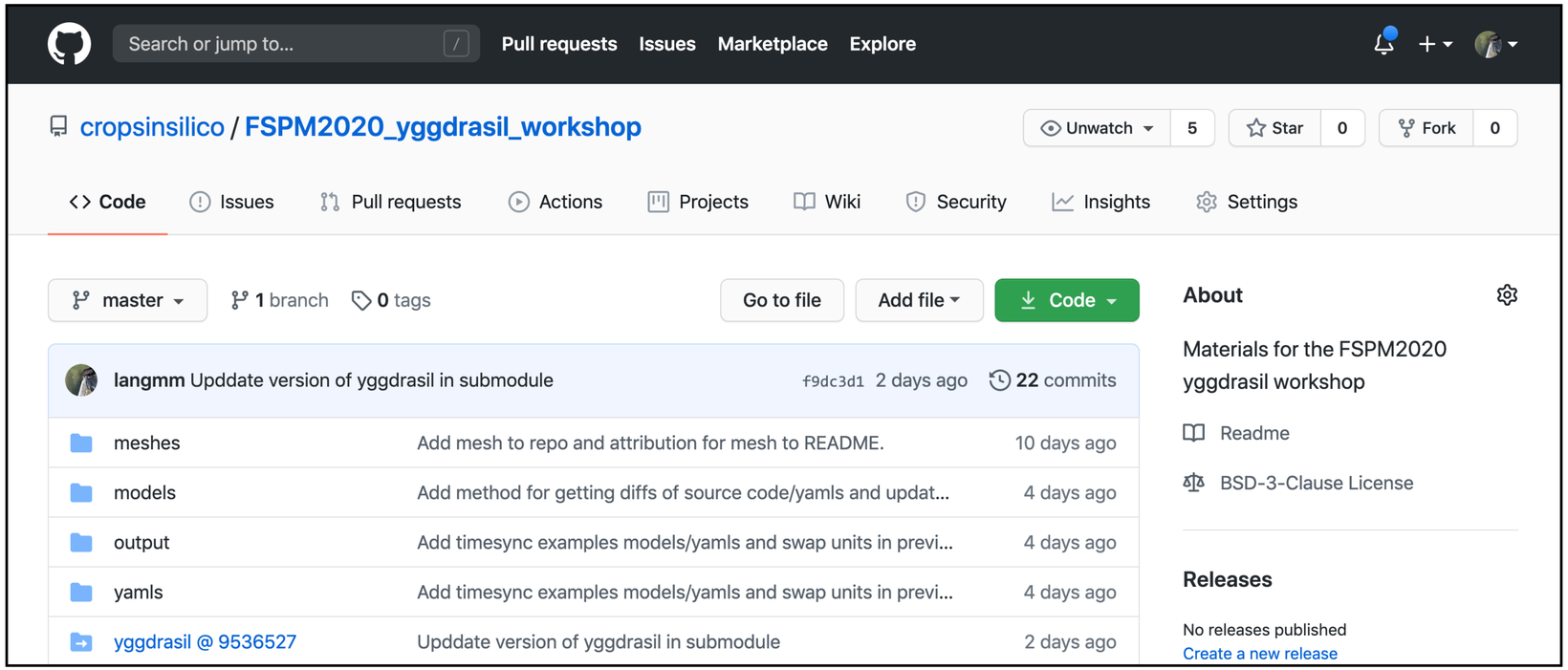
MEAGAN LANG

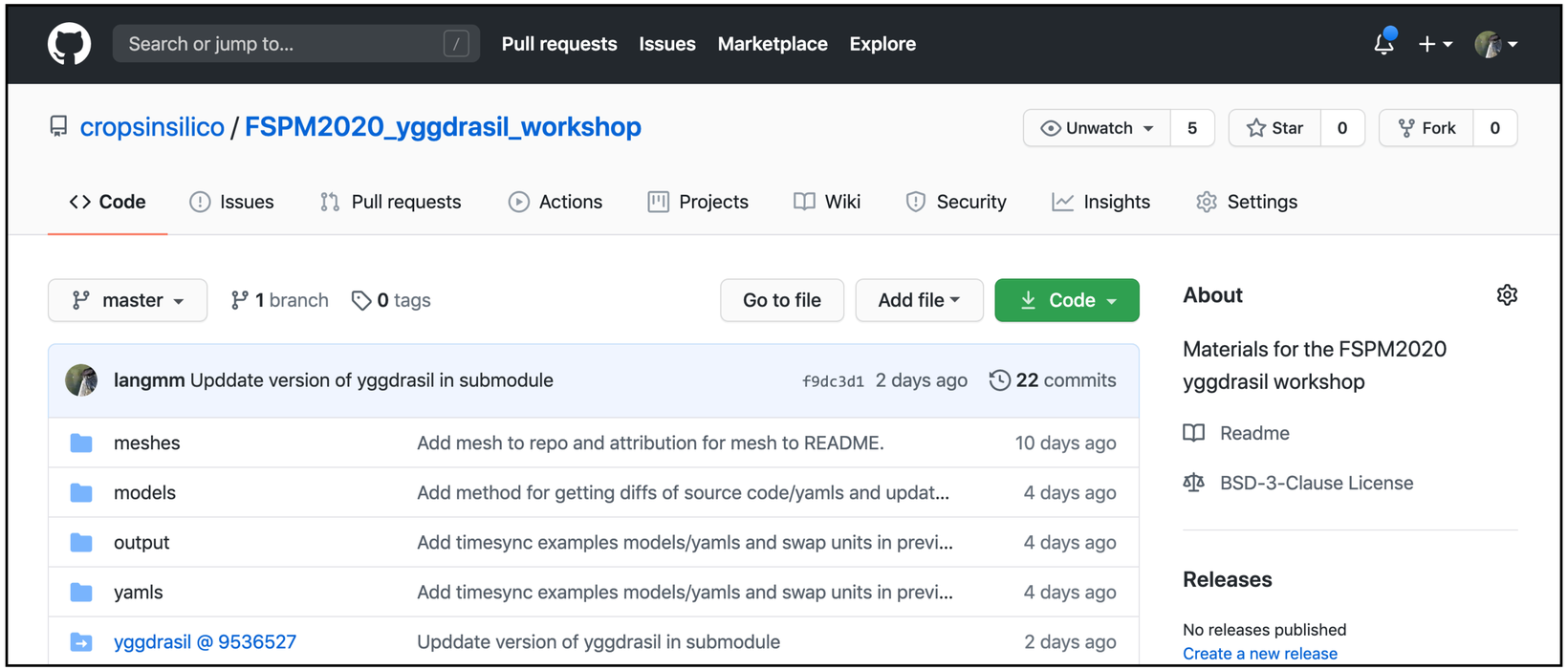
YGGDRASIL

THE CROPS IN SILICO PROJECT'S OPEN-SOURCE PYTHON PACKAGE FOR CONNECTING COMPUTATIONAL MODELS ACROSS PROGRAMMING LANGUAGES

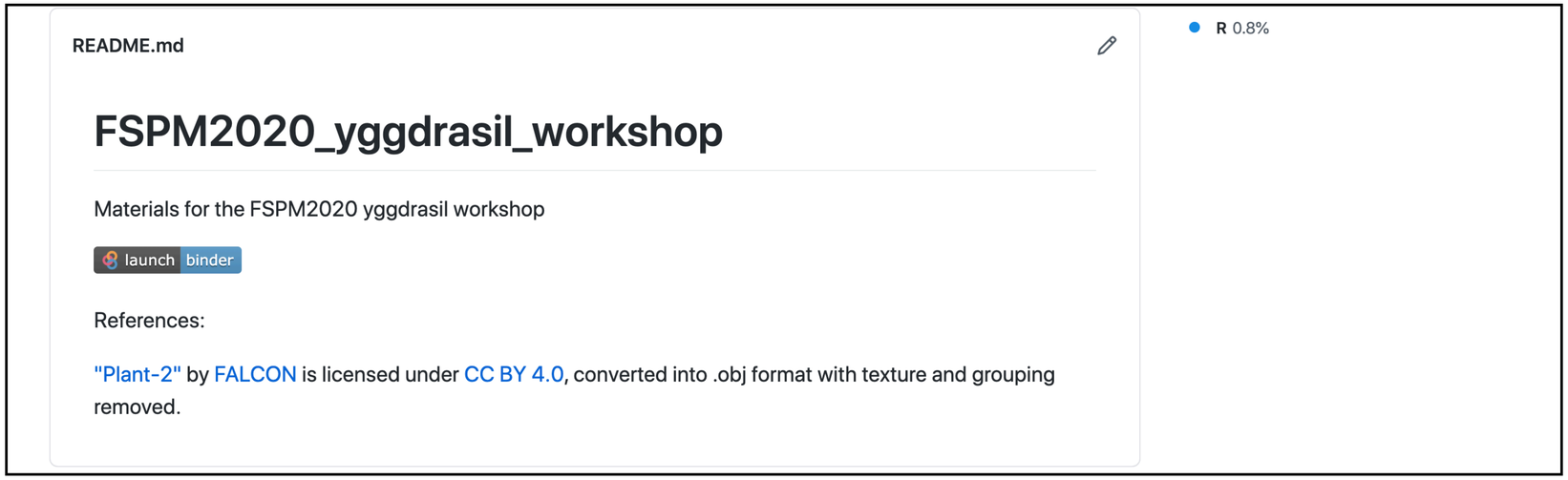
BUT FIRST…

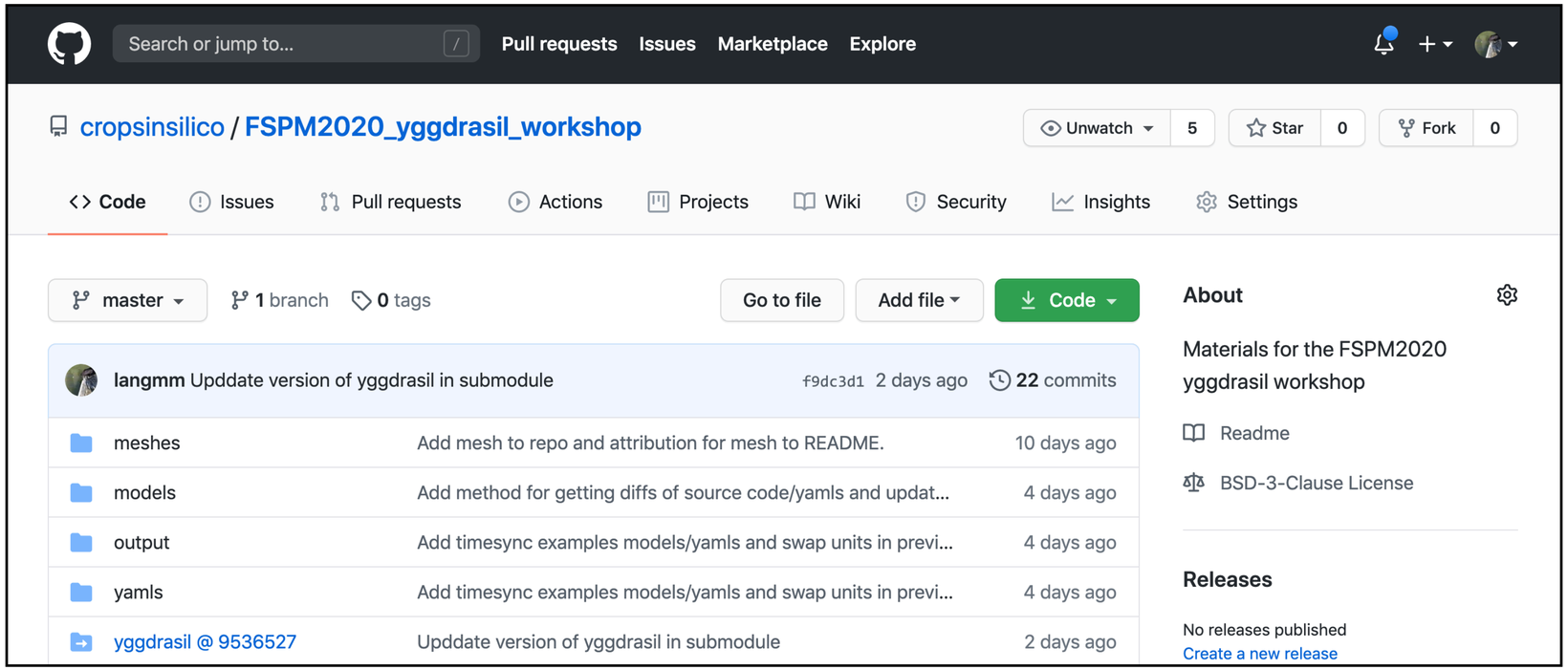
NOTEBOOK PREP


Search or jump to...
Pull requests
Issues
Marketplace
Explore
cropsinsilico / FSPM2020_yggdrasil_workshop
Unwatch 5
Star 0
Fork 0
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
master
1 branch
0 tags
Go to file
Add file
Code
langmm Upddate version of yggdrasil in submodule
f9dc3d1 2 days ago
22 commits
meshes Add mesh to repo and attribution for mesh to README. 10 days ago
models Add method for getting diffs of source code/yamls and updat... 4 days ago
output Add timesync examples models/yamls and swap units in previ... 4 days ago
yamls Add timesync examples models/yamls and swap units in previ... 4 days ago
yggdrasil @ 9536527 Upddate version of yggdrasil in submodule 2 days ago
About
Materials for the FSPM2020 yggdrasil workshop
Readme
BSD-3-Clause License
Releases
No releases published
Create a new release

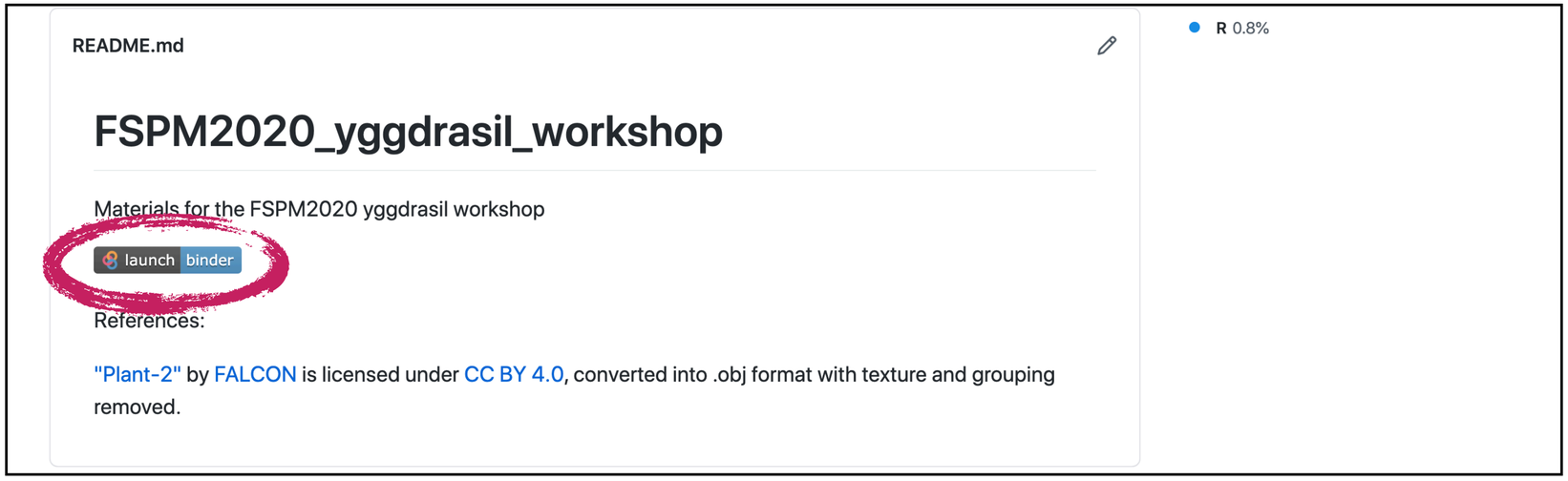

Search or jump to...
Pull requests
Issues
Marketplace
Explore
cropsinsilico / FSPM2020_yggdrasil_workshop
Unwatch 5
Star 0
Fork 0
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
master
1 branch
0 tags
Go to file
Add file
Code
langmm Upddate version of yggdrasil in submodule
f9dc3d1 2 days ago
22 commits
meshes
Add mesh to repo and attribution for mesh to README.
10 days ago
models
Add method for getting diffs of source code/yamls and updat...
4 days ago
output
Add timesync examples models/yamls and swap units in previ...
4 days ago
yamls
Add timesync examples models/yamls and swap units in previ...
4 days ago
yggdrasil @ 9536527
Upddate version of yggdrasil in submodule
2 days ago
About
Materials for the FSPM2020 yggdrasil workshop
Readme
BSD-3-Clause License
Releases
No releases published
Create a new release


SCROLL


README.md
FSPM2020_yggdrasil_workshop
Materials for the FSPM2020 yggdrasil workshop
launch binder
References:
"Plant-2" by FALCON is licensed under CC BY 4.0, converted into .obj format with texture and grouping removed.
R 0.8%


Search or jump to...
Pull requests
Issues
Marketplace
Explore
cropsinsilico / FSPM2020_yggdrasil_workshop
Unwatch 5
Star 0
Fork 0
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
master
1 branch
0 tags
Go to file
Add file
Code
langmm Upddate version of yggdrasil in submodule
f9dc3d1 2 days ago
22 commits
meshes
Add mesh to repo and attribution for mesh to README.
10 days ago
models
Add method for getting diffs of source code/yamls and updat...
4 days ago
output
Add timesync examples models/yamls and swap units in previ...
4 days ago
yamls
Add timesync examples models/yamls and swap units in previ...
4 days ago
yggdrasil @ 9536527
Upddate version of yggdrasil in submodule
2 days ago
About
Materials for the FSPM2020 yggdrasil workshop
Readme
BSD-3-Clause License
Releases
No releases published
Create a new release

