The Galaxy Toolshed

usegalaxy.org/toolshed

The Galaxy API

API Foundation

- Technologies
  - Representational State Transfer (REST)
    - Sessionless operations via HTTP
  - JavaScript Object Notation (JSON)

GO

Other API Interfaces

- Library permissions
- Forms
- Sample tracking requests and samples
- Manage users, roles, and quotas
- Execute tools and workflows

End-to-end pipelines

- Automatically upload data retrieved from instruments
- Start a workflow
- Move outputs to a data library

Beneath the Clouds

Building your own Galaxy production service

usegalaxy.org/production

Galaxy runs out of the box!

- Simple download, setup, and install design:

```
% hg clone ...
% sh run.sh
```

- Great for development!
- Not designed to support multiple users in a production environment with default configuration

Development-oriented defaults

- SQLite database
- One process
- Built-in HTTP server
- Local job execution

Start Fresh

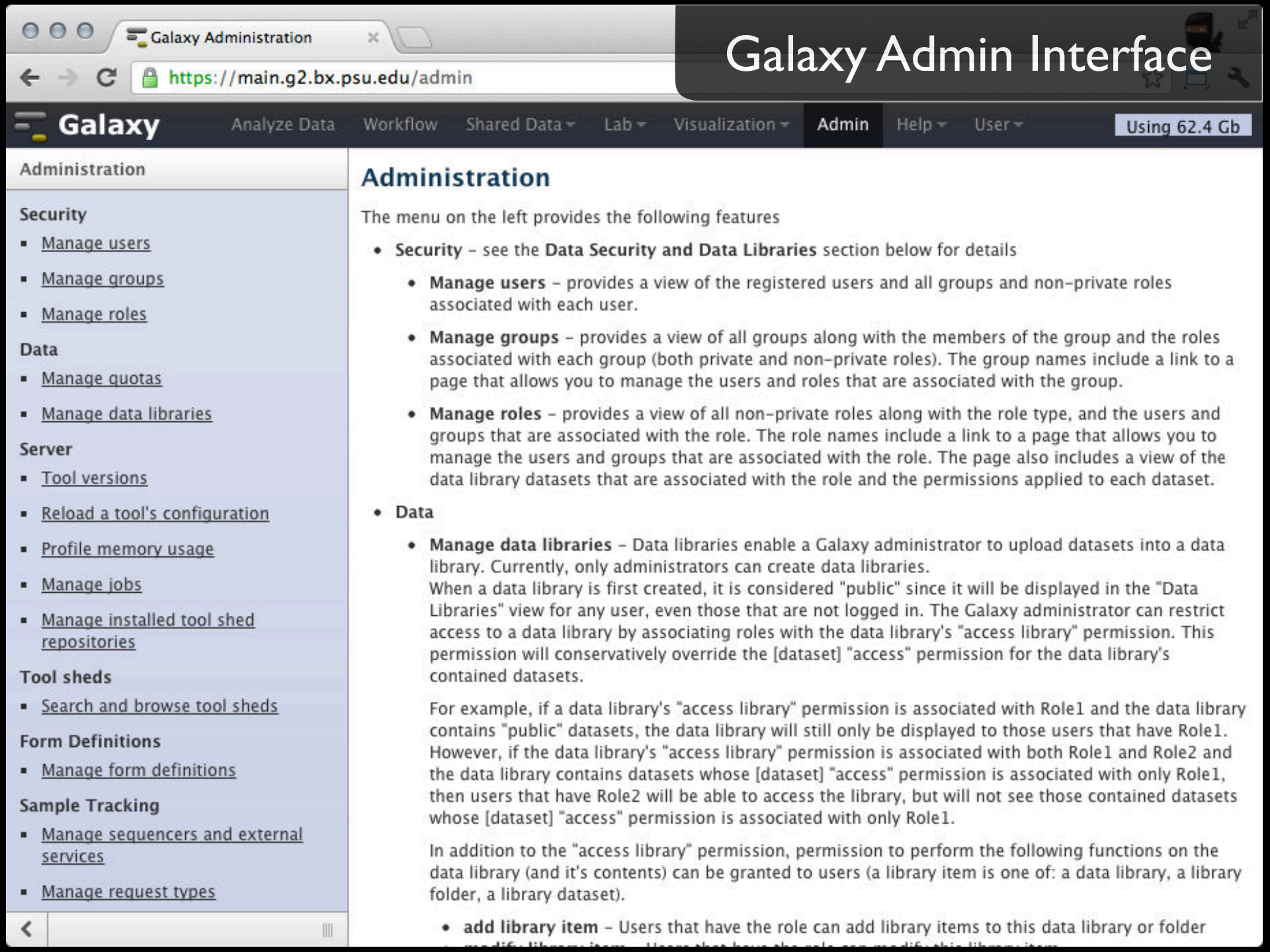
- Don’t use an old Galaxy installation - check out a new copy
- Use a dedicated non-root user
- Start and stop with your OS’ system service method (e.g. init.d, service)
- Don’t share the database or database user
- Use a dedicated Python or virtualenv
- If you plan to use a cluster, put galaxy in a shared filesystem

