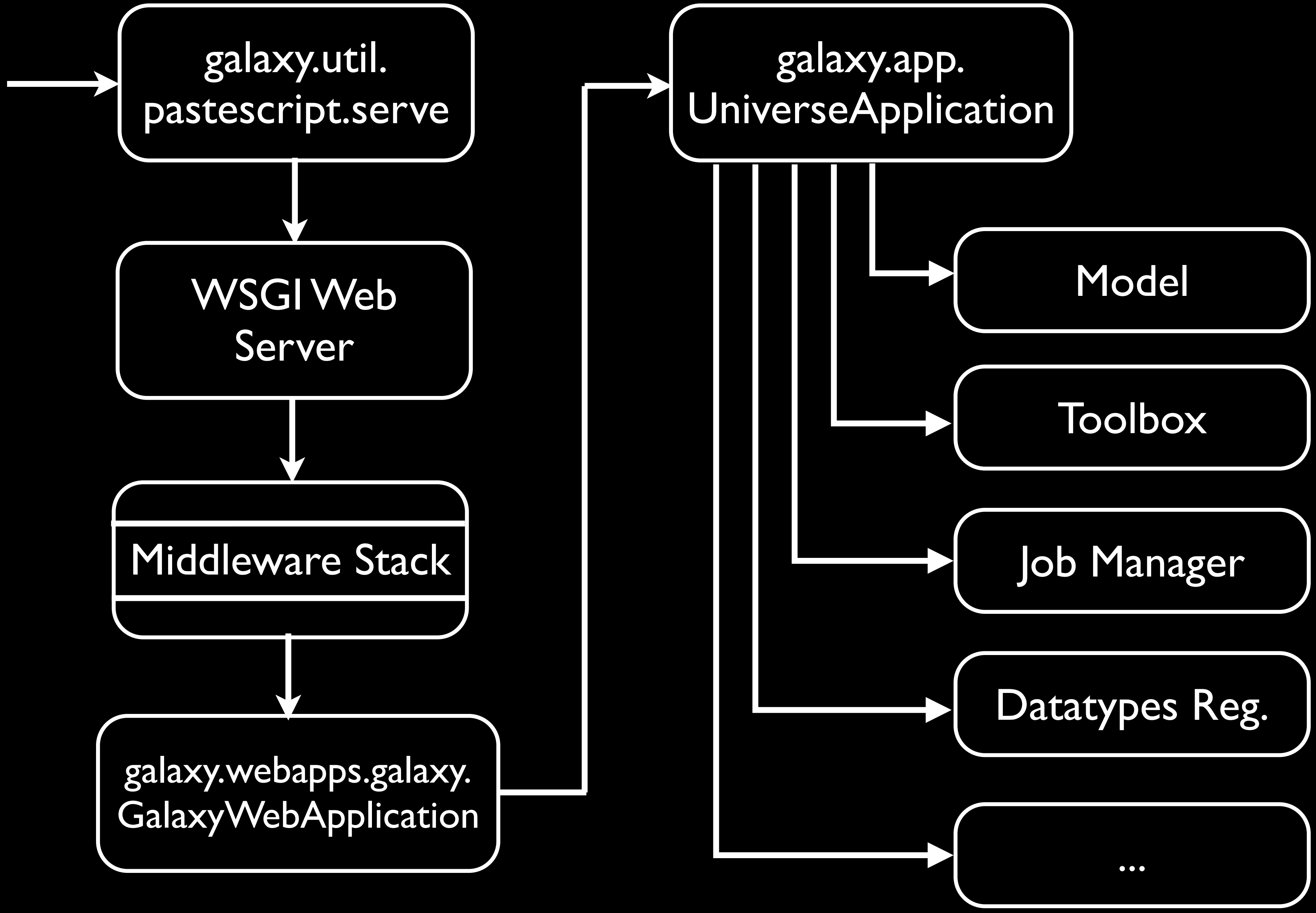
# Galaxy

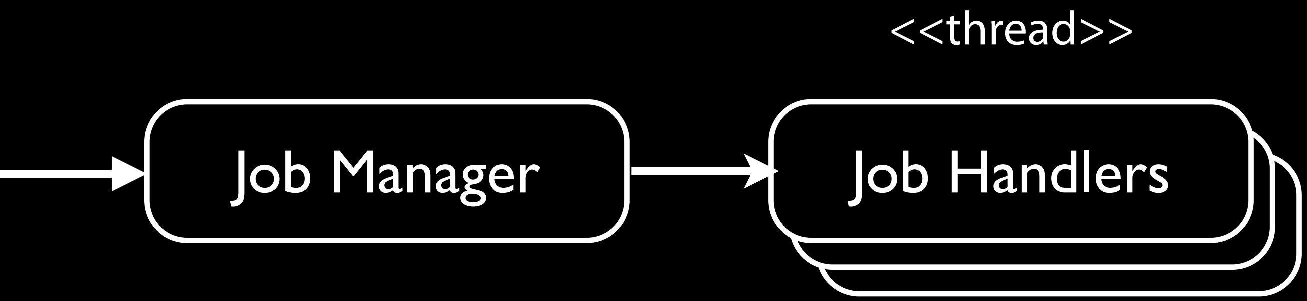
## Code and storage architecture

www.getgalaxy.org

@jxtx / @natefoo / #usegalaxy

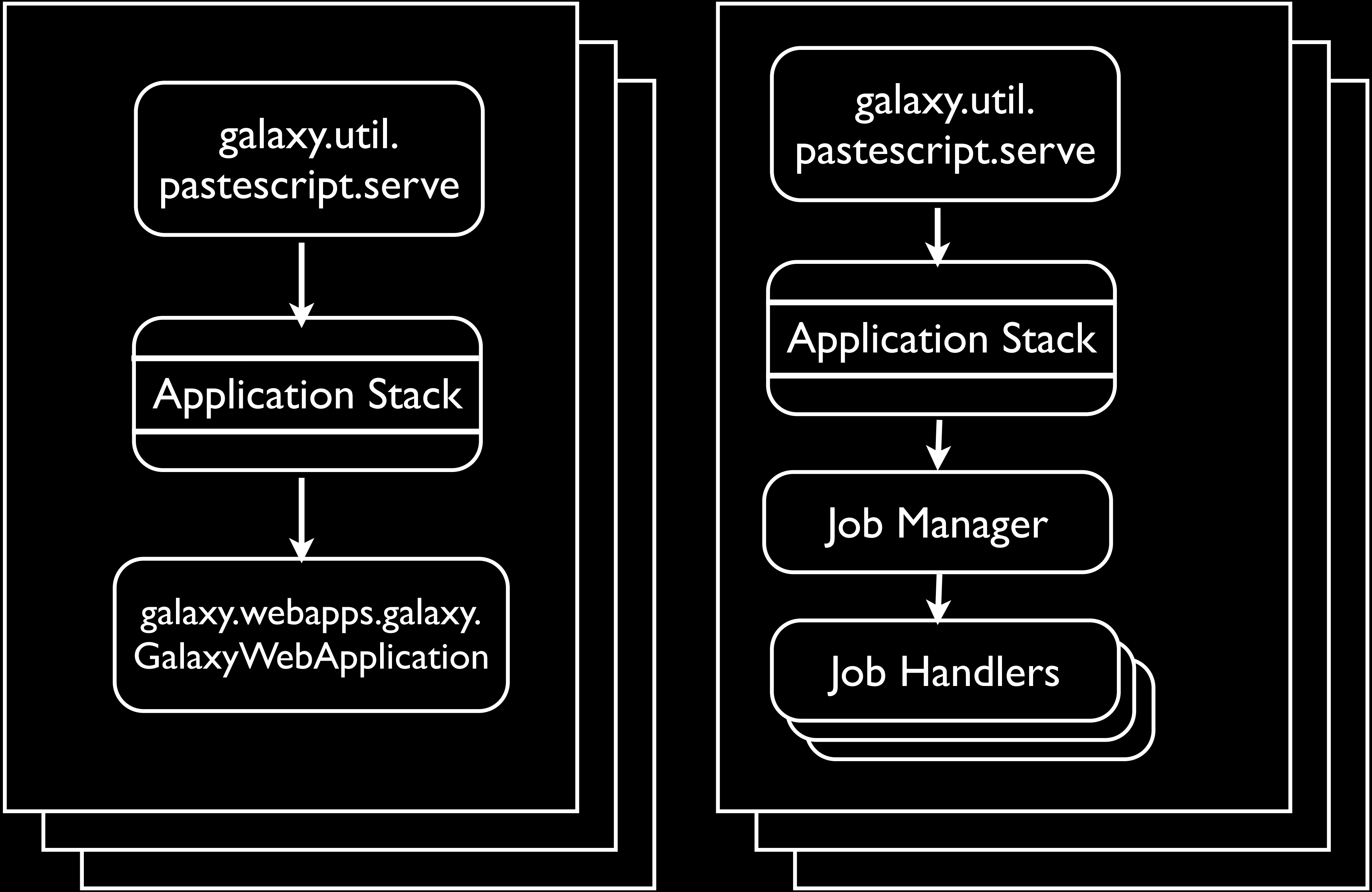
## Galaxy application architecture


galaxy.util.
pastescript.serve
WSGI Web
Server
Middleware Stack
galaxy.webapps.galaxy.
GalaxyWebApplication
galaxy.app.
UniverseApplication
Model
Toolbox
Job Manager
Datatypes Reg.
...

