for Scala, Java, Python, R,...
- Computations in memory (as much as possible, otherwise pilling to the disks)



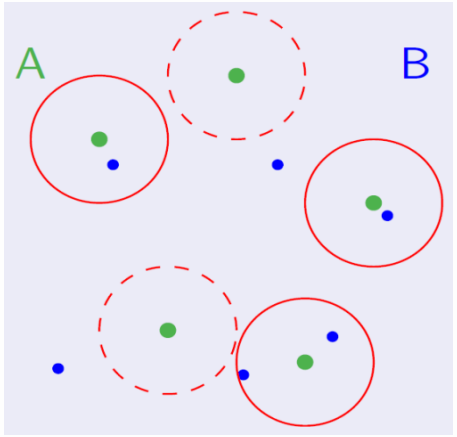
Credit: G. Fedak, INRIA

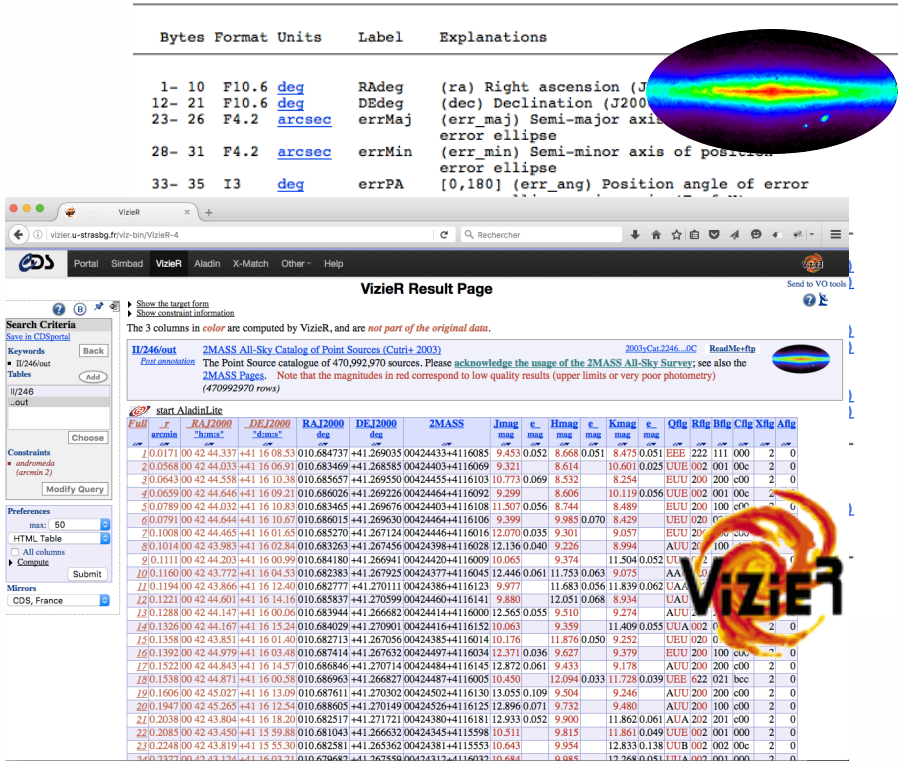
Reminder (2): Use case & data

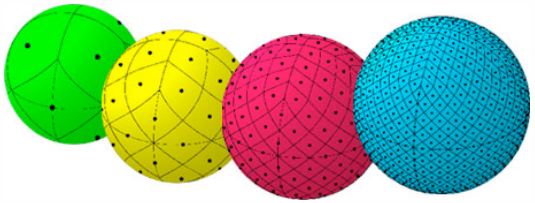
- Cross-matching of large source catalogues:
  - 2MASS[1], 470,992,970
  - SDSS[2] DR9, 469,053,874
  - …