Galaxy Config Basics

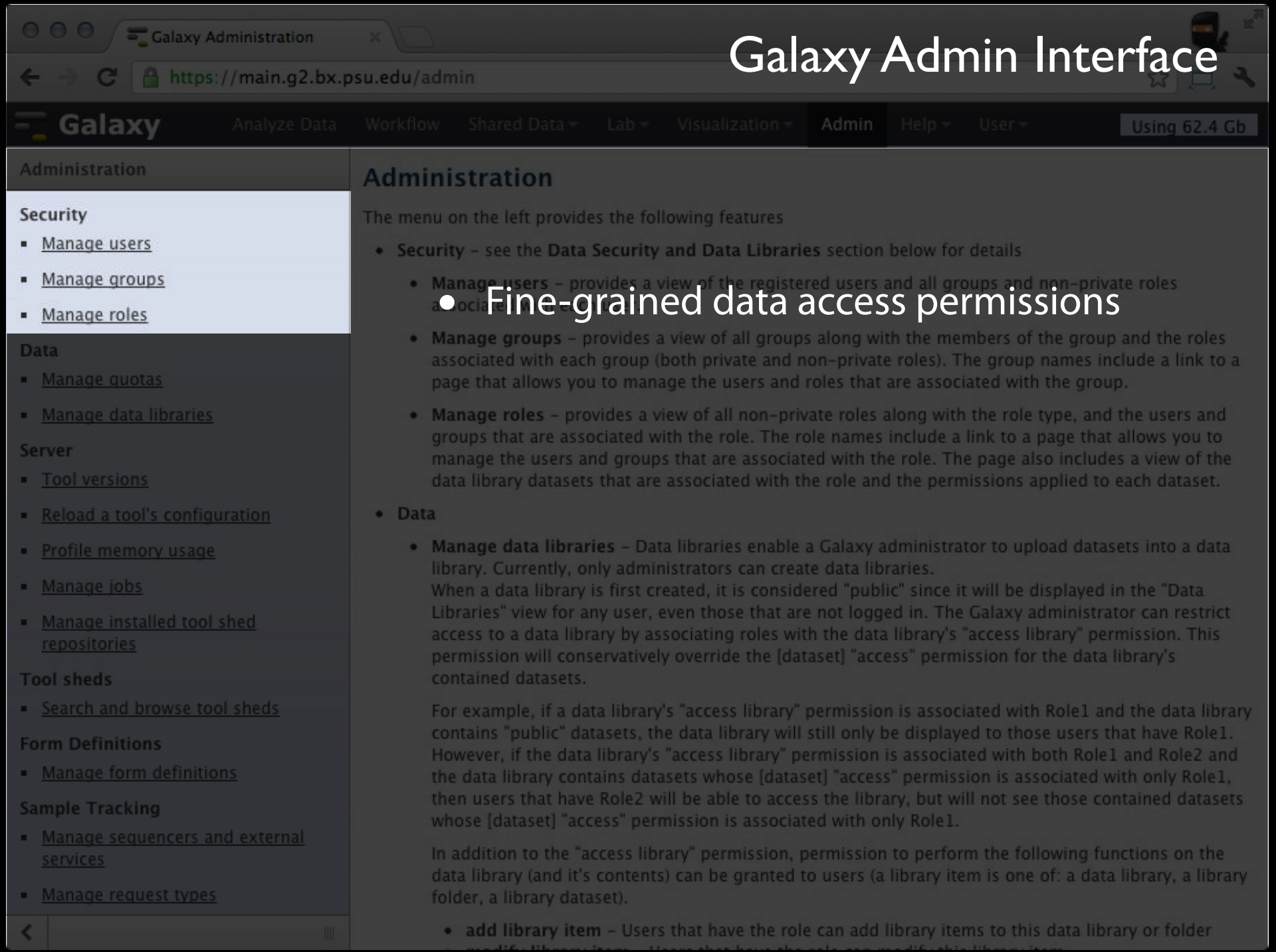
- Use the sample config (copy before starting):
  - `% cp universe_wsgi.ini.sample universe_wsgi.ini`
  - Read the full sample config
- Set:
- `use_interactive = False` - Not even safe (exposes config)
- `debug = False` - You'll still be able to see tracebacks in the log file, doesn't load response in memory

```
# ---- Galaxy ----------------------------------------------------------

# Configuration of the Galaxy application.

[app:main]

# By default, Galaxy uses a SQLite database at 'database/universe.sqlite'.  You
# may use a SQLAlchemy connection string to specify an external database
# instead.  This string takes many options which are explained in detail in the
# config file documentation.
#database_connection = sqlite:///./database/universe.sqlite
database_connection = postgres:///galaxy

# -- Data Libraries

# These library upload options are described in much more detail in the wiki:
# http://wiki.g2.bx.psu.edu/Admin/Data%20Libraries/Uploading%20Library%20Files

# Add an option to the library upload form which allows administrators to
# upload a directory of files.
#library_import_dir = None
library_import_dir = /Users/nate/import
```

