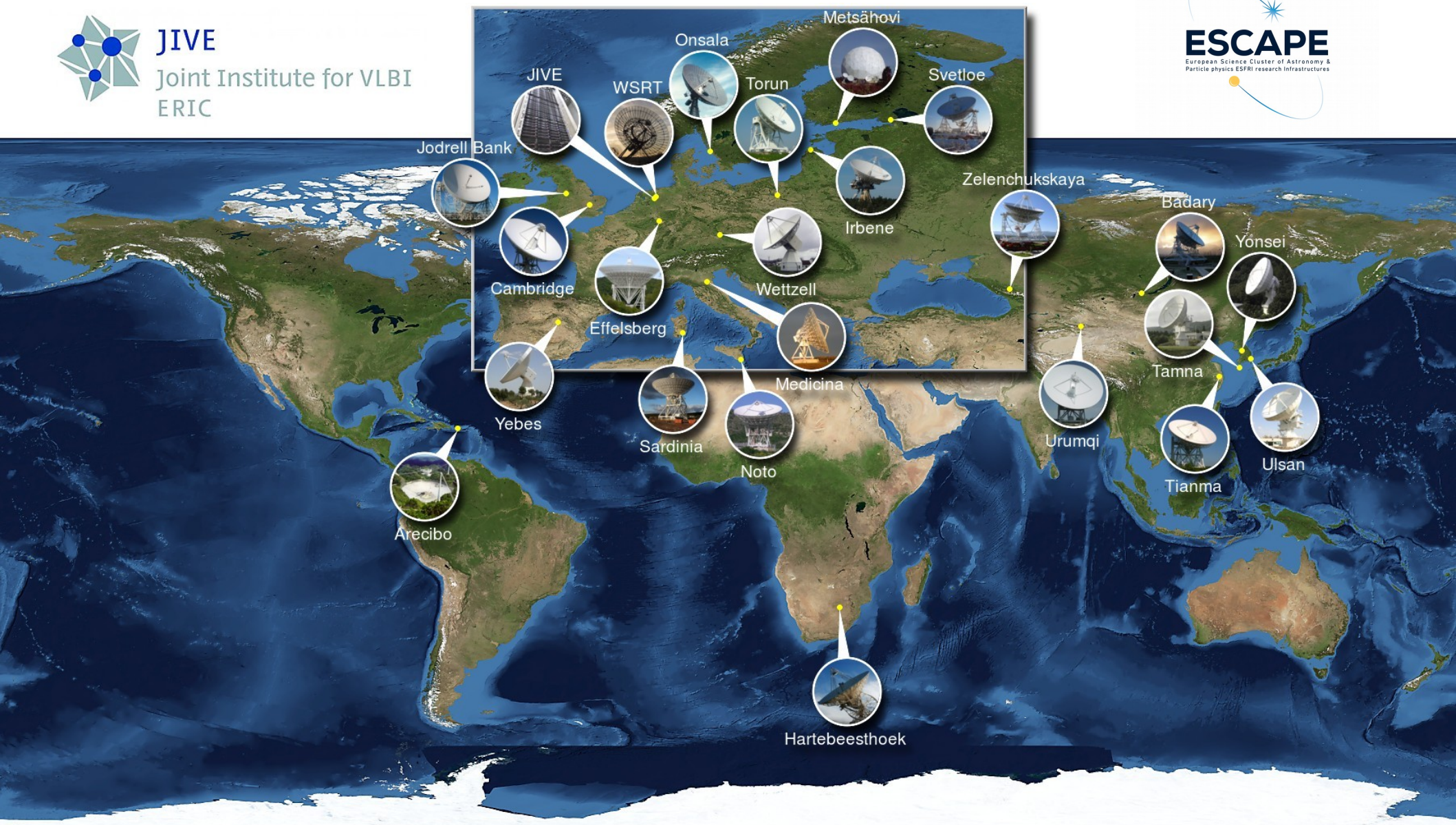
Mining the EVN Archive Using JupyterLab and the Virtual Obervatory