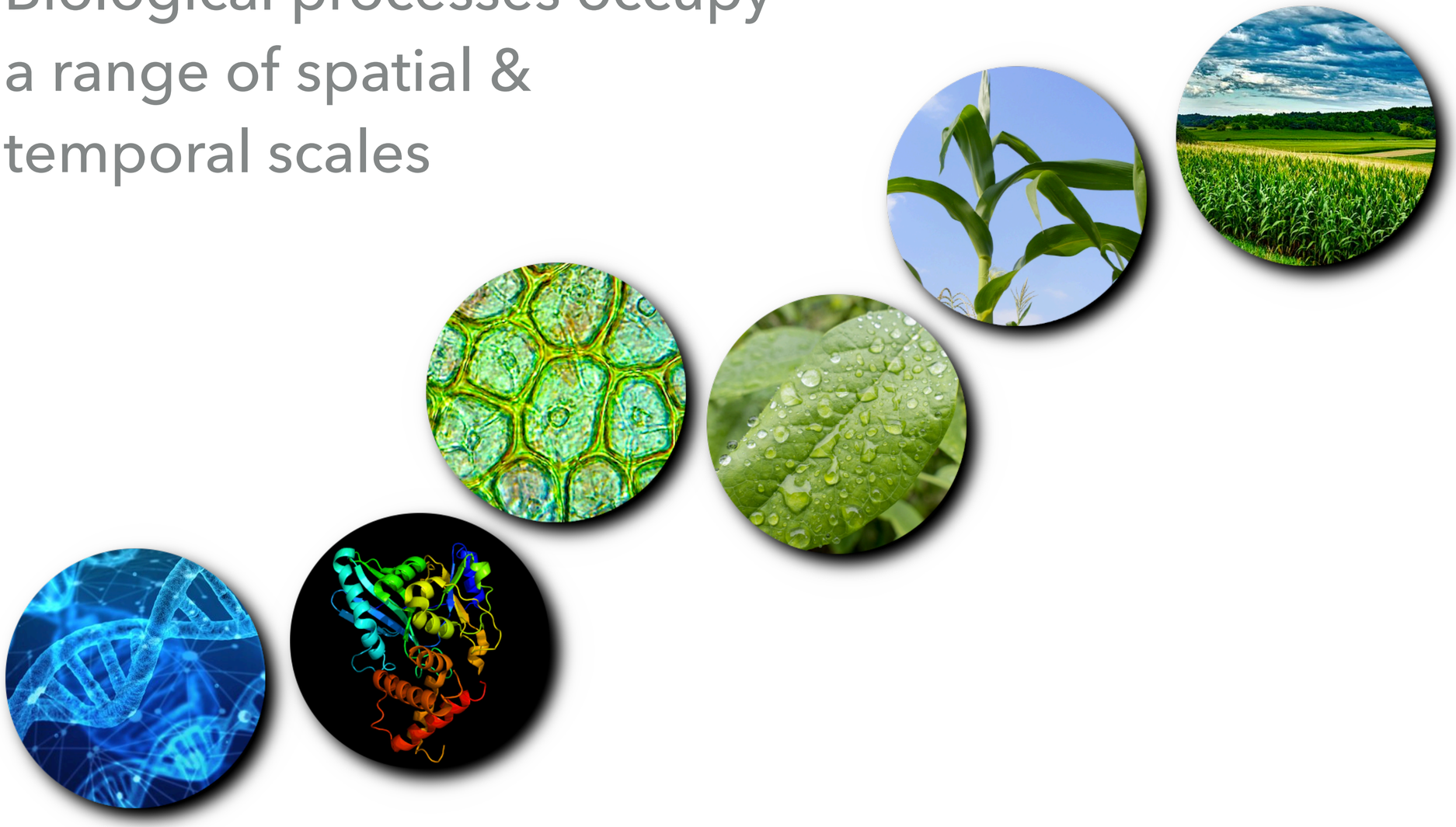
SCROLL


README.md
FSPM2020_yggdrasil_workshop
Materials for the FSPM2020 yggdrasil workshop
launch binder
References:
"Plant-2" by FALCON is licensed under CC BY 4.0, converted into .obj format with texture and grouping removed.
R 0.8%

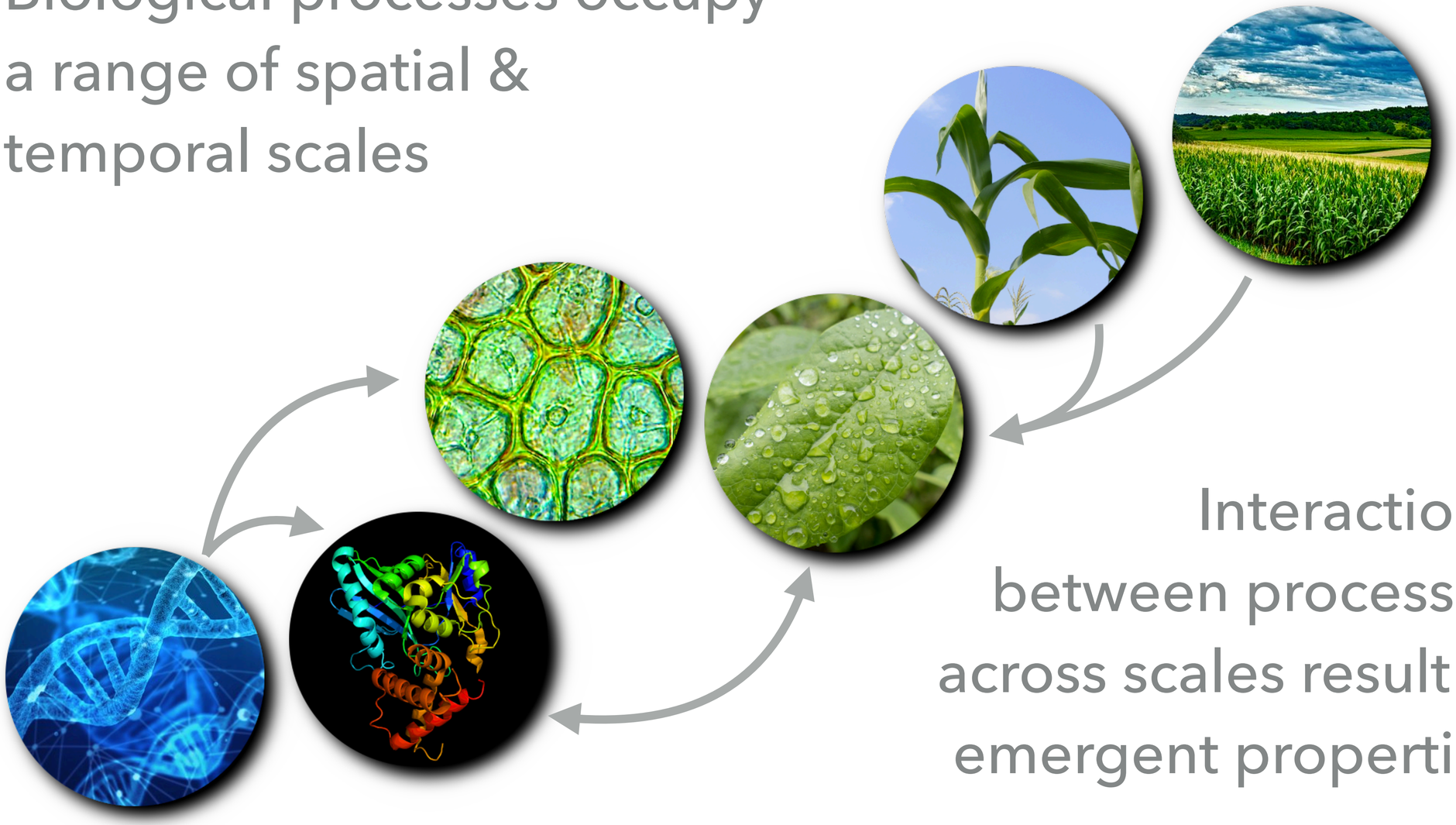
OK BACK TO THE…

INTRODUCTION

Biological processes occupy
a range of spatial &
temporal scales

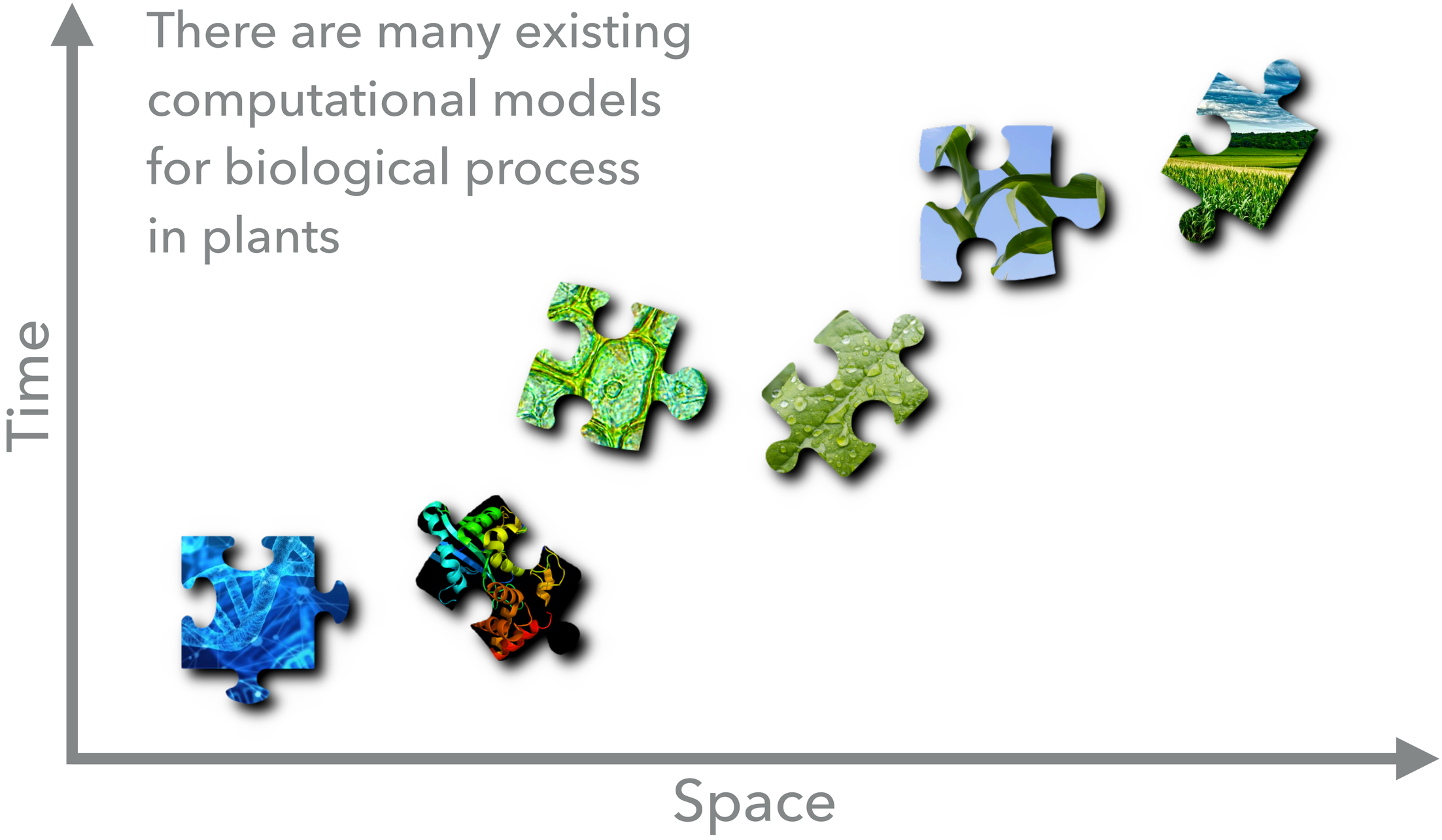


Biological processes occupy
a range of spatial &
temporal scales

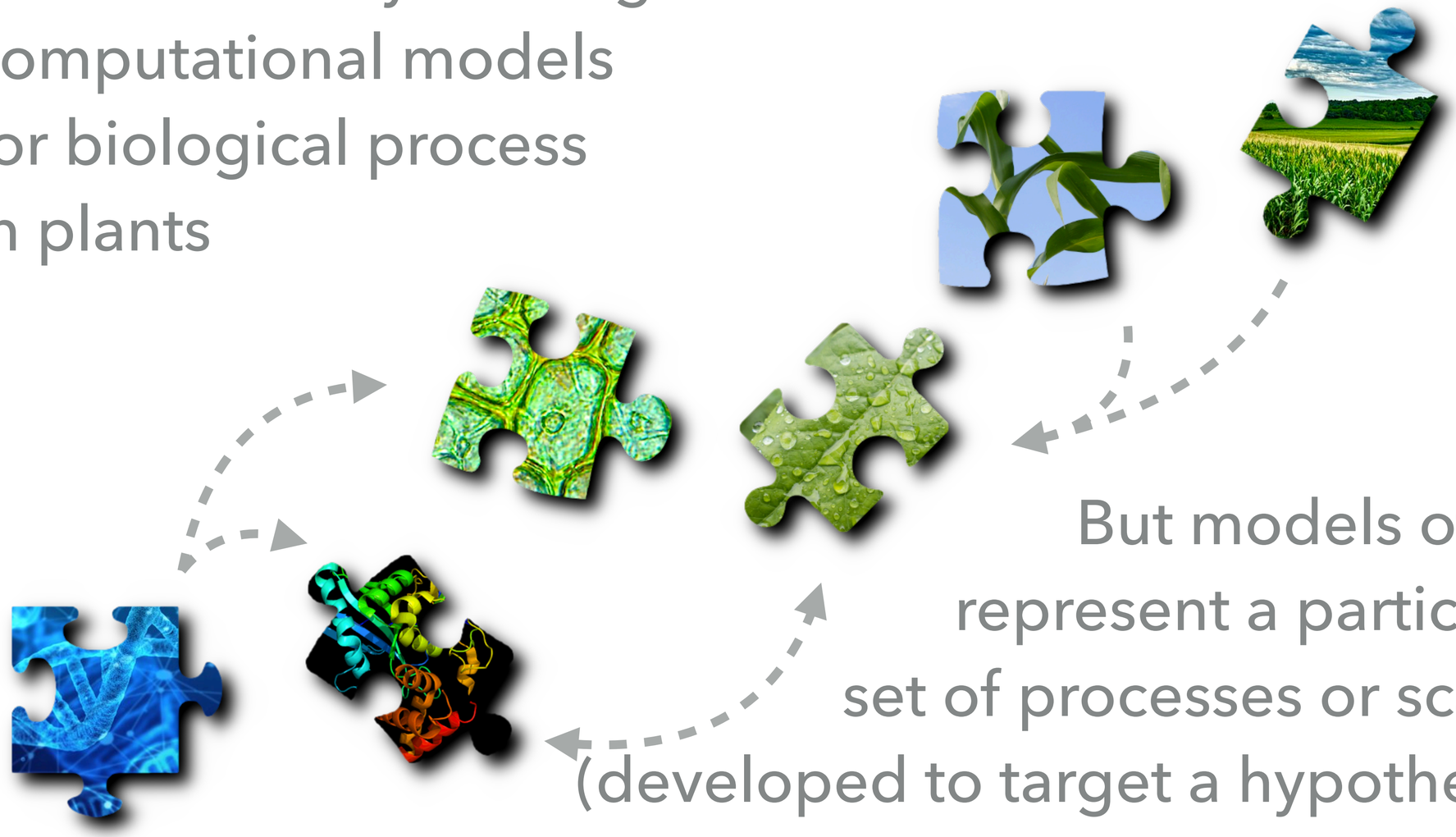


Interactions
between processes
across scales result in
emergent properties

There are many existing computational models for biological process in plants



There are many existing computational models for biological process in plants



But models often represent a particular set of processes or scales (developed to target a hypothesis)

Time

Space

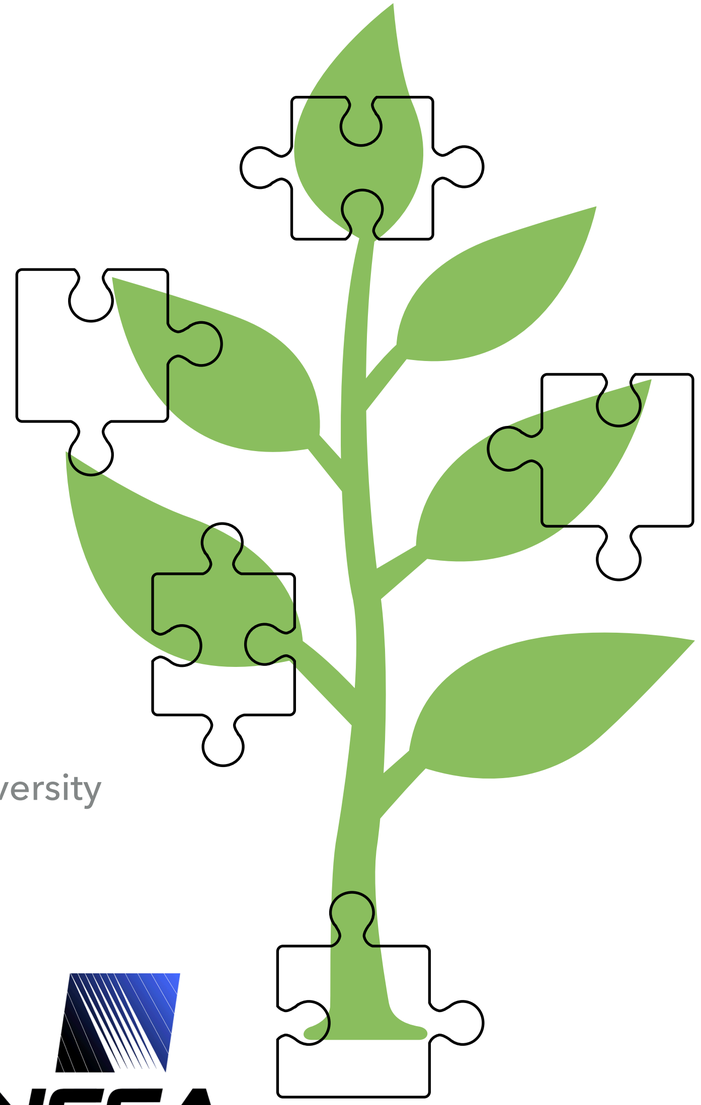
Crops[1] *in silico*

ILLINOIS

In partnership with
Oxford University
Pennsylvania State University
Purdue University
University of Nebraska

