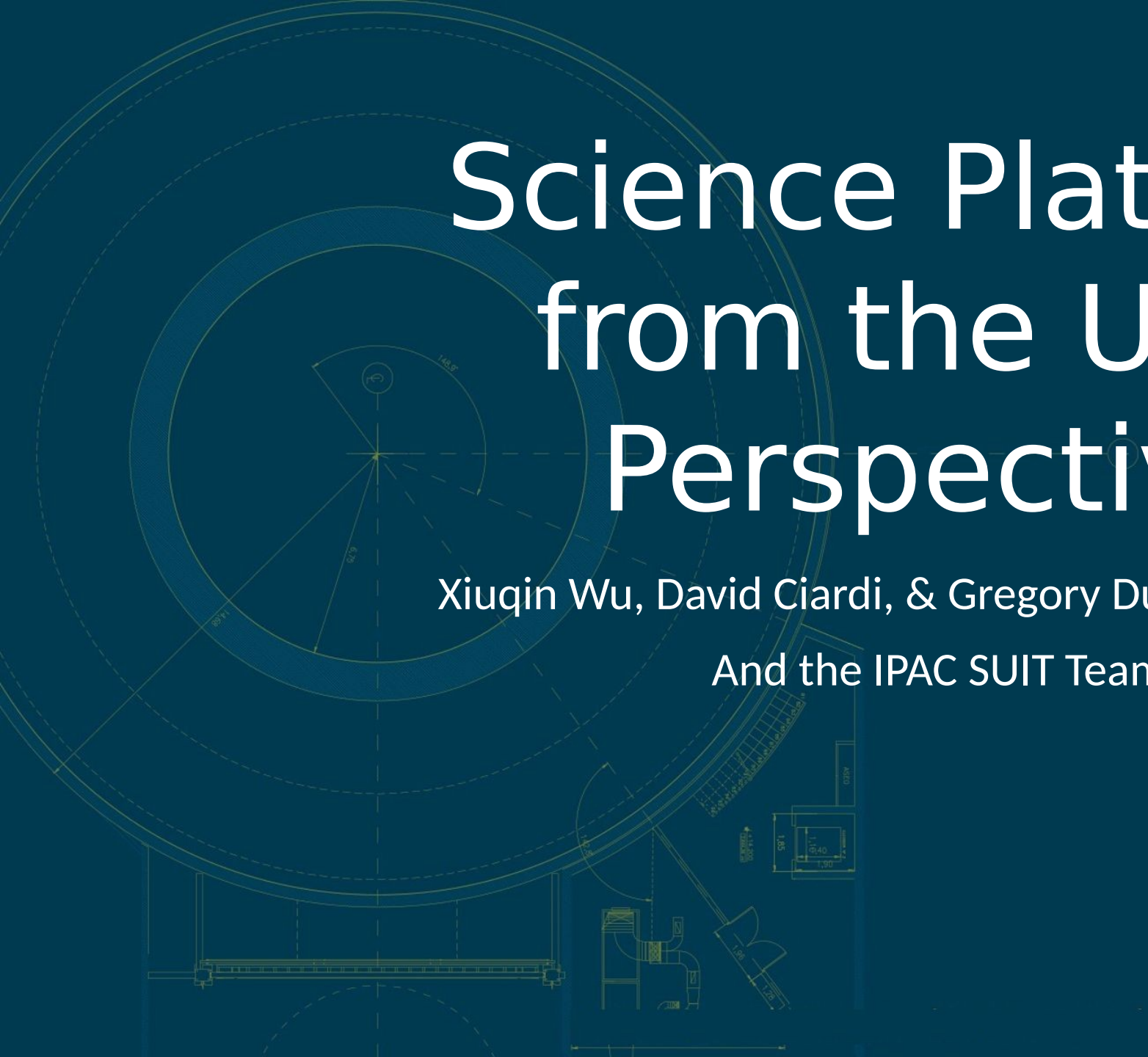
Science Platform from the User Perspective

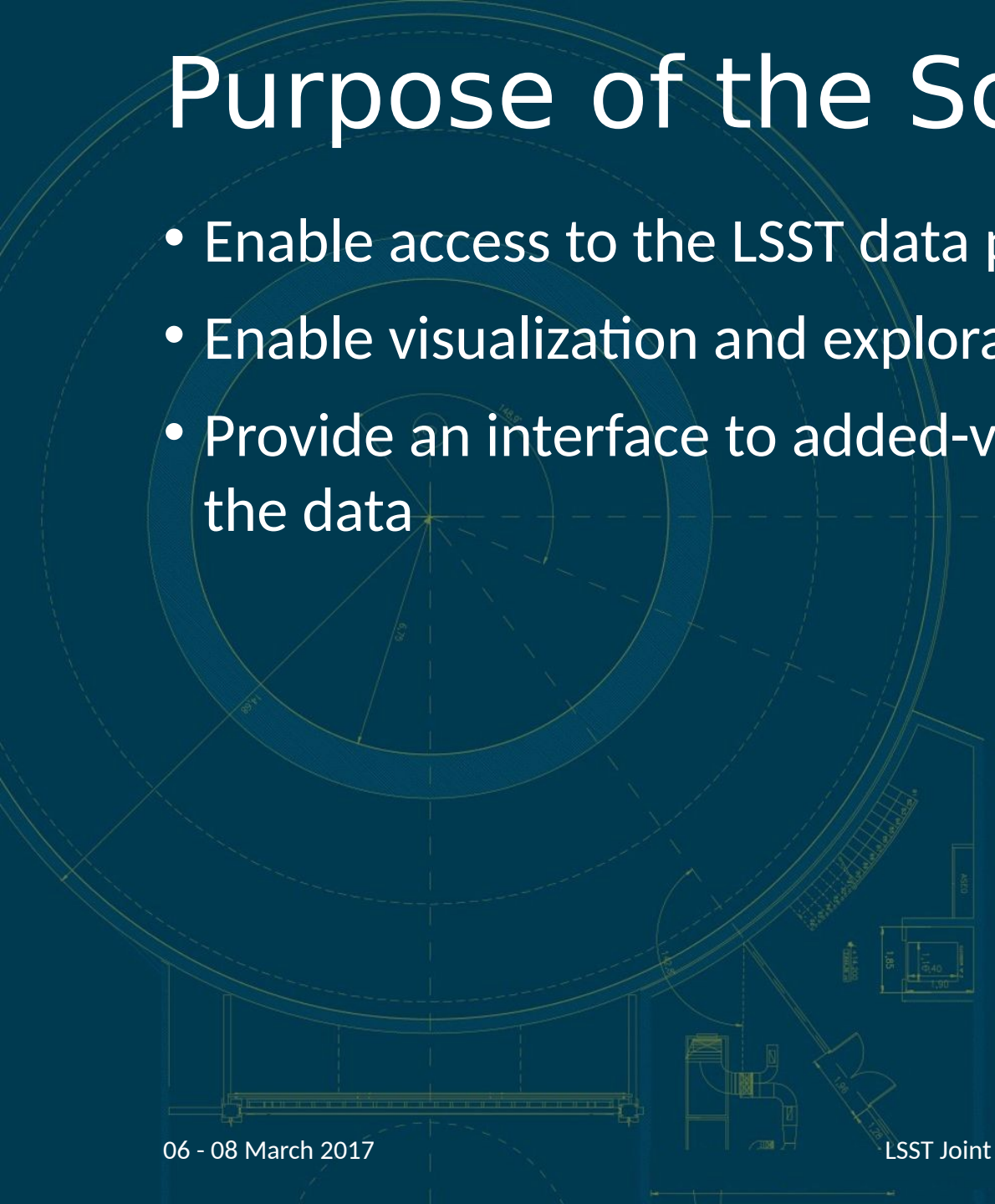
Xiuqin Wu, David Ciardi, & Gregory Dubois-Felsmann

And the IPAC SUIT Team



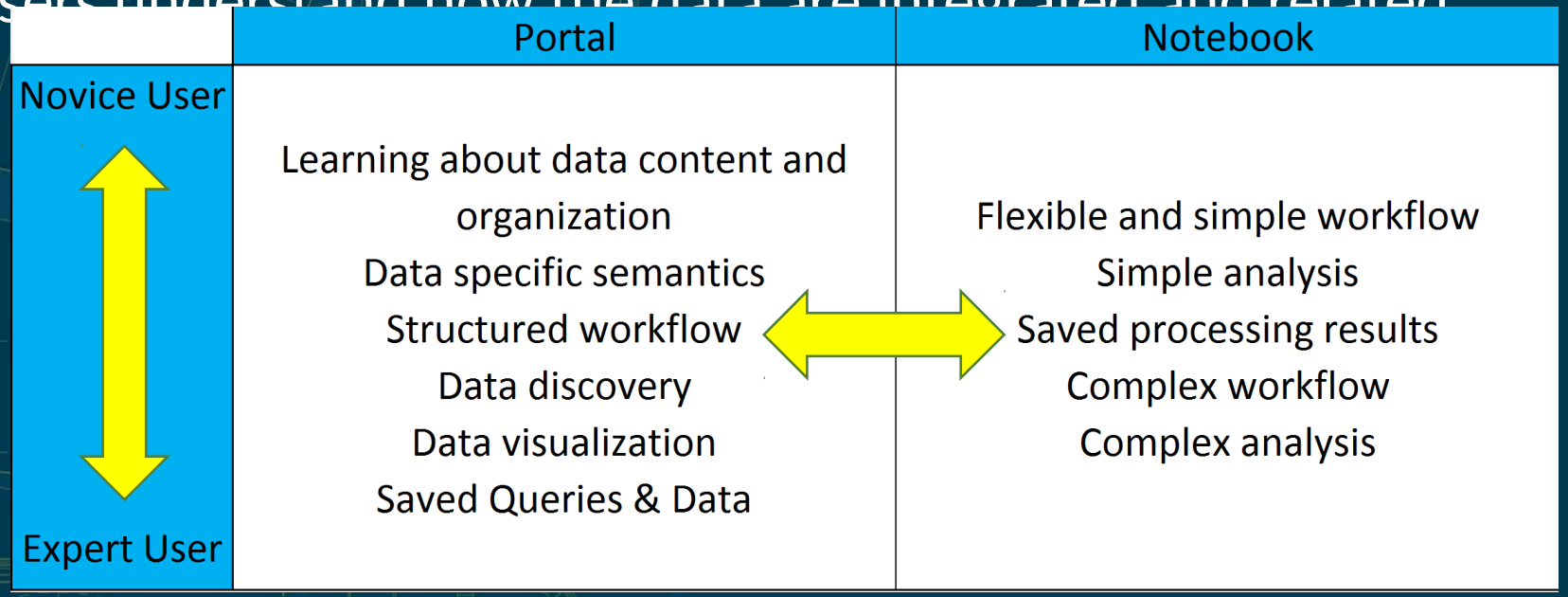
Purpose of the Science Platform

- Enable access to the LSST data products
- Enable visualization and exploration of the LSST data
- Provide an interface to added-value processing and analysis close to the data



Customers of the Science Platform

- LSST Data Rights Scientists, Science Collaborations, and LSST Project
- The Platform needs to be simple enough to engage the general users and flexible enough to meet the needs of the experienced users
- There is a continuum of users, and users will grow and adapt with time.
- Platform needs to enable the users to utilize the contextual information of the LSST data so the users understand how the data are integrated and related



Scientific Priorities for a Minimum Viable System

- Provide a portal with a structured workflow that utilizes the LSST semantic contextual information about the data to enable users to explore and understand the data
- Provide a flexible computational environment (Python Notebook) that has access to the Data Products and the computational infrastructure
- Enable the users to move between the portal and the flexible environments as their needs dictate

Vision of the Science Platform

Portal

- Enables discovery and exploration of the LSST data with structured workflows.
- Contextual information about the semantics of the LSST data is used to guide the workflow and the users.