Galaxy Admin Interface

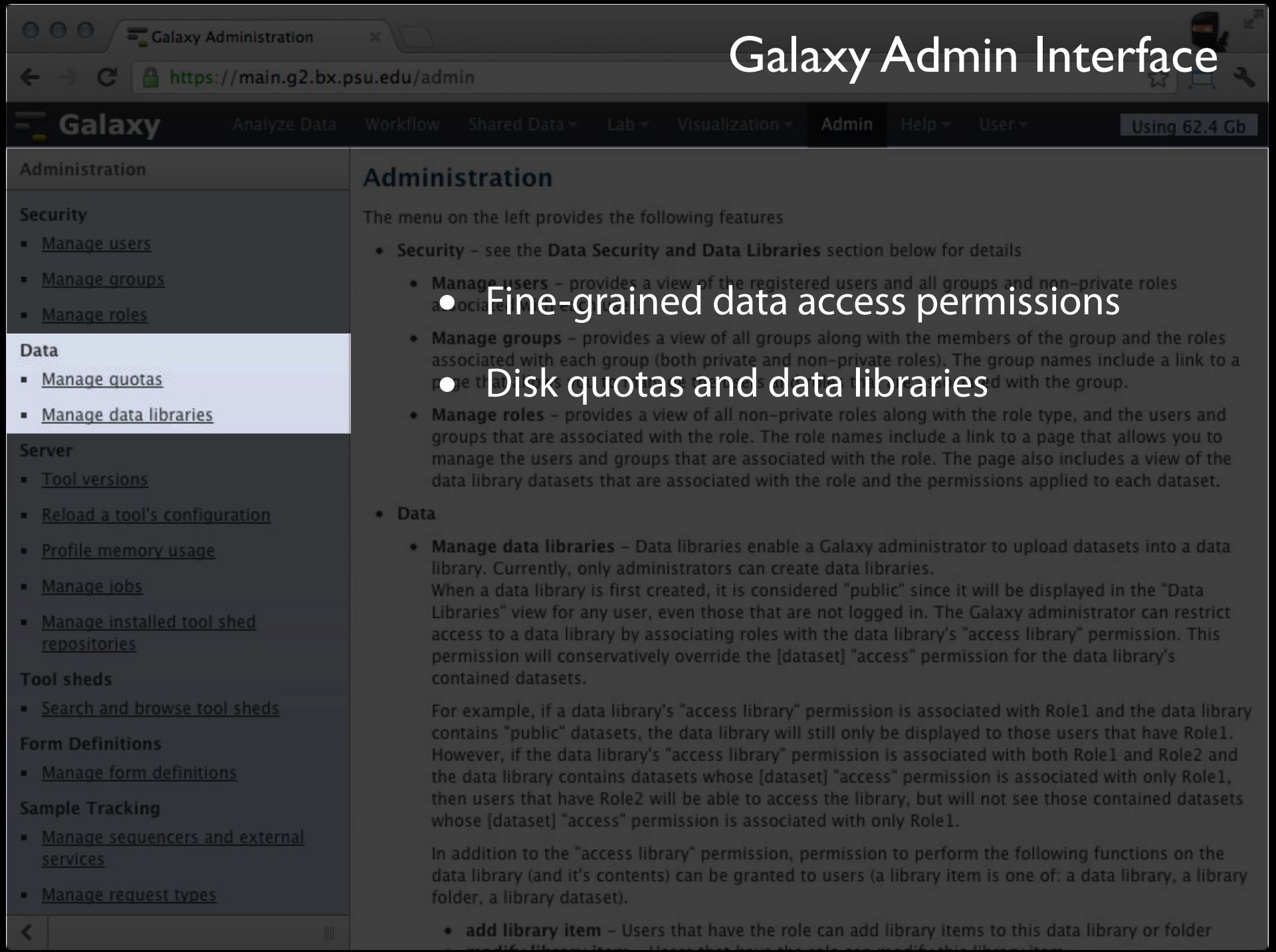


Galaxy Admin Interface



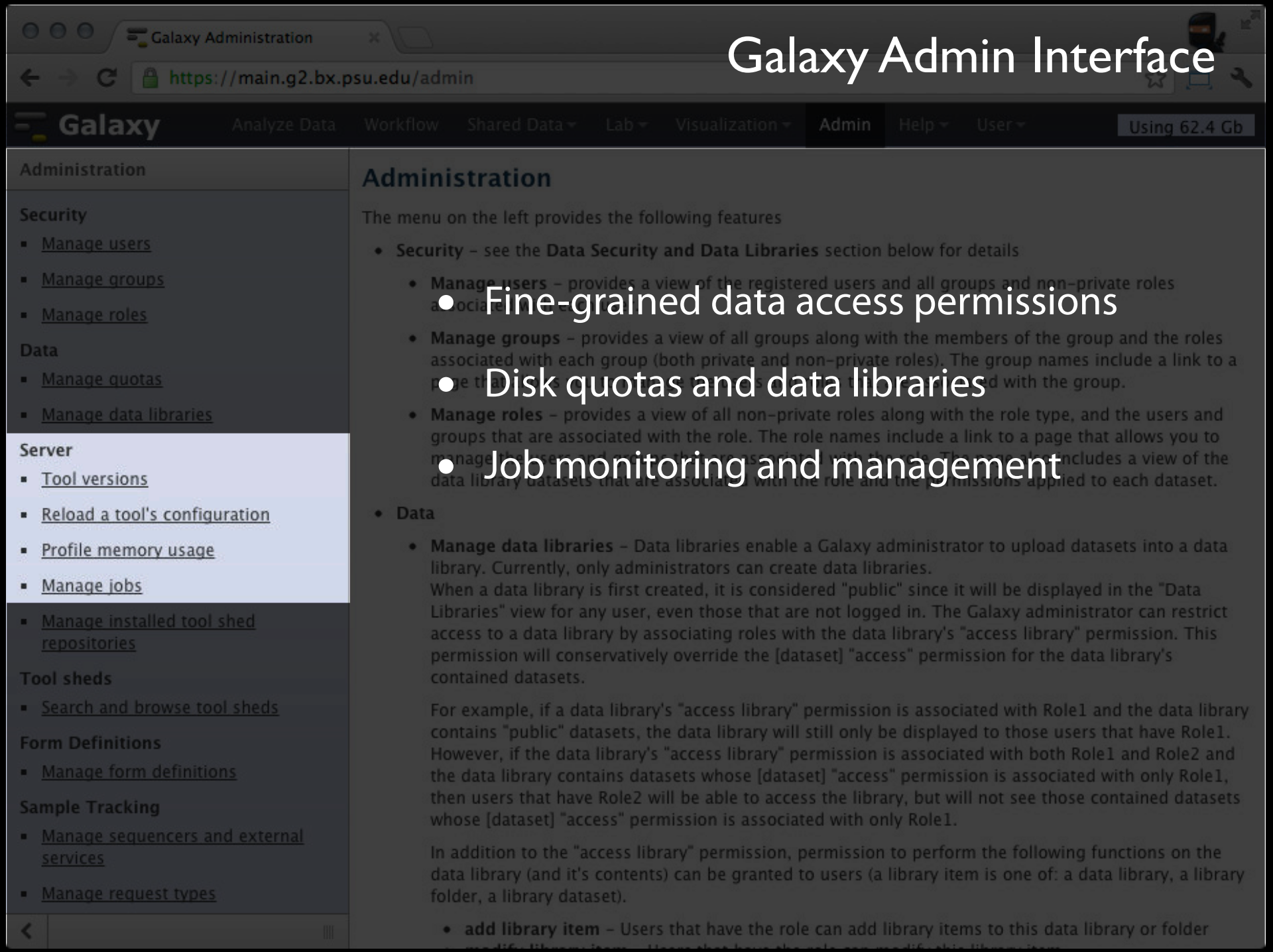
- Fine-grained data access permissions

Galaxy Admin Interface



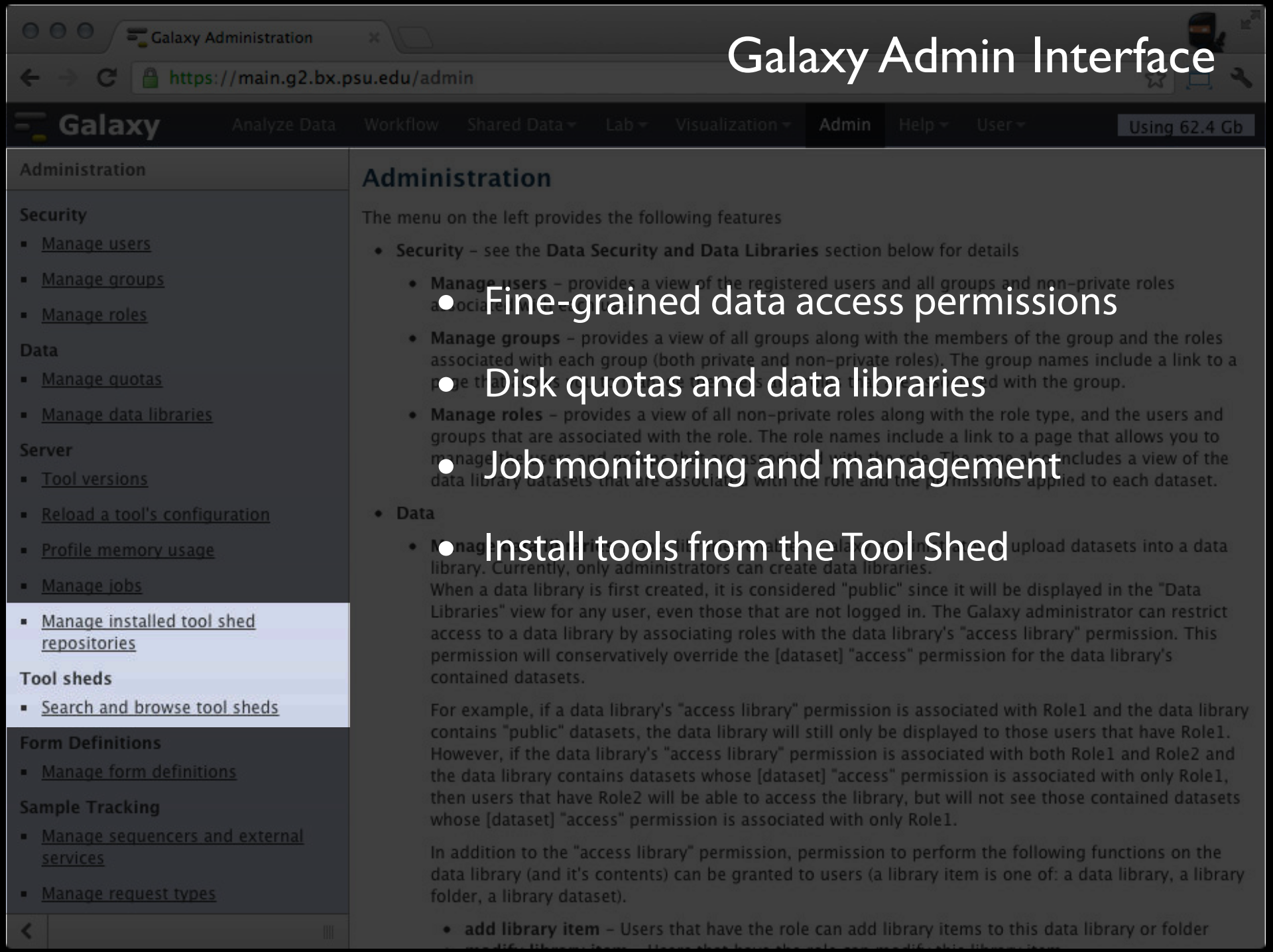
- Fine-grained data access permissions
- Disk quotas and data libraries