Funded by

FFAR ISEE NCSA



Models are pieces from different puzzles

We need an adapter

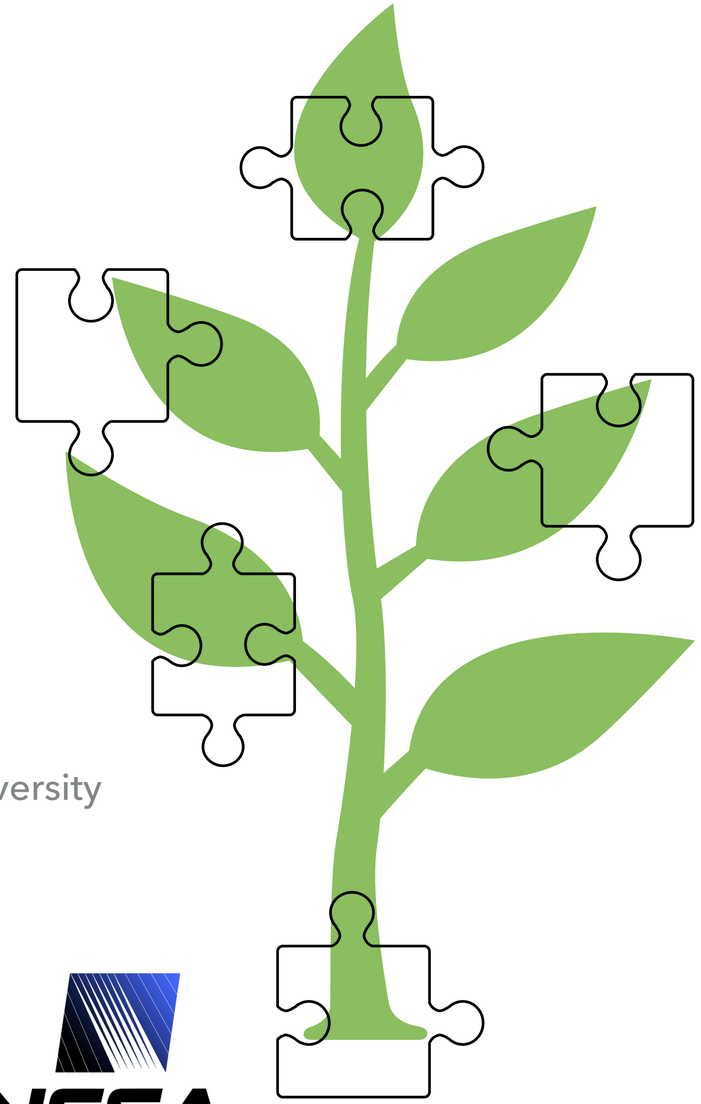
Crops[1] *in silico*

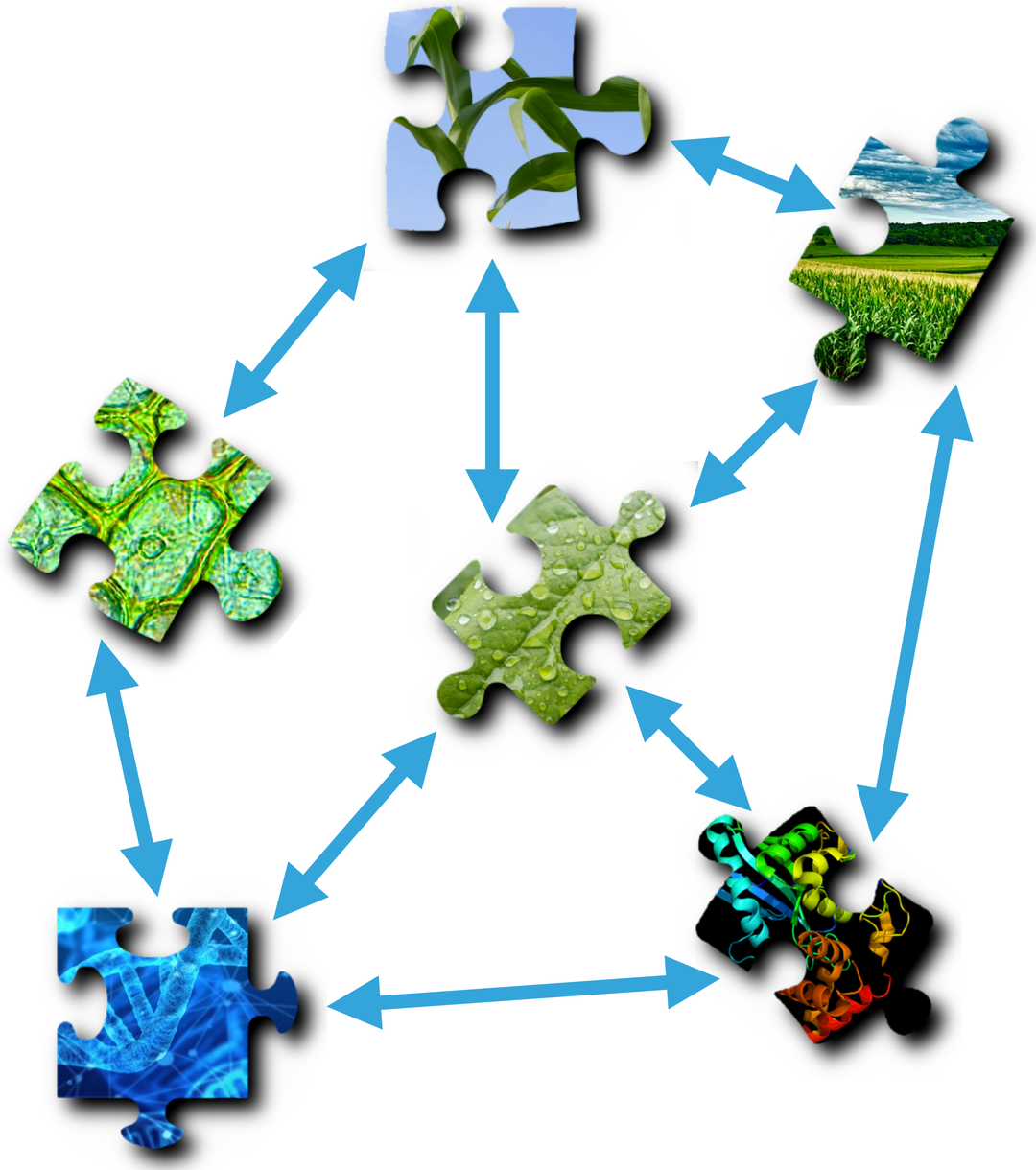
ILLINOIS

In partnership with
Oxford University
Pennsylvania State University
Purdue University
University of Nebraska

Funded by

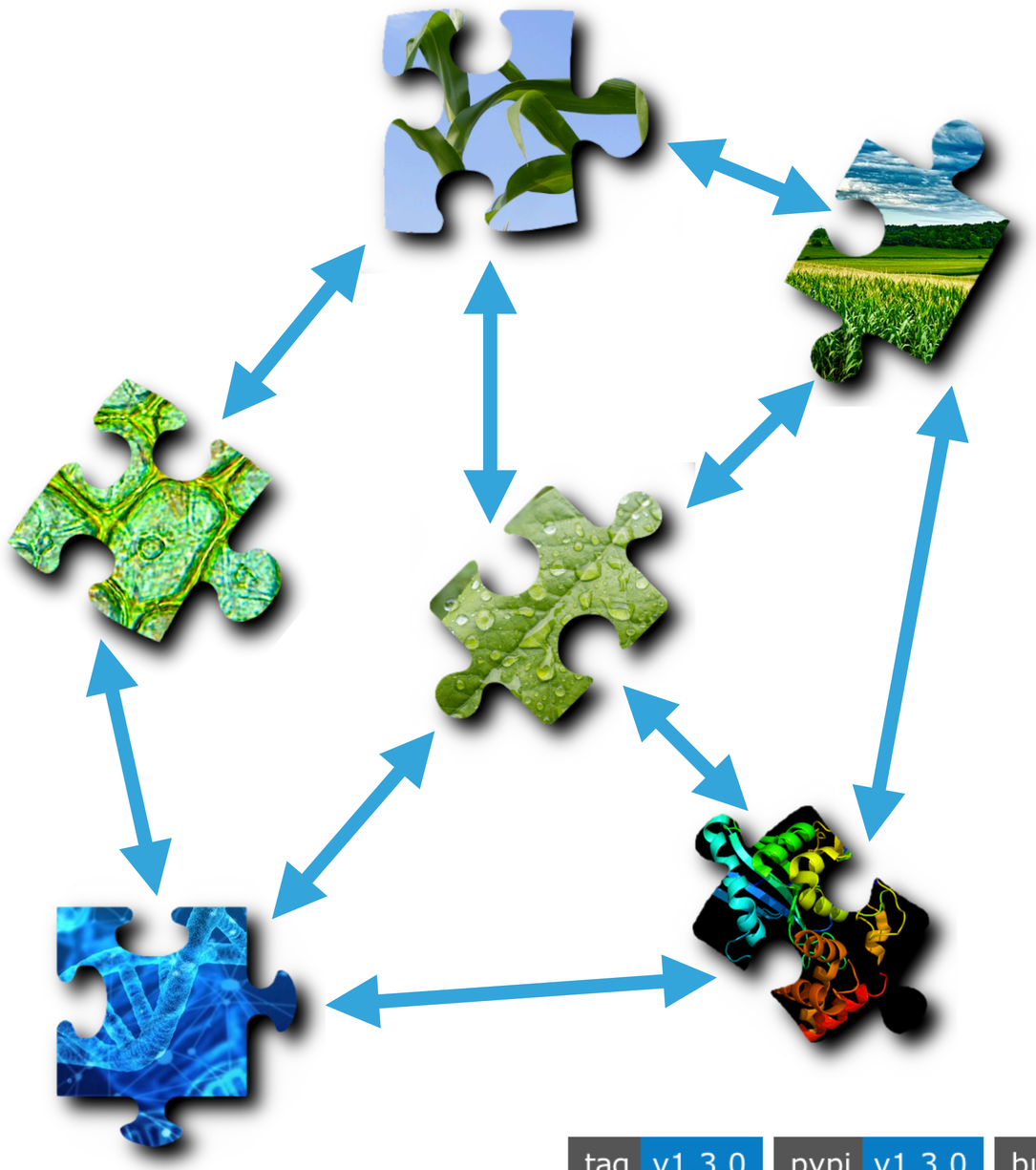
FFAR ISEE NCSA





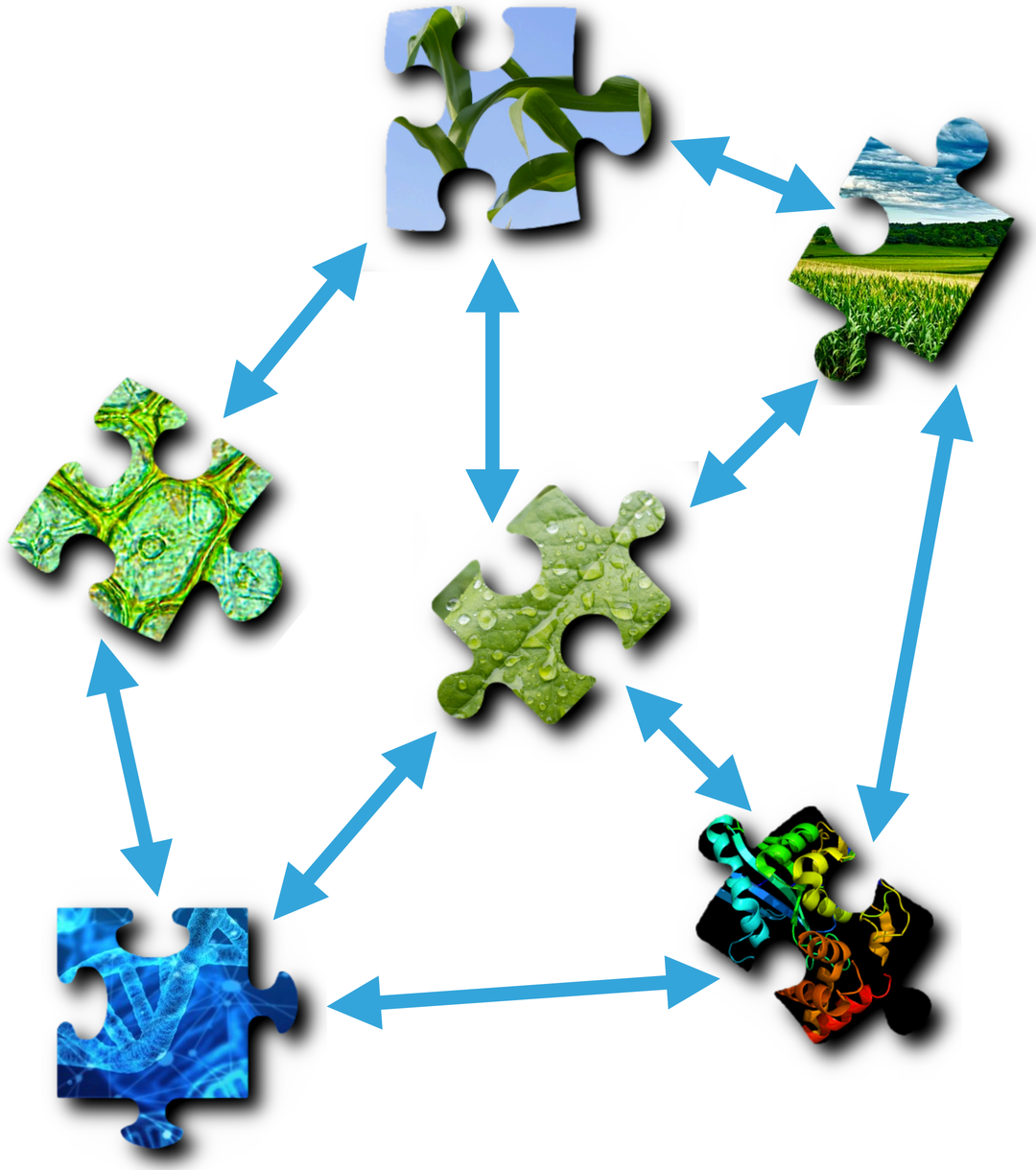
YGGDRASIL:

OPEN SOURCE PYTHON PACKAGE FOR CONNECTING MODELS ACROSS SCALES AND LANGUAGES



YGGDRASIL:

OPEN SOURCE PYTHON PACKAGE FOR CONNECTING MODELS ACROSS SCALES AND LANGUAGES



LANGUAGES

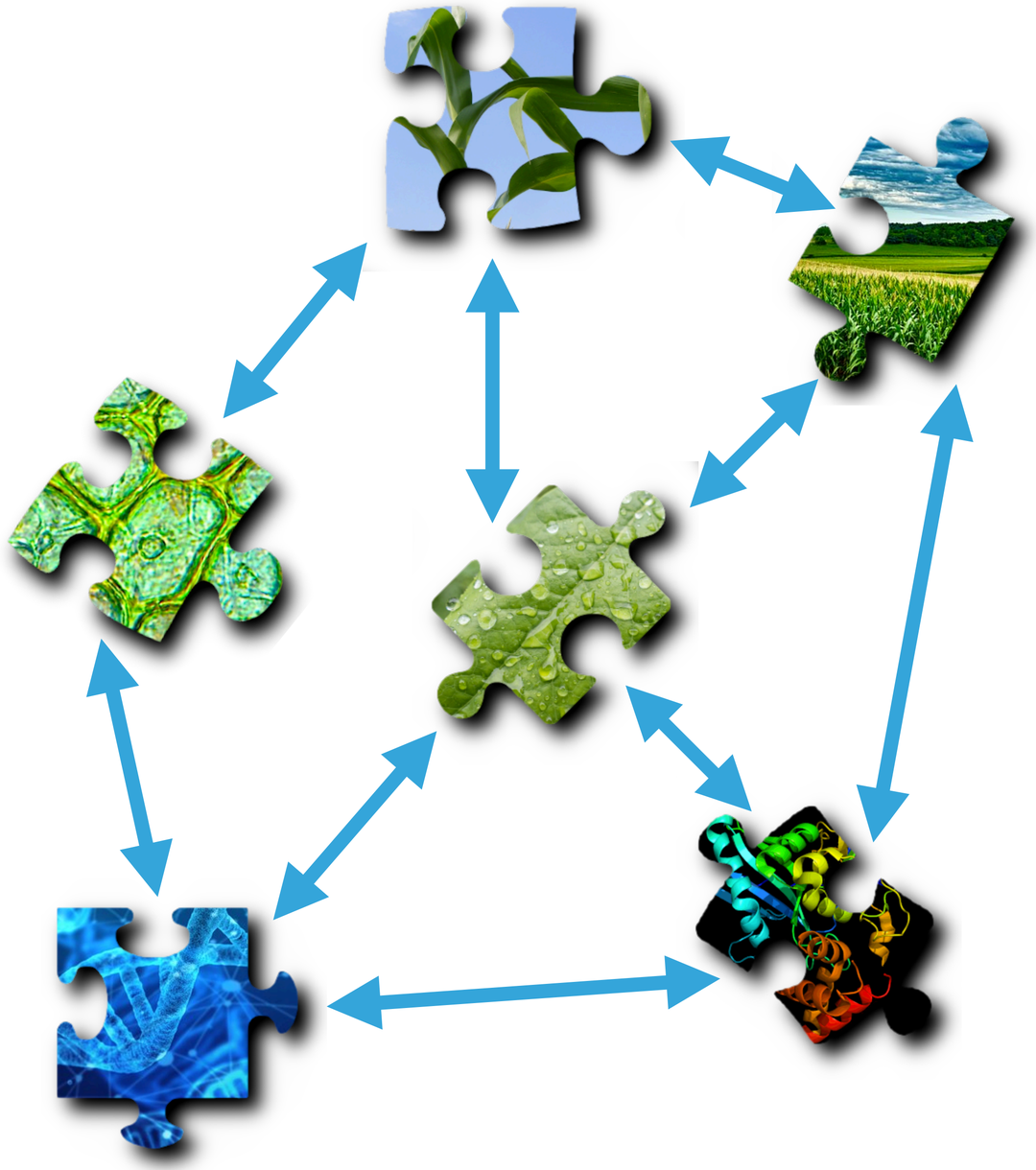
PYTHON

MATLAB

R

C/C++

FORTRAN (’03)



LANGUAGES

PYTHON

MATLAB

R

C/C++

FORTRAN ('03)

DOMAIN SPECIFIC LANGUAGES

SBML

OPENSIMROOT XML

COMMUNICATION

PARALLEL EXECUTION

MODEL A

MODEL B

PARALLEL EXECUTION

MODEL A

MODEL B

ASYNCHRONOUS COMMUNICATION

(SEND FIRST)

MODEL A

MODEL B

ASYNCHRONOUS COMMUNICATION

(SEND FIRST)