Make it easy to share, analyze, and visualize results between the structured workflow environment and the flexible workflow environment

Notebook

- Enables exploration and analysis of the LSST data with flexible workflows.
- Environment has access to tools developed by LSST as well as outside tools

Unifying Infrastructure

The *Aspects* of the Science Platform

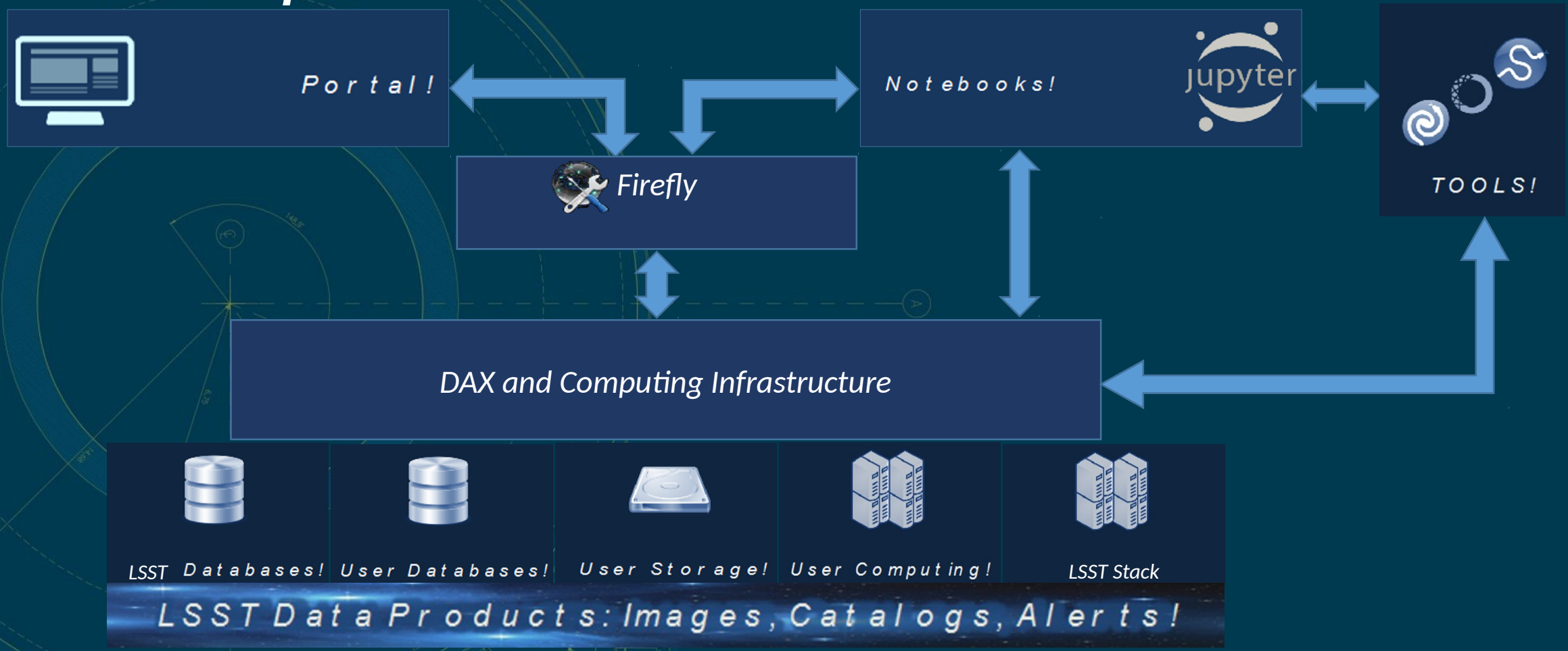
- The Science Platform concept rests on the provision of two* windows into the LSST data products, storage, and computing: the Portal and the Notebook
  - Both rest on a foundation of the DAX services for data access, and the NCSA provisioning services and identity management for flexible computing and storage access
  - Both are client-server web applications
  - Asynchronous DAX services and user identity management allow queries and results, and user data, to be shared between the two
  - The common use of web technologies allows components to be used equally well in either environment

Portal	Notebook
Firefly JavaScript App	Jupyter front end
Firefly servers and load balancer	Jupyter and JupyterHub servers
Python servers for extensions	IPython kernel servers
	Flexible computing for parallelized work

- "Aspects" emphasizes the underlying unity of the system

* There is a third: simple remote access to LSST's Web APIs, VO and otherwise

The *Aspects* of the Science Platform



- The individual components are important.
- But connections that enable workflow are crucial to the users.

The Portal Aspect of the Platform

- Data Discovery – lays out the full breadth of the LSST data with documentation
- Data Connections – presents the semantic links between data
- Data Query – supports both basic (UI-driven) and ADQL/SQL query building
  - Provides access to both the LSST data and standards-compliant external data
- Data Visualization – provides LSST-aware visualizations

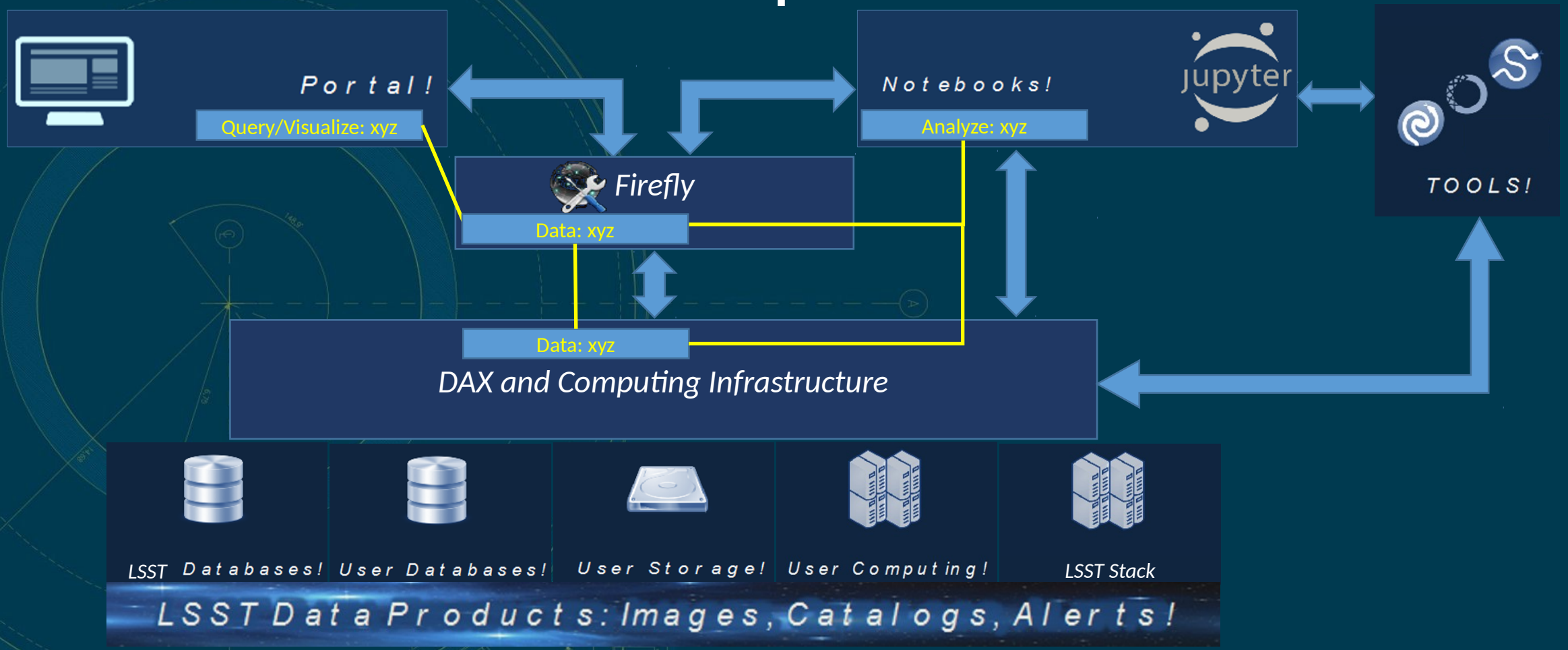
- Exploratory Data Analysis – brushing and linking, filtering, synthetic columns, histogramming, basic astronomical tools such as time-series analysis
- Client-server model enables scalable handling of large result sets
- User Data Access – enables read-write use of workspace and Level 3 Data Products
- Connection to Python – uses the Science Platform's computing resources to enable LSST-specific extensions to the core Firefly capabilities
  - Example: the visualization of the LSST footprint / de-blending model

- Provides structured environment for the novice
- Provides a starting point for the expert, with a seamless transition to interactive analysis in the Notebook Aspect
- Dedicated portals support Observatory processes (QC/QA, commissioning)

The Notebook Aspect of the Platform

- Interactive Python computing environment
  - JupyterHub-based session management
  - Python processes run with the user's identity
- User-customizable and persistent environments
  - Default configuration for new users based on pre-installed LSST stack
  - Allows the use of the full variety of community data analysis tools (e.g., SciPy, AstroPy, matplotlib, Bokeh)
  - Should support multiple saved environments per user to match stack releases to data releases
- Access to LSST data products via both Butler Python API and DAX services
- Access to additional compute resources at the Data Access Centers
  - Should support both "batch" and "interactive parallel" models (c.f. Dask)
- Access to "user workspace" storage as well as DAX-mediated Level 3 data products
- Visualization
  - Evolved version of afw.display supports a variety of back ends
    - Firefly back end supports both in-notebook visualization and push-to-portal (i.e., both 1-window and 2-window approaches). JupyterLab environment supports visualizations in flexible tabs.
  - Users are free to use any other Python-accessible notebook-aware visualization toolkits

Portal-Notebook Aspect Connections



- Connections enable sharing of data between components which enables more complex workflows and analysis

Portal-Notebook Aspect Connections

- Users are not confined to one aspect or the other
  - We can easily achieve seamless connections between the two aspects of the Science Platform
    - Enabled by the common DAX architecture and computing infrastructure
    - Enabled by the use of Javascript tools for visualization, and the rich Firefly APIs
  - Find or create data in one aspect; view or analyze that data in the other aspect
- Queries are shareable across the Portal and the Notebook, e.g.:
  - Build a query in the Portal UI; verify the results by browsing it in the UI; access the results from the Notebook for further analysis
  - Code a complex SQL query in the Notebook; browse the results in the Portal
  - Capture a query formulated in the Portal as code reusable later as a Notebook-driven query
  - Connections can be made either through identity management (DAX knows the queries you have recently performed) or through simple UI actions (e.g., copy/paste of a query token between windows)
- Moving from data discovery to analysis
  - We expect to provide a “one click” means of launching a fresh notebook with pre-loaded access to the results of data identified in the Portal

Portal-Notebook Aspect Connections

- The Firefly data model is accessible from the Notebook, e.g.:
  - afw.display calls can be used to set up overlays of tabular data and images
  - Results of Firefly operations (synthetic columns, interactive filtering/brushing) are retrievable in the notebook
- The Portal can be driven from the Notebook, e.g.:
  - Results of complex computations can be pushed to the Portal and explored, with immediate linkage to associated LSST data products
  - Multiple users can share a Portal view – the system can be used for collaboration
- LSST-aware visualizations developed for the Portal are all naturally usable in the Notebook
  - User experience is common across aspects.
  - Duplication of development effort is minimized
    - Some additional effort is required to package visualizations as widgets

An open question

What exactly is the Python (Notebook) user's interface to the data?