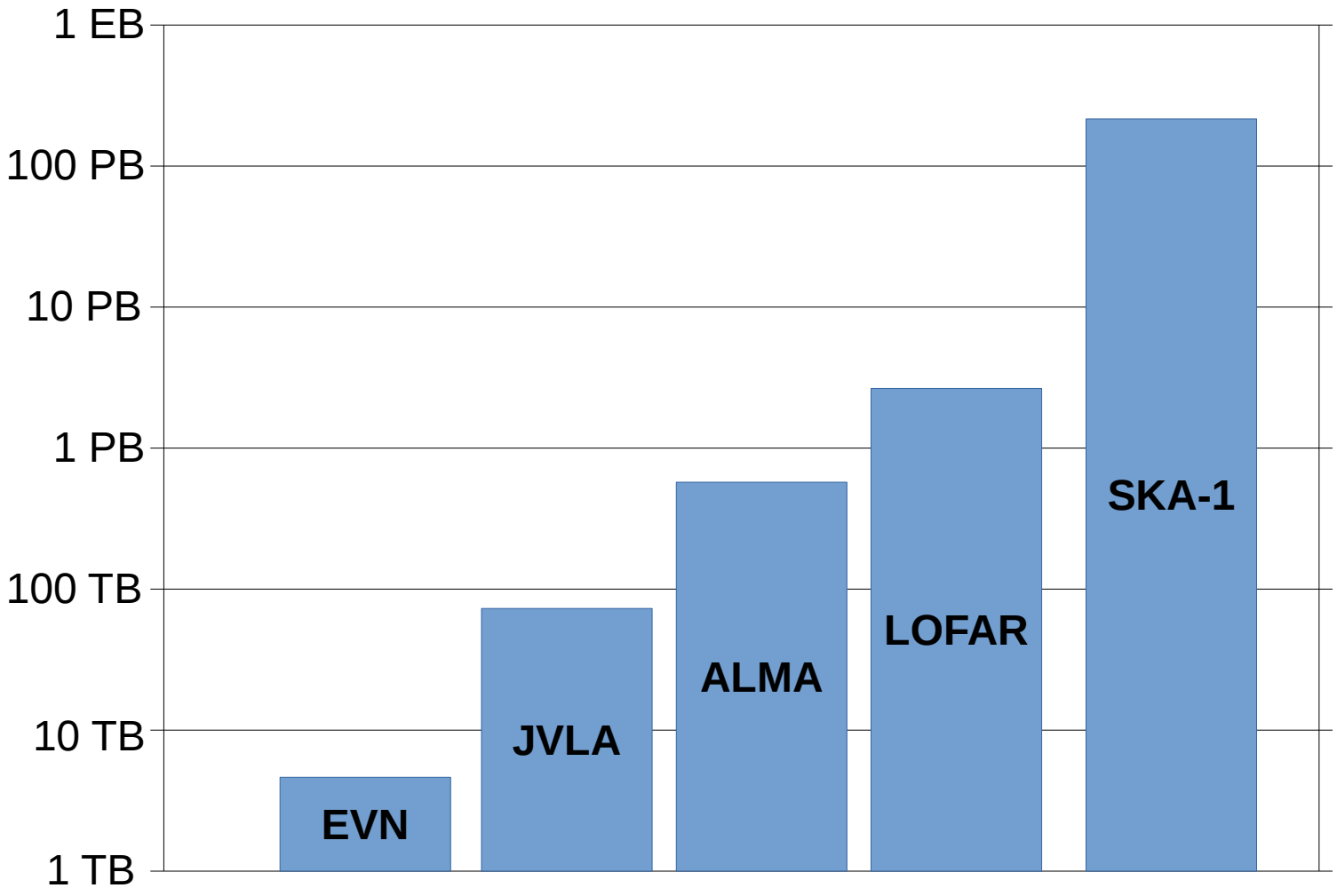
Aard Keimpema *(keimpema@jive.eu)*

JIVE
Joint Institute for VLBI ERIC

ESCAPE
European Science Cluster of Astronomy & Particle physics ESFRI research Infrastructures