MODEL A

SEND

MODEL B

ASYNCHRONOUS COMMUNICATION

(SEND FIRST)

MODEL A

SEND

MODEL B

ASYNCHRONOUS COMMUNICATION

(SEND FIRST)

MODEL A

SEND

MODEL B

ASYNCHRONOUS COMMUNICATION

(SEND FIRST)

MODEL A

SEND

MODEL B

RECEIVE

ASYNCHRONOUS COMMUNICATION

(SEND FIRST)

MODEL A

SEND

MODEL B

RECEIVE

ASYNCHRONOUS COMMUNICATION

(RECEIVE FIRST)

MODEL A

MODEL B

ASYNCHRONOUS COMMUNICATION

(RECEIVE FIRST)

MODEL A

MODEL B

RECEIVE

ASYNCHRONOUS COMMUNICATION

(RECEIVE FIRST)

MODEL A

SEND

MODEL B

RECEIVE

ASYNCHRONOUS COMMUNICATION

(RECEIVE FIRST)

MODEL A

SEND

MODEL B

RECEIVE

ASYNCHRONOUS COMMUNICATION

(RECEIVE FIRST)

MODEL A

SEND

MODEL B

RECEIVE

COMM:

COMMUNICATION OBJECT ALLOWING MODELS TO SEND/RECEIVE MESSAGES TO/FROM OTHER MODELS

COMM CLASSES

INPUT

Receive messages from another model

OUTPUT

Send messages to another model

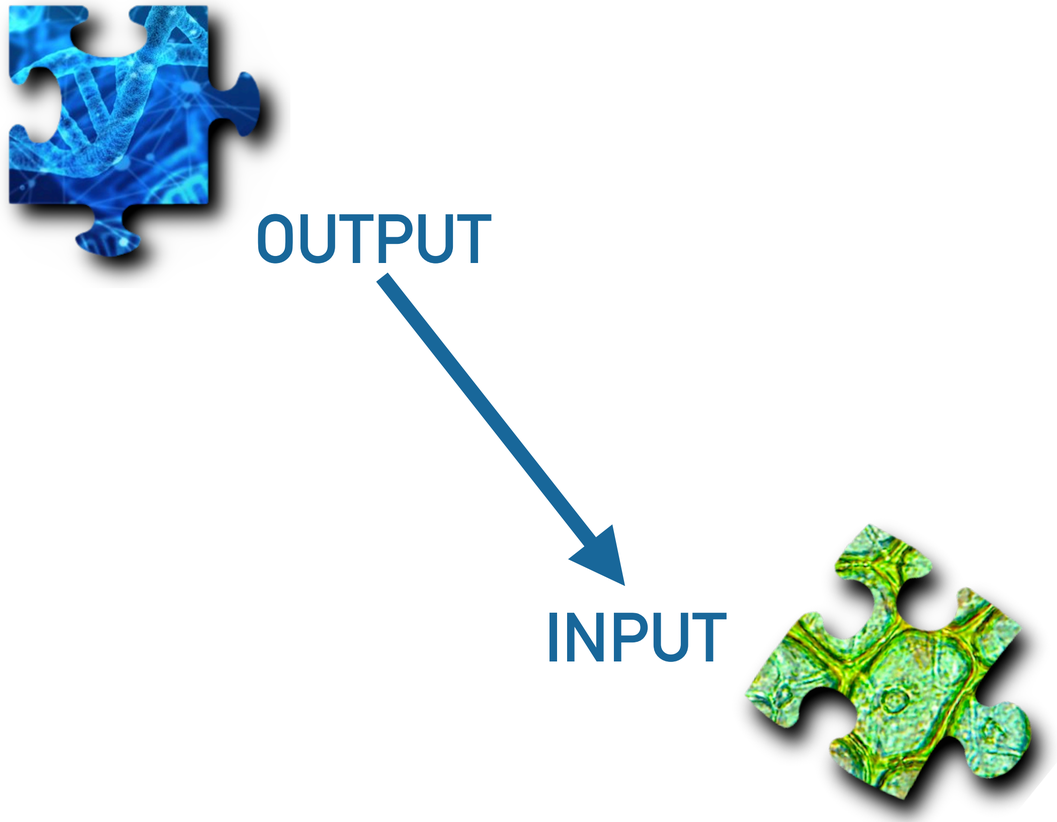
COMM CLASSES

INPUT

Receive messages from another model

OUTPUT

Send messages to another model



COMM CLASSES

INPUT

Receive messages from another model

OUTPUT

Send messages to another model

SERVER

Receive requests from client models and send responses

CLIENT

Send requests to a server model and receive messages (“call” a server model)

COMM CLASSES

CLIENT

INPUT

Receive messages from another model

OUTPUT

Send messages to another model

SERVER

SERVER

Receive requests from client models and send responses

CLIENT

Send requests to a server model and receive messages (“call” a server model)

COMM CLASSES

CLIENT

INPUT

Receive messages from another model

OUTPUT

Send messages to another model

SERVER

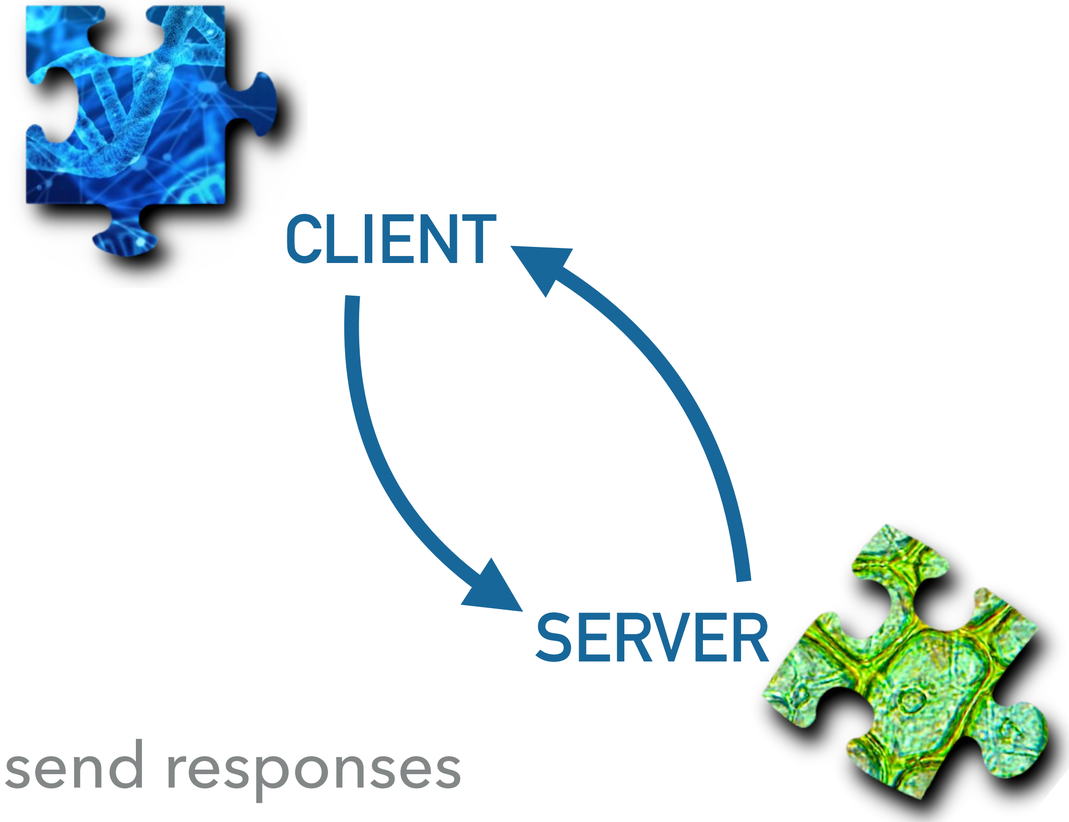
SERVER

Receive requests from client models and send responses

CLIENT

Send requests to a server model and receive messages (“call” a server model)

COMM CLASSES



INPUT

Receive messages from another model

OUTPUT

Send messages to another model

SERVER

Receive requests from client models and send responses

CLIENT

Send requests to a server model and receive messages (“call” a server model)

COMM CLASSES

INPUT

Receive messages from another model

OUTPUT

Send messages to another model

SERVER

Receive requests from client models and send responses

CLIENT

Send requests to a server model and receive messages (“call” a server model)

TIMESYNC

Send requests to a set of time-dependent models and receive time-dependent variables (“call” a time step synchronization)

COMM CLASSES

TIMESYNC

INPUT

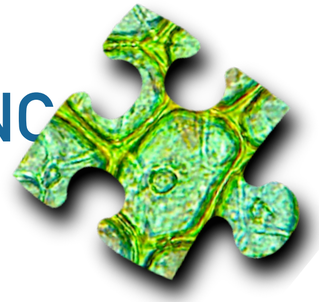
Receive messages from another model

SYNC

OUTPUT

Send messages to another model

TIMESYNC



SERVER

Receive requests from client models and send responses

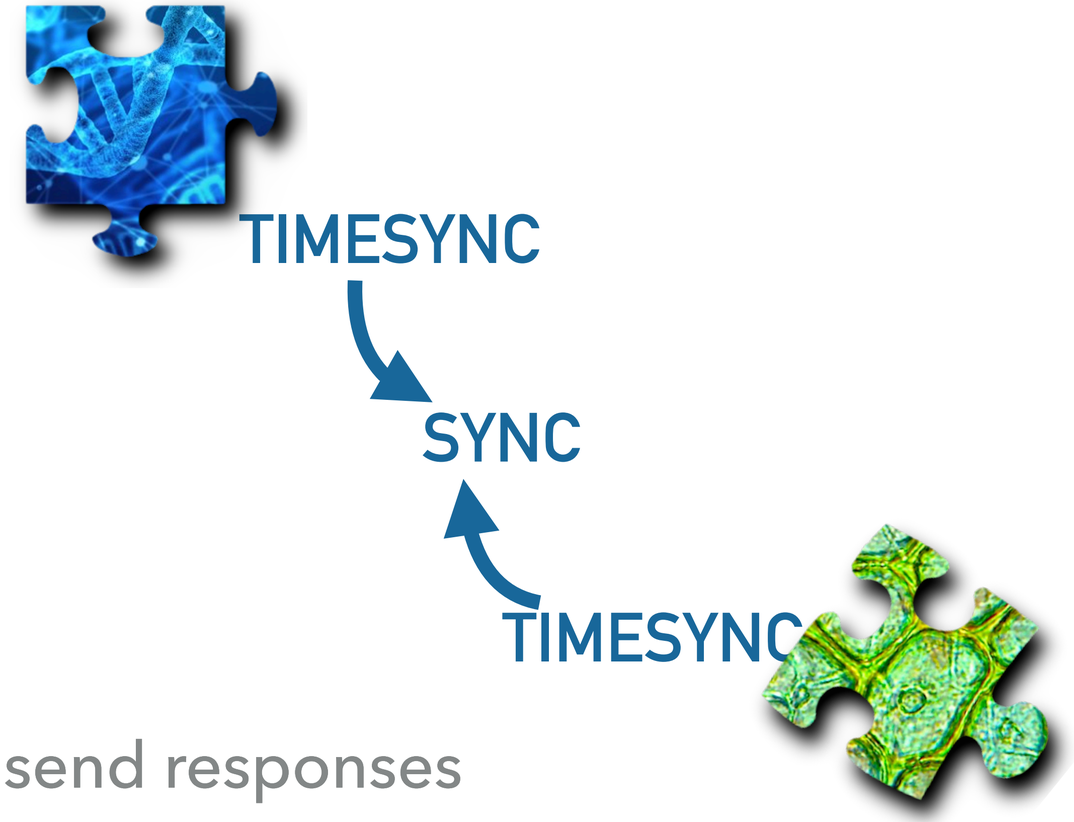
CLIENT

Send requests to a server model and receive messages (“call” a server model)

TIMESYNC

Send requests to a set of time-dependent models and receive time-dependent variables (“call” a time step synchronization)

COMM CLASSES



INPUT

Receive messages from another model

OUTPUT

Send messages to another model

SERVER

Receive requests from client models and send responses

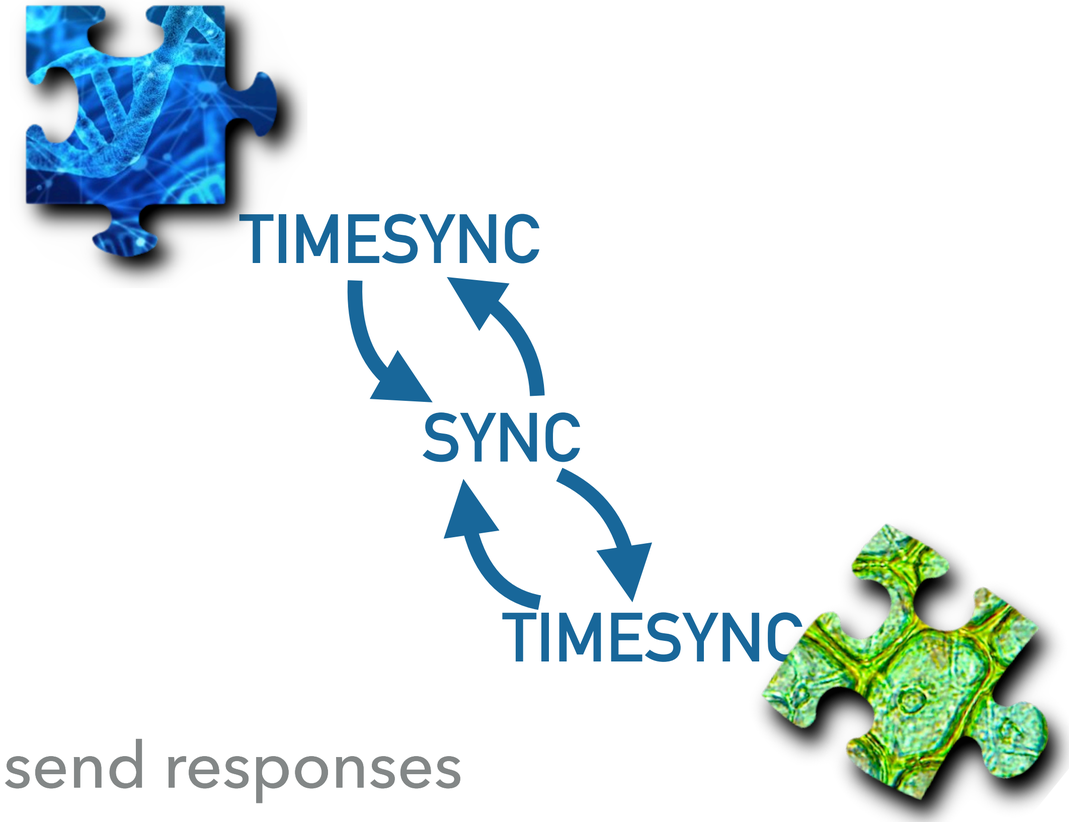
CLIENT

Send requests to a server model and receive messages (“call” a server model)

TIMESYNC

Send requests to a set of time-dependent models and receive time-dependent variables (“call” a time step synchronization)

COMM CLASSES



INPUT

Receive messages from another model

OUTPUT

Send messages to another model

SERVER

Receive requests from client models and send responses

CLIENT

Send requests to a server model and receive messages (“call” a server model)

TIMESYNC

Send requests to a set of time-dependent models and receive time-dependent variables (“call” a time step synchronization)

COMM METHODS

IPC

Interprocess communication, only available on Unix (Linux & Mac)

ZEROMQ

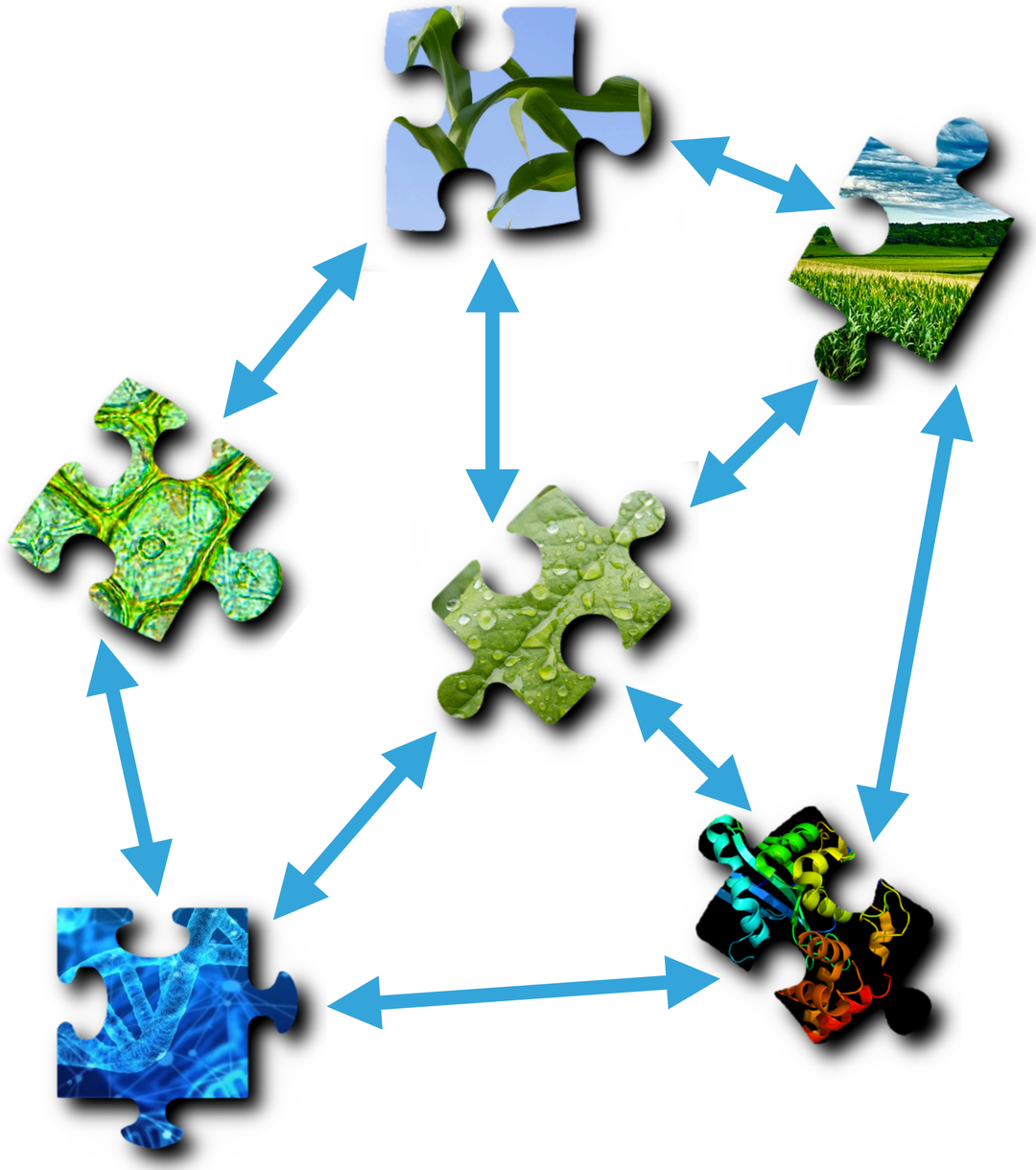
Broker-less communication sockets via TCP, IPC, UDP, inproc, etc.; available on all OSs