Galaxy Admin Interface



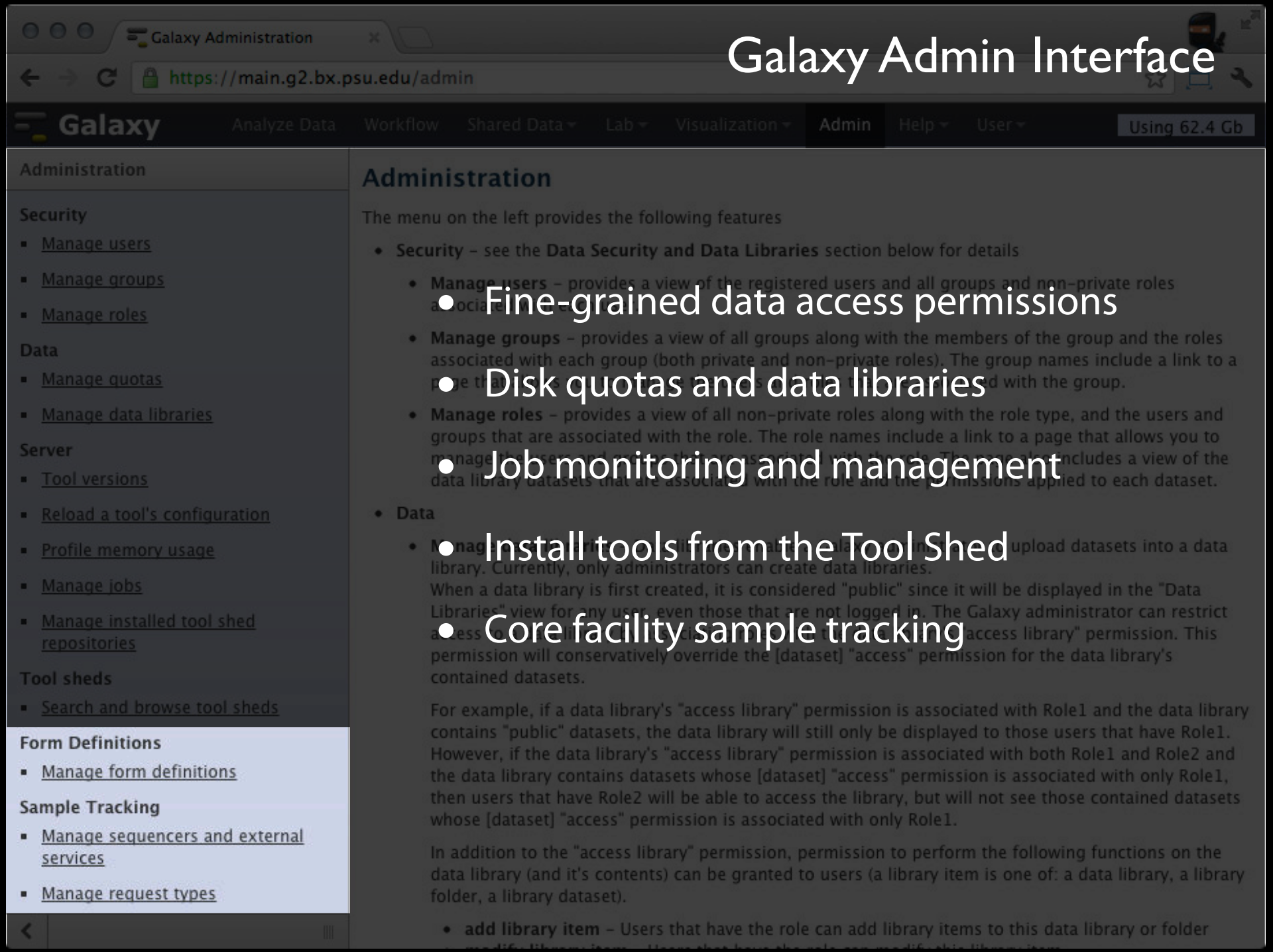
- Fine-grained data access permissions
- Disk quotas and data libraries
- Job monitoring and management