Yearly archivable data



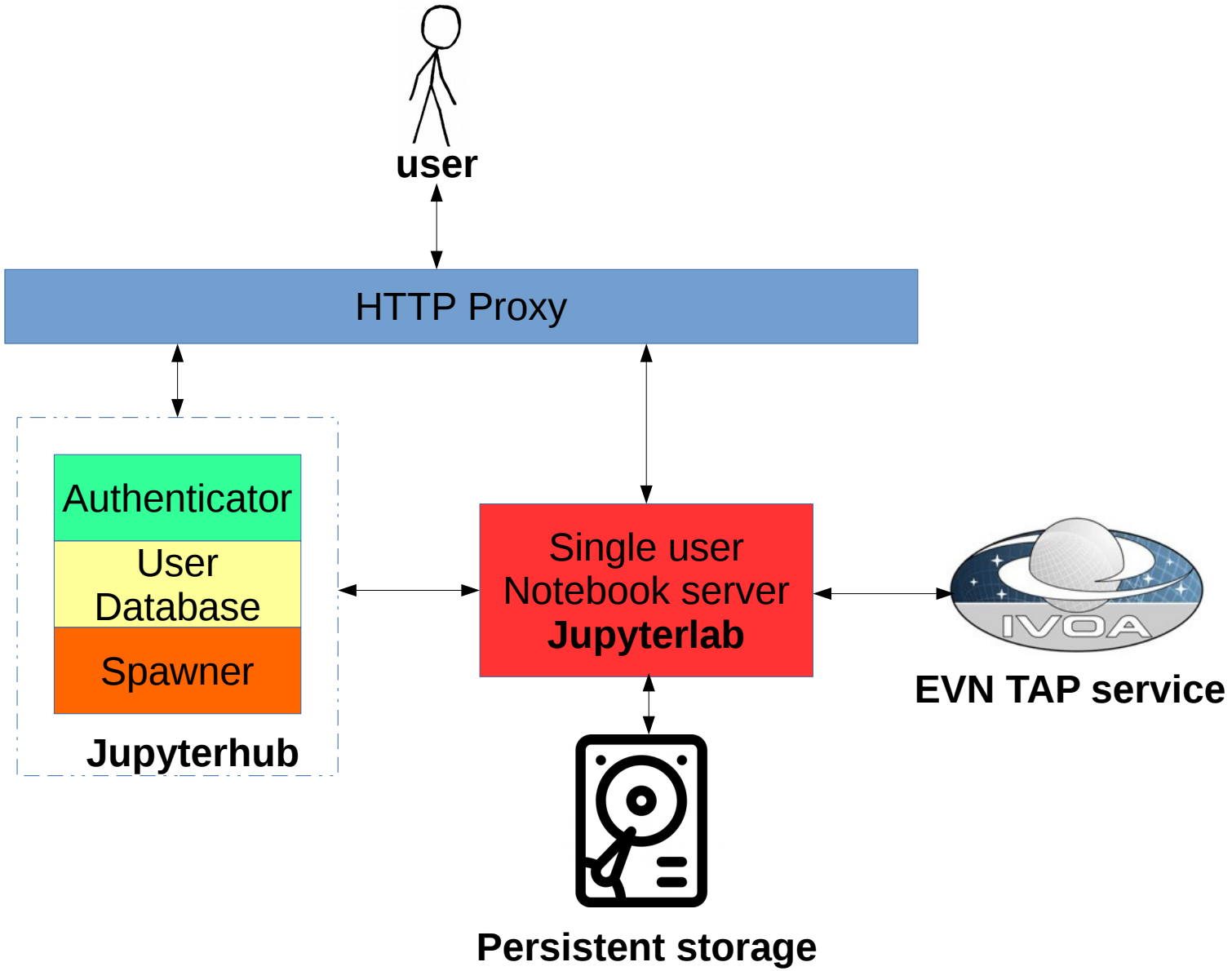
Interactive Jupyter pipelines

- Data reduction should be done where the data is stored
- Existing non-interactive CASA pipelines: ALMA / JVLA, LOFAR, MEERKATHI
- Jupyter offers many advantages
  - User friendly: Notebooks are easy and intuitive to use; all results are embedded in a single document
  - Easy to deploy: Off-the-shelf solutions to deploy multi-user services, e.g. Jupyter hub + Kubernetis
  - Accountability: Data reduction process is self-documenting and fully repeatable

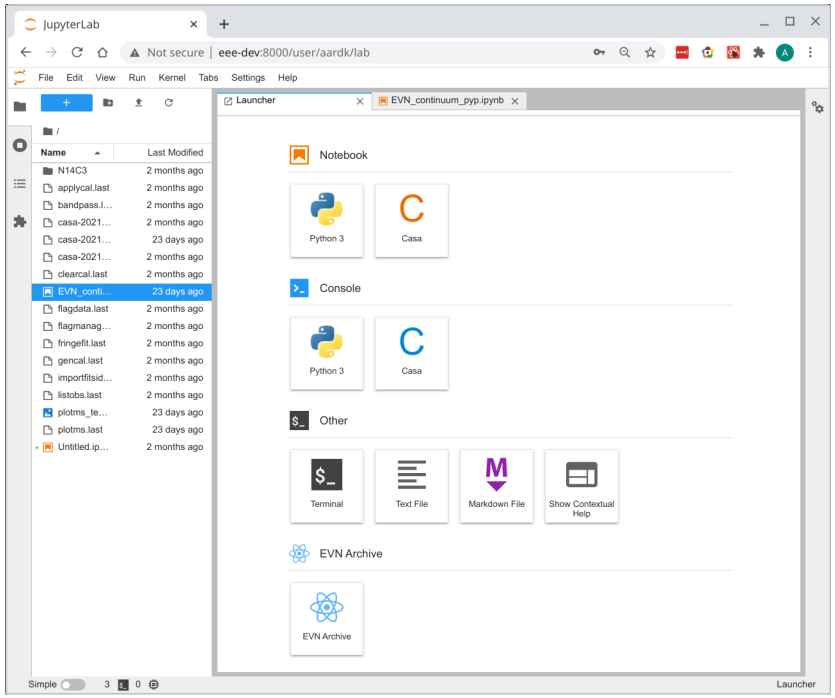
EVN Archive Portal

- Provide access to EVN Archive through Jupyterhub
- Users can pipeline any experiment and results are stored in persistent storage
- Users can submit improved pipeline results back into the archive
- Archive will accessed through VO queries using a JupyterLab plugin
- EVN TAP service (see Mark Kettenis' talk)
- All experiments older than 1yr are public
- Will be made available in the ESAP in WP5