RABBITMQ

Brokered communication; requires a RabbitMQ server, but allows for more reliable communication with remote machines

FILES

Slow due to interaction with the disk, but useful for input/output of parameters in a generic way that allows the same model to be used w/ files or other models

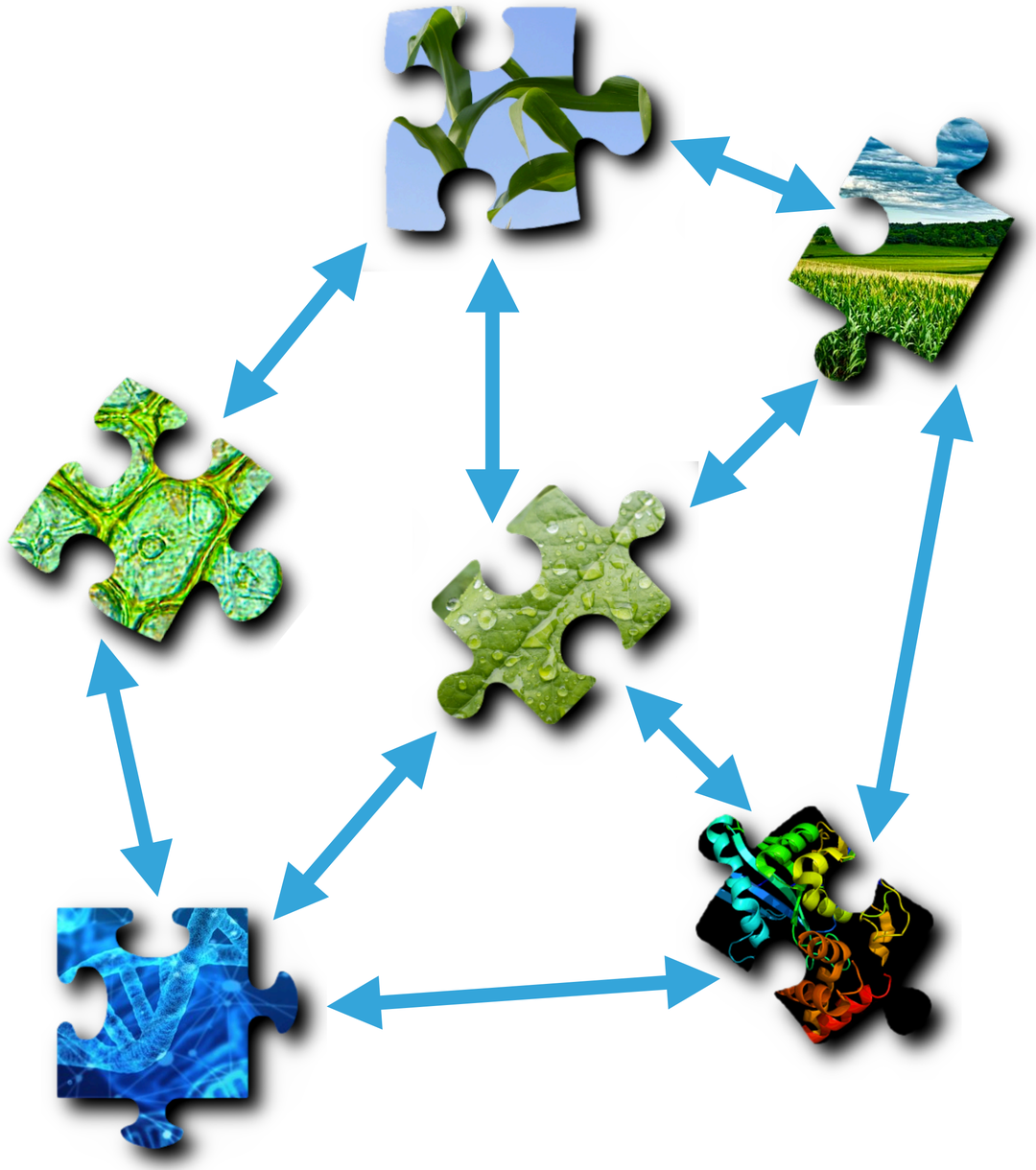


UNITS (AUTOMATED CONVERSION)

UNYT (PYTHON)

MATLAB SYMBOLIC UNITS

R UNITS



UNITS (AUTOMATED CONVERSION)

UNYT (PYTHON)

MATLAB SYMBOLIC UNITS

R UNITS

DATA FORMATS

SCALARS/ARRAYS

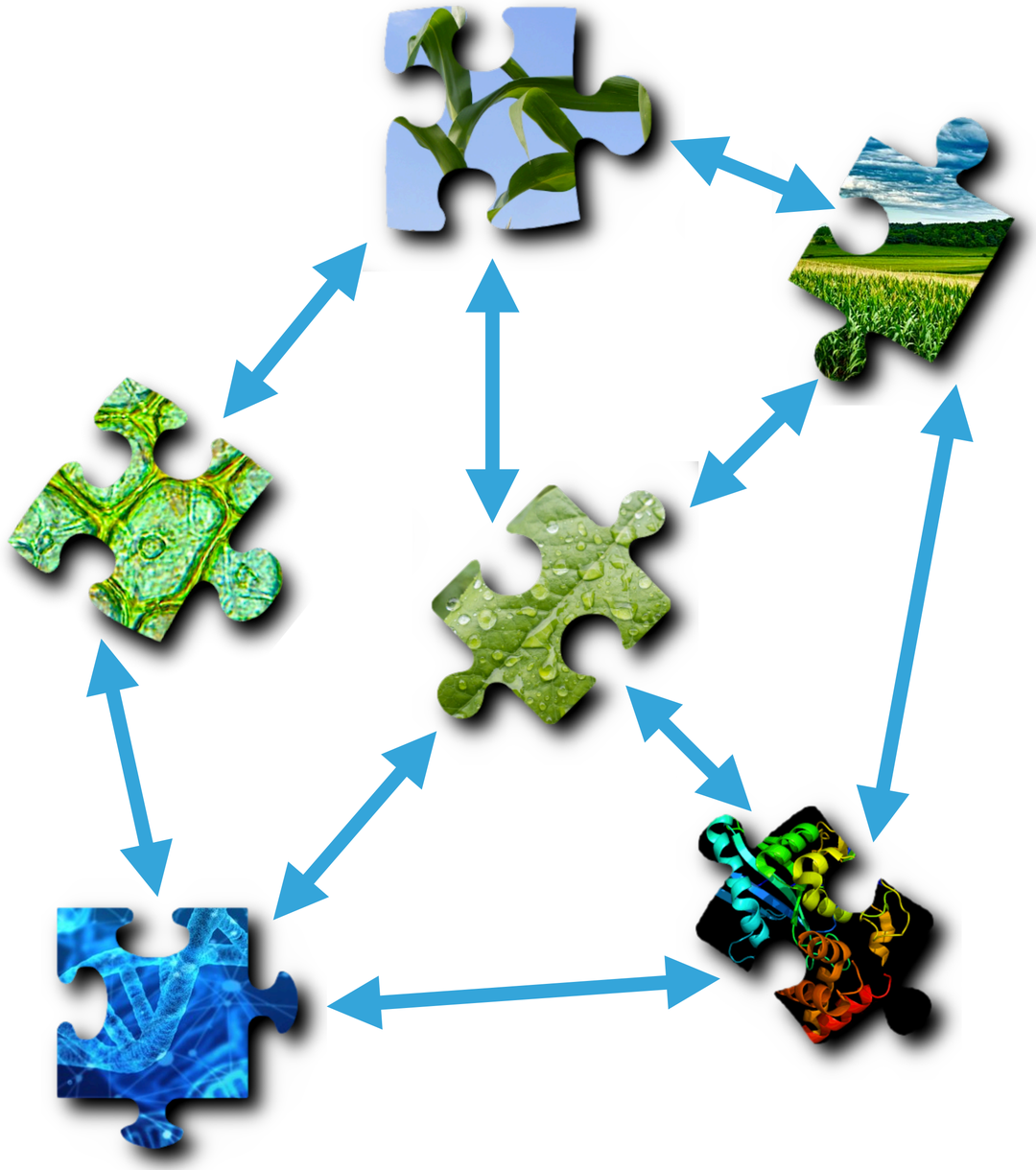
DELIMITED TABLES

PANDAS/R DATA FRAMES

PLY/OBJ

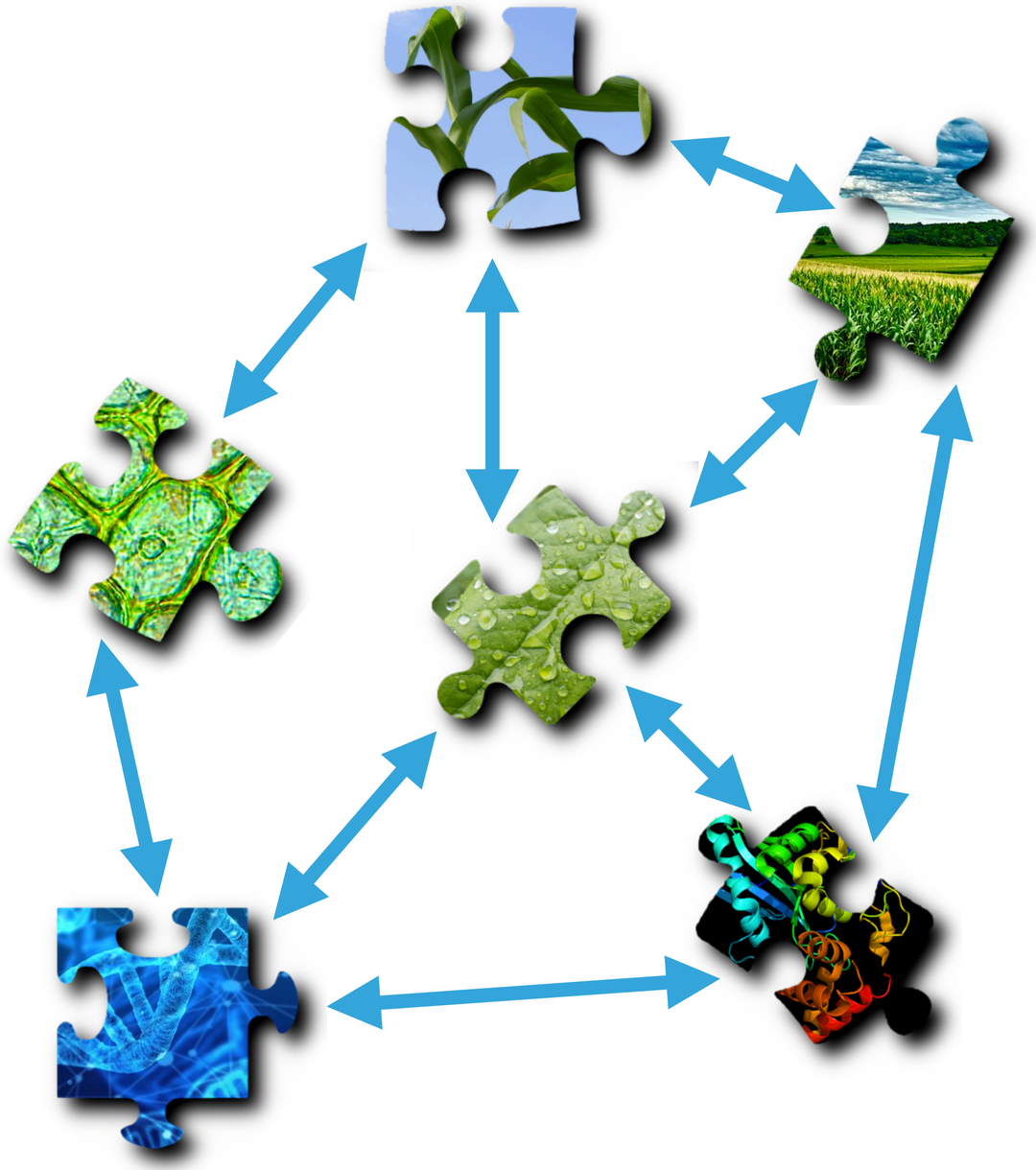
PYTHON PICKLES/
R .RDATA/MATLAB .MAT

EXECUTION



INTEGRATION:

NETWORK OF MODELS
RUN USING YGGDRASIL



YAML SPECIFICATION:

INPUT FILE(S) TO YGGDRASIL CONTAINING INFO ON MODELS & CONNECTIONS

YAML SPECIFICATION:

INPUT FILE(S) TO YGGDRASIL CONTAINING INFO ON MODELS & CONNECTIONS

```yaml
model:
  name: GrowthModel
  language: python
  args: ./src/growth.py
  inputs:
    - light
  outputs:
    - growth_rate
```

YAML SPECIFICATION:

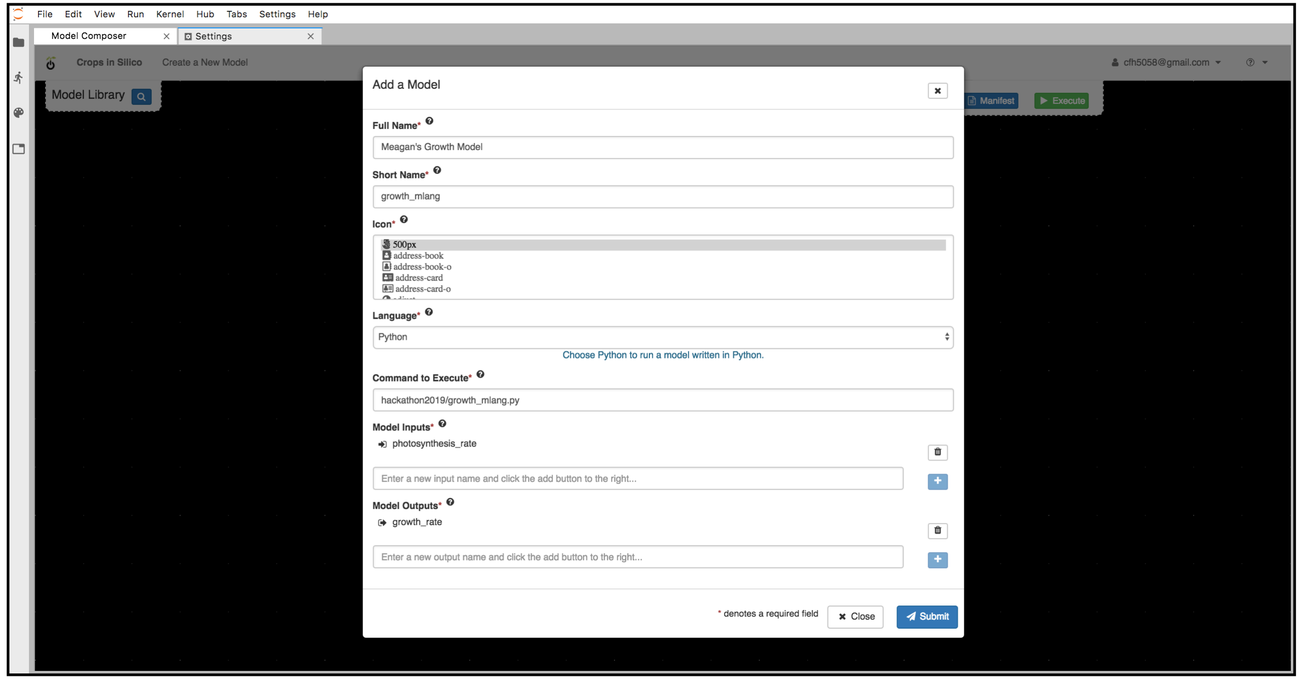
INPUT FILE(S) TO YGGDRASIL CONTAINING INFO ON MODELS & CONNECTIONS

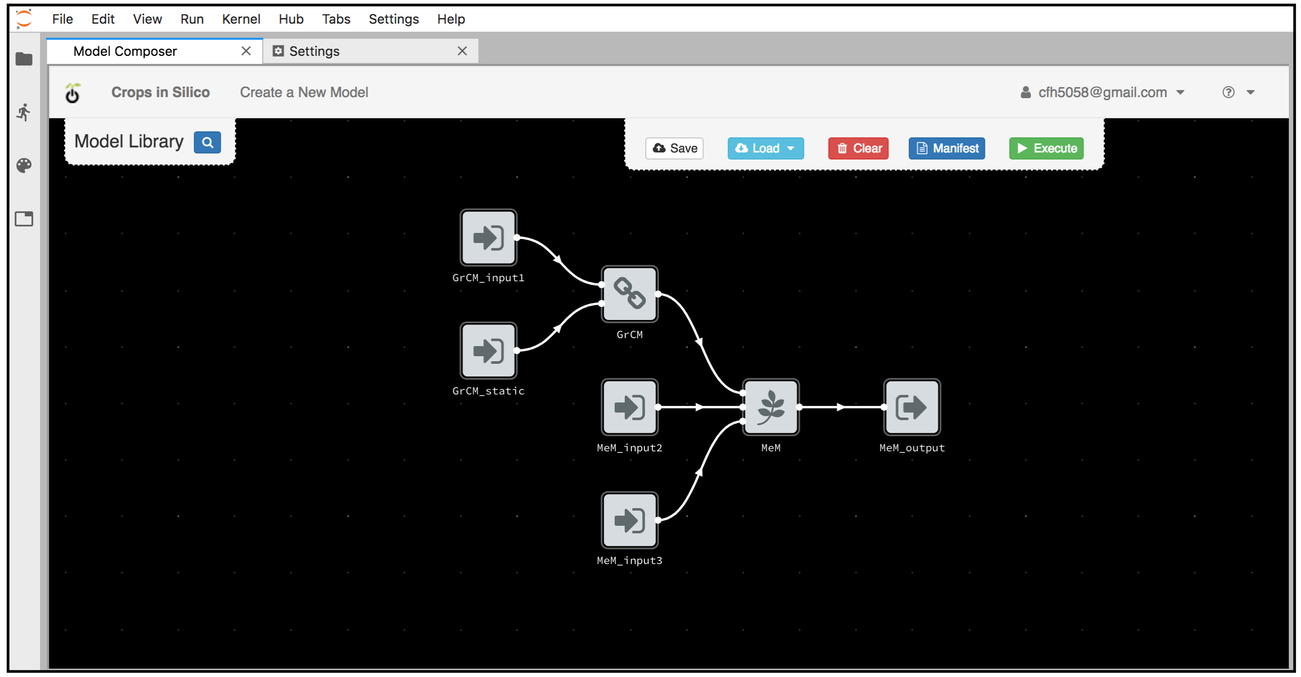
```yaml
model:
  name: GrowthModel
  language: python
  args: ./src/growth.py
  inputs:
    - light
  outputs:
    - growth_rate

connections:
  - input: LightModel:light
    output: GrowthModel:light
  - input: growth_rate
    output: ./Output/growth.txt
    filetype: table
    field_names: growth_rate
```