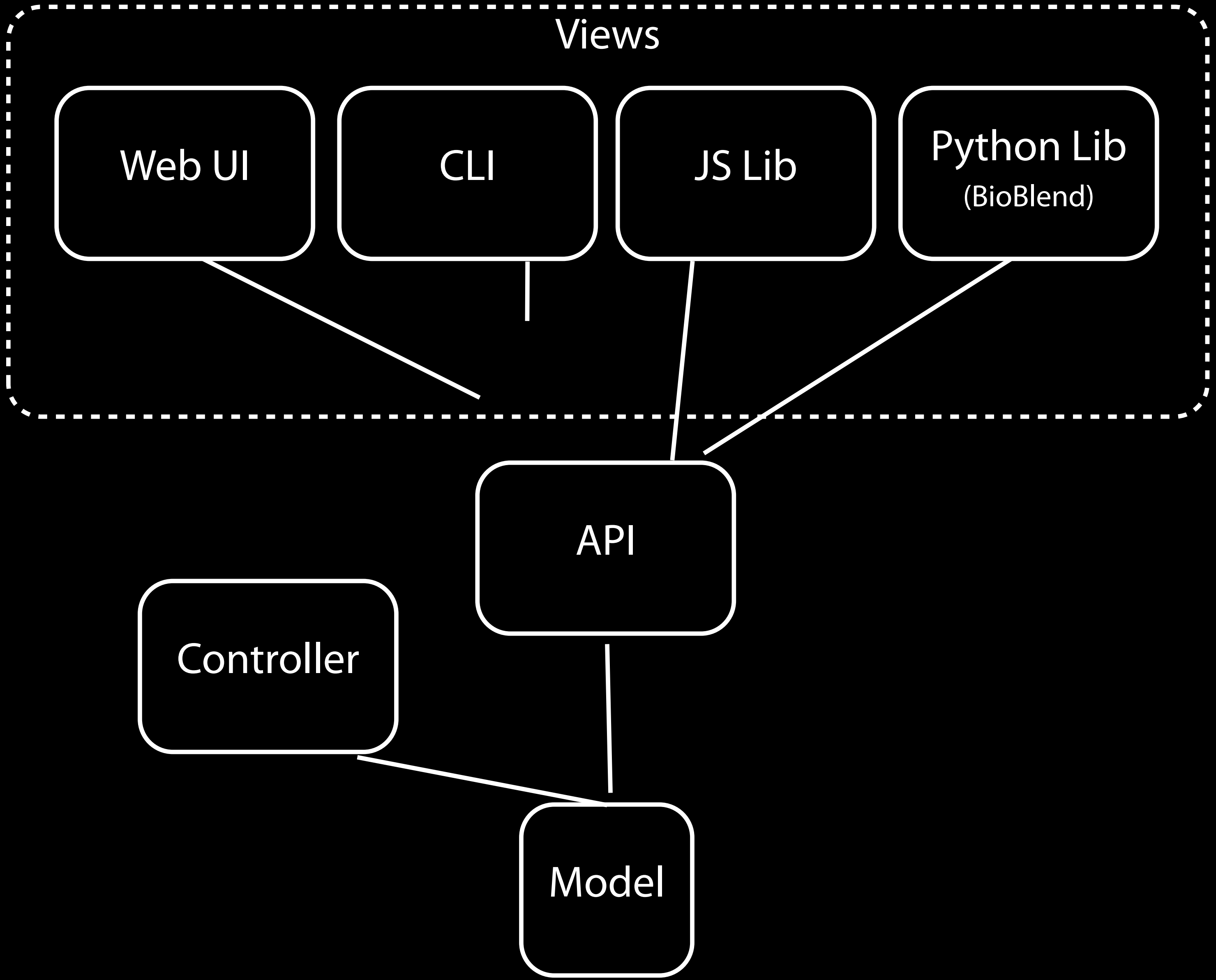


<<thread>>
Job Manager
Job Handlers

## Splitting by process



# UI Architecture



## Code organization and standards

http://wiki.galaxyproject.org/Develop/BestPractices

(let’s look at the code)

These are best practices.

“foolish consistency is the hobgoblin of little minds”

## Standards

Galaxy mostly follows PEP-8, readability is the ultimate goal

Avoid “from module import *”

Comment lines should be under 79 characters, code lines can be up to 200 characters if it improves readability

Docstrings need to be reStructured Text (RST) and Sphinx markup compatible, documentation is automatically generated

Whitespace: whatever is most readable, both for blank lines and space around operators

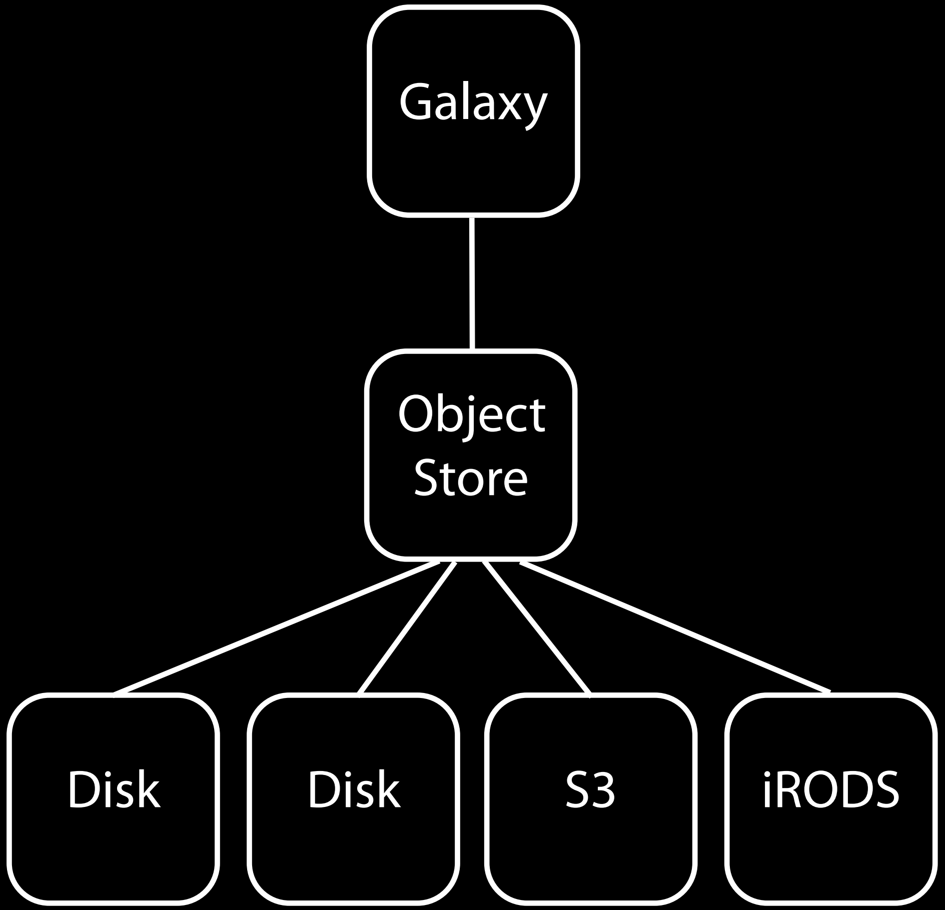
What about Javascript?

## Galaxy data storage

Metadata stored in relational database
(PostgreSQL Please!)