Full sky (all the sources), cone or HEALPix cell

Fuzzy join between 2 tables



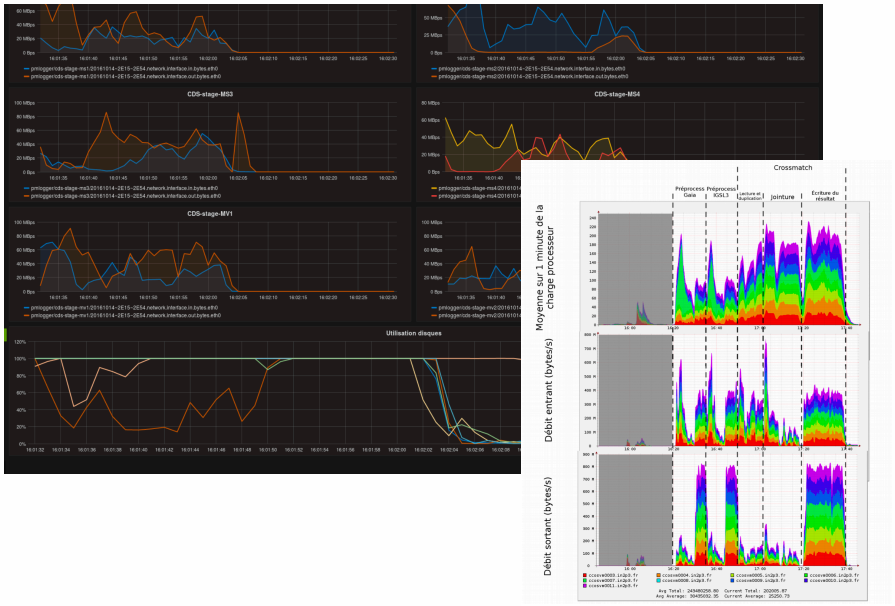




Credits: http://healpix.jpl.nasa.gov/

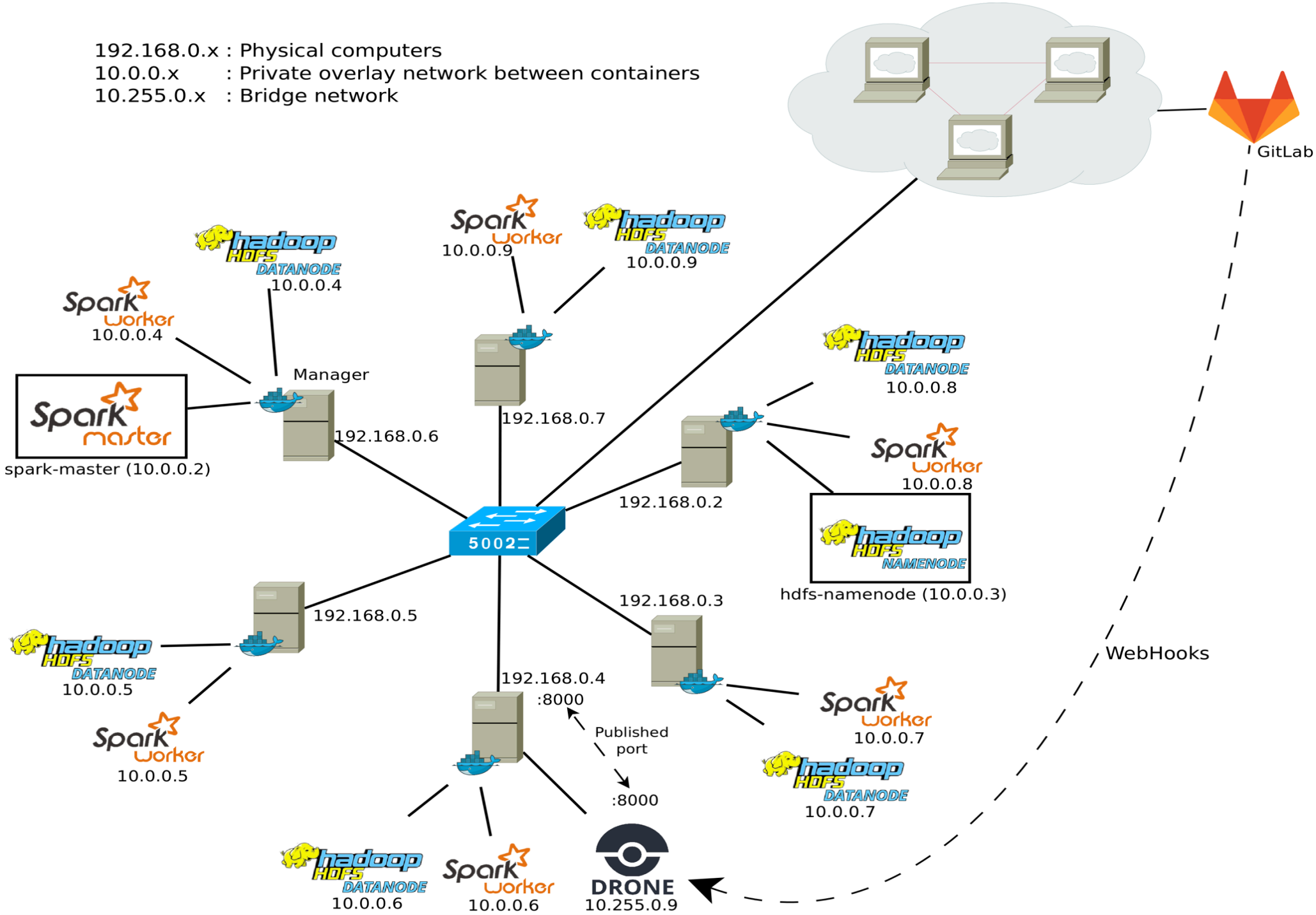
Most significant experiment

- GAIA (1.1 billion sources) vs IGSL3 (1.2 billions)
- 1.6 billion associations in 10 minutes vs 30 minutes for the production X-Match Server*
- Bottlenecks are
  - network for Loading phase
  - mainly CPU during the X-Match phase (improvable)
  - co-location of data (resolved through a homemade solution)



- 9 nodes (CentOS / OpenStack, remote data storage)
- 24 threads (-> 216 threads), 64GB (-> 576 GB)
- -> possible X-Match of billion sources …

* Rebuild Join not included


192.168.0.x : Physical computers
10.0.0.x : Private overlay network between containers
10.255.0.x : Bridge network
GitLab
Spark Worker 10.0.0.9
hadoop HDFS DATANODE 10.0.0.9
hadoop HDFS DATANODE 10.0.0.4
Spark Worker 10.0.0.4
Manager
Spark master
spark-master (10.0.0.2)
192.168.0.6
192.168.0.7
hadoop HDFS DATANODE 10.0.0.8
Spark Worker 10.0.0.8
192.168.0.2
hadoop HDFS NAMENODE
hdfs-namenode (10.0.0.3)
5002
192.168.0.5
192.168.0.3
hadoop HDFS DATANODE 10.0.0.5
Spark Worker 10.0.0.5
192.168.0.4 :8000
Published port
:8000
Spark Worker 10.0.0.7
hadoop HDFS DATANODE 10.0.0.7
WebHooks
hadoop HDFS DATANODE 10.0.0.6
Spark Worker 10.0.0.6
DRONE 10.255.0.9

Next steps

- Improvement of the algorithms
- After this phase we will test with Amazon WS
  - AWS offers a wide variety of tools including Apache Spark
  - Comparison with all the previous investigations
- Summary of this work and status during Victoria Interop

Bringing the code to the data

- How to allow users to execute code near our data ?
  - Which code ? On which data ?, …
  - Hardware resources, accounts, security, etc.
- Development of Jupyter Notebooks
  - Submit X-Match jobs to Spark from Python notebooks
  - Ipyaladin, Aladin Lite embeding in a Notebook (see Thomas's talk in Apps)

Spark in a notebook

jupyter X-Match.onefile Last Checkpoint: 10/12/2017 (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Hide Code Trusted Python 3