YAML SPECIFICATION:

INPUT FILE(S) TO YGGDRASIL CONTAINING INFO ON MODELS & CONNECTIONS

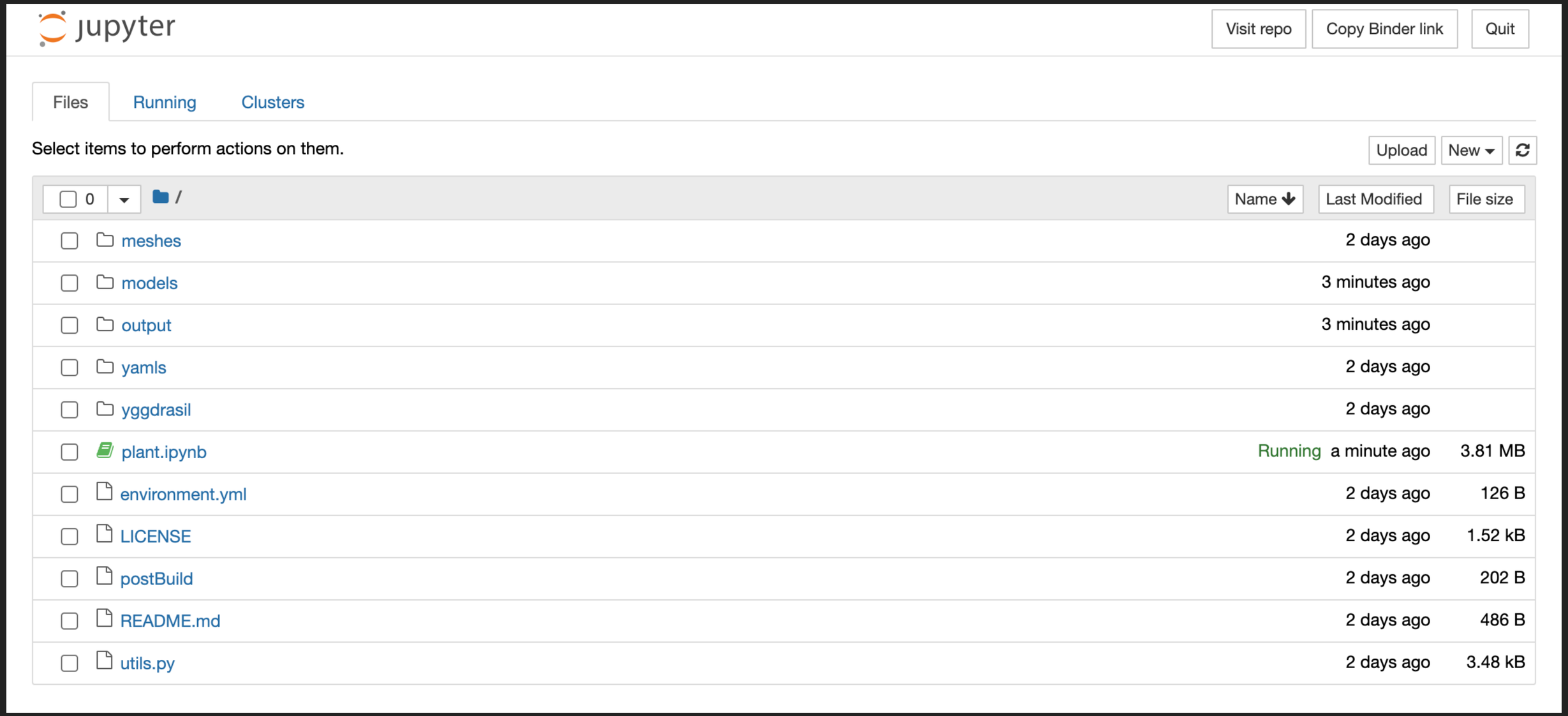


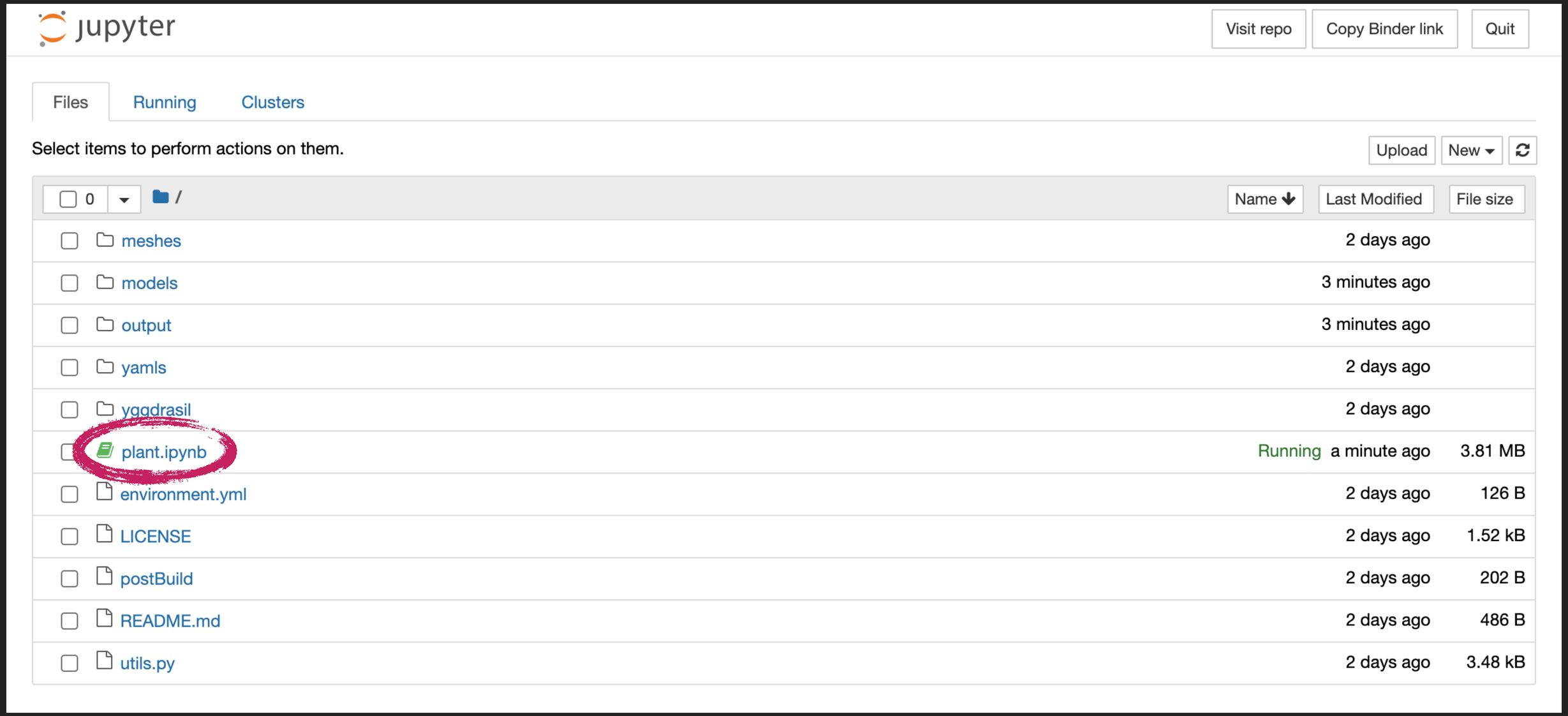


GUI:

GRAPHICAL USER INTERFACE IN DEVELOPMENT FOR YGGDRASIL CAN GENERATE THE YAML

NOTEBOOK TIME!


jupyter
Visit repo
Copy Binder link
Quit
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
0
/
Name
Last Modified
File size
meshes 2 days ago
models 3 minutes ago
output 3 minutes ago
yamls 2 days ago
yggdrasil 2 days ago
plant.ipynb Running a minute ago 3.81 MB
environment.yml 2 days ago 126 B
LICENSE 2 days ago 1.52 kB
postBuild 2 days ago 202 B
README.md 2 days ago 486 B
utils.py 2 days ago 3.48 kB


jupyter
Visit repo
Copy Binder link
Quit
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
0
/
Name
Last Modified
File size
meshes
2 days ago
models
3 minutes ago
output
3 minutes ago
yamls
2 days ago
yggdrasil
2 days ago
plant.ipynb
Running a minute ago
3.81 MB
environment.yml
2 days ago
126 B
LICENSE
2 days ago
1.52 kB
postBuild
2 days ago
202 B
README.md
2 days ago
486 B
utils.py
2 days ago
3.48 kB

Introduction

(NOTE: This notebook is intended for use with the slides found here).

This is a Jupyter notebook. It allows us to run code (in this case Python) alongside text in different "cells". This cell is a markdown cell that can display text and html, the next cell is a code cell.

In the code cells (prefixed by `In [ ]:`), you can assign variables, perform calculations or call external functions/classes. You can run code cells by selecting the cell (so that a blue or green box appears around it) and then clicking the run button or holding `Shift+Enter`. Then a number will appear inside the brackets indicating the order of when the cell was executed.

Output from the cell will be displayed below it with the `Out[#]:` prefix where the number in the brackets indicates the input cell that generated it.

In [ ]:
```
x = 1
y = 3
z = (x + y)**3
z
```

The cell below imports external Python code for use here (specifically a package `trimesh` for loading and displaying 3D meshes in the notebook and some utilities for viewing file contents).

jupyter plant (unsaved changes)

Visit repo | Copy Binder link

File Edit View Insert Cell Kernel Widgets Help | Not Trusted | Python 3

Run | Markdown | Download | GitHub | Binder

Memory: 117 / 2048 MB

TEXT CELL

Introduction

(NOTE: This notebook is intended for use with the slides found here).

This is a Jupyter notebook. It allows us to run code (in this case Python) alongside text in different "cells". This cell is a markdown cell that can display text and html, the next cell is a code cell.

In the code cells (prefixed by `In [ ]:`), you can assign variables, perform calculations or call external functions/classes. You can run code cells by selecting the cell (so that a blue or green box appears around it) and then clicking the run button or holding `Shift+Enter`. Then a number will appear inside the brackets indicating the order of when the cell was executed.

Output from the cell will be displayed below it with the `Out[#]:` prefix where the number in the brackets indicates the input cell that generated it.

In [ ]:
```
x = 1
y = 3
z = (x + y)**3
z
```

The cell below imports external Python code for use here (specifically a package `trimesh` for loading and displaying 3D meshes in the notebook and some utilities for viewing file contents).

jupyter plant (unsaved changes)

Visit repo | Copy Binder link

File Edit View Insert Cell Kernel Widgets Help | Not Trusted | Python 3

Run | Markdown | Download | GitHub | Binder

Memory: 117 / 2048 MB

Introduction

(NOTE: This notebook is intended for use with the slides found here).

This is a Jupyter notebook. It allows us to run code (in this case Python) alongside text in different "cells". This cell is a markdown cell that can display text and html, the next cell is a code cell.

In the code cells (prefixed by `In [ ]:`), you can assign variables, perform calculations or call external functions/classes. You can run code cells by selecting the cell (so that a blue or green box appears around it) and then clicking the run button or holding `Shift+Enter`. Then a number will appear inside the brackets indicating the order of when the cell was executed.

Output from the cell will be displayed below it with the `Out[#]:` prefix where the number in the brackets indicates the input cell that generated it.

CODE CELL

In [ ]:
```
x = 1
y = 3
z = (x + y)**3
z
```

The cell below imports external Python code for use here (specifically a package `trimesh` for loading and displaying 3D meshes in the notebook and some utilities for viewing file contents).

NOTEBOOK INTRO

Code cells can contain any valid Python code

Run cells by holding shift and pressing enter

(shift + enter)

In [ ]:
```
x = 1
y = 3
z = (x + y)**3
z
```

Code cells can contain any valid Python code

Run cells by holding shift and pressing enter

(shift + enter)

In [1]:
```
x = 1
y = 3
z = (x + y)**3
z
```

Out[1]: 64

Code cells can contain any valid Python code

Run cells by holding shift and pressing enter

(shift + enter)

In [1]:
```
x = 1
y = 3
z = (x + y)**3
z
```

Out[1]: 64

Output appears below

Number in bracket is the order of execution

("*" indicates the cell is still running)

We need some tools!

trimesh - package for loading/displaying meshes in the notebook
utils - various helper scripts in this demo's repo for displaying output
yggdrasil - the method for running yggdrasil integration

In [ ]:
```
import trimesh
from utils import *
from yggdrasil.runner import run
```

We need some tools!

trimesh - package for loading/displaying meshes in the notebook
utils - various helper scripts in this demo's repo for displaying output
yggdrasil - the method for running yggdrasil integration

In [2]:
```
import trimesh
from utils import *
from yggdrasil.runner import run
```

We need some tools!

trimesh - package for loading/displaying meshes in the notebook
utils - various helper scripts in this demo's repo for displaying output
yggdrasil - the method for running yggdrasil integration

In [2]:
```
import trimesh
from utils import *
from yggdrasil.runner import run
```

No output, so nothing appears below

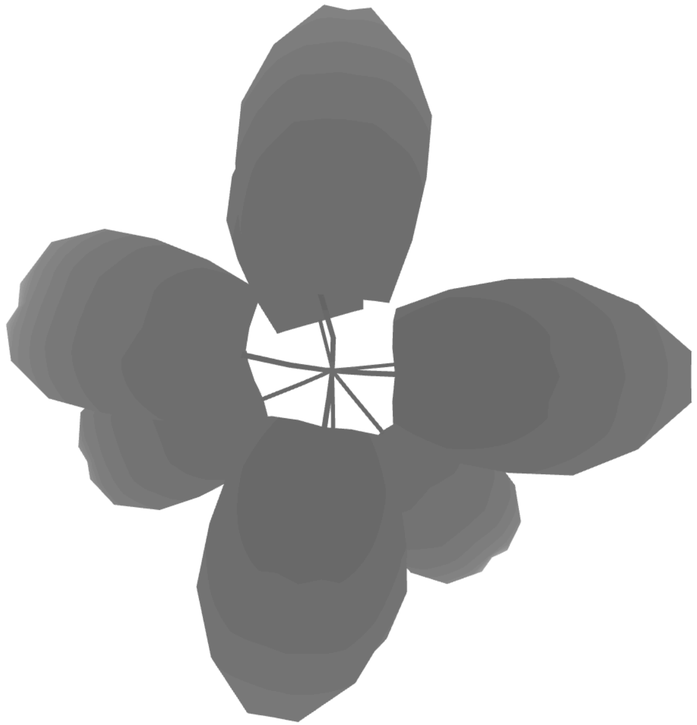
Lets load & display some 3D mesh data

In [ ]:
```
fname = 'meshes/plants-2.obj'
mesh = trimesh.load_mesh(fname)
mesh.show()
```

Lets load & display some 3D mesh data

In [3]:
```
fname = 'meshes/plants-2.obj'
mesh = trimesh.load_mesh(fname)
mesh.show()
```

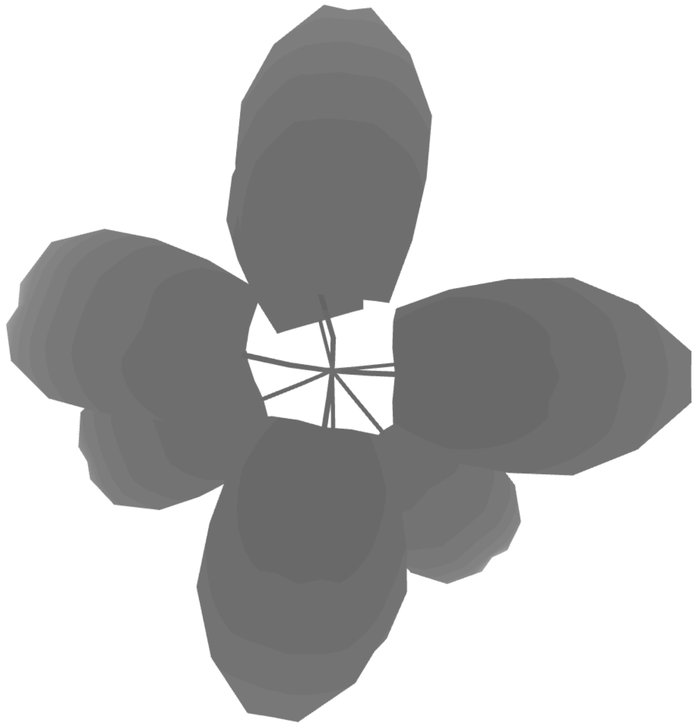
Out[3]:



Lets load & display some 3D mesh data

In [3]:
```
fname = 'meshes/plants-2.obj'
mesh = trimesh.load_mesh(fname)
mesh.show()
```

Out[3]:



Click and drag
to move mesh

TOY PLANT MODEL

In [ ]:

```
from models import plant_v0
print_python_source(plant_v0.run)
```

In [4]:
```
from models import plant_v0
print_python_source(plant_v0.run)
```

```
def run(mesh, tmin, tmax, tstep):
    mass = 2000.0
    t = tmin
    i = 0

    while t <= tmax:

        # Grow mesh
        # (pretend this is a biologically complex calculation)
        scale = mass * t / 3.5e5
        mesh.vertices[:, 2] += mesh.vertices[:, 2] * scale

        # Save mesh for this timestep
        filename_mesh = os.path.join(_dir, f'../output/mesh_{i:03d}.obj')
        with open(filename_mesh, 'w') as fd:
            mesh.export(fd, 'obj')

        # Advance time step
        t += tstep
        i += 1

    return mesh
```