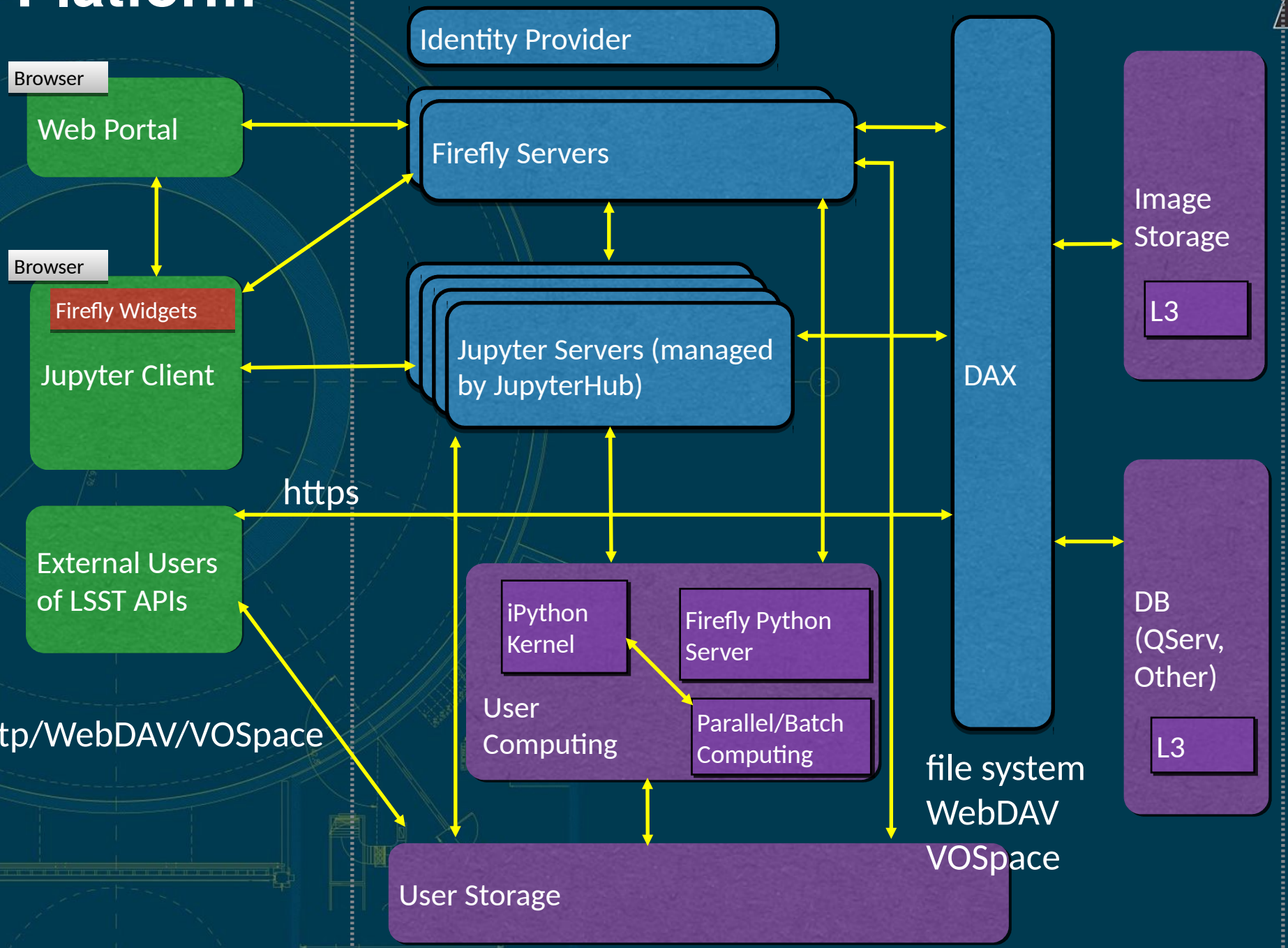
- The IPython kernels run locally *within* the Data Access Center – they have access to whatever is visible to machines on the internal net
- They also have full access to the DAX APIs (VO and otherwise)
  - However, we currently don't plan LSST-aware Python wrappers for these
  - People can (already) call these (Web) APIs from Python and parse the results, but that seems like a below-threshold offering from a project of this scale

- Will we provide the means to "reconstitute" LSST-Python-domain versions of all the data products?
- Will we provide access to all of the Level 1 and Level 2 DPs through the Butler?

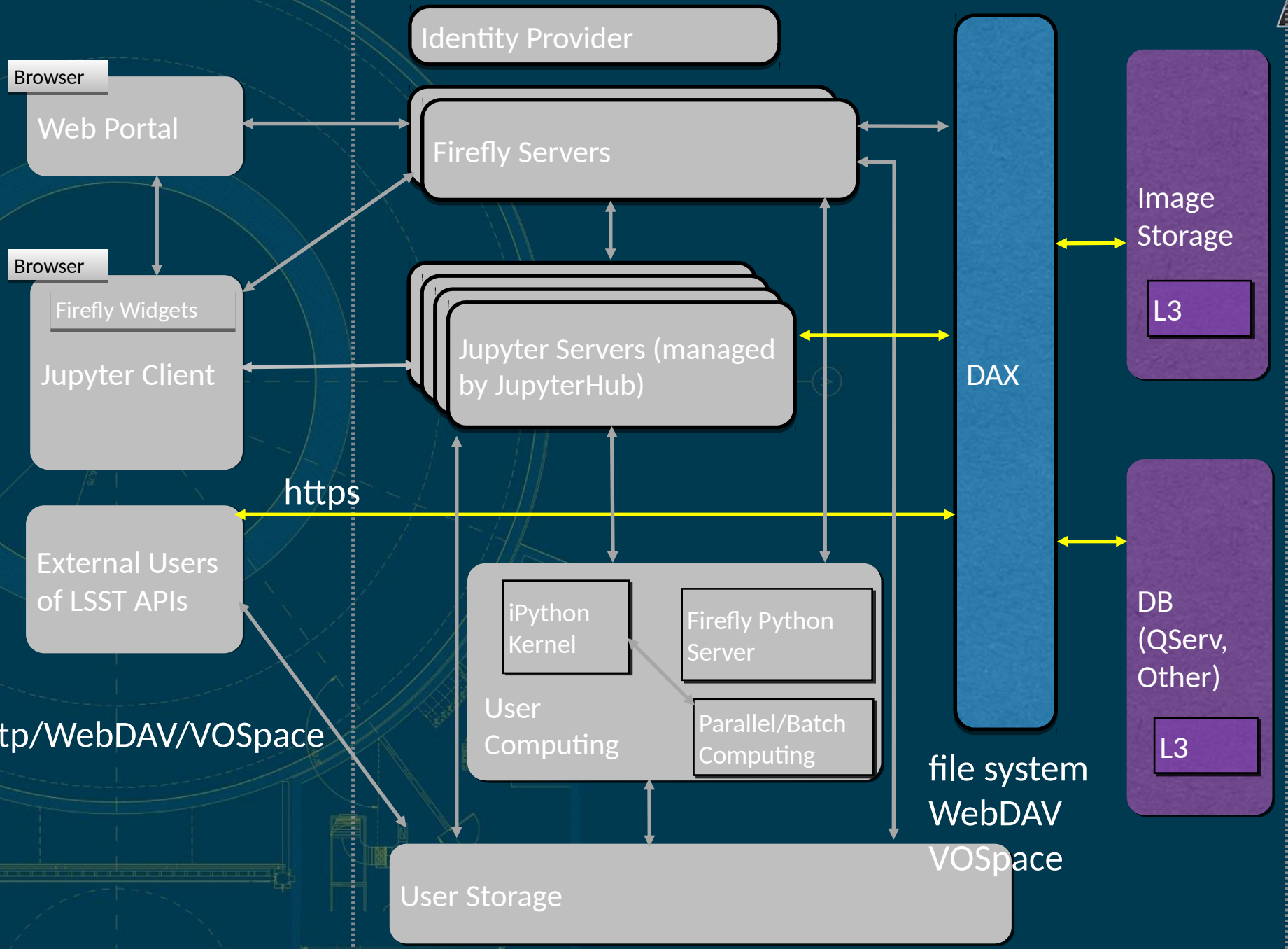
Science Platform Components

- We have identified major components necessary to support the Science Platform vision
- The following set of “wire-diagrams” outline these components and the necessary connections
- The Science Platform is an integrated service with contributions from multiple groups within the DM sub-system
- We have tried to identify areas of responsibility for the DM teams
  - These slides are a “first pass” and need input from the teams for assessment of completeness
  - The identified responsibilities are intended to be starting points of the conversation with the DM team at the JTM
  - There are areas of overlap between teams that need to be identified and scoped