Run Code

Hide/show code Export to HTML Export to PDF via HTML Export to PDF via Latex

In [2]:

```
import pyspark
from pyspark.sql import SparkSession
import math
import healpy

class XMatch:
    def __init__(self,file1,file2):
        self.spark = SparkSession\ #Creation of SparkSession and SparkContext if nessessary
                    .builder\
                    .master("spark://130.79.128.185:7077")\
                    .appName("X-Match")\
                    .getOrCreate()
        self.result=None
        print("---------- ",file1," X ",file2," ----------")
        #Load file1
        s=loadBinary(self.spark,file1)
        self.s1=s.flatMap(fDuplicate()) # Duplicate the sources on the border of the pixels
        print(self.s1.count(),"sources loaded from ",file1," after duplication")
        #Load file2
        self.s2=loadBinary(self.spark,file2)
        print(self.s2.count(),"sources loaded from ",file2)

    def performXMatch(self):
        r = self.s1.join(self.s2).filter(filterF) # That's the CrossMatch
        print(r.count()," matching sources found")
```

Spark in a notebook (2)

jupyter X-Match Last Checkpoint: 10/09/2017 (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Hide Code Trusted Python 3

Run Code Hide/show code Export to HTML Export to PDF via HTML Export to PDF via Latex

In [1]:
```
import xmatch

xm=xmatch.XMatch()
xm.loadCatXYZ("lsdss.csv","lsdss")
xm.loadCatXYZ("tmass.csv","tmass")
xm.loadCat("lsdss.csv","lsdss")
xm.loadCat("tmass.csv","tmass")
xm.shuffleAll()
xm.__del__()
```

In [*]:
```
import xmatch
import time

xm=xmatch.XMatch() # Instanciate SparkContext and HDFS reader
xm.performXMatch("tmass","lsdss",5/3600)
```

```
----------  tmass  X  lsdss  ----------
426186 sources loaded from  tmass
783234 sources loaded from  tmass  after duplication
15836 sources loaded from  lsdss
```

In [5]:
```
import xmatch

xm=xmatch.XMatch()
xm.performXMatch("tmass","sdss",3/3600)
xm.downloadResult("result.test")
xm.__del__()
```

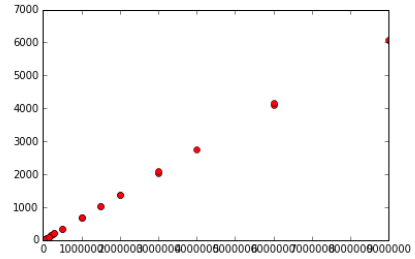
```
----------  tmass  X  sdss  ----------
739325 sources loaded from  tmass  after duplication
1583685 sources loaded from  sdss
134550  matching sources found
Results downloaded in result.test
```

sanchez@cds-stage-ms3: ~

```
[I 08:37:10.582 NotebookApp] Saving file at /X-Match/X-Match.onefile.ipynb
[I 08:39:10.581 NotebookApp] Saving file at /X-Match/X-Match.onefile.ipynb
[I 08:41:10.580 NotebookApp] Saving file at /X-Match/X-Match.onefile.ipynb
[I 08:43:10.580 NotebookApp] Saving file at /X-Match/X-Match.onefile.ipynb
[I 08:43:55.118 NotebookApp] Kernel started: eaf78ba4-0873-4157-a089-8dd0a0ece3b
2
[I 08:43:55.705 NotebookApp] Adapting to protocol v5.1 for kernel eaf78ba4-0873-
4157-a089-8dd0a0ece3b2
[I 08:45:55.125 NotebookApp] Saving file at /X-Match/X-Match.ipynb
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLeve
l(newLevel).
17/10/23 08:46:06 WARN NativeCodeLoader: Unable to load native-hadoop library fo
r your platform... using builtin-java classes where applicable
[I 08:47:10.702 NotebookApp] Kernel restarted: eaf78ba4-0873-4157-a089-8dd0a0ece
3b2
[I 08:47:11.178 NotebookApp] Adapting to protocol v5.1 for kernel eaf78ba4-0873-
4157-a089-8dd0a0ece3b2
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLeve
l(newLevel).
17/10/23 08:47:13 WARN NativeCodeLoader: Unable to load native-hadoop library fo
r your platform... using builtin-java classes where applicable
[Stage 5:============================>                            (8 + 0) / 16]
```