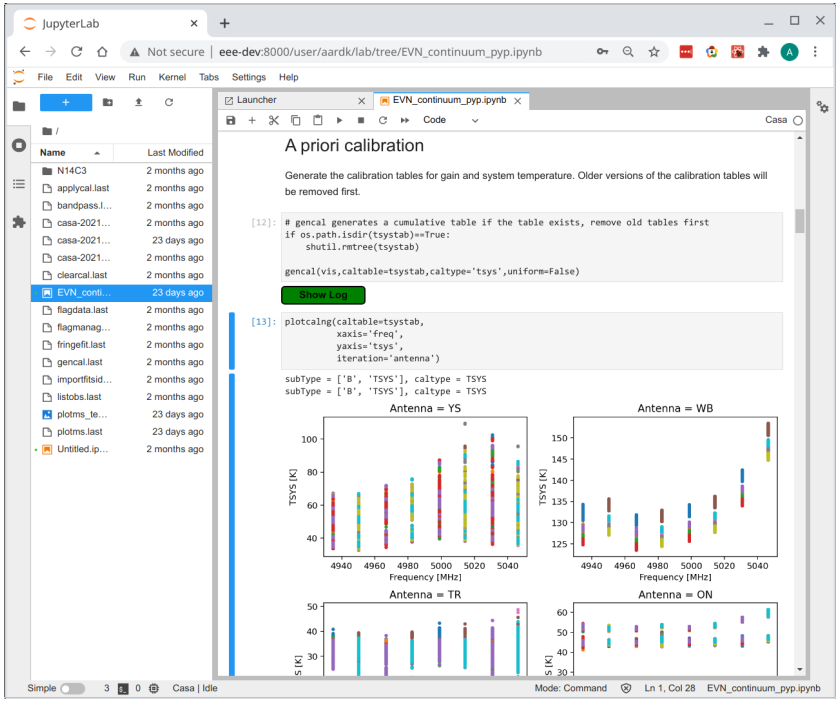
EVN Archive Portal



Relationship JupyterLab and Notebook



JupyterLab



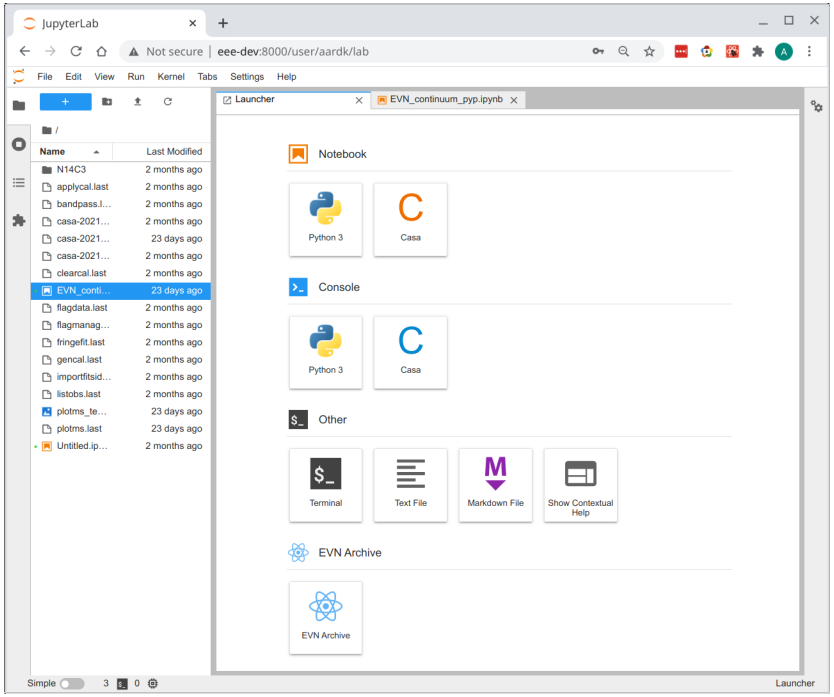
Jupyter notebook inside JupyterLab

- Jupyter notebooks are the interactive documents
- JupyterLab is a user interface
  - Can have multiple notebooks open
  - Also run non-notebook applications, e.g. shell