Dataset files stored ~~on filesystem~~
in objectstore

# Data Abstraction



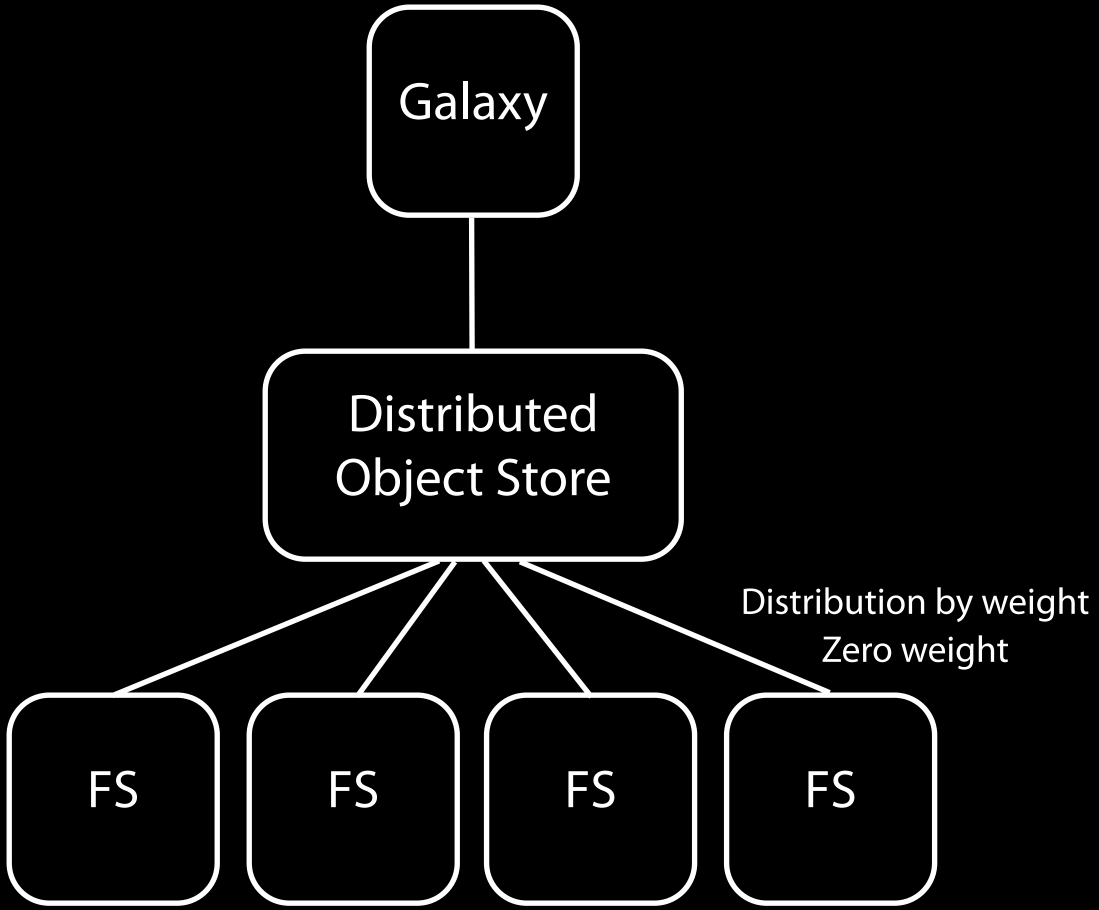
# Data Abstraction

```
~~>>> fh = open( dataset.file_path, 'w' )~~
~~>>> fh.write( 'foo' )~~
~~>>> fh.close()~~
~~>>> fh = open( dataset.file_path, 'r' )~~
~~>>> fh.read()~~
```

```
>>> update_from_file( dataset, file_name='foo.txt' )
>>> get_data( dataset )
>>> get_data( dataset, start=42, count=4096 )
```

# Data Abstraction

## Distributed Object Store



# Data Abstraction

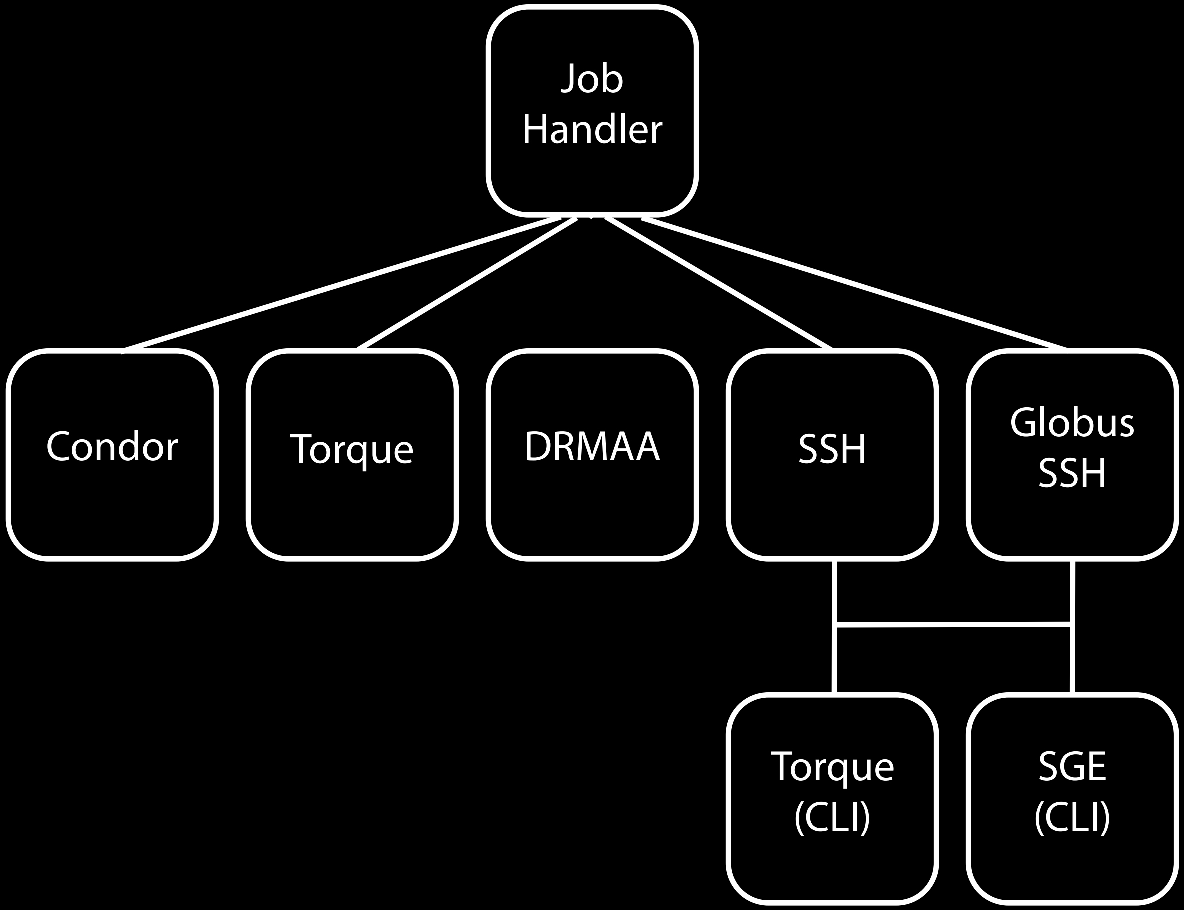
## Benefits

- Grow beyond original capacity
- Avoid migrating data offline
- Tier storage
- Let your users bring their own storage
- Use resources w/o a shared filesystem (with iRODS)
- Remove IO bottlenecks

## Job scheduling and running

# Job Control

job_conf.xml



## Extension: adding a new job runner

Running as actual user

User and group specific storage (?)

## Galaxy workflow system

Workflow representation: data flow graph

Scheduling like any other job

New: fix and rerun from failure point (workflow automatically paused)

Future: background scheduling, decision points during scheduling

## Galaxy API

# Galaxy API

## Technologies

- Representational State Transfer (REST)
  - Sessionless operations via HTTP
- JavaScript Object Notation (JSON)
- Uses a key (rather than user/password)

# Galaxy API

Core Interfaces

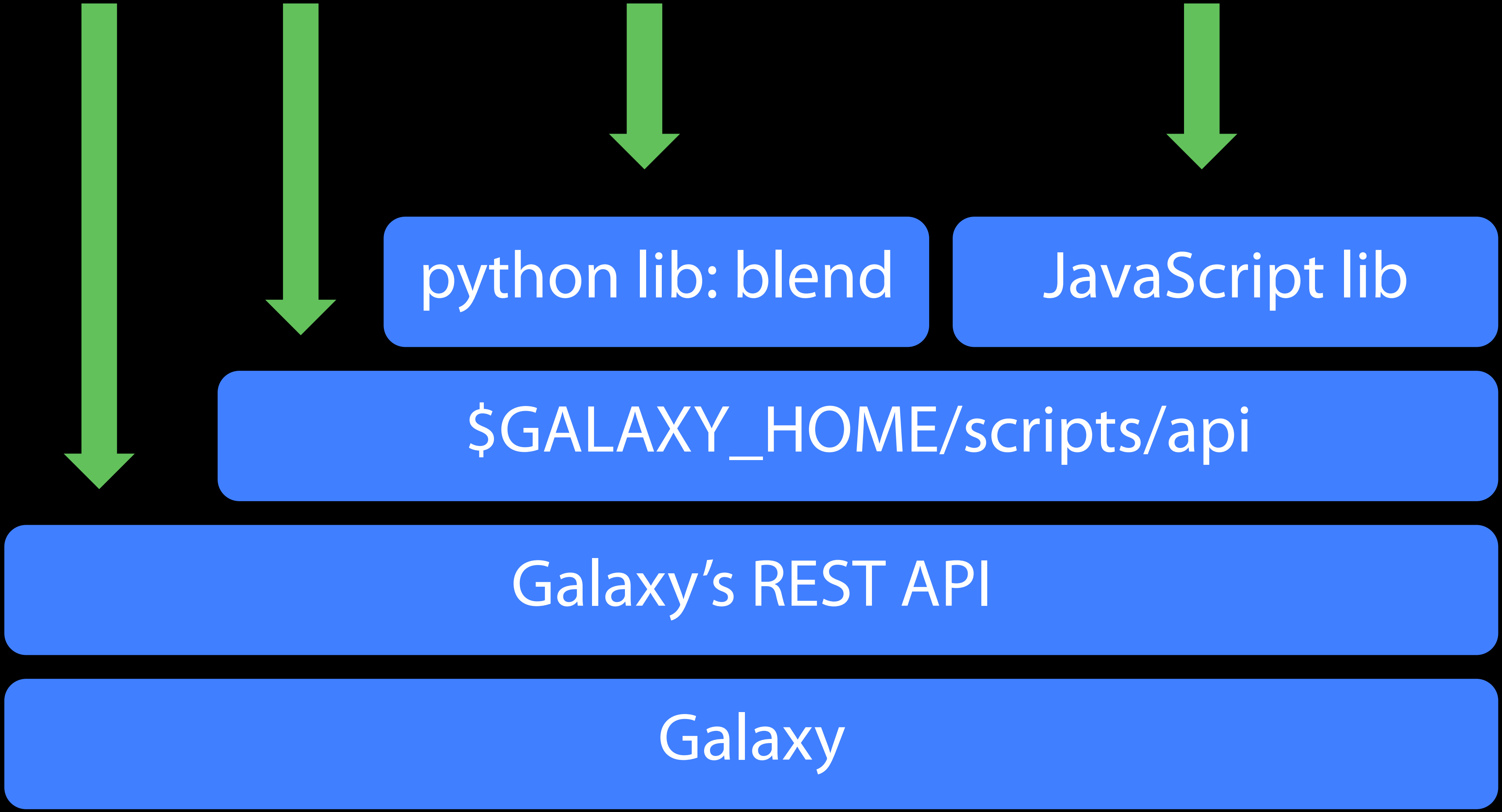
- Library permissions
- Forms
- Server configuration (view)
- Sample tracking requests and samples
- Manage users, roles, and quotas
- Execute tools and workflows