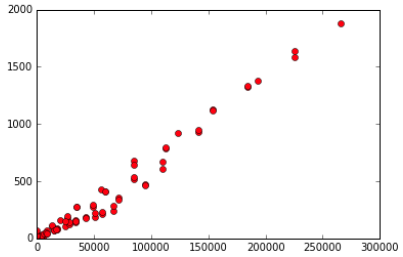
Switching the previous developments to Python

In [1]:
```
import time
import random
import healpy
import math
import numpy
import kdMatch as km
import itertools
import matplotlib.pyplot as plt
radiusDeg=20/3600

fcart=km.fCartesianMatch(radiusDeg)
a=[1,10,50,100,500,1000,2000,3000]
mn=itertools.product(a,a)
lmxn=[]
lt=[]
for b in mn:
    n=b[0]
    m=b[1]
    mxn=n*m

    cn=[(healpy.ang2vec(random.random(),random.random(),lonlat=True),"") for i
    cm=[(healpy.ang2vec(random.random(),random.random(),lonlat=True),"") for i
    sample=(cn,cm)

    t1=int(round(time.time() * 1000))
    result=fcart(sample)
    t2=int(round(time.time() * 1000))
    lt.append(t2-t1)
    lmxn.append(mxn)

plt.plot(lmxn, lt, 'ro')
plt.show()
droite=numpy.polyfit(lmxn, lt, 1)
print("après régression linéaire : y = ",droite[0]," x + ",droite[1] )
```



```
après régression linéaire : y =  0.000681781894157  x +  1.29801560841
```