Galaxy Admin Interface



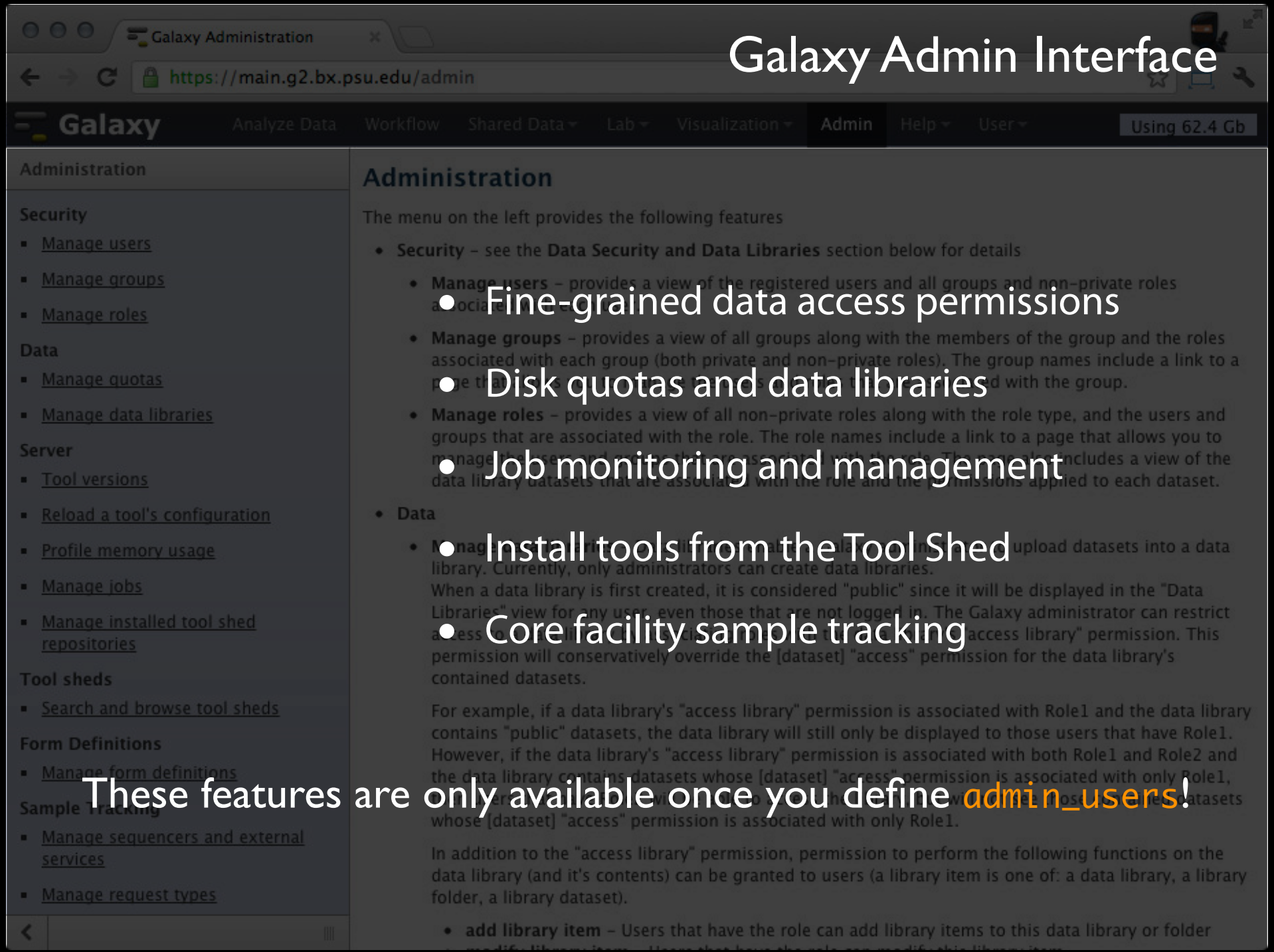
- Fine-grained data access permissions
- Disk quotas and data libraries
- Job monitoring and management
- Install tools from the Tool Shed

Galaxy Admin Interface



- Fine-grained data access permissions
- Disk quotas and data libraries
- Job monitoring and management
- Install tools from the Tool Shed
- Core facility sample tracking

Galaxy Admin Interface



- Fine-grained data access permissions
- Disk quotas and data libraries
- Job monitoring and management
- Install tools from the Tool Shed
- Core facility sample tracking

These features are only available once you define `admin_users`!

A few more options

- Enable browsers like UCSC, GBrowse and Galaxy Trackster
  - `ucsc_display_sites, gbrowse_display_sites, enable_tracks`
- “sudo” for Galaxy: `allow_user_impersonation`
- Publishing features: `enable_pages`
- Disk quotas: `enable_quotas`

Get a real database

- SQLite is serverless
- Galaxy is a heavy database consumer
- Locking will be an immediate issue
- Consumes Galaxy server process resources
- Migrating data is no fun
- Setup is very easy:

```
database_connection = postgres://
```

```
victory# apt-get install postgresql
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  libpq5 postgresql-9.1 postgresql-client-9.1 postgresql-client-common
postgresql-common
Suggested packages:
  oidentd ident-server locales-all postgresql-doc-9.1
The following NEW packages will be installed:
  libpq5 postgresql postgresql-9.1 postgresql-client-9.1 postgresql-client-
common postgresql-common
0 upgraded, 6 newly installed, 0 to remove and 0 not upgraded.
Need to get 0 B/8,951 kB of archives.
After this operation, 22.6 MB of additional disk space will be used.
Do you want to continue [Y/n]? Y
 ... magic happens ...
victory# su - postgres
postgres@victory:~$ createuser -SDR galaxy
postgres@victory:~$ createdb -O galaxy galaxy
postgres@victory:~$
```

Offload the menial tasks: Proxy

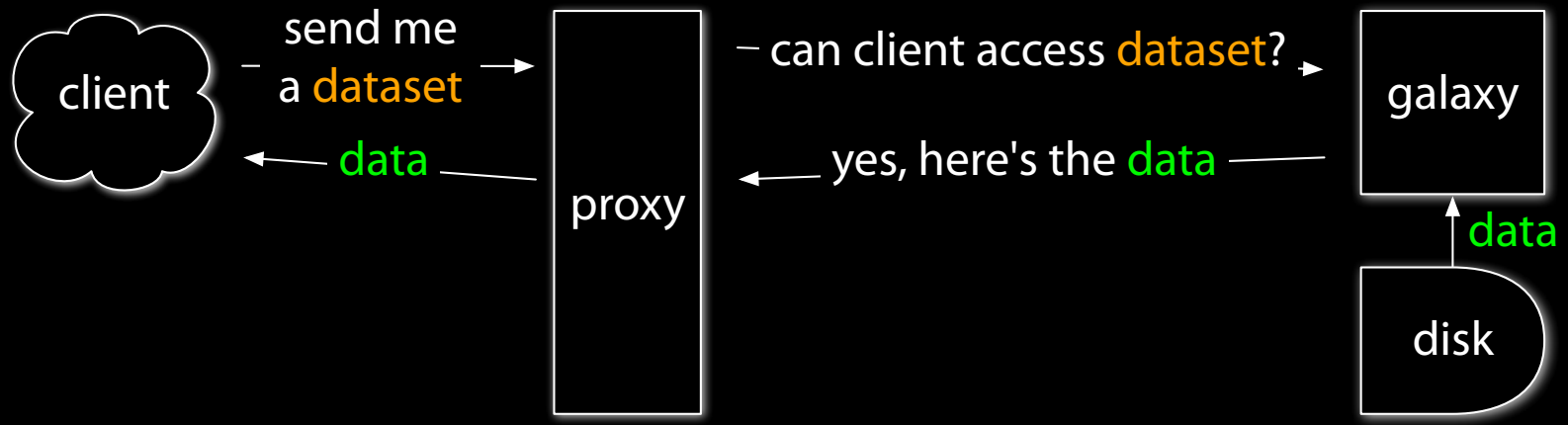
- Directly serve static content faster than Galaxy's HTTP server
- Reduce load on the application
- Caching and compression
- Load balancing (more on that later)
- Hook your local authentication and authorization system