# Galaxy API

Core Interfaces

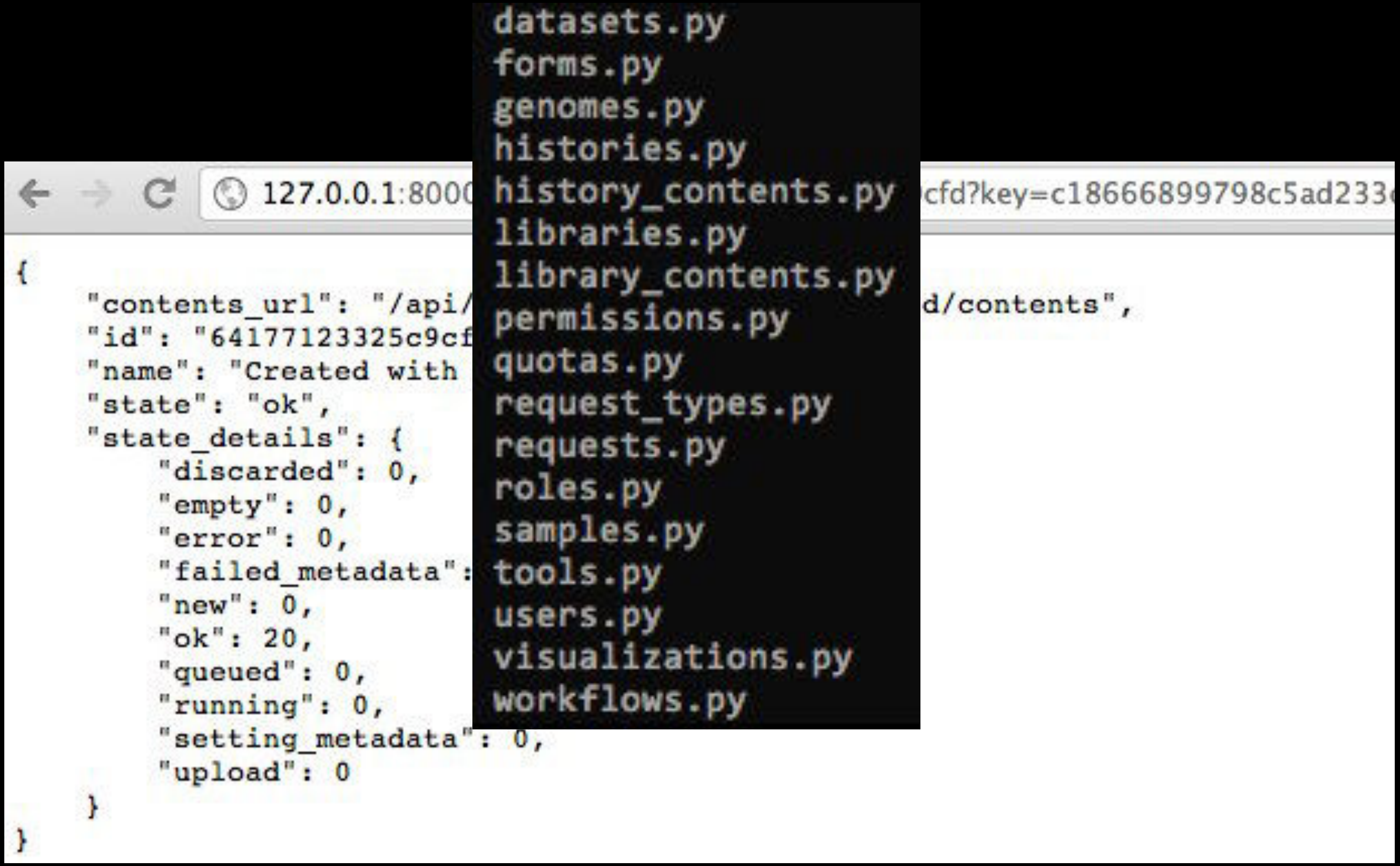
- Histories and History Datasets
- Libraries and Library Datasets
- Tools
- Workflows

# API Interactions with Galaxy



# Bare Bones Galaxy API

- create (POST)
- display (GET)
- update (PUT)
- delete (DELETE)



127.0.0.1:8000 cfd?key=c18666899798c5ad233

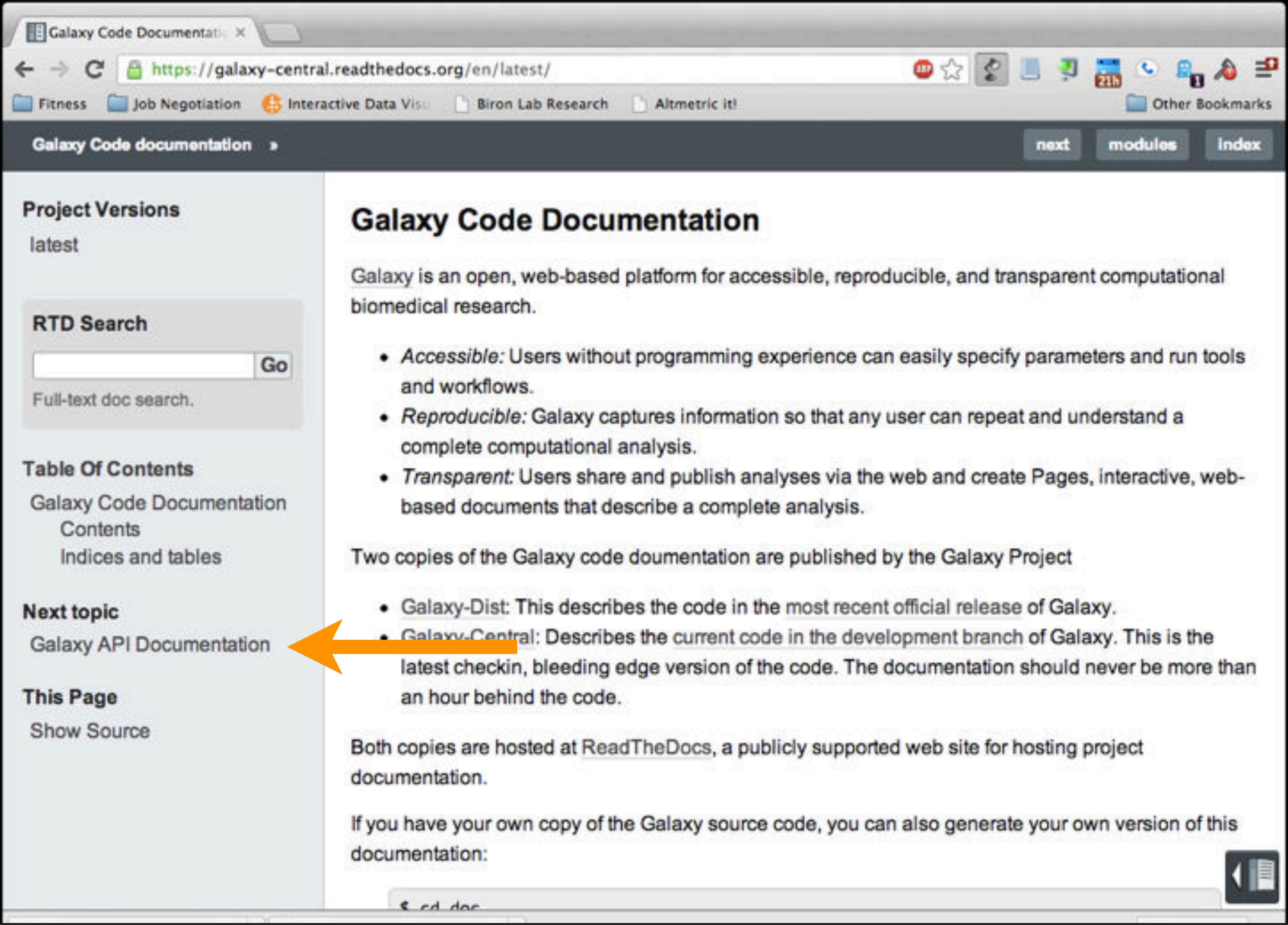
```
{
    "contents_url": "/api/                    d/contents",
    "id": "64177123325c9cf
    "name": "Created with
    "state": "ok",
    "state_details": {
        "discarded": 0,
        "empty": 0,
        "error": 0,
        "failed_metadata":
        "new": 0,
        "ok": 20,
        "queued": 0,
        "running": 0,
        "setting_metadata": 0,
        "upload": 0
    }
}
```

```
datasets.py
forms.py
genomes.py
histories.py
history_contents.py
libraries.py
library_contents.py
permissions.py
quotas.py
request_types.py
requests.py
roles.py
samples.py
tools.py
users.py
visualizations.py
workflows.py
```