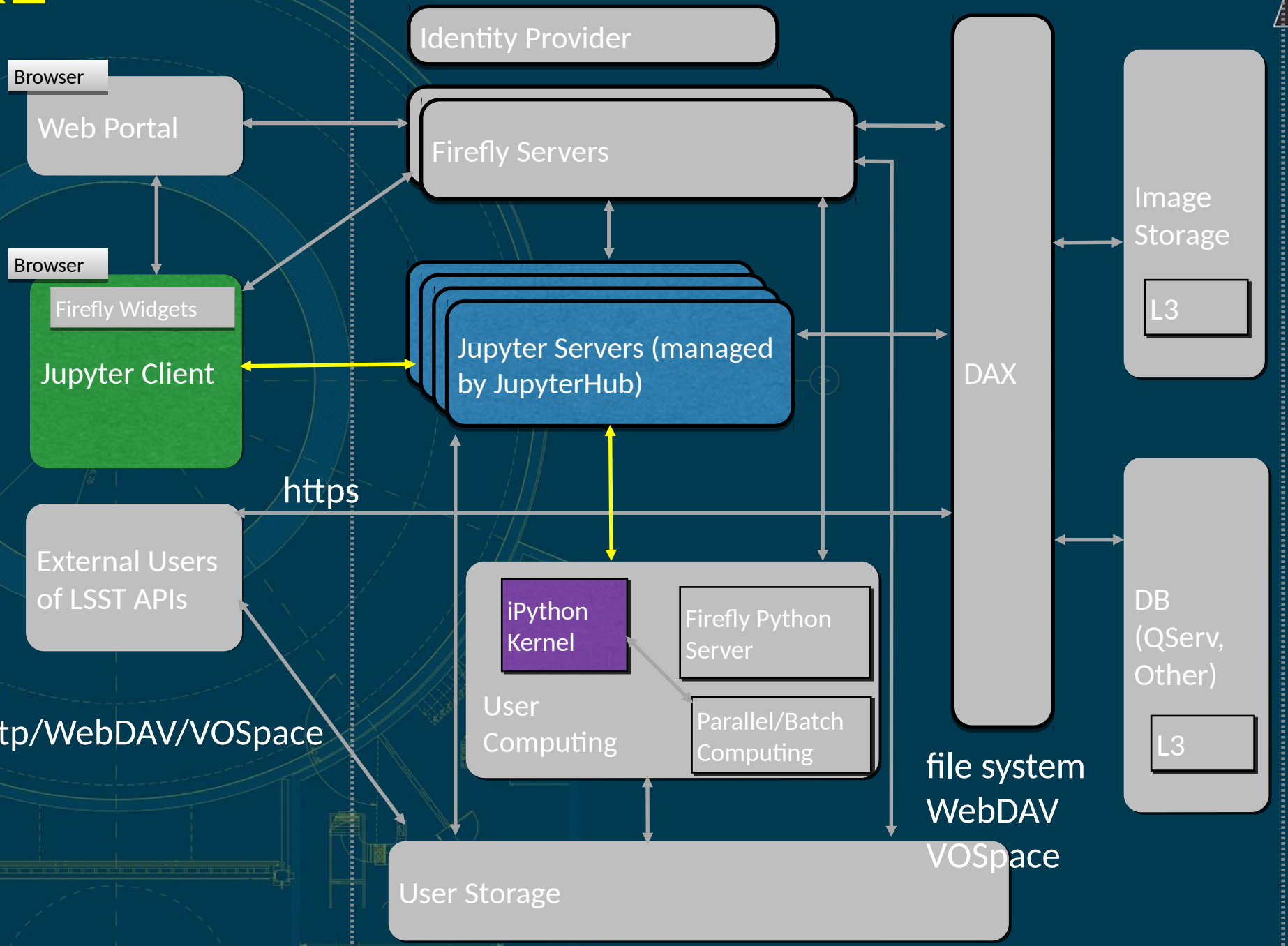
Science Platform



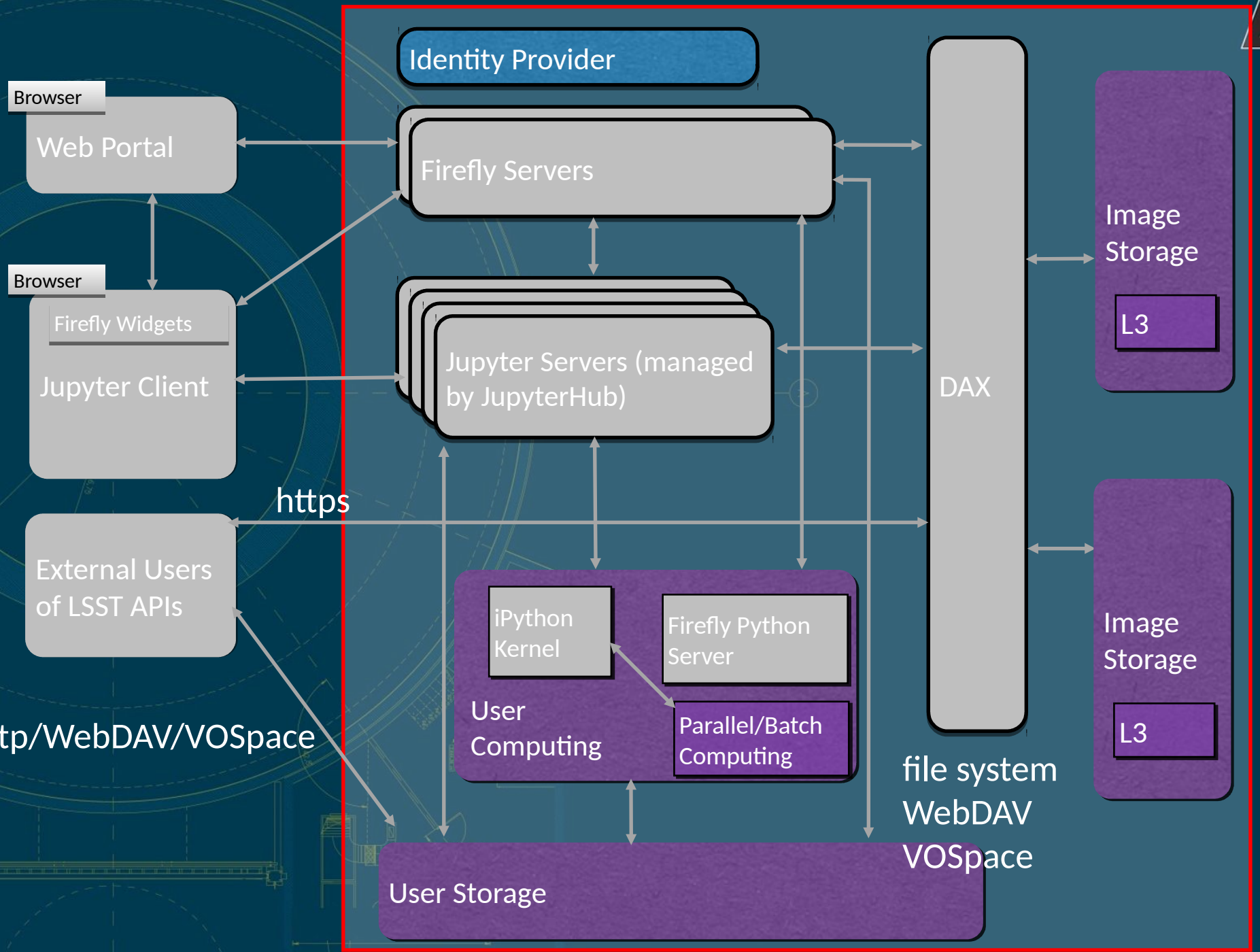
~SLAC


Identity Provider
Browser
Web Portal
Firefly Servers
Image Storage
L3
Browser
Firefly Widgets
Jupyter Client
Jupyter Servers (managed by JupyterHub)
DAX
https
External Users of LSST APIs
iPython Kernel
Firefly Python Server
User Computing
Parallel/Batch Computing
DB (QServ, Other)
L3
sftp/WebDAV/VOSpace
file system
WebDAV
VOSpace
User Storage

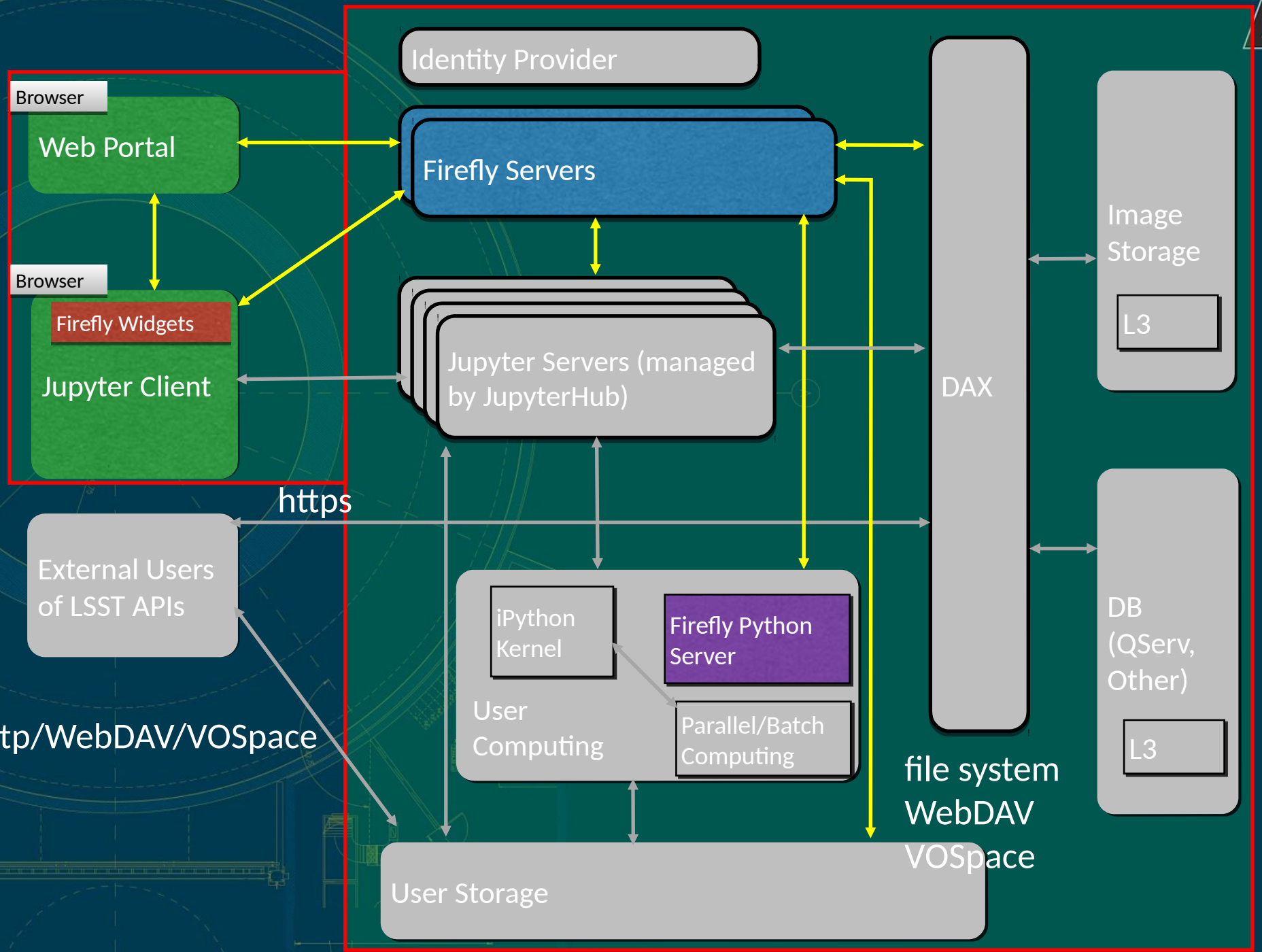
~SQuaRE


Identity Provider
Browser
Web Portal
Firefly Servers
Image Storage
L3
Browser
Firefly Widgets
Jupyter Client
Jupyter Servers (managed by JupyterHub)
DAX
https
External Users of LSST APIs
iPython Kernel
Firefly Python Server
User Computing
Parallel/Batch Computing
DB (QServ, Other)
L3
sftp/WebDAV/VOSpace
file system
WebDAV
VOSpace
User Storage

~NCSA


Identity Provider
Browser
Web Portal
Firefly Servers
Image Storage
L3
Browser
Firefly Widgets
Jupyter Client
Jupyter Servers (managed by JupyterHub)
DAX
https
External Users of LSST APIs
iPython Kernel
Firefly Python Server
User Computing
Parallel/Batch Computing
Image Storage
L3
sftp/WebDAV/VOSpace
file system
WebDAV
VOSpace
User Storage

~IPAC


Identity Provider
Browser
Web Portal
Firefly Servers
Image Storage
L3
Browser
Firefly Widgets
Jupyter Client
Jupyter Servers (managed by JupyterHub)
DAX
https
External Users of LSST APIs
iPython Kernel
Firefly Python Server
User Computing
Parallel/Batch Computing
DB (QServ, Other)
L3
sftp/WebDAV/VOSpace
file system
WebDAV
VOSpace
User Storage

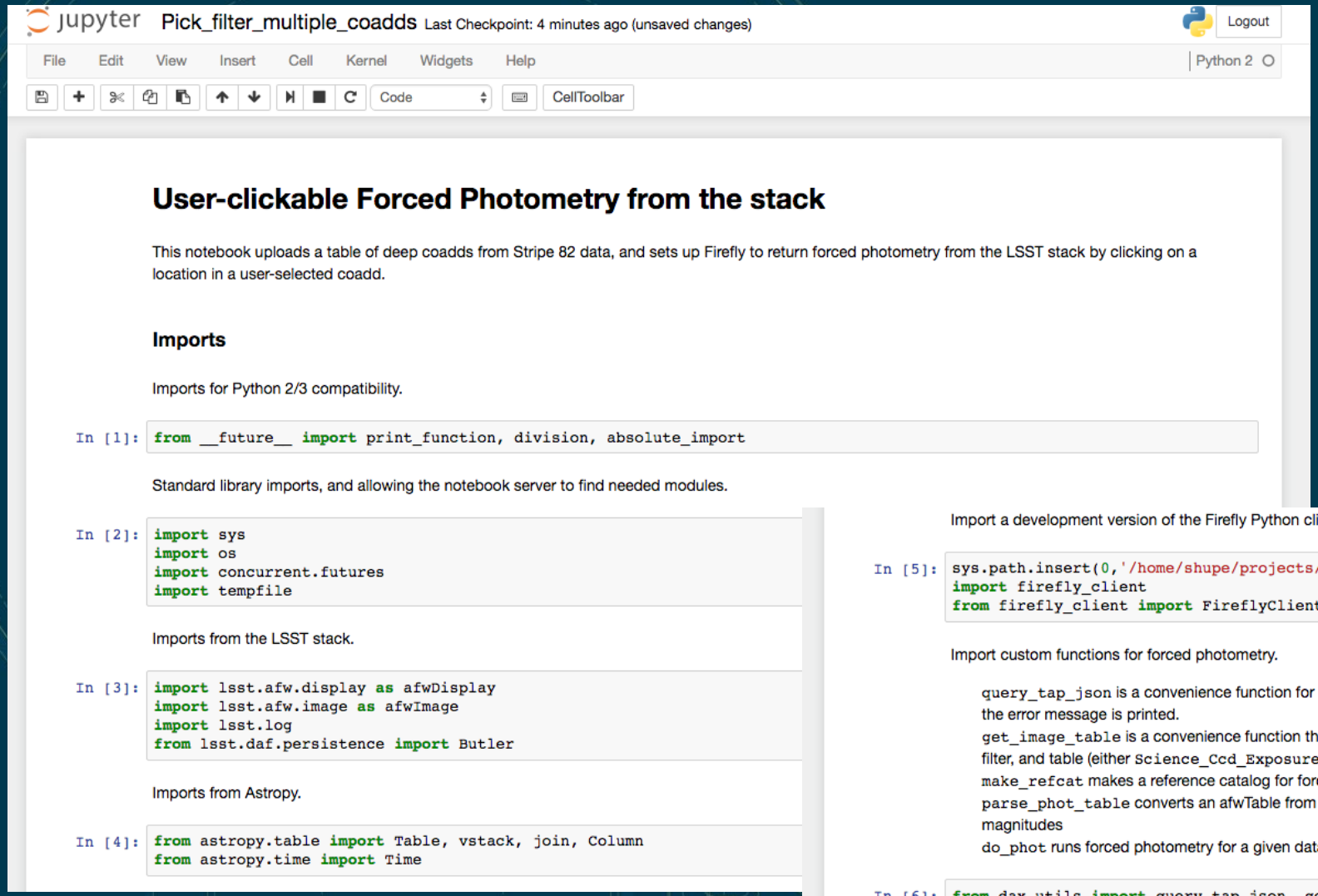
SUIT Functionality for a Minimum Viable System