JupyterLab

Jupyterlab client

JupyterLab server



- Tornado webserver
- Written in python
- Launched jupyter kernels

- Client side uses Lumino JavaScript library
- Implemented in TypeScript

https://jupyterlab.readthedocs.io/en/stable/

JupyterLab extensions

- Expand functionality of JupyterLab
- **Client side extensions**
  - Written in TypeScript or JavaScript
  - Communicates with server through HTTP requests
  - Not limited to Lumino widgets, e.g. can embed React components
- **Server side extensions**
  - Written in python
  - Communicate with client using JSON objects
- Extension tutorial:
  https://jupyterlab.readthedocs.io/en/stable/extension/extension_tutorial.html
- Extension templates:
  https://github.com/jupyterlab/extension-cookiecutter-ts

Minimal plugin

Create project using cookiecutter

```
pip install cookiecutter
cookiecutter  https://github.com/jupyterlab/extension-cookiecutter-ts
```

Minimal extension

Create project using cookiecutter

```
pip install cookiecutter
cookiecutter  https://github.com/jupyterlab/extension-cookiecutter-ts
```

Cookie cutter asks some basic questions

```
author_name []: Aard Keimpema
author_email []: keimpema@jive.eu
python_name [myextension]: minimal-plugin
labextension_name [minimal-plugin]:
project_short_description [A JupyterLab extension.]: Minimal JupyterLab
plugin
has_server_extension [n]: y
has_binder [n]:
repository [https://github.com/github_username/minimal-plugin]:
```

Minimal extension

Create project using cookiecutter

```
pip install cookiecutter
cookiecutter  https://github.com/jupyterlab/extension-cookiecutter-ts
```

Resulting in the following files

```
install.json     MANIFEST.in      pyproject.toml   src
jupyter-config   minimal-plugin   README.md        style
LICENSE          package.json     setup.py         tsconfig.json
```

Minimal extension

Create project using cookiecutter

```
pip install cookiecutter
cookiecutter  https://github.com/jupyterlab/extension-cookiecutter-ts
```

Resulting in the following files

```
install.json     MANIFEST.in      pyproject.toml   src
jupyter-config   minimal-plugin   README.md        style
LICENSE          package.json     setup.py         tsconfig.json
```

```
handlers.py   __init__.py   _version.py
```

Server side plugin (Python)

```
handler.ts   index.ts
```

Client side plugin (TypeScript)

Minimal extension

Create project using cookiecutter

```
pip install cookiecutter
cookiecutter  https://github.com/jupyterlab/extension-cookiecutter-ts
```

Resulting in the following files

```
install.json     MANIFEST.in      pyproject.toml   src
jupyter-config   minimal-plugin   README.md        style
LICENSE          package.json     setup.py         tsconfig.json
```

JavaScript dependencies

Python dependencies

TypeScript configuration

Minimal extension

Create project using cookiecutter

```
pip install cookiecutter
cookiecutter  https://github.com/jupyterlab/extension-cookiecutter-ts
```

Resulting in the following files

```
install.json     MANIFEST.in      pyproject.toml   src
jupyter-config   minimal-plugin   README.md        style
LICENSE          package.json     setup.py         tsconfig.json
```

Install plugin

```
# Install server
pip install -e .
jupyter serverextension enable --py minimal-plugin --sys-prefix

# Install client
jlpm build
jupyter labextension install .
jupyter lab build
```

Minimal plugin

Create project using cookiecutter

```
pip install cookiecutter
cookiecutter  https://github.com/jupyterlab/extension-cookiecutter-ts
```

Resulting in the following files

```
install.json     MANIFEST.in      pyproject.toml   src
jupyter-config   minimal-plugin   README.md        style
LICENSE          package.json     setup.py         tsconfig.json
```

Install plugin

```
# Install server
pip install -e .
jupyter serverextension enable --py minimal-plugin --sys-prefix

# Install client
jlpm build
jupyter labextension install .
jupyter lab build
```

Jupyter version of **yarn** package manager

Server side Request handler

- Clients communicate with Server through HTTP Requests
- Requests are handled using Tornado python package
  - https://www.tornadoweb.org/
- First step: Register request handler
- Server now listening to http://HOSTNAME/minimal-plugin/get_example

```
def setup_handlers(web_app):
    host_pattern = ".*$"

    base_url = web_app.settings["base_url"]
    route_pattern = url_path_join(base_url, "minimal-plugin", "get_example")
    handlers = [(route_pattern, RouteHandler)]
    web_app.add_handlers(host_pattern, handlers)
```