Proxy Options

- Server
  - nginx
    - Designed with proxy as the primary purpose
    - Has an upload module
    - The proxy used for usegalaxy.org and Cloud Galaxy
  - Apache
    - More authentication and other 3rd party modules

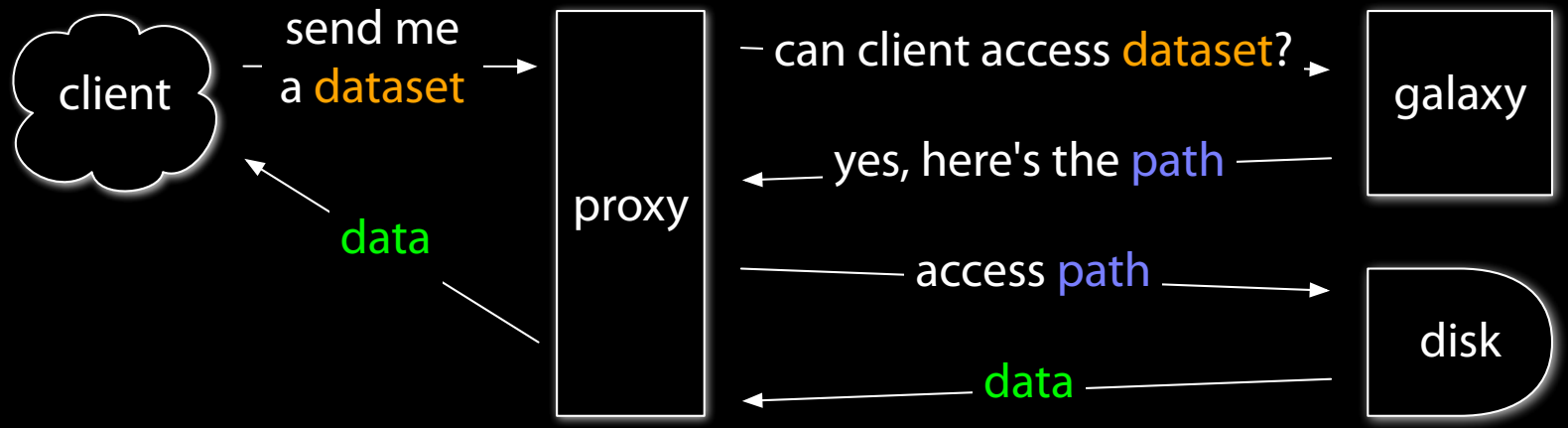
Downloading data

from Galaxy



Downloading data

from the proxy



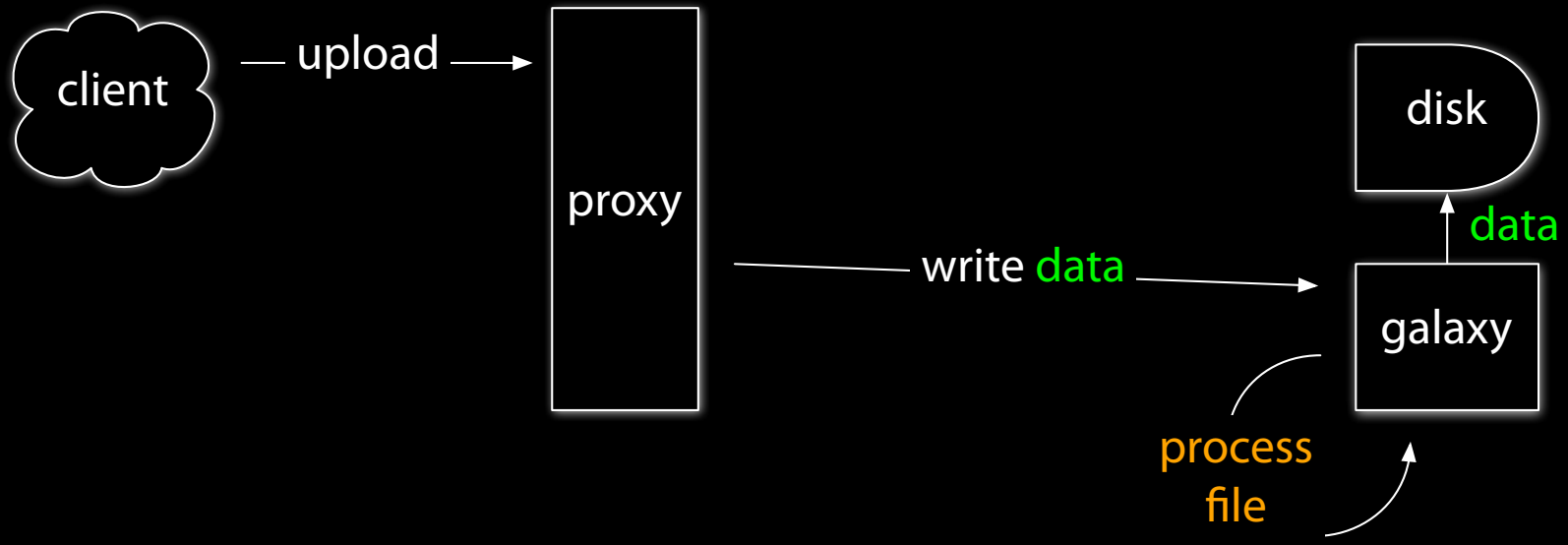
Downloading data

from the proxy

- The proxy server can send files much faster than Galaxy's internal HTTP server and file I/O methods
- Reduce load on the application, free the process
- Restartability
- Security is maintained: the proxy consults Galaxy for authZ
- Proxy server requires minimal config and then:
  - nginx: `nginx_x_accel_redirect_base = /_download`
  - Apache: `apache_xsendfile = True`