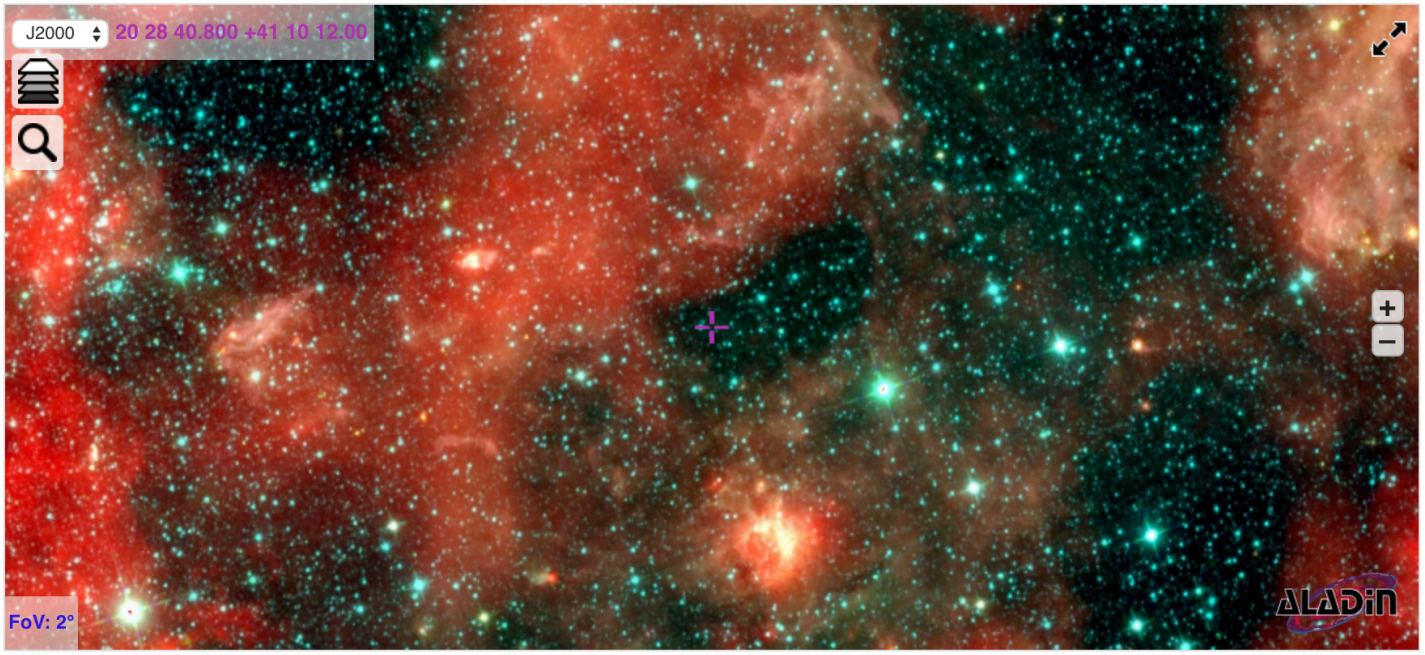
In [2]:
```
import time
import random
import healpy
import math
import kdMatch as km
import itertools
import matplotlib.pyplot as plt
radiusDeg=20/3600

fkd=km.fKDMatch(radiusDeg)
a=[1,10,50,100,500,1000,2500,5000,7500,10000]
mn=itertools.product(a,a)
lmxn=[]
lt=[]
for b in mn:
    n=b[0]
    m=b[1]
    mxn=(n+m)*math.log2(min(m,n))

    cn=[(healpy.ang2vec(random.random(),random.random(),lonlat=True),"") for i in range(0,n)]
    cm=[(healpy.ang2vec(random.random(),random.random(),lonlat=True),"") for i in range(0,m)]
    sample=(cn,cm)

    t1=int(round(time.time() * 1000))
    result=fkd(sample)
    t2=int(round(time.time() * 1000))
    lt.append(t2-t1)
    lmxn.append(mxn)

plt.plot(lmxn, lt, 'ro')
plt.show()
droite=numpy.polyfit(lmxn, lt, 1)
print("après régression linéaire : y = ",droite[0]," x + ",droite[1] )
```



```
après régression linéaire : y =  0.00694964313111  x +  -23.6947540134
```

Ipyaladin

In [2]:
```
import ipyaladin
aladin = ipyaladin.Aladin(target = 'Cygnus X')
aladin
```