Server side Request handler

- Subclasses Jupyter's *APIHandler*
- *APIHandler* in turn subclasses *Tornado.web.RequestHandler*
- Override get method for HTTP GET requests
- Always responds with JSON object

```
from jupyter_server.base.handlers import APIHandler
from jupyter_server.utils import url_path_join
import tornado

class RouteHandler(APIHandler):

    @tornado.web.authenticated
    def get(self):
        self.finish(json.dumps({
            "data": "This is /minimal-plugin/get_example endpoint!"
        }))
```

Client side

```
/**
 * Initialization data for the minimal-plugin extension.
 */
const extension: JupyterFrontEndPlugin<void> = {
  id: 'minimal-plugin:plugin',
  autoStart: true,
  activate: (app: JupyterFrontEnd) => {
    console.log('JupyterLab extension minimal-plugin is activated!');

    requestAPI<any>('get_example')
      .then(data => {
        console.log(data);
      })
      .catch(reason => {
        console.error(
         The minimal-plugin server extension appears to be missing.\n${reason}`
        );
      });
  }
};
```

Client side

```
/**
 * Initialization data for the minimal-plugin extension.
 */
const extension: JupyterFrontEndPlugin<void> = {
  id: 'minimal-plugin:plugin',
  autoStart: true,
  activate: (app: JupyterFrontEnd) => {
    console.log('JupyterLab extension minimal-plugin is activated!');

    requestAPI<any>('get_example')
      .then(data => {
        console.log(data);
      })
      .catch(reason => {
        console.error(
         The minimal-plugin server extension appears to be missing.\n${reason}`
        );
      });
  }
};
```

TypeScript datatypes

Client side

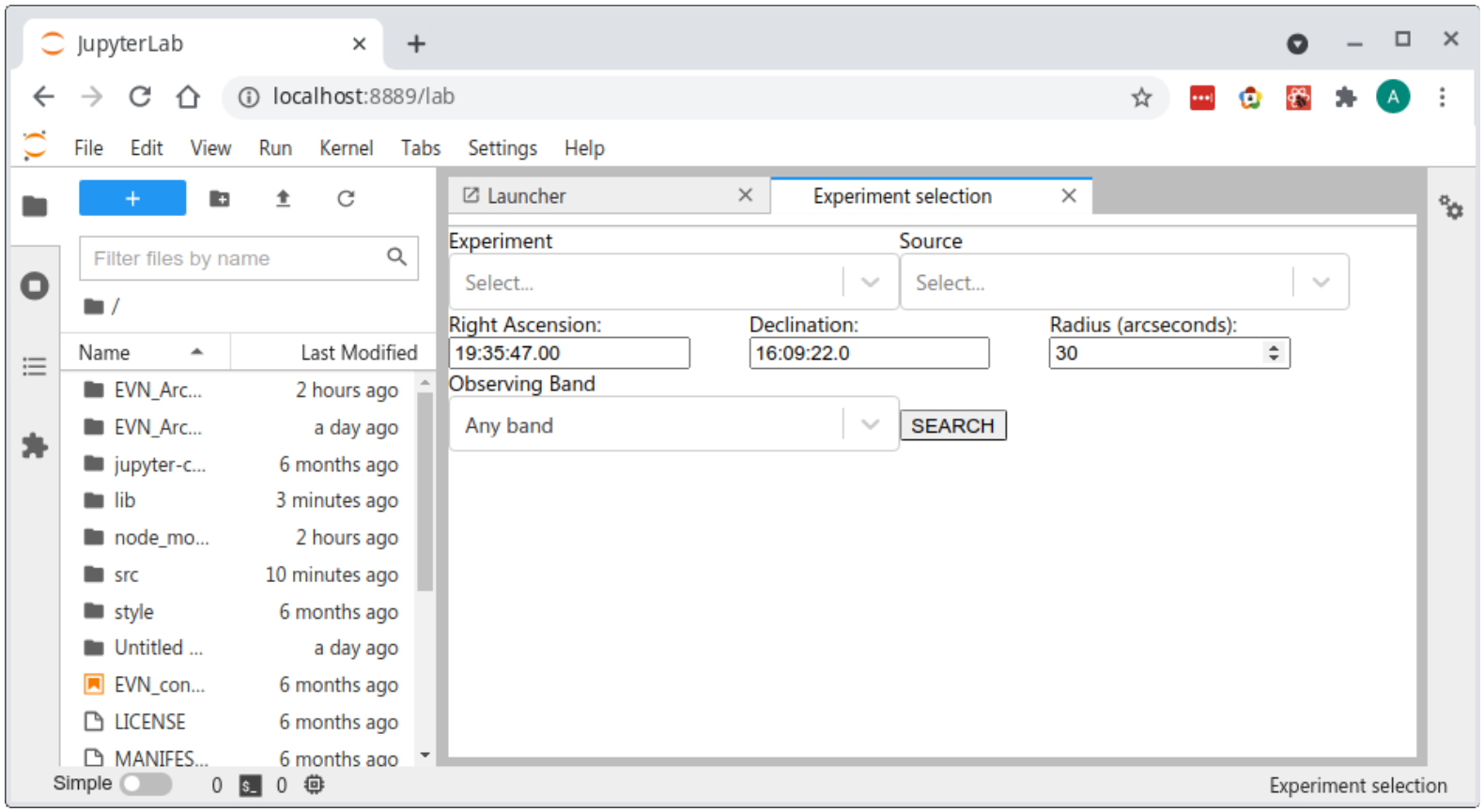
```
/**
 * Initialization data for the minimal-plugin extension.
 */
const extension: JupyterFrontEndPlugin<void> = {
  id: 'minimal-plugin:plugin',
  autoStart: true,
  activate: (app: JupyterFrontEnd) => {
    console.log('JupyterLab extension minimal-plugin is activated!');

    requestAPI<any>('get_example')
      .then(data => {
        console.log(data);
      })
      .catch(reason => {
        console.error(
         The minimal-plugin server extension appears to be missing.\n${reason}`
        );
      });
  }
};
```