# Making the Calls

Wrapper methods exist (in /scripts/api/) to make the API calls easier:

```
./{action}.py <api key> http://<ip>/api/{module}/[unit] [args]

action: create | display | update | delete

api_key: obtained from the UI

module: datasets | forms | histories | libraries | permissions |
quotas | requests | roles | samples | tools | users |
visualizations | workflows

unit: dataset_id / history_id / library_id / …

args: name / key-value pair / …
```


Galaxy Code Documentation
https://galaxy-central.readthedocs.org/en/latest/
Fitness | Job Negotiation | Interactive Data Visu | Biron Lab Research | Altmetric it! | Other Bookmarks

Galaxy Code documentation » | next | modules | index

**Project Versions**
latest

**RTD Search**
Go
Full-text doc search.

**Table Of Contents**

**Next topic**
Galaxy API Documentation

**This Page**
Show Source

## Galaxy Code Documentation

Galaxy is an open, web-based platform for accessible, reproducible, and transparent computational biomedical research.

- *Accessible:* Users without programming experience can easily specify parameters and run tools and workflows.
- *Reproducible:* Galaxy captures information so that any user can repeat and understand a complete computational analysis.
- *Transparent:* Users share and publish analyses via the web and create Pages, interactive, web-based documents that describe a complete analysis.

Two copies of the Galaxy code doumentation are published by the Galaxy Project

- Galaxy-Dist: This describes the code in the most recent official release of Galaxy.
- Galaxy-Central: Describes the current code in the development branch of Galaxy. This is the latest checkin, bleeding edge version of the code. The documentation should never be more than an hour behind the code.

Both copies are hosted at ReadTheDocs, a publicly supported web site for hosting project documentation.

If you have your own copy of the Galaxy source code, you can also generate your own version of this documentation:

```
$ cd doc
```

http://galaxy-central.readthedocs.org

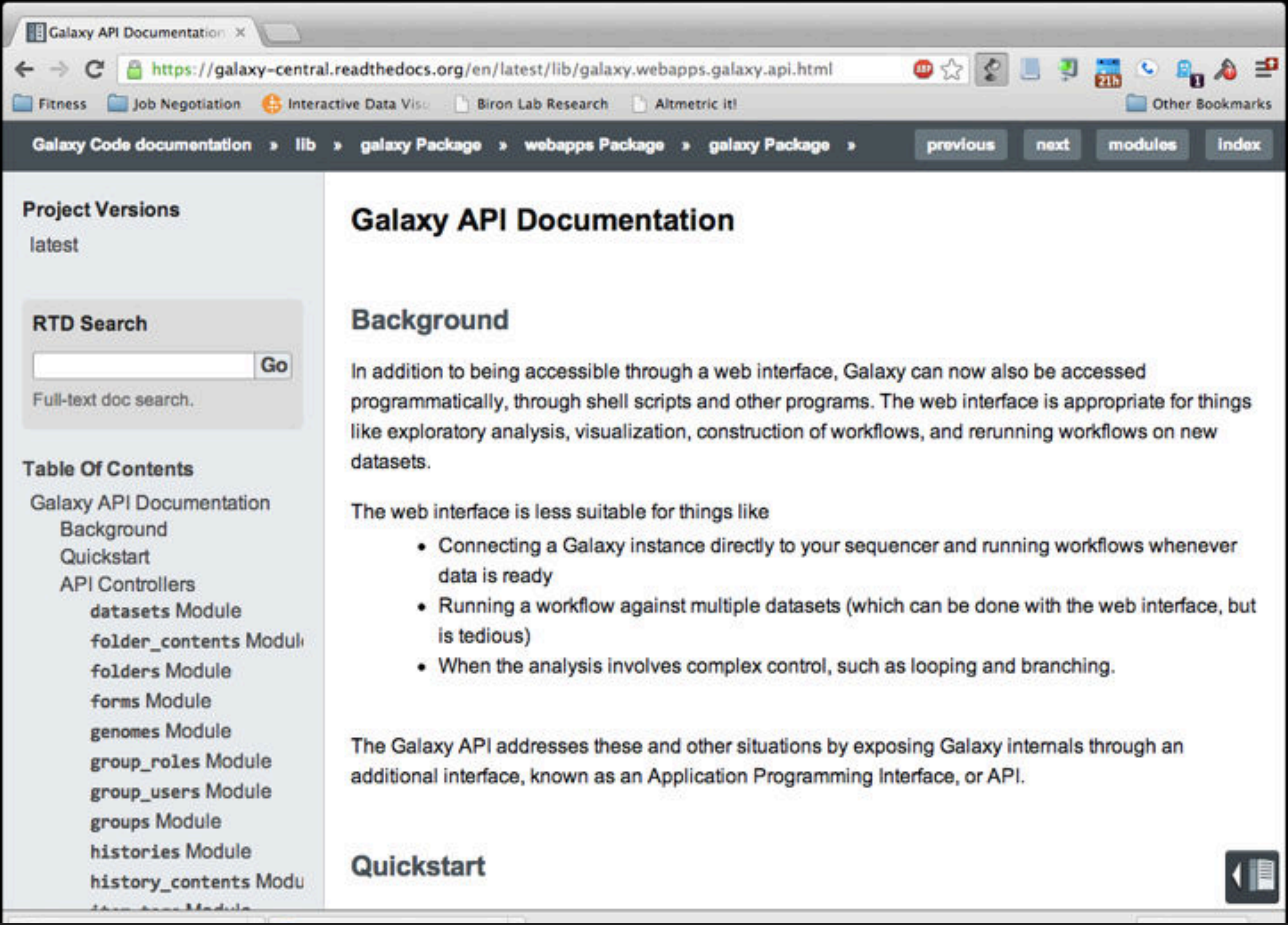


Table Of Contents

## Galaxy API Documentation

### Background

In addition to being accessible through a web interface, Galaxy can now also be accessed programmatically, through shell scripts and other programs. The web interface is appropriate for things like exploratory analysis, visualization, construction of workflows, and rerunning workflows on new datasets.