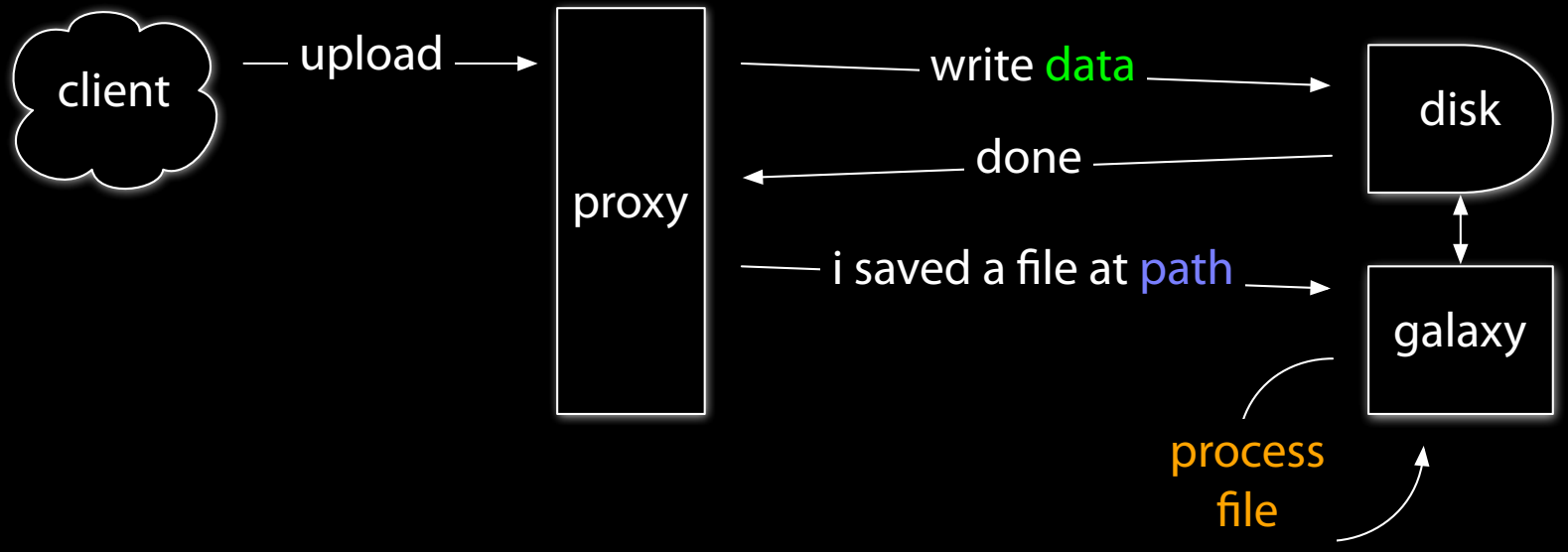
Uploading data

to Galaxy



Uploading data

to the proxy (nginx)



Uploading data

to the proxy

- The proxy is also better at receiving files than Galaxy
- Again, reduce load on the application, free the process
- Again, restartability
- More reliable
- Slightly more complicated to set up, and nginx only

Uploading data

from a local filesystem

- Many browsers have file size limitations
- Interrupted uploads cannot be resumed
- You may want to upload directly from a server
- Perhaps your data is already on a filesystem locally accessible to the Galaxy server

Uploading data

from a local filesystem

- For data libraries
- For histories

Uploading data

from a local filesystem

- Non-admin users may also upload to libraries from the local filesystem if granted permission and `user_library_import_dir` is set

Uploading data

via FTP

- FTP not explicitly required, cp, scp, sftp, whatever method your users have to place data on the server
- Full config example for ProFTPd with user authentication against Galaxy available in Wiki

Caching data locally

- Some data (e.g. sequences and associated indexes) are useful to many but should automatically be recognized as available by tools
- Placing in a data library and requiring history import every time would be a nuisance
- Avoid duplication and wasted time repeatedly building indexes on the same sequences
- Manage locally cached data in Galaxy

Limitless tool resources: Use a cluster (or two)

- Move intensive processing (tool execution) to other hosts
- Utilize existing resources
- No job interruption upon restart
- Per-tool cluster options
- DRMAA supports most other DRMs
- It's easy: Set `start_job_runners` and `default_cluster_job_runner` and go!

Per-tool Job Control

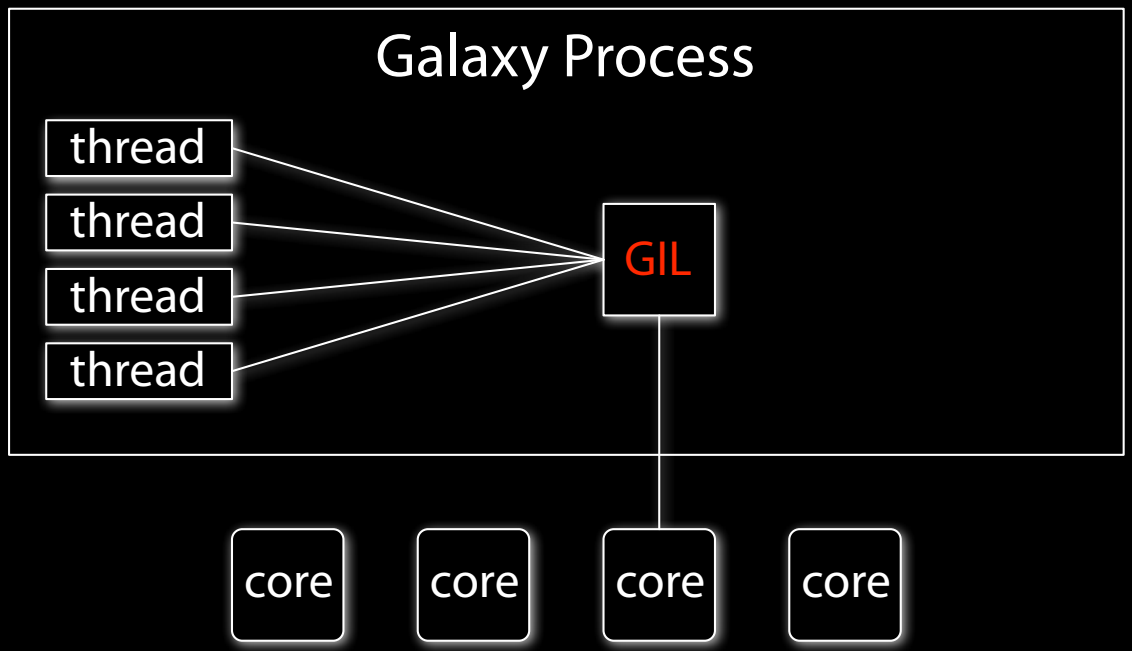
- `default_cluster_job_runner = pbs:///`
- `fastq_groomer = pbs://othercluster.example.org/groomerq`
- `bowtie_wrapper = pbs:///ngsq/-l nodes=1:ppn=8/`