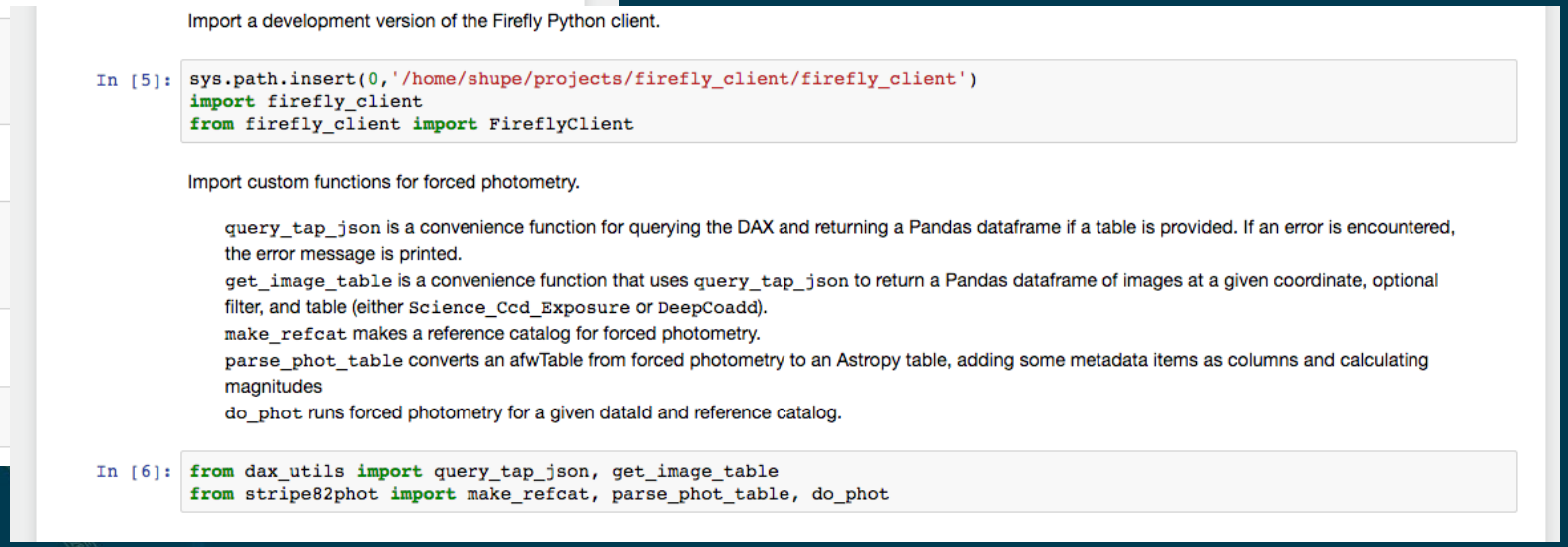
- Searching and access to Level 1 and Level 2 data products (catalogs, images, and alerts)
- Integration with Authentication and Authorization of Users
- Portal-based workflow accounting for the semantics of the LSST data
- Common computational and storage resources shared across the Science Platform visible from both the portal and the notebook environments
- Data exploration connected across the portal and the notebook
- Distributed computational model to enable analysis next to the data
- Alert subscription front-end to the alert mini-broker
- Availability of the Project Documentation

Demo: Creating an Integrated Environment

- Purpose: Demonstrate the distributed computing model comprising the Jupyter-Python Notebook, the Firefly visualization tools, and the Database APIs – all within the single environment of the PDAC
- Demo Science Case: Forced Photometry Driven from Firefly using the Stack
- Highlights
  - Queried the PDAC from both portal and the notebook
  - Invoked a Firefly visualization in the portal from the notebook
  - Used Firefly visualization tools within the notebook
  - Enabled portal to invoke LSST stack and return results through call-back
  - Enabled use of data from call-back in the portal and the notebook
  - Within the portal enabled an LSST-contextual workflow by connecting data types
  - Demonstrates the Science Platform vision of Firefly, Notebook, and Python kernel all running remotely at PDAC, only the browser is local
- Caveats
  - Demonstration is about functionality and does not represent the APIs the project expects to deliver
    - Some Python helper-code written as glue – to query the DAX and run the photometry task from the stack
    - Incomplete APIs: afw.display, Butler access to tabular data,
  - Admin tunnels into PDAC used to simulate a JupyterHub environment
  - Not yet available to the general PDAC user

Demo: Initial Setup





Python imports including

- LSST-stack modules
- Astropy table and time functions
- Firefly client
- DAX query helper functions

Demo: Initialize Firefly and Define Function

Firefly setup

Set up the FireflyClient to point to the local Firefly server.

In [7]:
```
mychannel = 'stripe82'
fc = FireflyClient('lsst-sui-tomcat01.ncsa.illinois.edu:8080', channel=mychannel)
```

At this point, make sure VPN to vpn.ncsa.illinois.edu is active. Then, open a browser window to http://lsst-sui-tomcat01.ncsa.illinois.edu:8080/firefly/lsst-pdac-triview.html;wsch=stripe82

Helper function to do forced photometry for a dataId and to return None if an exception is encountered. The dataId for SDSS data is specified by run, field, camcol, filter.

In [8]:
```
def do_one(dataId):
    global src_cat
    try:
        rval = do_phot(dataId=dataId, refCat=src_cat)
    except:
        return
    return(rval)
```

- Initialize Firefly client to point to the Firefly server on the PDAC
- Add the callback function on the Firefly server for the Forced Photometry function.
- Add an extension to the Firefly browser to make a Forced Photometry button appear in Point mode.

Add the callback (on the Firefly server).

In [11]:
```
plistner = fc.add_listener(callback_forcedphot)
```

Add the Forced Phot extension for all image displays

In [12]:
```
fc.add_extension(ext_type='POINT', plot_id=None, title='Forced Phot',
                 tool_tip='fetch forced photometry at selected point',
                extension_id='fphot',
                 image_src='./button_forced-photometry.png');
```

Demo: Query DAX via Portal or Notebook

Coordinates and filter specification, and image retrieval

A very nice variable source is at RA=45.804433, Dec=0.905573. (For name resolvers, it is V* GI Cet.) Here it is used just for image retrieval, along with the specified filter name.

Other high-amplitude variables with P > 50 days, rAmpl > 3.0:

25.612807 0.291621, ID=1340590, rAmpl=3.217, P=2958d. (2SLAQ J014227.07+001729.8 -- Star.)
28.930942 0.468687, ID=1261335, rAmpl=4.146, P=3321 d. (V* FL Cet -- CV of AM Her type (polar) -- the period is really 87 minutes.)

Shorter periods:

10.62013 -0.869339, ID=196130, iAmpl=8.059, P=0.6d. (SDSS J004228.83-005210.4, White Dwarf.)

QSOs:

2.87667 0.964417 ID=68411, iAmpl=2.218, P=2875 days, ([VV2006] J001130.4+005751 -- Quasar.)
319.572495 0.221337, id=2655567, iAmpl=3.113, P=1453 days
UGC 2479 is 45.167525 0.020461

You may specify any coordinates in the DC_2013 Stripe 82 processing.

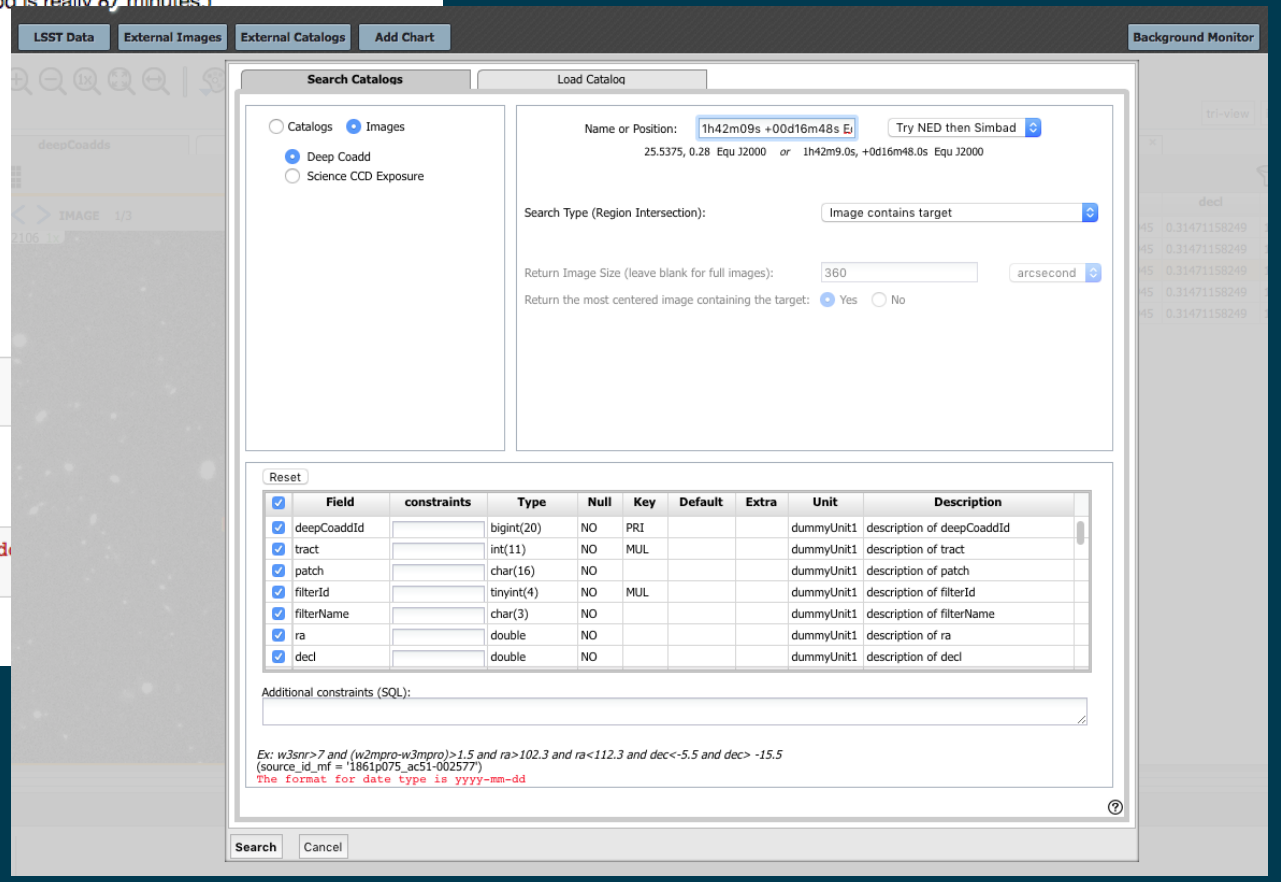
In [13]:
```
myra = 25.6
mydec = 0.3
```

Query the DeepCoadd table for coadds covering the user-specified RA and Dec, for any filter

In [14]:
```
df_coadds = get_image_table(ra=myra, dec=mydec, filter_name=None, table_name='DeepCoad
df_coadds.sort_values(['deepCoaddId'], inplace=True)
```

Verify that df_coadds has at least five entries for the five filters

- Query the DAX on the DeepCoadd table for Stripe 82 at the starting RA and Dec.