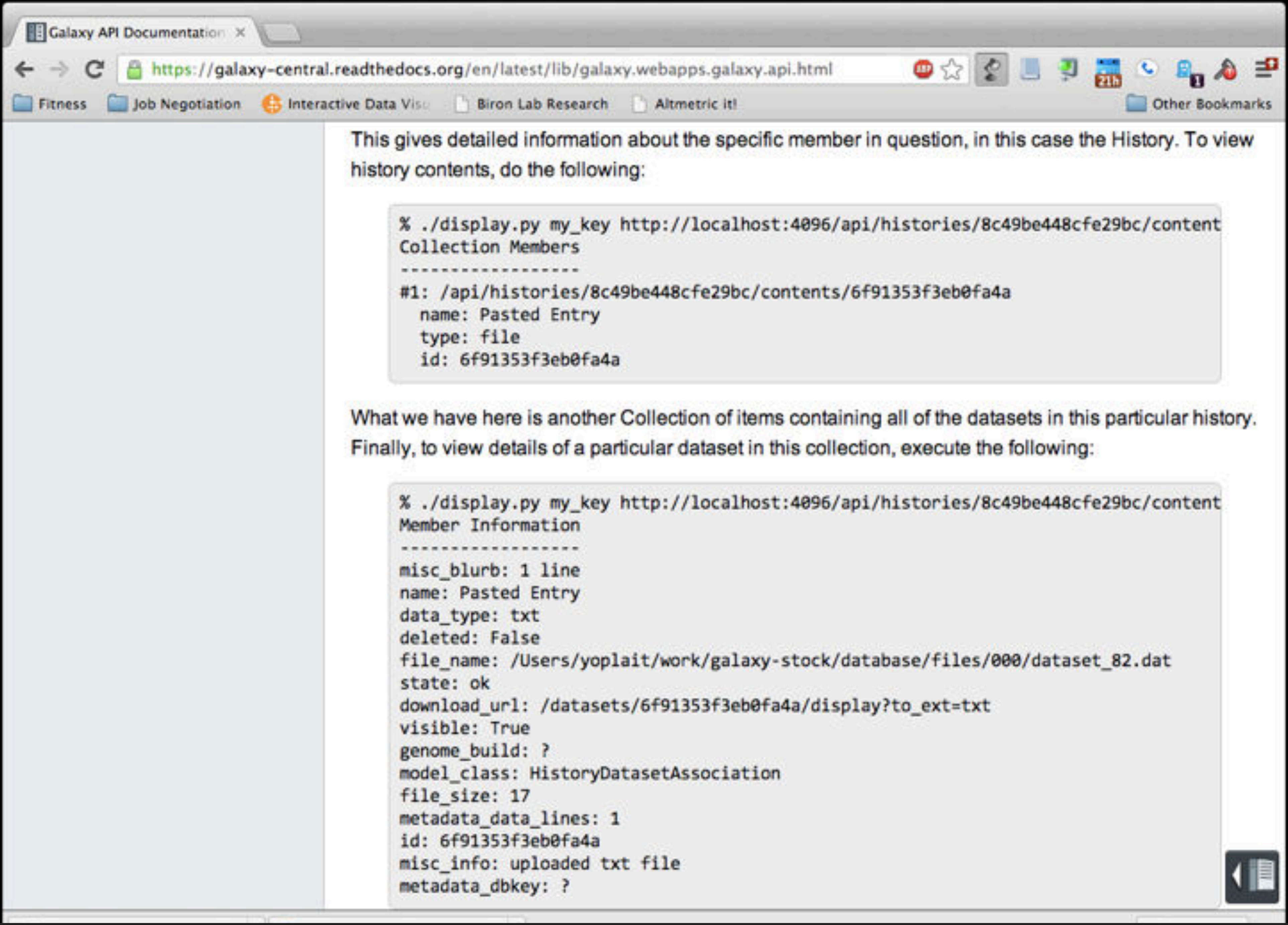
The web interface is less suitable for things like

- Connecting a Galaxy instance directly to your sequencer and running workflows whenever data is ready
- Running a workflow against multiple datasets (which can be done with the web interface, but is tedious)
- When the analysis involves complex control, such as looping and branching.

The Galaxy API addresses these and other situations by exposing Galaxy internals through an additional interface, known as an Application Programming Interface, or API.

### Quickstart

http://galaxy-central.readthedocs.org



This gives detailed information about the specific member in question, in this case the History. To view history contents, do the following:

```
% ./display.py my_key http://localhost:4096/api/histories/8c49be448cfe29bc/content
Collection Members
------------------
#1: /api/histories/8c49be448cfe29bc/contents/6f91353f3eb0fa4a
  name: Pasted Entry
  type: file
  id: 6f91353f3eb0fa4a
```

What we have here is another Collection of items containing all of the datasets in this particular history. Finally, to view details of a particular dataset in this collection, execute the following:

```
% ./display.py my_key http://localhost:4096/api/histories/8c49be448cfe29bc/content
Member Information
------------------
misc_blurb: 1 line
name: Pasted Entry
data_type: txt
deleted: False
file_name: /Users/yoplait/work/galaxy-stock/database/files/000/dataset_82.dat
state: ok
download_url: /datasets/6f91353f3eb0fa4a/display?to_ext=txt
visible: True
genome_build: ?
model_class: HistoryDatasetAssociation
file_size: 17
metadata_data_lines: 1
id: 6f91353f3eb0fa4a
misc_info: uploaded txt file
metadata_dbkey: ?
```

http://galaxy-central.readthedocs.org

## Extending Galaxy

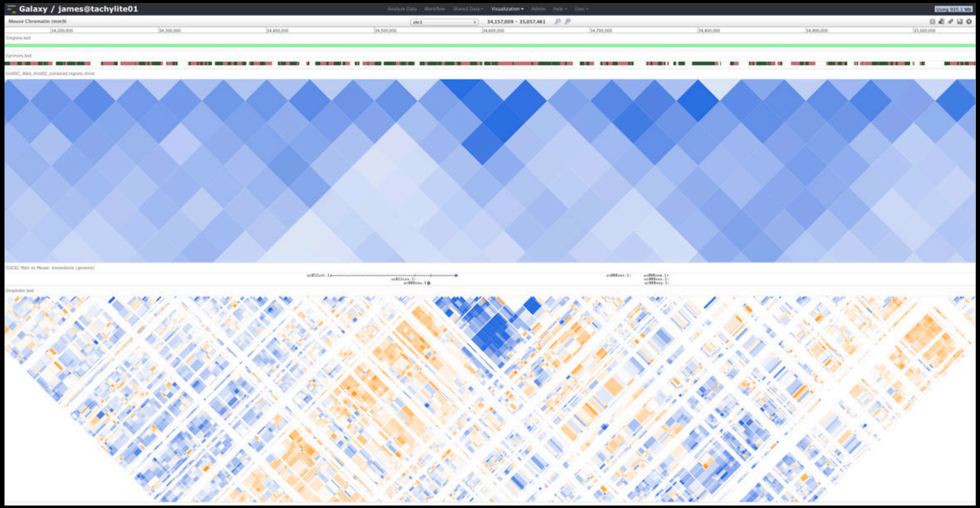
Tools & Datatypes

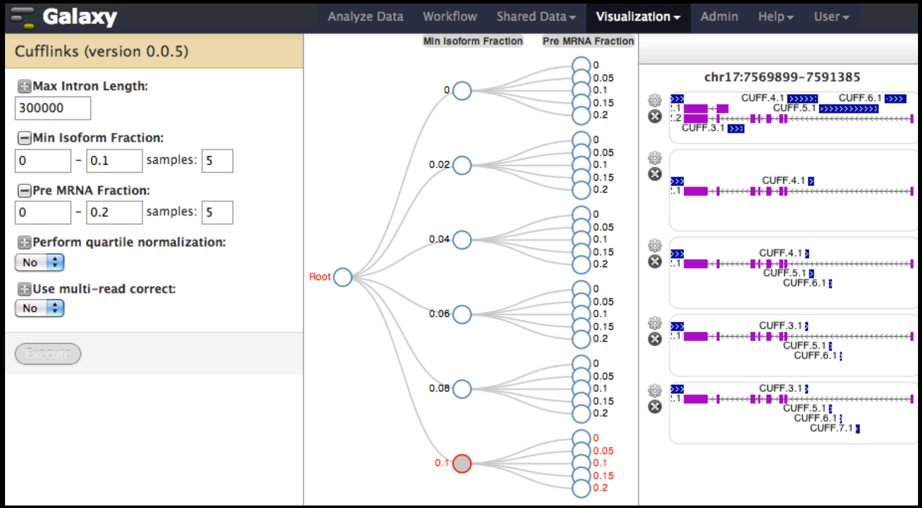
Hopefully everyone understands this by now?

There is another session,
“Introduction to Tool and Data Source Configuration”

## Visualizations

(See talk from Carl Eberhard)





Cufflinks (version 0.0.5)

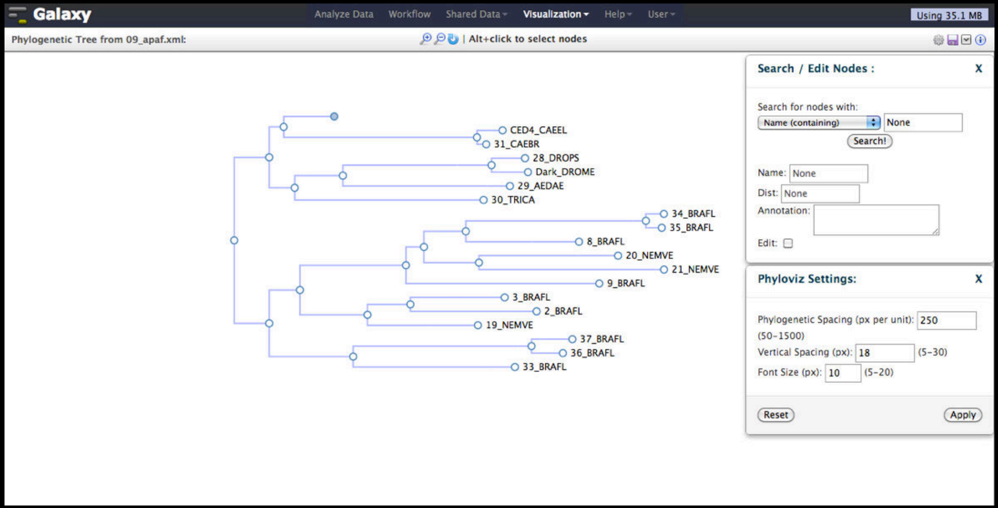
Max Intron Length: 300000

Min Isoform Fraction: 0 - 0.1 samples: 5

Pre MRNA Fraction: 0 - 0.2 samples: 5

Perform quartile normalization: No

Use multi-read correct: No



Phylogenetic Tree from 09_apaf.xml:

Search / Edit Nodes :

Search for nodes with: Name (containing) None

Name: None

Dist: None

Annotation:

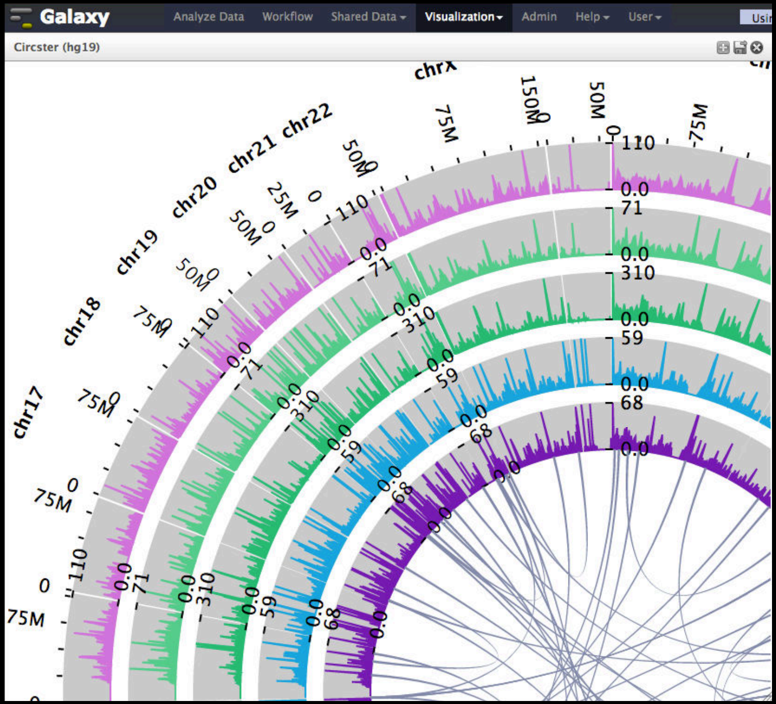
Edit: ☐

Phyloviz Settings:

Phylogenetic Spacing (px per unit): 250 (50-1500)

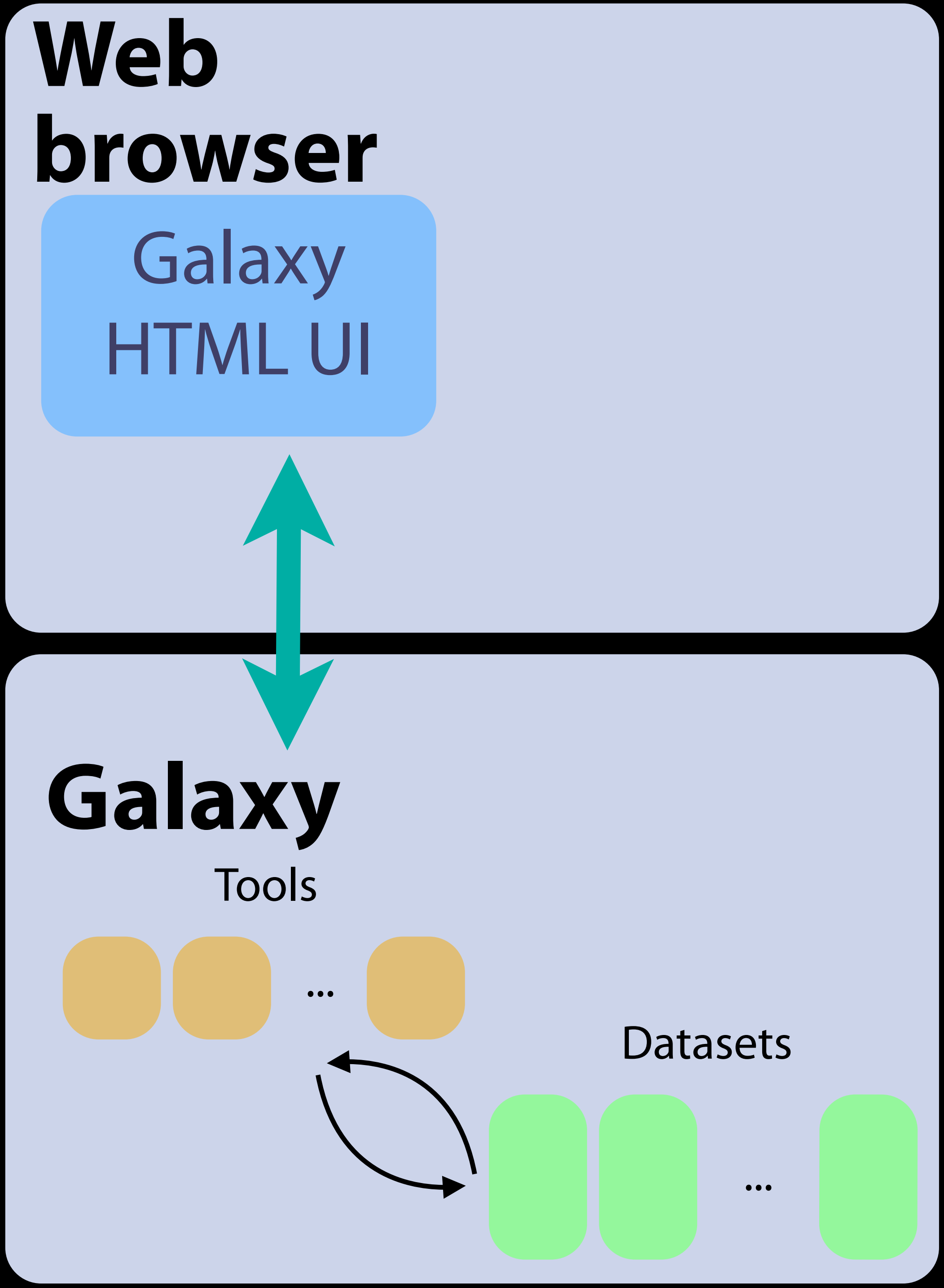
Vertical Spacing (px): 18 (5-30)

Font Size (px): 10 (5-20)