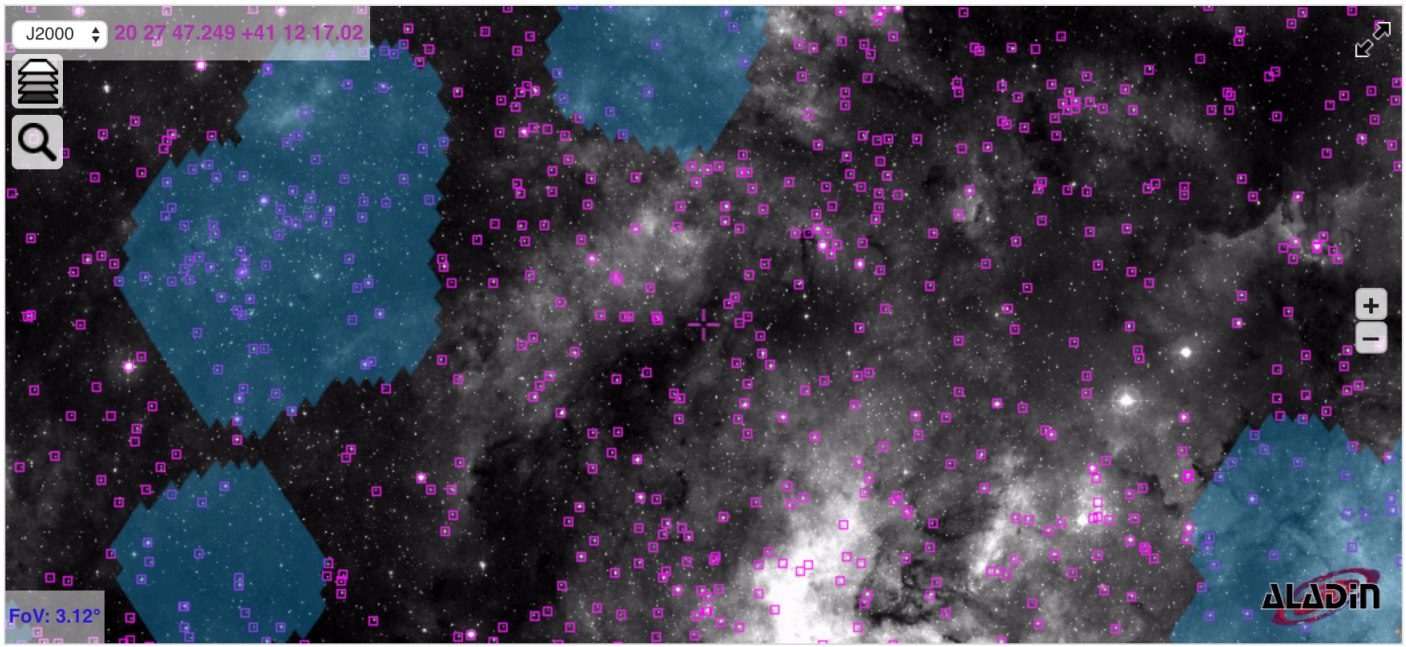
In [3]:
```
aladin.fov = 2
aladin.survey = 'P/allWISE/color'
```

Aladin Lite integration in notebook and control of displayed target and field of view

Ipyaladin (2)

In [24]:

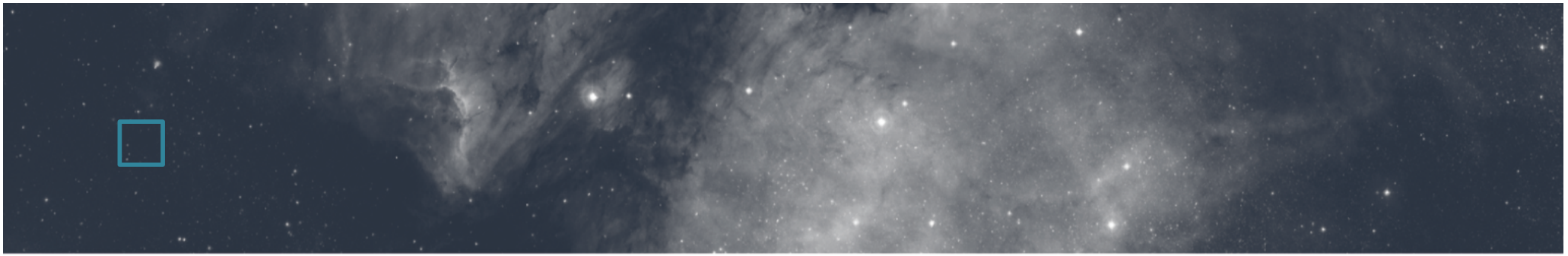
```
from astroquery.vizier import Vizier
tgas_result = Vizier.query_region('Cygnus X', radius='3 deg', catalog='I/337/tgas')
aladin.add_table(tgas_result[0])

aladin.add_moc_from_URL('http://saada.unistra.fr/PNRed/Moc.fits',
                        {'color': '#009bd6', 'opacity': 0.4})
```



Add overlays: Astropy tables and MOCs

Conclusion

- Exciting experiments, implementation and testing of various technologies, switching between languages (Java, Python, Scala, JS), etc.
- Promising results concerning the X-Match, confident to do better
- Notebooks open the door to a simple way to execute code near the data



H2020-Astronomy ESFRI and Research Infrastructure Cluster (Grant Agreement number: 653477).