Demo: Results Return in Portal or Notebook

Add a URL for downloading the image by ID

```
In [16]: df_coadds['img_url'] = df_coadds.deepCoaddId.map(lambda x:
             'http://lsst-qserv-dax01.ncsa.illinois.edu:5000/image/v0/deepCoadd/id?id=' + str(x))
```

Verify that `df_coadds` has at least five entries for the five filters

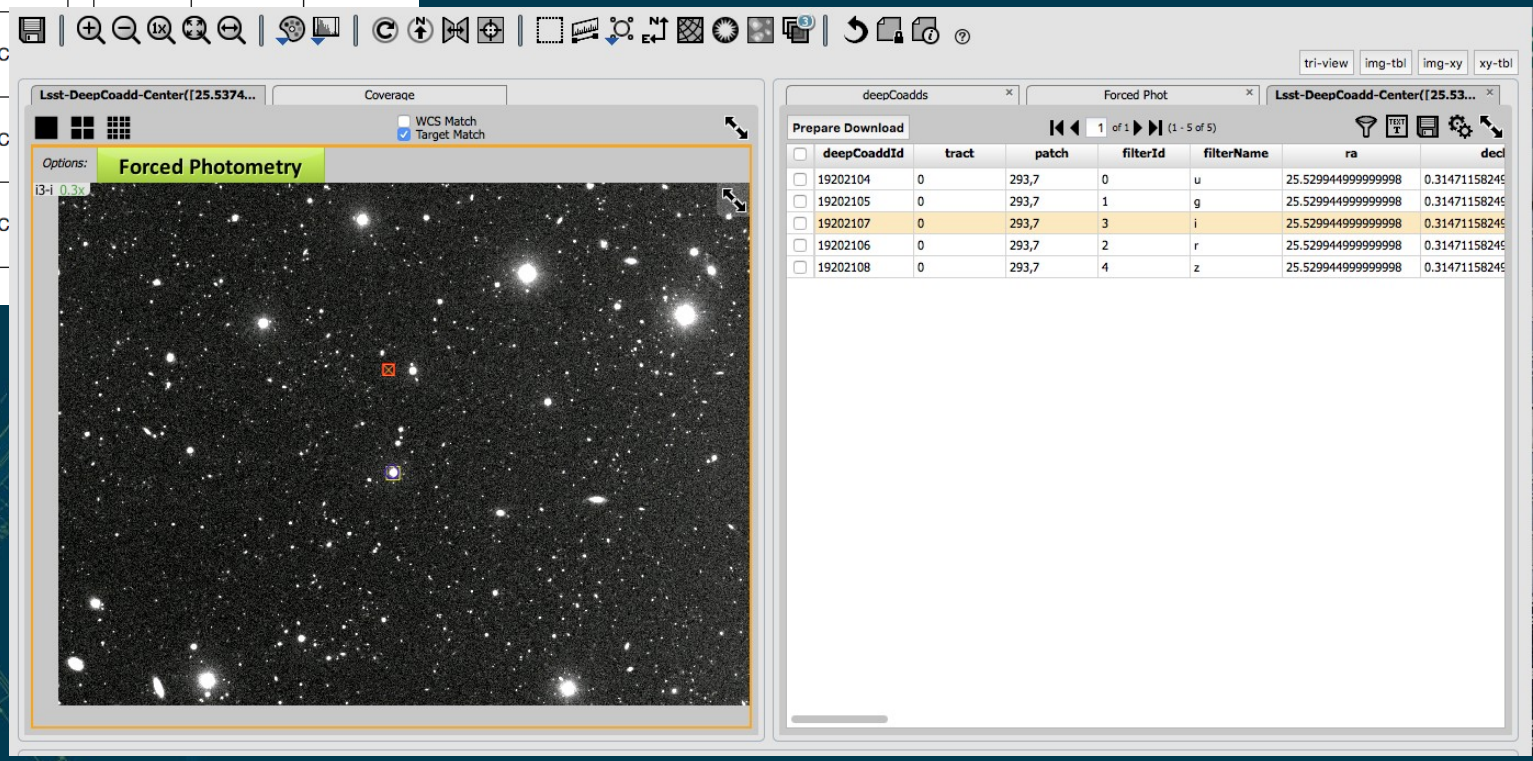
```
In [15]: df_coadds
```

Out[15]:

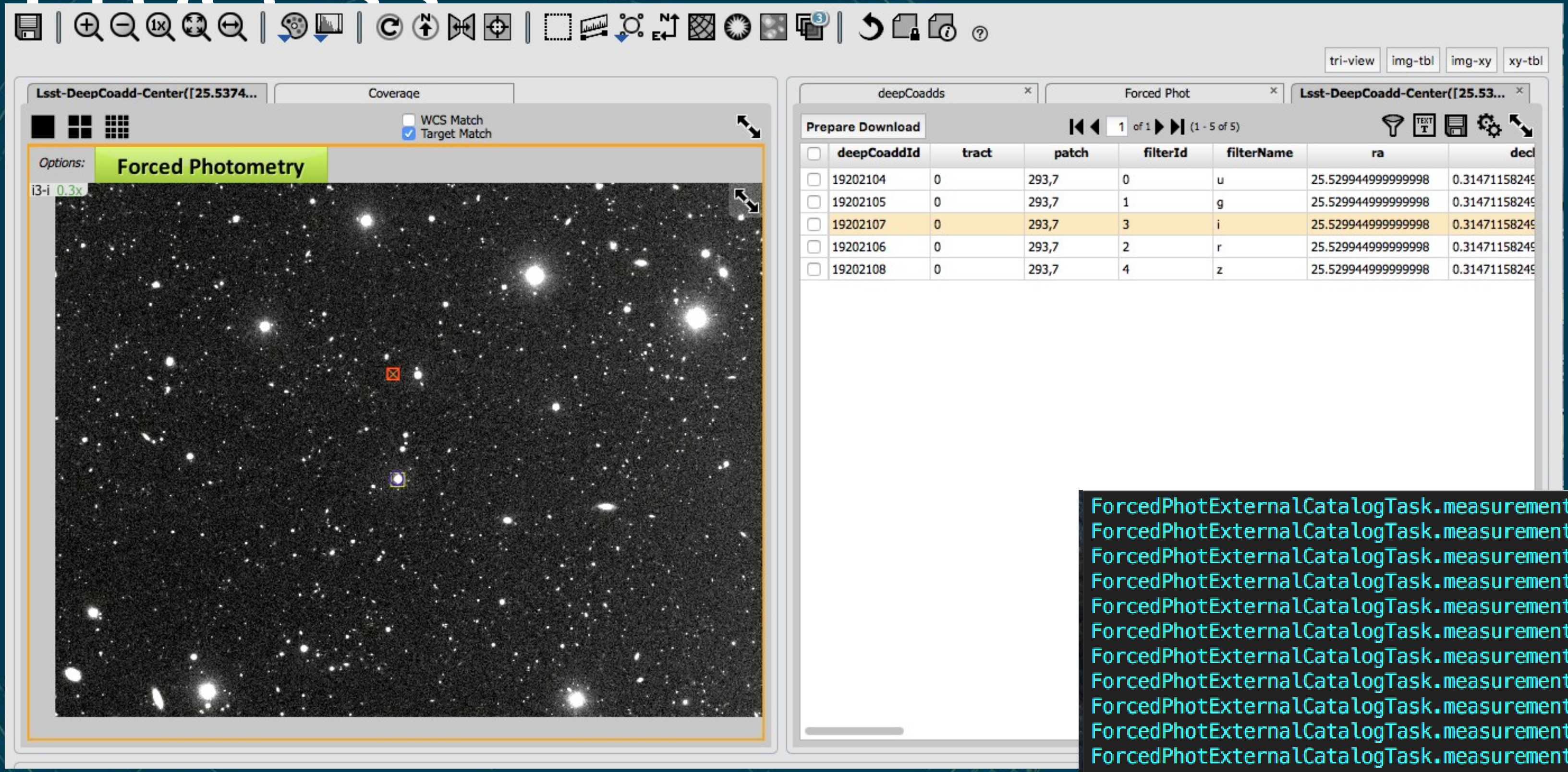
	deepCoaddId	tract	patch	filterId	filterName	ra	decl	htmId20	equinox	raDeSys	...	corner3Ra	corner3Decl	corner4Ra
0	19202104	0	293,7	0	u	25.529945	0.314712	17147177442654	2000.0	ICRS	...	25.416645	0.422899	25.416645
1	19202105	0	293,7	1	g	25.529945	0.314712	17147177442654	2000.0	ICRS	...	25.416645	0.422899	25.416645
3	19202106	0	293,7	2	r	25.529945	0.314712	17147177442654	2000.0	IC				
2	19202107	0	293,7	3	i	25.529945	0.314712	17147177442654	2000.0	IC				
4	19202108	0	293,7	4	z	25.529945	0.314712	17147177442654	2000.0	IC				

5 rows × 34 columns

- Results available in portal and notebook
- Data specific connections enabled in context of portal: images connected to table in portal
- Output driven by notebook commands
- Forced Photometry function available for images



Demo: Function Triggers Python Process



- Forced Photometry function triggers execution of Python LSST-stack process

```
ForcedPhotExternalCatalogTask.measurement INFO: Performing forced measurement on 1 sources
ForcedPhotExternalCatalogTask.measurement INFO: Performing forced measurement on 1 sources
ForcedPhotExternalCatalogTask.measurement INFO: Performing forced measurement on 1 sources
ForcedPhotExternalCatalogTask.measurement INFO: Performing forced measurement on 1 sources
ForcedPhotExternalCatalogTask.measurement INFO: Performing forced measurement on 1 sources
ForcedPhotExternalCatalogTask.measurement INFO: Performing forced measurement on 1 sources
ForcedPhotExternalCatalogTask.measurement INFO: Performing forced measurement on 1 sources
ForcedPhotExternalCatalogTask.measurement INFO: Performing forced measurement on 1 sources
ForcedPhotExternalCatalogTask.measurement INFO: Performing forced measurement on 1 sources
ForcedPhotExternalCatalogTask.measurement INFO: Performing forced measurement on 1 sources
ForcedPhotExternalCatalogTask.measurement INFO: Performing forced measurement on 1 sources
ForcedPhotExternalCatalogTask.measurement INFO: Performing forced measurement on 1 sources
ForcedPhotExternalCatalogTask.measurement INFO: Performing forced measurement on 1 sources
ForcedPhotExternalCatalogTask.measurement INFO: Performing forced measurement on 1 sources
ForcedPhotExternalCatalogTask.measurement INFO: Performing forced measurement on 1 sources
ForcedPhotExternalCatalogTask.measurement INFO: Performing forced measurement on 1 sources
ForcedPhotExternalCatalogTask.measurement INFO: Performing forced measurement on 1 sources
ForcedPhotExternalCatalogTask.measurement INFO: Performing forced measurement on 1 sources
ForcedPhotExternalCatalogTask.measurement INFO: Performing forced measurement on 1 sources
ForcedPhotExternalCatalogTask.measurement INFO: Performing forced measurement on 1 sources
ForcedPhotExternalCatalogTask.measurement INFO: Performing forced measurement on 1 sources
ForcedPhotExternalCatalogTask.measurement INFO: Performing forced measurement on 1 sources
ForcedPhotExternalCatalogTask.measurement INFO: Performing forced measurement on 1 sources
```

Demo: Data Interaction Enables Exploration

Table widget of photometry

```
In [19]: from firefly_widgets import connect
         connect('http://lsst-sui-tomcat01:8080')

In [20]: from firefly_widgets import TableViewer

In [21]: tval = 'placeholder'

In [22]: tv = TableViewer(url_or_path=tval, title='Forced Phot',
                          tbl_id = 'Forced Phot',
                          width='600px', height='400px')

In [23]: tv
```

placeholder | ${temp-files}/upload_1919959...