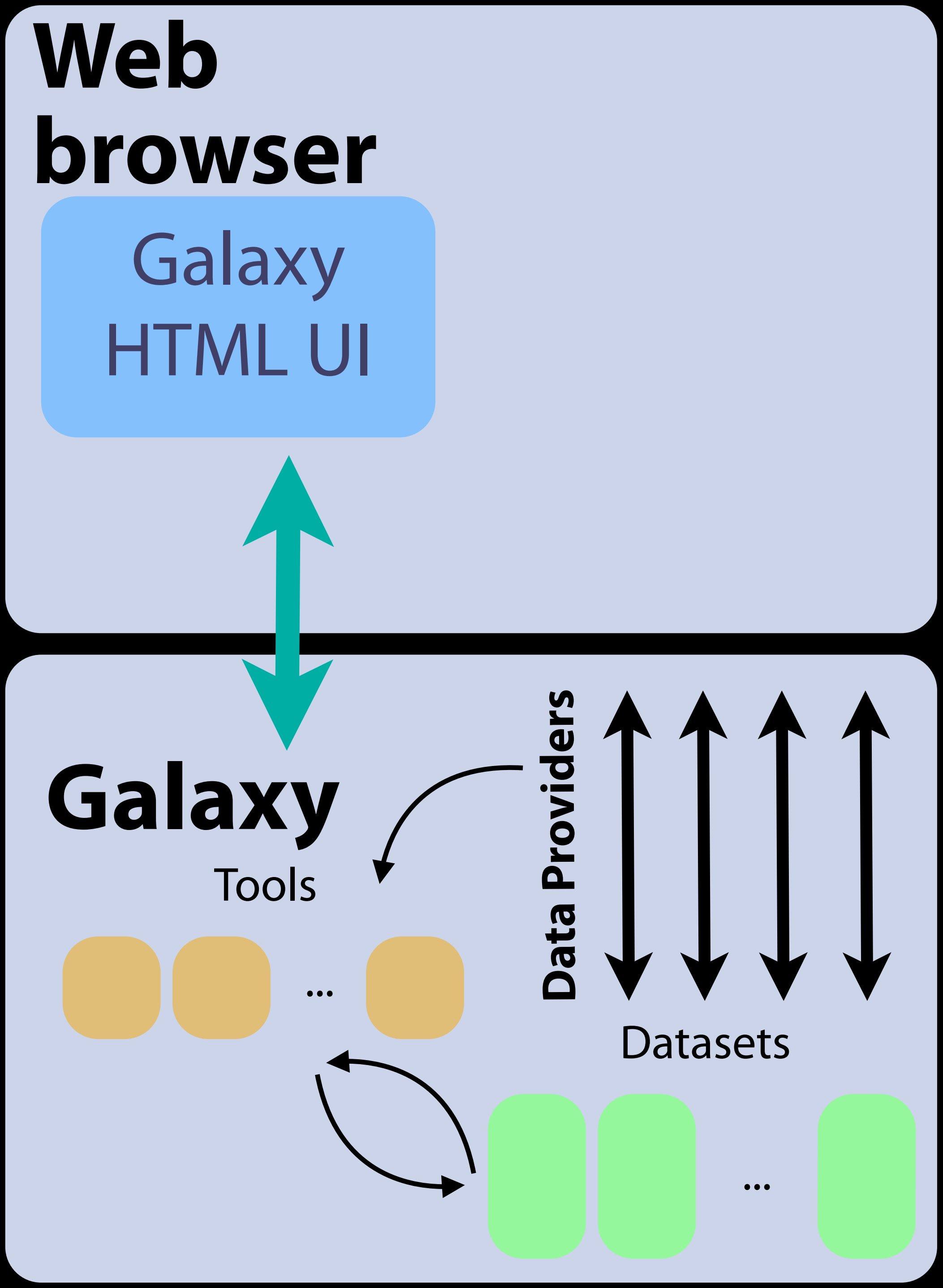


# Visualization and visual analytics

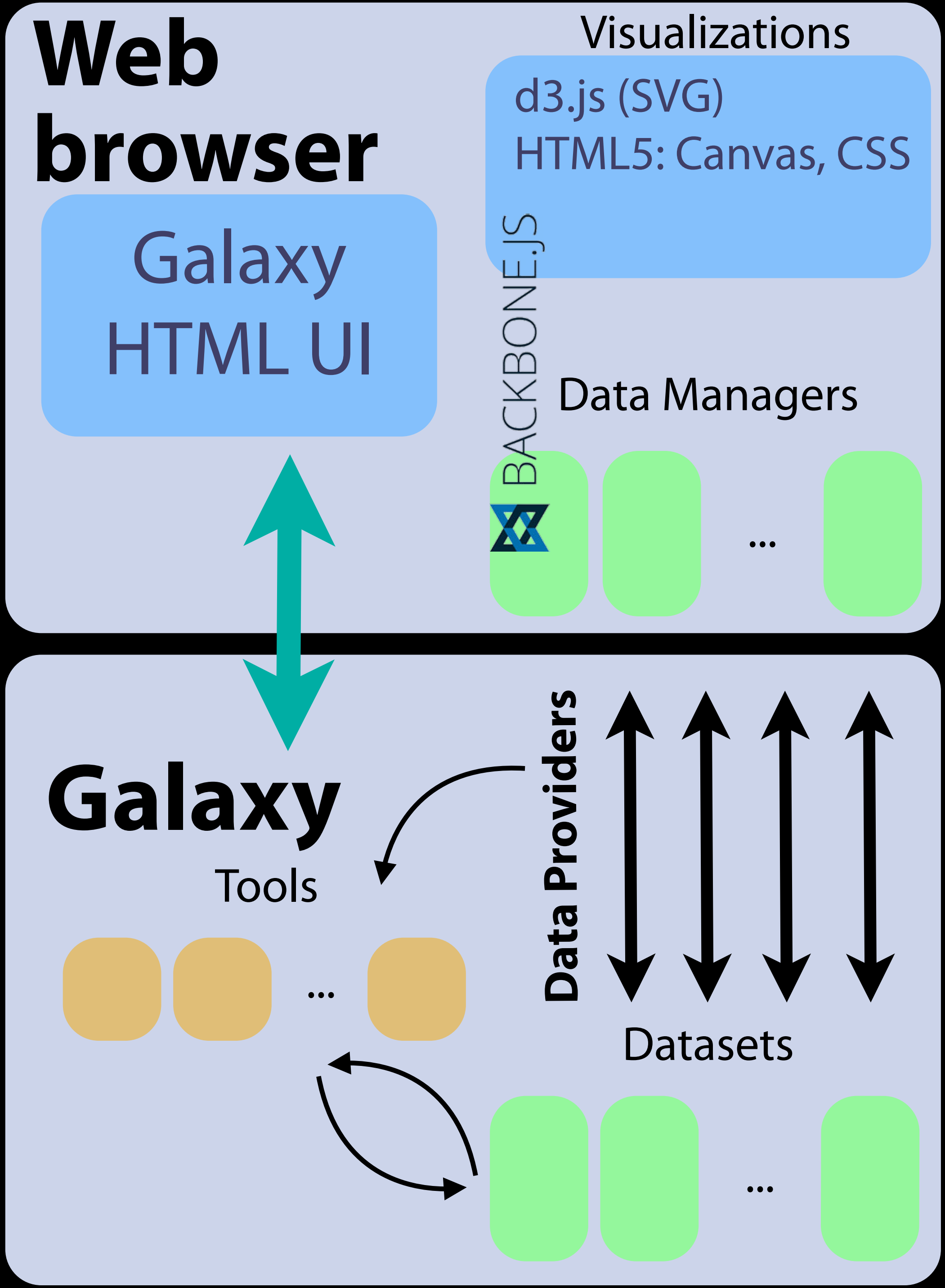
# Architecture



## Architecture



# Architecture



# Technical Highlights

Rough pluggable support for generic JavaScript visualizations + data providers

Data providers = fast, random access to data in Python, JS

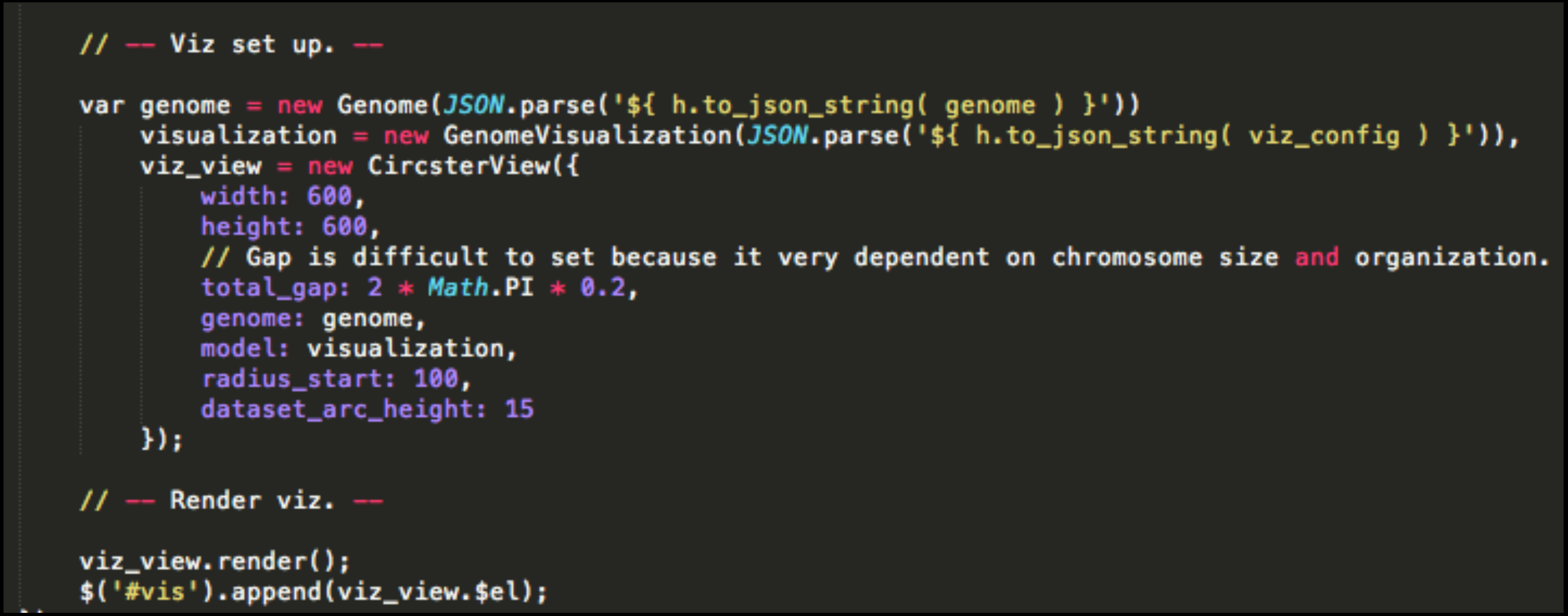
API: run tools, run them on data subsets

Backbone + HTML5 objects for Web-based genomic visualizations

- e.g. data managers, linear and circular views

JS binding to Galaxy API (blendJS?)

- visualizations, tools, datasets
- custom Galaxy UIs



```
// -- Viz set up. --

var genome = new Genome(JSON.parse('${ h.to_json_string( genome ) }'))
    visualization = new GenomeVisualization(JSON.parse('${ h.to_json_string( viz_config ) }')),
    viz_view = new CircsterView({
        width: 600,
        height: 600,
        // Gap is difficult to set because it very dependent on chromosome size and organization.
        total_gap: 2 * Math.PI * 0.2,
        genome: genome,
        model: visualization,
        radius_start: 100,
        dataset_arc_height: 15
    });

// -- Render viz. --

viz_view.render();
$('#vis').append(viz_view.$el);
```

## Eggs and dependency packaging

## New controllers and webapps

Q&A