Thin wrapper around JupyterLab's ServerConnection.makeRequest function

Register extension

Prototype plugin



Client side

- React GUI
- Search parameters selection
- Select data sets

Server side

- Perform VO queries using PyVO
- Datasets are downloaded at server side

Example: Cone search

URL: http://HOSTNAME/EVN-Archive/search?ra=0.0&dec=0.0&radius=1.0

```
class SearchHandler(APIHandler):
  @tornado.web.authenticated
  def get(self):
    q = { 'ra': self.get_query_argument('ra', default=''),
          'dec': self.get_query_argument('dec', default=''),
          'radius': self.get_query_argument('radius', default=''),
        }
    service = pyvo.dal.TAPService('http://evn-vo.jive.eu/tap')
    result = service.search(f"SELECT target_name, obs_id, t_exptime, s_ra, s_dec,
                            DISTANCE(POINT('ICRS', {q.ra}, {q.dec}),
                                     POINT('ICRS', s_ra, s_dec)) AS dist
                            FROM ivoa.obscore
                            WHERE 1=CONTAINS(POINT('ICRS', {q.ra}, {q.dec}),
                                             CIRCLE('ICRS', s_ra, s_dec,
                                                    {q.radius}))
                            ORDER BY dist ASC")
    response = [{'src': row['target_name'].decode(), 'ra': row['s_ra'],
                 'dec': row['s_dec'], 't_exp': row['t_exp_time'],
                 'dist': row['dist'] } for row in result]
    self.finish(json.dumps(response))
```

Example: Cone search

URL: http://HOSTNAME/EVN-Archive/search?ra=0.0&dec=0.0&radius=1.0

Get parameters, through tornado get_query_argument

```
class SearchHandler(APIHandler
  @tornado.web.authenticated
  def get(self):
    q = { 'ra': self.get_query_argument('ra', default=''),
          'dec': self.get_query_argument('dec', default=''),
          'radius': self.get_query_argument('radius', default=''),
        }
    service = pyvo.dal.TAPService('http://evn-vo.jive.eu/tap')
    result = service.search(f"SELECT target_name, obs_id, t_exptime, s_ra, s_dec,
                            DISTANCE(POINT('ICRS', {q.ra}, {q.dec}),
                                     POINT('ICRS', s_ra, s_dec)) AS dist
                            FROM ivoa.obscore
                            WHERE 1=CONTAINS(POINT('ICRS', {q.ra}, {q.dec}),
                                             CIRCLE('ICRS', s_ra, s_dec,
                                                    {q.radius}))
                            ORDER BY dist ASC")
    response = [{'src': row['target_name'].decode(), 'ra': row['s_ra'],
                 'dec': row['s_dec'], 't_exp': row['t_exp_time'],
                 'dist': row['dist'] } for row in result]
    self.finish(json.dumps(response))
```

Example: Cone search

URL: http://HOSTNAME/EVN-Archive/search?ra=0.0&dec=0.0&radius=1.0

```
class SearchHandler(APIHandler):
  @tornado.web.authenticated
  def get(self):
    q = { 'ra': self.get_query_argument('ra', default=''),
          'dec': self.get_query_argument('dec', default=''),
          'radius': self.get_query_argument('radius', default='
        }
    service = pyvo.dal.TAPService('http://evn-vo.jive.eu/tap')
    result = service.search(f"SELECT target_name, obs_id, t_exptime, s_ra, s_dec,
                            DISTANCE(POINT('ICRS', {q.ra}, {q.dec}),
                                     POINT('ICRS', s_ra, s_dec)) AS dist
                            FROM ivoa.obscore
                            WHERE 1=CONTAINS(POINT('ICRS', {q.ra}, {q.dec}),
                                             CIRCLE('ICRS', s_ra, s_dec,
                                                    {q.radius}))
                            ORDER BY dist ASC")
    response = [{'src': row['target_name'].decode(), 'ra': row['s_ra'],
                 'dec': row['s_dec'], 't_exp': row['t_exp_time'],
                 'dist': row['dist'] } for row in result]
    self.finish(json.dumps(response))
```