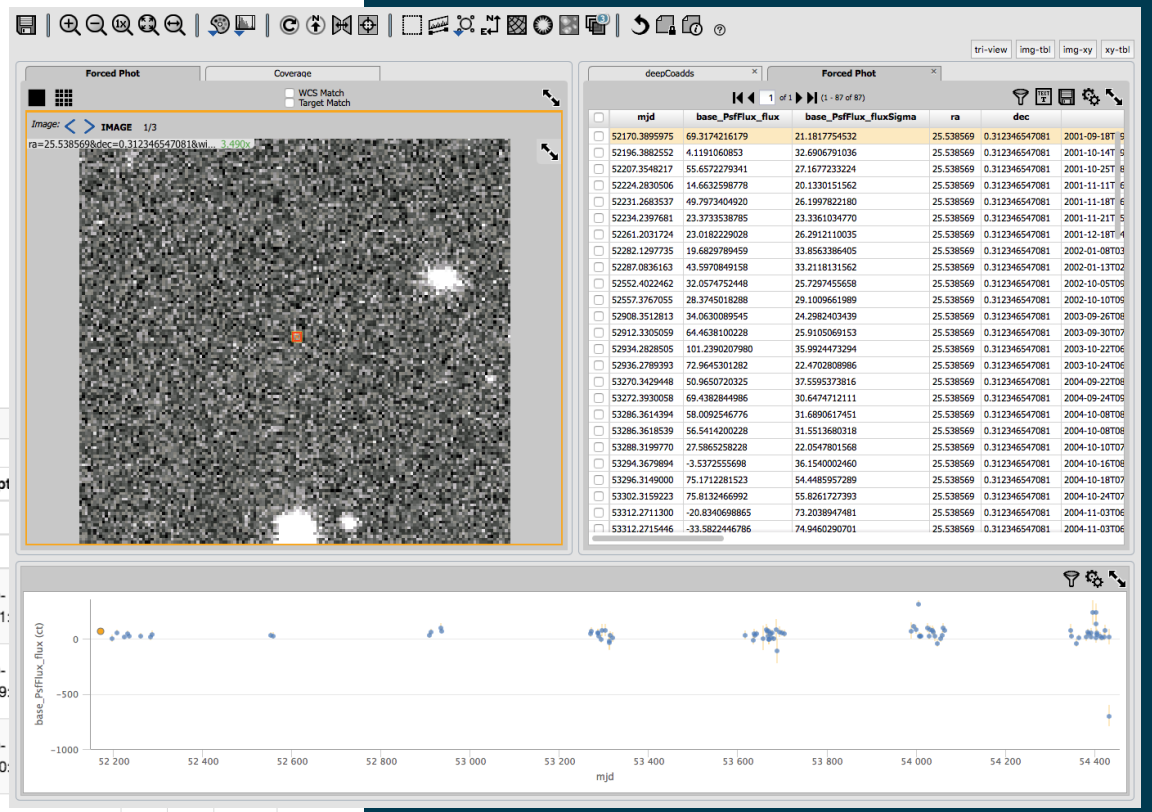
1 of 1 (1 - 87 of 87)

mjd	base_PsfFlux_flux	base_PsfFlux_fluxSigma	ra	dec
52170.3895975	69.3174216179	21.1817754532	25.538569	0.312346547081
52196.3882552	4.1191060853	32.6906791036	25.538569	0.312346547081
52207.3548217	55.6572279341	27.1677233224	25.538569	0.312346547081
52224.2830506	14.6632598778	20.1330151562	25.538569	0.312346547081
52231.2683537	49.7973404920	26.1997822180	25.538569	0.312346547081
52234.2397681	23.3733538785	23.3361034770	25.538569	0.312346547081
52261.2031724	23.0182229028	26.2912110035	25.538569	0.312346547081
52282.1297735	19.6829789459	33.8563386405	25.538569	0.312346547081
52287.0836163	43.5970849158	33.2118131562	25.538569	0.312346547081
52552.4022462	32.0574752448	25.7297455658	25.538569	0.312346547081
52557.3767055	28.3745018288	29.1009661989	25.538569	0.312346547081
52908.3512813	34.0630089545	24.2982403439	25.538569	0.312346547081
52912.3305059	64.4638100228	25.9105069153	25.538569	0.312346547081
52934.2828505	101.2390207980	35.9924473294	25.538569	0.312346547081
52936.2789393	72.9645301282	22.4702808986	25.538569	0.312346547081

```
In [24]: tv.selection()
```

Out[24]: <Table length=87>

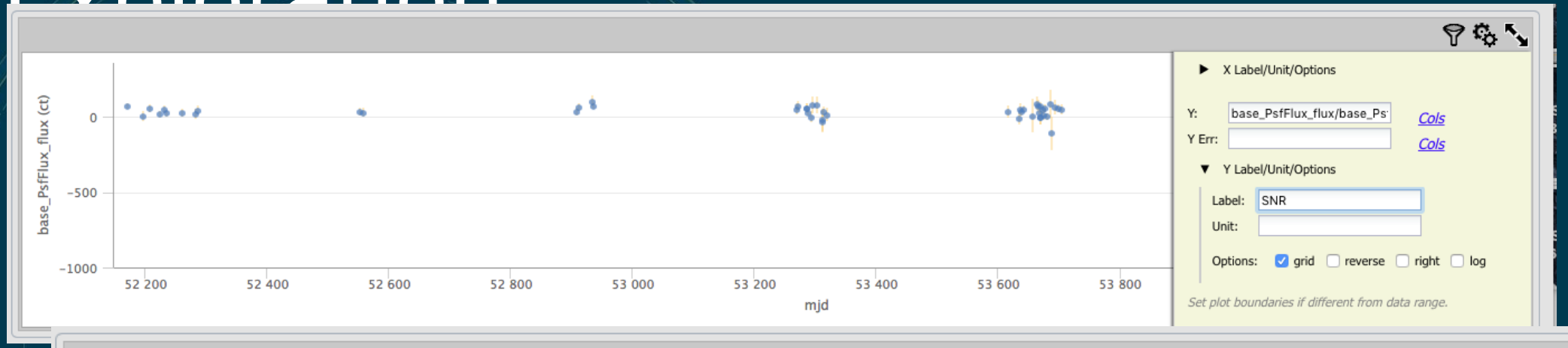
mjd	base_PsfFlux_flux	base_PsfFlux_fluxSigma	ra	dec	expMidpt				
	ct	ct		deg					
float64	float64	float64	float64	float64	str30				
52170.3895975	66.784811304	21.177059098	25.538569	0.312302546427	2001-09-18T09:21:				
52196.3882552	11.1389714544	32.7448112994	25.538569	0.312302546427	2001-10-14T09:19:				
52207.3548217	47.1346952678	27.0856588179	25.538569	0.312302546427	2001-10-25T08:30:				
52224.2830506	6.54654962581	20.0134894456	25.538569	0.312302546427	2001-11-11T06:47:35.571328512Z	2700	36	4	r
52231.2683537	47.7502228036	26.1841560388	25.538569	0.312302546427	2001-11-18T06:26:25.759168000Z	2728	583	4	r

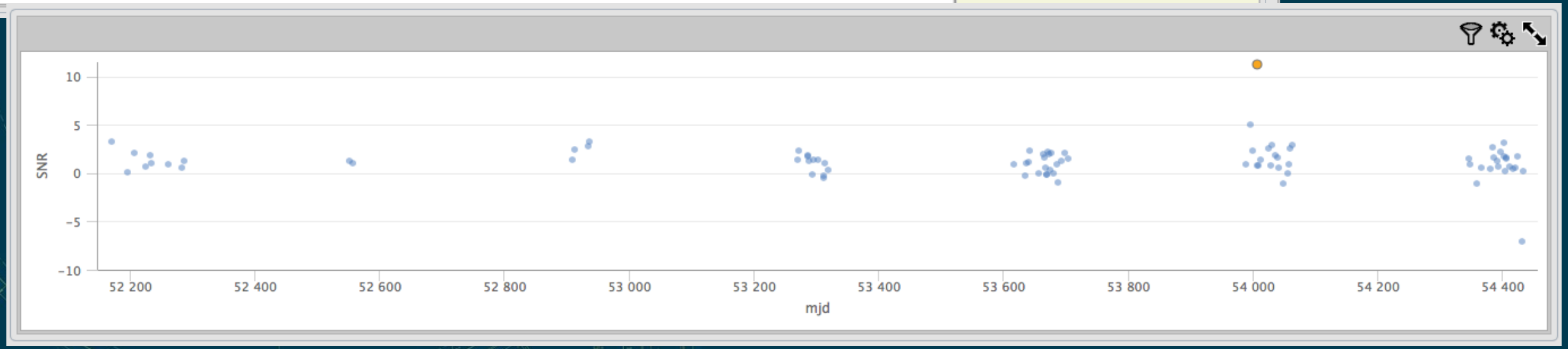


- Results available in Notebook as Python object and as Firefly table
- Interactive tools available inline in notebook as well as a structured workflow in browser

- Data specific connections enabled in context of portal: images connected to table in portal
- Connected data model enables selection and visualization of single visit images