In [ ]:

```
mesh = plant_v0.run(mesh, 0, 28, 1)
mesh.show()
```

In [5]:
```
mesh = plant_v0.run(mesh, 0, 28, 1)
mesh.show()
```

Out[5]:

In [ ]:
```
print_yaml('yamls/plant_v0.yml')
```

In [6]:
```
print_yaml('yamls/plant_v0.yml')
```

```
1: model:
2:   name: plant
3:   language: python
4:   args: [../models/plant_v0.py, 0.0, 48.0, 6.0]
```

```
In [6]: print_yaml('yamls/plant_v0.yml')
```

```
1: model:
2:     name: plant
3:     language: python
4:     args: [../models/plant_v0.py, 0.0, 48.0, 6.0]
```

YAMLs provide info needed to run model

name - name used by other models to identify the model
language - programming language of the model
args - arguments required to run the model
(including the path to the source code)

In [ ]:

```
run('yamls/plant_v0.yml')
display_last_timestep()
```

In [7]:
```
run('yamls/plant_v0.yml')
display_last_timestep()
```

```
YggRunner(runner): Starting I/O drivers and models on system Meagan
YggRunner(runner): plant finished running.
YggRunner(runner): plant finished exiting.
YggRunner(runner): All models completed
YggRunner(runner):                  init 0.000000
YggRunner(runner):          load drivers 0.321367
YggRunner(runner):         start drivers 0.024131
YggRunner(runner):             run models 1.076655
YggRunner(runner):        close channels 0.000083
YggRunner(runner):               clean up 0.000533
YggRunner(runner): ========================================
YggRunner(runner):                 Total 1.422769
```

In [7]:
```
run('yamls/plant_v0.yml')
display_last_timestep()
```

Out[7]:

CALLING MODELS AS FUNCTIONS

In [ ]:

```
from models import light
print_python_source(light.light)
```

In [8]:
```
from models import light
print_python_source(light.light)
```

```
1: def light(height, time):
2:     intensity = (
3:         80.0 *
4:         (1.0 + np.sin(2.0 * np.pi * time / units.add_units(365.0, 'days'))) /
5:         (np.abs(200 - height)**2))
6:     return intensity
```

In [ ]:
```
print_yaml('yamls/light.yml')
```

In [9]:
```
print_yaml('yamls/light.yml')
```

```
1: model:
2:    name: light
3:    language: python
4:    args: ../models/light.py
5:    function: light
6:    is_server: True
```

```
In [9]: print_yaml('yamls/light.yml')
```

```
1: model:
2:   name: light
3:   language: python
4:   args: ../models/light.py
5:   function: light
6:   is_server: True
```

YAMLs provide info needed to run model

function - name of the function that yggdrasil should wrap
is_server - specify that the model should be treated as a server that other models can call

```
In [9]: print_yaml('yamls/light.yml')
```

```
1: model:
2:   name: light
3:   language: python
4:   args: ../models/light.py
5:   function: light
6:   is_server: True
```

YAMLs provide info needed to run model

function - name of the function that yggdrasil should wrap
is_server - specify that the model should be treated as a server that other models can call

NO MODIFICATION TO LIGHT MODEL SOURCE CODE NECESSARY

In [ ]:
```
print_yaml_diff('yamls/plant_v0.yml', 'yamls/plant_v1.yml')
```

In [10]:
```
print_yaml_diff('yamls/plant_v0.yml', 'yamls/plant_v1.yml')
```

```
 1:   model:
 2:     name: plant
 3:     language: python
    -   args: [../models/plant_v0.py, 0.0, 48.0, 6.0]
    ?                                                 ^

 4: +   args: [../models/plant_v1.py, 0.0, 48.0, 6.0]
    ?                                                 ^

 5: +   client_of: light
 6: +   outputs:
 7: +     - name: light
 8: +       default_file:
 9: +         name: ../output/light_%03d.pkl
10: +         filetype: pickle
11: +         is_series: True
```

```
In [10]: print_yaml_diff('yamls/plant_v0.yml', 'yamls/plant_v1.yml')
```

```
 1:   model:
 2:     name: plant
 3:     language: python
    -   args: [../models/plant_v0.py, 0.0, 48.0, 6.0]
    ?                                                 ^

 4: +   args: [../models/plant_v1.py, 0.0, 48.0, 6.0]
    ?                                                 ^

 5: +   client_of: light
 6: +   outputs:
 7: +     - name: light
 8: +       default_file:
 9: +         name: ../output/light_%03d.pkl
10: +         filetype: pickle
11: +         is_series: True
```

client_of - specify that the server model that this model should call as a client

```
In [10]: print_yaml_diff('yamls/plant_v0.yml', 'yamls/plant_v1.yml')
```

```
 1:   model:
 2:     name: plant
 3:     language: python
    -   args: [../models/plant_v0.py, 0.0, 48.0, 6.0]
    ?                                                 ^

 4: +   args: [../models/plant_v1.py, 0.0, 48.0, 6.0]
    ?                                                 ^

 5: +   client_of: light
 6: +   outputs:
 7: +     - name: light
 8: +       default_file:
 9: +         name: ../output/light_%03d.pkl
10: +         filetype: pickle
11: +         is_series: True
```

outputs - description of output channels the model uses

default_file - description of file where output should be sent if the channel isn't connected to an input channel

In [ ]:

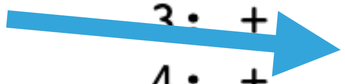
```
from models import plant_v1
print_python_source_diff(plant_v0.run, plant_v1.run)
```

In [11]:
```
from models import plant_v1
print_python_source_diff(plant_v0.run, plant_v1.run)
```

```
 1:    def run(mesh, tmin, tmax, tstep):
 2:        mass = 2000.0
 3: +      light_rpc = YggRpcClient('light_plant')
 4: +      light_out = YggOutput('light')
 5: +      mass = units.add_units(mass, 'g')
 6: +      tmin = units.add_units(tmin, 'hrs')
 7: +      tmax = units.add_units(tmax, 'hrs')
 8: +      tstep = units.add_units(tstep, 'hrs')
 9:        t = tmin
10:        i = 0
```

```
In [11]: from models import plant_v1
         print_python_source_diff(plant_v0.run, plant_v1.run)
```

Interface classes



```
 1:    def run(mesh, tmin, tmax, tstep):
 2:        mass = 2000.0
 3: +      light_rpc = YggRpcClient('light_plant')
 4: +      light_out = YggOutput('light')
 5: +      mass = units.add_units(mass, 'g')
 6: +      tmin = units.add_units(tmin, 'hrs')
 7: +      tmax = units.add_units(tmax, 'hrs')
 8: +      tstep = units.add_units(tstep, 'hrs')
 9:        t = tmin
10:        i = 0
```

```
In [11]: from models import plant_v1
         print_python_source_diff(plant_v0.run, plant_v1.run)
```

Interface classes

"server model name"_"client model name"

```
 1:    def run(mesh, tmin, tmax, tstep):
 2:        mass = 2000.0
 3: +      light_rpc = YggRpcClient('light_plant')
 4: +      light_out = YggOutput('light')
 5: +      mass = units.add_units(mass, 'g')
 6: +      tmin = units.add_units(tmin, 'hrs')
 7: +      tmax = units.add_units(tmax, 'hrs')
 8: +      tstep = units.add_units(tstep, 'hrs')
 9:        t = tmin
10:        i = 0
```

```
In [11]: from models import plant_v1
         print_python_source_diff(plant_v0.run, plant_v1.run)
```

```
 1:    def run(mesh, tmin, tmax, tstep):
 2:        mass = 2000.0
 3: +      light_rpc = YggRpcClient('light_plant')
 4: +      light_out = YggOutput('light')
 5: +      mass = units.add_units(mass, 'g')
 6: +      tmin = units.add_units(tmin, 'hrs')
 7: +      tmax = units.add_units(tmax, 'hrs')
 8: +      tstep = units.add_units(tstep, 'hrs')
 9:        t = tmin
10:        i = 0
```

Interface classes

"server model name"_"client model name"

output channel in yaml

In [11]:
```
from models import plant_v1
print_python_source_diff(plant_v0.run, plant_v1.run)
```

```
 1:    def run(mesh, tmin, tmax, tstep):
 2:        mass = 2000.0
 3: +      light_rpc = YggRpcClient('light_plant')
 4: +      light_out = YggOutput('light')
 5: +      mass = units.add_units(mass, 'g')
 6: +      tmin = units.add_units(tmin, 'hrs')
 7: +      tmax = units.add_units(tmax, 'hrs')
 8: +      tstep = units.add_units(tstep, 'hrs')
 9:        t = tmin
10:        i = 0
```

Add units to variables

In [11]:
```
from models import plant_v1
print_python_source_diff(plant_v0.run, plant_v1.run)
```

```
 1:    def run(mesh, tmin, tmax, tstep):
 2:        mass = 2000.0
 3: +      light_rpc = YggRpcClient('light_plant')
 4: +      light_out = YggOutput('light')
 5: +      mass = units.add_units(mass, 'g')
 6: +      tmin = units.add_units(tmin, 'hrs')
 7: +      tmax = units.add_units(tmax, 'hrs')
 8: +      tstep = units.add_units(tstep, 'hrs')
 9:        t = tmin
10:        i = 0
11:
12:        while t <= tmax:
13:
14: +          # Get light data by calling light model
15: +          flag, light = light_rpc.call(mesh.vertices[:, 2], t)
16: +          if not flag:
17: +              raise Exception("Error calling light model")
18: +
```

```
In [11]: from models import plant_v1
         print_python_source_diff(plant_v0.run, plant_v1.run)
```

```
 1:    def run(mesh, tmin, tmax, tstep):
 2:        mass = 2000.0
 3: +      light_rpc = YggRpcClient('light_plant')
 4: +      light_out = YggOutput('light')
 5: +      mass = units.add_units(mass, 'g')
 6: +      tmin = units.add_units(tmin, 'hrs')
 7: +      tmax = units.add_units(tmax, 'hrs')
 8: +      tstep = units.add_units(tstep, 'hrs')
 9:        t = tmin
10:        i = 0
11:
12:        while t <= tmax:
13:
14: +          # Get light data by calling light model
15: +          flag, light = light_rpc.call(mesh.vertices[:, 2], t)
16: +          if not flag:
17: +              raise Exception("Error calling light model")
18: +
```

Interface
"call"
(= send + receive)

```
In [11]: from models import plant_v1
         print_python_source_diff(plant_v0.run, plant_v1.run)
```

```
 1:    def run(mesh, tmin, tmax, tstep):
 2:        mass = 2000.0
 3: +      light_rpc = YggRpcClient('light_plant')
 4: +      light_out = YggOutput('light')
 5: +      mass = units.add_units(mass, 'g')
 6: +      tmin = units.add_units(tmin, 'hrs')
 7: +      tmax = units.add_units(tmax, 'hrs')
 8: +      tstep = units.add_units(tstep, 'hrs')
 9:        t = tmin
10:        i = 0
11:
12:        while t <= tmax:
13:
14: +          # Get light data by calling light model
15: +          flag, light = light_rpc.call(mesh.vertices[:, 2], t)
16: +          if not flag:
17: +              raise Exception("Error calling light model")
18: +
```

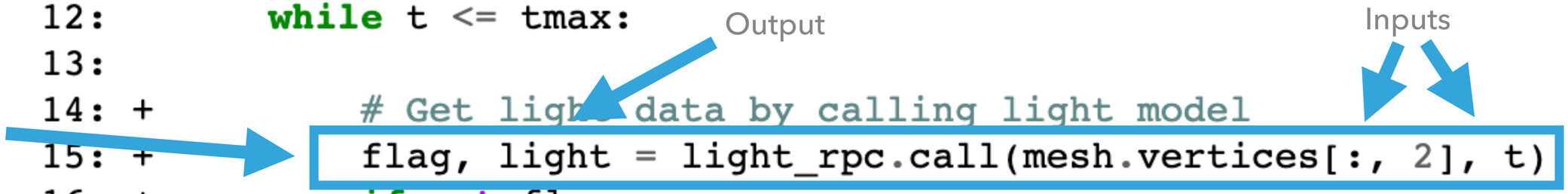
Interface
"call"
(= send + receive)

Inputs

In [11]:
```
from models import plant_v1
print_python_source_diff(plant_v0.run, plant_v1.run)
```

```
 1:    def run(mesh, tmin, tmax, tstep):
 2:        mass = 2000.0
 3: +      light_rpc = YggRpcClient('light_plant')
 4: +      light_out = YggOutput('light')
 5: +      mass = units.add_units(mass, 'g')
 6: +      tmin = units.add_units(tmin, 'hrs')
 7: +      tmax = units.add_units(tmax, 'hrs')
 8: +      tstep = units.add_units(tstep, 'hrs')
 9:        t = tmin
10:        i = 0
11:
12:        while t <= tmax:
13:
14: +          # Get light data by calling light model
15: +          flag, light = light_rpc.call(mesh.vertices[:, 2], t)
16: +          if not flag:
17: +              raise Exception("Error calling light model")
18: +
```

Interface
"call"
(= send + receive)



In [11]:
```
from models import plant_v1
print_python_source_diff(plant_v0.run, plant_v1.run)
```

```
19:           # Grow mesh
20:           # (pretend this is a biologically complex calculation)
    -         scale = mass * t / 3.5e5
21: +         scale = units.get_data(mass * light / units.add_units(1.0, 'kg'))
22:           mesh.vertices[:, 2] += mesh.vertices[:, 2] * scale
```

```
In [11]: from models import plant_v1
         print_python_source_diff(plant_v0.run, plant_v1.run)
```

Use the light data in calc

```
19:             # Grow mesh
20:             # (pretend this is a biologically complex calculation)
    -           scale = mass * t / 3.5e5
21: +           scale = units.get_data(mass * light / units.add_units(1.0, 'kg'))
22:             mesh.vertices[:, 2] += mesh.vertices[:, 2] * scale
```

In [11]:
```
from models import plant_v1
print_python_source_diff(plant_v0.run, plant_v1.run)
```

```
19:           # Grow mesh
20:           # (pretend this is a biologically complex calculation)
    -         scale = mass * t / 3.5e5
21: +         scale = units.get_data(mass * light / units.add_units(1.0, 'kg'))
22:           mesh.vertices[:, 2] += mesh.vertices[:, 2] * scale

                                      ...

29: +         # Send light to output
30: +         flag = light_out.send(light)
31: +         if not flag:
32: +             raise Exception("Error sending light to output")
33: +
```

```
In [11]: from models import plant_v1
         print_python_source_diff(plant_v0.run, plant_v1.run)
```

```
19:           # Grow mesh
20:           # (pretend this is a biologically complex calculation)
    -         scale = mass * t / 3.5e5
21: +         scale = units.get_data(mass * light / units.add_units(1.0, 'kg'))
22:           mesh.vertices[:, 2] += mesh.vertices[:, 2] * scale
```

...

Send light data to output comm

```
29: +         # Send light to output
30: +         flag = light_out.send(light)
31: +         if not flag:
32: +             raise Exception("Error sending light to output")
33: +
```

In [ ]:
```
run(['yamls/plant_v1.yml', 'yamls/light.yml'], production_run=True)
```

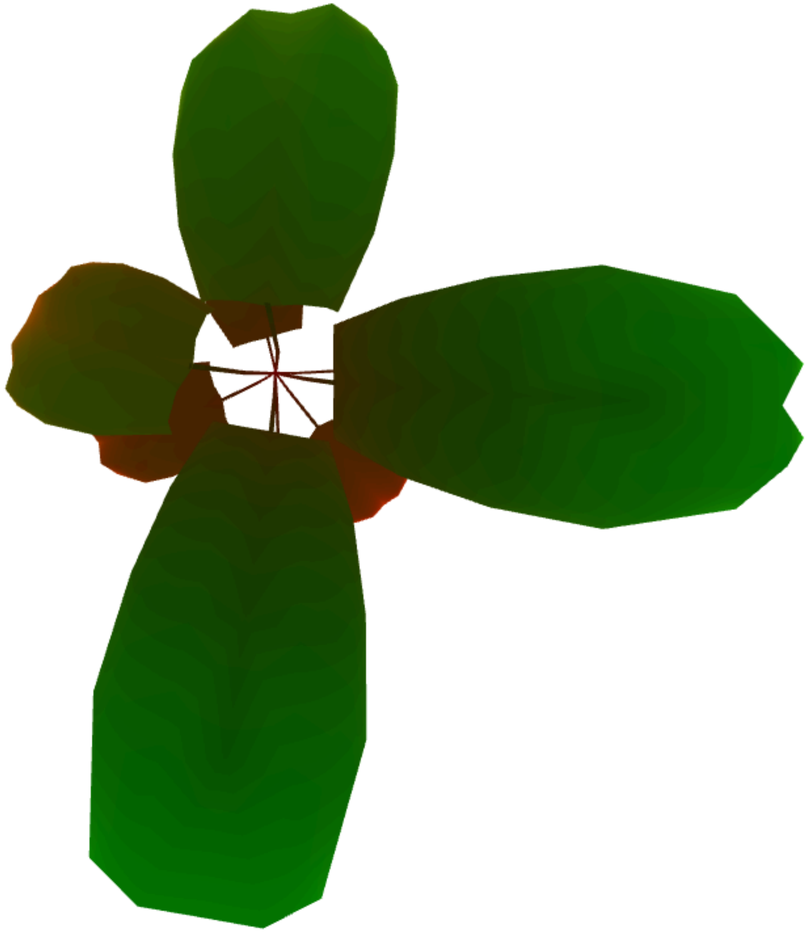
In [12]:
```
run(['yamls/plant_v1.yml', 'yamls/light.yml'], production_run=True)
```

```
YggRunner(runner): Starting I/O drivers and models on system Meagans-Air
YggRunner(runner): plant finished running.
End of input from temp_height.
YggRunner(runner): plant finished exiting.
YggRunner(runner): light finished running.
YggRunner(runner): light finished exiting.
YggRunner(runner): All models completed
YggRunner(runner):                  init 0.000000
YggRunner(runner):         load drivers 0.071073
YggRunner(runner):        start drivers 0.106543
YggRunner(runner):           run models 15.581171
YggRunner(runner):       close channels 0.000185
YggRunner(runner):              clean up 0.034680
YggRunner(runner): ========================================
YggRunner(runner):                 Total 15.793652
```