Job users on the cluster

- By default, jobs run as the user Galaxy is started as
- If your Galaxy users and cluster system users are identical, you may wish to run jobs on the cluster as the actual user
- Galaxy uses sudo to change ownership of relevant files and submit the job to the cluster as the correct system user
- Configurable for your specific environment

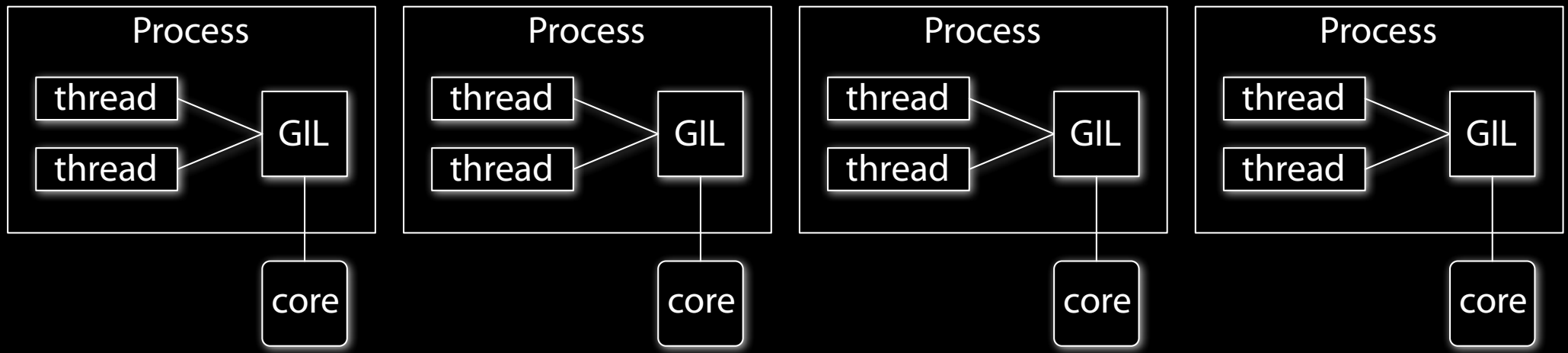
Python and threading

- Galaxy is multi-threaded. No problem, right?
- Problem... Enter the Global Interpreter Lock



- Guido says: "run multiple processes instead of threads!"

Opening the bottleneck



- One job manager - responsible for dispatching jobs to handlers
- Many job handlers - responsible for preparing and finishing jobs, monitoring cluster queue(s)
- Many web servers

Defining extra servers is easy

```
[server:web_0]
port = 8000
[server:web_1]
port = 8001
[server:web_2]
port = 8002
...
```

```
[server:manager]
port = 8099

[server:handler0]
port = 8100
[server:handler1]
port = 8101
...
```

Unload the Galaxy server

- Setting metadata is CPU intensive and will make Galaxy unresponsive
  - Make a new process (better yet, run on the cluster!)
  - All you need is: `set_metadata_externally = True`
- Run the data source tools on the cluster if they have access to the Internet
  - Remove `tool = local:///` from config file

Data Management

- The Galaxy philosophy
  - Data is never overwritten
  - Data is never deleted

Data Management

filesystem choices

- Storage can easily be the bottleneck
  - Your storage must scale with your cluster
- Transparent compression and deduplication can reduce usage drastically
- Suggestions
  - ZFS: usegalaxy.org relies on ZFS on Solaris
  - ZFS on FreeBSD stable, Native ZFS on Linux coming
  - Btrfs may be viable soon

Data Management

creating data

- By default, all Galaxy history and library datasets are assigned an ID and stored in `galaxy-dist/database/files/`
- Single directory = single massive filesystem
- Galaxy has a dataset abstraction layer to decouple from a single local filesystem: Object Store
  - Disk backend: single filesystem
  - Distributed backend: multiple filesystems
  - Amazon S3 backend in development

Data Management

creating data

```
object_store = distributed
```

```
<?xml version="1.0"?>
<backends maxpctfull="90">
    <backend id="pool1" type="disk" weight="5">
        <files_dir path="/pool/pool1/files"/>
        <extra_dir type="temp" path="/pool/pool1/tmp"/>
        <extra_dir type="job_work" path="/pool/pool1/work"/>
    </backend>
    <backend id="pool2" type="disk" weight="1">
        <files_dir path="/pool/pool2/files"/>
        <extra_dir type="temp" path="/pool/pool2/tmp"/>
        <extra_dir type="job_work" path="/pool/pool2/work"/>
    </backend>
</backends>
```

Data Management

cleaning data

- Data is never removed from disk unless
  - `allow_user_dataset_purge = True`
  - users click “delete permanently”
- Solution: cleanup_datasets.py
  - Run from cron to remove data from disk that has been deleted by the user (but not “deleted permanently”)
  - Configurable deletion policy allows removal after data has been deleted for a specified number of days

Monitoring

- Monitor Galaxy
  - Provided methods:
    - With cron/email using `galaxy-dist/cron/check_galaxy.sh`
    - With Nagios using `galaxy-dist/contrib/nagios`
  - The provided scripts upload files and run jobs

Collecting Statistics

- The Galaxy Reports webapp
  - Modify `galaxy-dist/reports_wsgi.ini` for your environment
  - Start the webapp with `sh run_reports.sh`

Collecting Statistics

All Jobs for May 2012

Click Total Jobs to see jobs for that day

Day	Date	User Jobs	Monitor Jobs	Total Jobs
Saturday	May 26, 2012	1115	155	1270
Friday	May 25, 2012	5138	193	5331
Thursday	May 24, 2012	4600	153	4753
Wednesday	May 23, 2012	4384	188	4572
Tuesday	May 22, 2012	5852	195	6047
Monday	May 21, 2012	3905	198	4103
Sunday	May 20, 2012	1423	193	1616
Saturday	May 19, 2012	1158	182	1340
Friday	May 18, 2012	4546	190	4736
Thursday	May 17, 2012	4679	199	4878
Wednesday	May 16, 2012	7378	185	7563
Tuesday	May 15, 2012	5962	200	6162
Monday	May 14, 2012	4803	179	4982
Sunday	May 13, 2012	1563	196	1759
Saturday	May 12, 2012	1555	151	1706
Friday	May 11, 2012	5778	57	5835
Thursday	May 10, 2012	4506	0	4506
Wednesday	May 09, 2012	4812	0	4812

usegalaxy.org/production

The Team