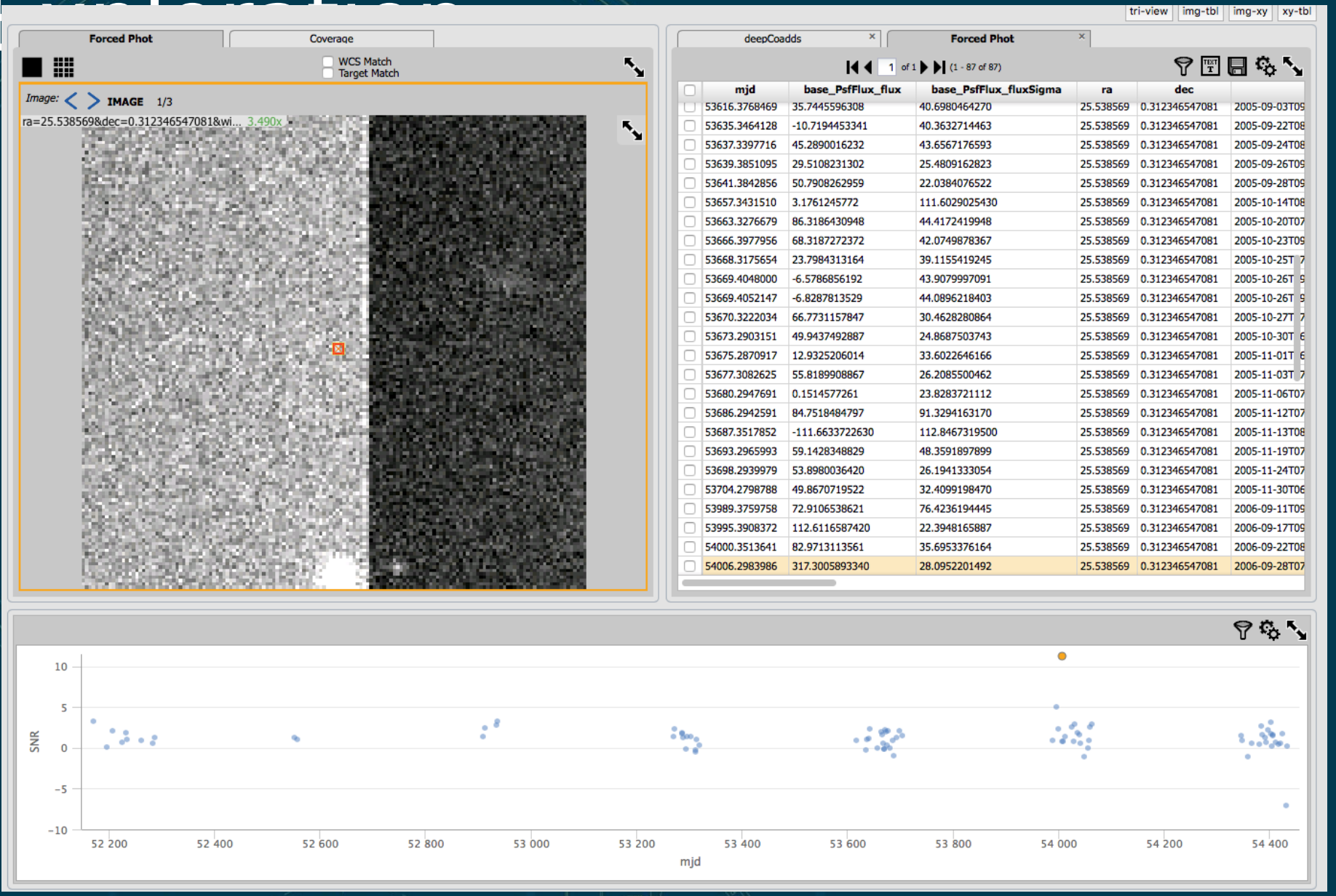
Demo: Data Interaction Enables Exploration





- Interactive plotting tools enable math for immediate exploration of the data
- Flux with errors plotted vs time changed to plotting SNR vs time

Demo: Data Interaction Enables Exploration

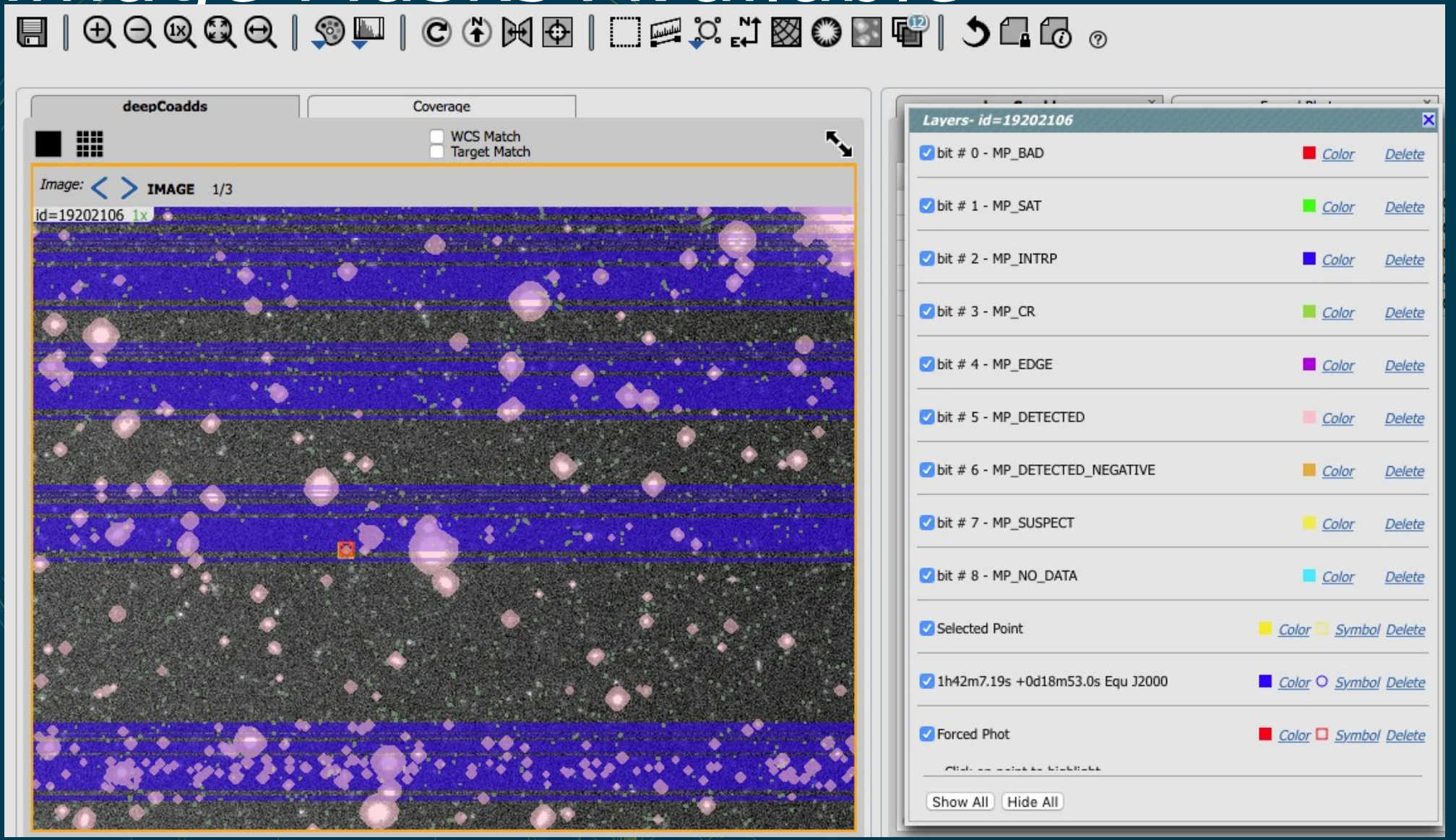


- Connection of images, tables, and plots enables exploration of data
- In this case, high SNR data point caused by image artifact clearly seen when data point selected and associated image displayed

Demo: Creating an Integrated Environment

- Purpose: Demonstrate the distributed computing model comprising the Jupyter-Python Notebook, the Firefly visualization tools, and the Database APIs – all within the single environment of the PDAC
- Demo Science Case: Forced Photometry Driven from Firefly using the Stack
- Highlights
  - Queried the PDAC from both portal and the notebook
  - Invoked a Firefly visualization in the portal from the notebook
  - Used Firefly visualization tools within the notebook
  - Enabled portal to invoke LSST stack and return results through call-back
  - Enabled use of data from call-back in the portal and the notebook
  - Within the portal enabled an LSST-contextual workflow by connecting data types
  - Demonstrates the Science Platform vision of Firefly, Notebook, and Python kernel all running remotely at PDAC, only the browser is local
- Caveats
  - Demonstration is about functionality and does not represent the APIs the project expects to deliver
    - Some Python helper-code written as glue – to query the DAX and run the photometry task from the stack
    - Incomplete APIs: afw.display, Butler access to tabular data,
  - Admin tunnels into PDAC used to simulate a JupyterHub environment
  - Not yet available to the general PDAC user

Image Masks Available



- Image data masks controllable by the API and the GUI
- Prioritized the functionality to support the Pipeline development
- Operational functionality on the PDAC for the Stripe82 data

The API aspect of the Science Platform

Encompasses all externally visible APIs provided from the Data Access Centers

- Web APIs
  - “Zero-install” access to LSST data for remote users
  - VO standard data retrieval protocols (TAP, SIA, etc.) implemented by the DAX team
  - Additional LSST-specific DAX APIs
    - Currently cutouts are made available in this way; we should evaluate a SODA solution
    - May include a Web “Remote Butler” API which provides the data-naming and data-relationship functions of the Butler
  - “Workspace” APIs for user data space access and management (WebDAV or VOSpace, tbd)
  - Enable the use of community tools that are based on VO standards

- Python APIs
  - “The stack” is intended to be installable by remote users
  - An unresolved question for the re-plan: Python APIs for retrieving / reconstituting LSST-domain Python data objects via remote use of the Web APIs (VO et al.)
    - Could be implemented by wrapping the Web Remote Butler in Python

Support for remote operation

- When you run in the Notebook Aspect, your *Python process is at the DAC*
- People who start working in the DAC environment and create notebooks will reasonably ask: can use my notebook off-site?
  - This is clearly related to the "API Aspect" of the Science Platform
  - We also should consider the case of completely offline operation
- Some considerations:
  - Read-only viewing: we will use Jupyter's widget facilities to ensure that a frozen snapshot of any LSST-developed visualizations is saved with notebooks
  - Remote operations: we need to understand the differences between data access capabilities in Python *at the DAC* and *remotely*
    - Will we support remotely-running Butlers accessing data in the DACs?
    - Will we design a system that facilitates exporting data subsets to remote systems and setting up Butler access there?
  - Disconnected operations:
    - If you have local data and run a local Jupyter server and IPython kernel, you should be able to work

Summary

- Defined a vision for the Science Platform
  - Fulfills the purposes of the platform
  - Enables use of the platform by a continuum of users from novice through expert
- Vision is have a portal and notebook environment connected via shared resources to enable a seamless workflow for both the novice and expert users
- Defined a minimum viable system for the science platform
  - Scientific Drivers
  - Minimum set of functionality
  - Identified the components and the connections necessary
- Built demonstration system that demonstrates the aspects of the Science Platform by connecting the Jupyter-Python Notebook, the Firefly visualization tools, and the Database APIs

About Firefly…

- Firefly is open-source software
- A set of tools in a client-server architecture
  - Integration of native astronomical image visualization (no pre-processing required)
  - Visualization of tabular data as plots, histograms, and image overlays integrated with image visualization
  - Interactive and distributed
  - Brushing and linking – shared data model across multiple images and tables

Server	Client
• Data model management, I/O, user session management • Data model supports server-side assistance with large data sets (e.g., decimation, paging) • Robust multithreaded Java framework serves large number of simultaneous users • Scalable deployment • Search Processor abstraction handles many sources of data – VO interfaces, files, custom • Framework for calling external code to extend server's capabilities (e.g., in Python) • Remote APIs for bidirectional data transfer, visualization, and session control	• Visualization and interactivity • Built on open-source React.js framework, Highcharts interactive plotting package • JavaScript APIs for customization of layout, features, and interactivity • Readily integrated into Jupyter / JupyterLab environment as "widgets"

Firefly Capabilities

Portal-building Toolkit	API	Server Infrastructure	Central Data Model
• Query-building • Name resolution • Layout and UI	• Embedded Firefly and Jupyter widgets (JavaScript API) • Remote control and data exchange (Remote/Python API)	• Service and data integration • Extensible and scalable • Security layer • Cache System • Background management	• Definable data relations • Multiple, synchronized views of data

Image Visualization	Tabular Data Visualization	2D Charts and Histograms	Analysis Tools
• Distributed astronomical image visualization • Sensitive to multiple WCS representations • Native FITS visualizer – no preprocessing required • Supports many DS9 like features: zoom, rotate, magnifier, thumbnail, color, stretch, 3-color, region annotations, etc. • Overlays tabular data	• Distributed data • Supports large data sets, greater then 1 million rows • DB-like filtering and sorting • Table viewing	• Tied to Data Model • Supports decimated plotting for large data sets	• Computations on columns • Adaptive histogramming • Time series analysis

Firefly Applications…

Specific collections of tools and behavior configured for different purposes

- In production in the IRSA archives
  - Many Tools: WISE, Spitzer, FinderChart, IRSAViewer, Planck, PTF
- PDACv1
  - Basic application demonstrating integration with DAX data sources
  - Time Series Viewer
- The Science Platform “Portal aspect” will evolve from these