EVN TAP Service

Example: Cone search

URL: http://HOSTNAME/EVN-Archive/search?ra=0.0&dec=0.0&radius=1.0

```
class SearchHandler(APIHandler):
  @tornado.web.authenticated
  def get(self):
    q = { 'ra': self.get_query_argument('ra', default=''),
          'dec': self.get_query_argument('dec', default=''),
          'radius': self.get_query_argument('radius', default=''),
        }
    service = pyvo.dal.TAPService('http://evn-vo.jive.eu/tap')
    result = service.search(f"SELECT target_name, obs_id, t_exptime, s_ra, s_dec,
                                DISTANCE(POINT('ICRS', {q.ra}, {q.dec}),
                                         POINT('ICRS', s_ra, s_dec)) AS dist
                            FROM ivoa.obscore
                            WHERE 1=CONTAINS(POINT('ICRS', {q.ra}, {q.dec}),
                                             CIRCLE('ICRS', s_ra, s_dec,
                                                    {q.radius}))
                            ORDER BY dist ASC")
    response = [{'src': row['target_name'].decode(), 'ra': row['s_ra'],
                 'dec': row['s_dec'], 't_exp': row['t_exp_time'],
                 'dist': row['dist'] } for row in result]
    self.finish(json.dumps(response))
```

Example: Cone search

URL: http://HOSTNAME/EVN-Archive/search?ra=0.0&dec=0.0&radius=1.0

```
class SearchHandler(APIHandler):
  @tornado.web.authenticated
  def get(self):
    q = { 'ra': self.get_query_argument('ra', default=''),
```

Example for ra = 141.762558, dec = 39.0391, radius = 2.0

target_name	obs_id	t_exptime	s_ra	s_dec	dist
		s	deg	deg	deg
object	object	float32	float64	float64	float64
4C39.25	DA193	1830.0	141.76255801851036	39.03912500509745	2.5005101589442766e-05
J0916+3854	N09M2	2856.0	139.20376904166667	38.907818472222225	1.993558853813117
J0916+3854	N08C1	4192.0	139.2037690375	38.90781847777778	1.9935588566014968
J0916+3854	N08C2	4194.0	139.2037690375	38.90781847777778	1.9935588566014968
J0916+3854	N08L2	3167.0	139.2037690375	38.90781847777778	1.9935588566014968
J0916+3854	N08K2	962.0	139.2037690375	38.90781847777778	1.9935588566014968
J0916+3854	N06U1	6832.0	139.2037690375	38.907818477777774	1.9935588566014975
J0916+3854	N05C2	3320.0	139.20376903333332	38.90781849166666	1.9935588587244752
J0916+3854	N05C1	8280.0	139.20376895416666	38.90781850277778	1.993558919244693

```
                'dist': row['dist'] } for row in result]
    self.finish(json.dumps(response))
```

Example: Cone search

URL: http://HOSTNAME/EVN-Archive/search?ra=0.0&dec=0.0&radius=1.0

```
class SearchHandler(APIHandler):
  @tornado.web.authenticated
  def get(self):
    q = { 'ra': self.get_query_argument('ra', default=''),
          'dec': self.get_query_argument('dec', default=''),
          'radius': self.get_query_argument('radius', default=''),
        }
    service = pyvo.dal.TAPService('http://evn-vo.jive.eu/tap')
    result = service.search(f"SELECT target_name, obs_id, t_exptime, s_ra, s_dec,
                            DISTANCE(POINT('ICRS', {q.ra}, {q.dec}),
                                     POINT('ICRS', s_ra, s_dec)) AS dist
                            FROM ivoa.obscore
                            WHERE 1=CONTAINS(POINT('ICRS', {q.ra}, {q.dec}),
                                             CIRCLE('ICRS', s_ra, s_dec,
                                                    {q.radius}))
                            ORDER BY dist ASC")
    response = [{'src': row['target_name'].decode(), 'ra': row['s_ra'],
                 'dec': row['s_dec'], 't_exp': row['t_exp_time'],
                 'dist': row['dist'] } for row in result]
    self.finish(json.dumps(response))
```

Search result as JSON object