In [ ]:
```
display_last_timestep(with_light=True)
```

In [13]:
```
display_last_timestep(with_light=True)
```

Out[13]:



We can easily swap out the light model for another
(e.g. one written in another language)

C++ Model

In [ ]:
```
print_yaml('yamls/light_cpp.yml')
print_source('models/light.cpp')
run(['yamls/plant_v1.yml', 'yamls/light_cpp.yml'], production_run=True)
display_last_timestep(with_light=True)
```

R Model

In [ ]:
```
print_yaml('yamls/light_R.yml')
print_source('models/light.R')
run(['yamls/plant_v1.yml', 'yamls/light_R.yml'], production_run=True)
display_last_timestep(with_light=True)
```

Fortran Model

In [ ]:
```
print_yaml('yamls/light_fortran.yml')
print_source('models/light.f90')
run(['yamls/plant_v1.yml', 'yamls/light_fortran.yml'], production_run=True)
display_last_timestep(with_light=True)
```

TIME STEP SYNCHRONIZATION

In [ ]:

```
from models import roots_v0
print_python_source(roots_v0.run)
```

In [14]:
```
from models import roots_v0
print_python_source(roots_v0.run)
```

```
 1: def run(tmin, tmax, tstep):
 2:     t = tmin
 3:     mass = 0
 4:
 5:     while t <= tmax:
 6:
 7:         # Calculate mass for the time step
 8:         # (pretend this is a biologically complex calculation)
 9:         mass += t * 0.2
10:
11:         # Advance time step
12:         t += tstep
13:
14:     return mass
```

In [ ]:
```
print_yaml('yamls/roots.yml')
```

In [15]:
```
print_yaml('yamls/roots.yml')
```

```
 1: models:
 2:   - name: roots
 3:     language: python
 4:     args: [../models/roots_v1.py, 0.0, 2.0, 0.5]
 5:     timesync: plant2root
 6:     outputs:
 7:       - name: mass
 8:         default_file:
 9:           name: ../output/mass.txt
10:           filetype: table
11:   - name: plant2root
12:     language: timesync
13:     aggregation: sum
```

```
In [15]: print_yaml('yamls/roots.yml')
```

```
 1: models:
 2:   - name: roots
 3:     language: python
 4:     args: [../models/roots_v1.py, 0.0, 2.0, 0.5]
 5:     timesync: plant2root
 6:     outputs:
 7:       - name: mass
 8:         default_file:
 9:           name: ../output/mass.txt
10:           filetype: table
11:   - name: plant2root
12:     language: timesync
13:     aggregation: sum
```

Name of dummy timesync “model”

timesync - name of the time-step synchronization dummy “model”

aggregation - method used to combine values from models being synchronized

```
In [15]: print_yaml('yamls/roots.yml')
```

```
 1: models:
 2:   - name: roots
 3:     language: python
 4:     args: [../models/roots_v1.py, 0.0, 2.0, 0.5]
 5:     timesync: plant2root
 6:     outputs:
 7:       - name: mass
 8:         default_file:
 9:           name: ../output/mass.txt
10:           filetype: table
11:   - name: plant2root
12:     language: timesync
13:     aggregation: sum
```

Output mass to default file

timesync - name of the time-step synchronization dummy “model”

aggregation - method used to combine values from models being synchronized

```
In [15]: print_yaml('yamls/roots.yml')
```

```
 1: models:
 2:   - name: roots
 3:     language: python
 4:     args: [../models/roots_v1.py, 0.0, 2.0, 0.5]
 5:     timesync: plant2root
 6:     outputs:
 7:       - name: mass
 8:         default_file:
 9:           name: ../output/mass.txt
10:           filetype: table
11:   - name: plant2root
12:     language: timesync
13:     aggregation: sum
```

Dummy timesync "model"

timesync - name of the time-step synchronization dummy "model"

aggregation - method used to combine values from models being synchronized

In [ ]:
```
from models import roots_v1
print_python_source_diff(roots_v0.run, roots_v1.run)
```

In [16]:
```
from models import roots_v1
print_python_source_diff(roots_v0.run, roots_v1.run)
```

```
1:    def run(tmin, tmax, tstep):
2: +      mass_out = YggOutput('mass')
3: +      plant2root = YggTimesync('plant2root')
4: +      tmin = units.add_units(tmin, 'days')
5: +      tmax = units.add_units(tmax, 'days')
6: +      tstep = units.add_units(tstep, 'days')
7:        t = tmin
8:        mass = 0
```

In [16]:
```
from models import roots_v1
print_python_source_diff(roots_v0.run, roots_v1.run)
```

Interface classes

```
1:    def run(tmin, tmax, tstep):
2: +      mass_out = YggOutput('mass')
3: +      plant2root = YggTimesync('plant2root')
4: +      tmin = units.add_units(tmin, 'days')
5: +      tmax = units.add_units(tmax, 'days')
6: +      tstep = units.add_units(tstep, 'days')
7:        t = tmin
8:        mass = 0
```

```
In [16]: from models import roots_v1
         print_python_source_diff(roots_v0.run, roots_v1.run)
```

Interface classes

Output channel from yaml

```
1:    def run(tmin, tmax, tstep):
2: +      mass_out = YggOutput('mass')
3: +      plant2root = YggTimesync('plant2root')
4: +      tmin = units.add_units(tmin, 'days')
5: +      tmax = units.add_units(tmax, 'days')
6: +      tstep = units.add_units(tstep, 'days')
7:        t = tmin
8:        mass = 0
```

```
In [16]: from models import roots_v1
         print_python_source_diff(roots_v0.run, roots_v1.run)
```

Interface classes

Output channel from yaml

timesync “model” name

```
1:    def run(tmin, tmax, tstep):
2: +      mass_out = YggOutput('mass')
3: +      plant2root = YggTimesync('plant2root')
4: +      tmin = units.add_units(tmin, 'days')
5: +      tmax = units.add_units(tmax, 'days')
6: +      tstep = units.add_units(tstep, 'days')
7:        t = tmin
8:        mass = 0
```

```
In [16]: from models import roots_v1
         print_python_source_diff(roots_v0.run, roots_v1.run)
```

```
1:    def run(tmin, tmax, tstep):
2: +      mass_out = YggOutput('mass')
3: +      plant2root = YggTimesync('plant2root')
4: +      tmin = units.add_units(tmin, 'days')
5: +      tmax = units.add_units(tmax, 'days')
6: +      tstep = units.add_units(tstep, 'days')
7:        t = tmin
8:        mass = 0
```

Add units to variables

In [16]:
```
from models import roots_v1
print_python_source_diff(roots_v0.run, roots_v1.run)
```

```
 1:    def run(tmin, tmax, tstep):
 2: +      mass_out = YggOutput('mass')
 3: +      plant2root = YggTimesync('plant2root')
 4: +      tmin = units.add_units(tmin, 'days')
 5: +      tmax = units.add_units(tmax, 'days')
 6: +      tstep = units.add_units(tstep, 'days')
 7:        t = tmin
 8:        mass = 0

 ...

14: +          mass += t * units.add_units(0.2, 'kg/days')
15: +
16: +          # Synchronize data for time step with the root model
17: +          plant_state = {'mass': mass}
18: +          flag, plant_state = plant2root.call(t, plant_state)
19: +          if not flag:
20: +              raise Exception("Error performing time-step synchr
21: +                              "with plant model.")
```

In [16]:
```
from models import roots_v1
print_python_source_diff(roots_v0.run, roots_v1.run)
```

```
1:     def run(tmin, tmax, tstep):
2: +       mass_out = YggOutput('mass')
3: +       plant2root = YggTimesync('plant2root')
4: +       tmin = units.add_units(tmin, 'days')
5: +       tmax = units.add_units(tmax, 'days')
6: +       tstep = units.add_units(tstep, 'days')
7:         t = tmin
8:         mass = 0

...

14: +           mass += t * units.add_units(0.2, 'kg/days')
15: +
16: +           # Synchronize data for time step with the root model
17: +           plant_state = {'mass': mass}
18: +           flag, plant_state = plant2root.call(t, plant_state)
19: +           if not flag:
20: +               raise Exception("Error performing time-step synchr
21: +                               "with plant model.")
```

Interface "call"
(= send + receive)

In [16]:
```
from models import roots_v1
print_python_source_diff(roots_v0.run, roots_v1.run)
```

```
 1:    def run(tmin, tmax, tstep):
 2: +      mass_out = YggOutput('mass')
 3: +      plant2root = YggTimesync('plant2root')
 4: +      tmin = units.add_units(tmin, 'days')
 5: +      tmax = units.add_units(tmax, 'days')
 6: +      tstep = units.add_units(tstep, 'days')
 7:        t = tmin
 8:        mass = 0

 ...

14: +          mass += t * units.add_units(0.2, 'kg/days')
15: +
16: +          # Synchronize data for time step with the root model
17: +          plant_state = {'mass': mass}
18: +          flag, plant_state = plant2root.call(t, plant_state)
19: +          if not flag:
20: +              raise Exception("Error performing time-step synchr
21: +                              "with plant model.")
```

Interface "call"
(= send + receive)

Inputs

In [16]:
```
from models import roots_v1
print_python_source_diff(roots_v0.run, roots_v1.run)
```

```
 1:    def run(tmin, tmax, tstep):
 2: +      mass_out = YggOutput('mass')
 3: +      plant2root = YggTimesync('plant2root')
 4: +      tmin = units.add_units(tmin, 'days')
 5: +      tmax = units.add_units(tmax, 'days')
 6: +      tstep = units.add_units(tstep, 'days')
 7:        t = tmin
 8:        mass = 0

 ...

14: +          mass += t * units.add_units(0.2, 'kg/days')
15: +
16: +          # Synchronize data for time step with the root model
17: +          plant_state = {'mass': mass}
18: +          flag, plant_state = plant2root.call(t, plant_state)
19: +          if not flag:
20: +              raise Exception("Error performing time-step synchr
21: +                              "with plant model.")
```

Interface "call" (= send + receive)

Output

Inputs

In [16]:
```
from models import roots_v1
print_python_source_diff(roots_v0.run, roots_v1.run)
```

```
23: +        # Send masses to output
24: +        flag = mass_out.send({'time': t,
25: +                              'total_mass': plant_state['mass'],
26: +                              'root_mass': mass,
27: +                              'plant_mass': plant_state['mass'] - mass})
28: +        if not flag:
29: +            raise Exception("Error sending masses to output")
30:
```

```
In [16]: from models import roots_v1
         print_python_source_diff(roots_v0.run, roots_v1.run)
```

Send masses to output

```
23: +        # Send masses to output
24: +        flag = mass_out.send({'time': t,
25: +                              'total_mass': plant_state['mass'],
26: +                              'root_mass': mass,
27: +                              'plant_mass': plant_state['mass'] - mass})
28: +        if not flag:
29: +            raise Exception("Error sending masses to output")
30:
```

In [ ]:
```
print_yaml_diff('yamls/plant_v1.yml', 'yamls/plant_v2.yml')
```

In [17]:
```
print_yaml_diff('yamls/plant_v1.yml', 'yamls/plant_v2.yml')
```

```
 1:   model:
 2:     name: plant
 3:     language: python
    -   args: [../models/plant_v1.py, 0.0, 48.0, 6.0]
    ?                                                ^

 4: +   args: [../models/plant_v2.py, 0.0, 48.0, 6.0]
    ?                                                ^

 5:     client_of: light
 6: +   timesync: plant2root
 7:     outputs:
 8:       - name: light
 9:         default_file:
10:           name: ../output/light_%03d.pkl
11:           filetype: pickle
12:           is_series: True
```

```
In [17]: print_yaml_diff('yamls/plant_v1.yml', 'yamls/plant_v2.yml')
```

```
 1:   model:
 2:     name: plant
 3:     language: python
    -   args: [../models/plant_v1.py, 0.0, 48.0, 6.0]
    ?                                                ^

 4: +   args: [../models/plant_v2.py, 0.0, 48.0, 6.0]
    ?                                                ^

 5:     client_of: light
 6: +   timesync: plant2root
 7:     outputs:
 8:       - name: light
 9:         default_file:
10:           name: ../output/light_%03d.pkl
11:           filetype: pickle
12:           is_series: True
```

timesync - name of the time-step synchronization dummy “model”

In [ ]:

```
from models import plant_v2
print_python_source_diff(plant_v1.run, plant_v2.run)
```

In [18]:
```
from models import plant_v2
print_python_source_diff(plant_v1.run, plant_v2.run)
```

```
 4:         light_out = YggOutput('light')
 5: +       plant2root = YggTimesync('plant2root')
 6:         mass = units.add_units(mass, 'g')
 7:         tmin = units.add_units(tmin, 'hrs')
 8:         tmax = units.add_units(tmax, 'hrs')
 9:         tstep = units.add_units(tstep, 'hrs')
10:         t = tmin
11:         i = 0
```

```
In [18]: from models import plant_v2
         print_python_source_diff(plant_v1.run, plant_v2.run)
```

Timesync interface class

```
 4:         light_out = YggOutput('light')
 5: +       plant2root = YggTimesync('plant2root')
 6:         mass = units.add_units(mass, 'g')
 7:         tmin = units.add_units(tmin, 'hrs')
 8:         tmax = units.add_units(tmax, 'hrs')
 9:         tstep = units.add_units(tstep, 'hrs')
10:         t = tmin
11:         i = 0
```

```
In [18]: from models import plant_v2
         print_python_source_diff(plant_v1.run, plant_v2.run)
```

Timesync interface class

timesync “model" name

```
 4:         light_out = YggOutput('light')
 5: +       plant2root = YggTimesync('plant2root')
 6:         mass = units.add_units(mass, 'g')
 7:         tmin = units.add_units(tmin, 'hrs')
 8:         tmax = units.add_units(tmax, 'hrs')
 9:         tstep = units.add_units(tstep, 'hrs')
10:         t = tmin
11:         i = 0
```

In [18]:
```
from models import plant_v2
print_python_source_diff(plant_v1.run, plant_v2.run)
```

```
 4:         light_out = YggOutput('light')
 5: +       plant2root = YggTimesync('plant2root')
 6:         mass = units.add_units(mass, 'g')
 7:         tmin = units.add_units(tmin, 'hrs')
 8:         tmax = units.add_units(tmax, 'hrs')
 9:         tstep = units.add_units(tstep, 'hrs')
10:         t = tmin
11:         i = 0

  ...

15: +           # Synchronize data for time step with the root model
16: +           root_state = {'mass': mass}
17: +           flag, root_state = plant2root.call(t, root_state)
18: +           if not flag:
19: +               raise Exception("Error performing time-step synchr
20: +                               "with root model.")
21: +           mass = root_state['mass']
22: +
```

```
In [18]: from models import plant_v2
         print_python_source_diff(plant_v1.run, plant_v2.run)
```

```
 4:         light_out = YggOutput('light')
 5: +       plant2root = YggTimesync('plant2root')
 6:         mass = units.add_units(mass, 'g')
 7:         tmin = units.add_units(tmin, 'hrs')
 8:         tmax = units.add_units(tmax, 'hrs')
 9:         tstep = units.add_units(tstep, 'hrs')
10:         t = tmin
11:         i = 0

  ...

15: +           # Synchronize data for time step with the root model
16: +           root_state = {'mass': mass}
17: +           flag, root_state = plant2root.call(t, root_state)
18: +           if not flag:
19: +               raise Exception("Error performing time-step synchr
20: +                               "with root model.")
21: +           mass = root_state['mass']
22: +
```

Interface "call" (= send + receive)

In [18]:
```
from models import plant_v2
print_python_source_diff(plant_v1.run, plant_v2.run)
```

```
 4:         light_out = YggOutput('light')
 5: +       plant2root = YggTimesync('plant2root')
 6:         mass = units.add_units(mass, 'g')
 7:         tmin = units.add_units(tmin, 'hrs')
 8:         tmax = units.add_units(tmax, 'hrs')
 9:         tstep = units.add_units(tstep, 'hrs')
10:         t = tmin
11:         i = 0

  ...

15: +           # Synchronize data for time step with the root model
16: +           root_state = {'mass': mass}
17: +           flag, root_state = plant2root.call(t, root_state)
18: +           if not flag:
19: +               raise Exception("Error performing time-step synchr
20: +                               "with root model.")
21: +           mass = root_state['mass']
22: +
```

Inputs

Interface "call"
(= send + receive)

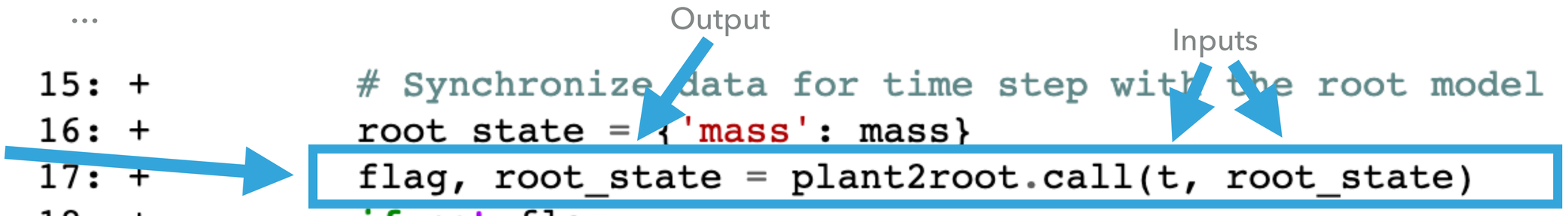
In [18]:
```
from models import plant_v2
print_python_source_diff(plant_v1.run, plant_v2.run)
```

```
 4:         light_out = YggOutput('light')
 5: +       plant2root = YggTimesync('plant2root')
 6:         mass = units.add_units(mass, 'g')
 7:         tmin = units.add_units(tmin, 'hrs')
 8:         tmax = units.add_units(tmax, 'hrs')
 9:         tstep = units.add_units(tstep, 'hrs')
10:         t = tmin
11:         i = 0

...

15: +           # Synchronize data for time step with the root model
16: +           root_state = {'mass': mass}
17: +           flag, root_state = plant2root.call(t, root_state)
18: +           if not flag:
19: +               raise Exception("Error performing time-step synchr
20: +                               "with root model.")
21: +           mass = root_state['mass']
22: +
```

Interface "call"
(= send + receive)

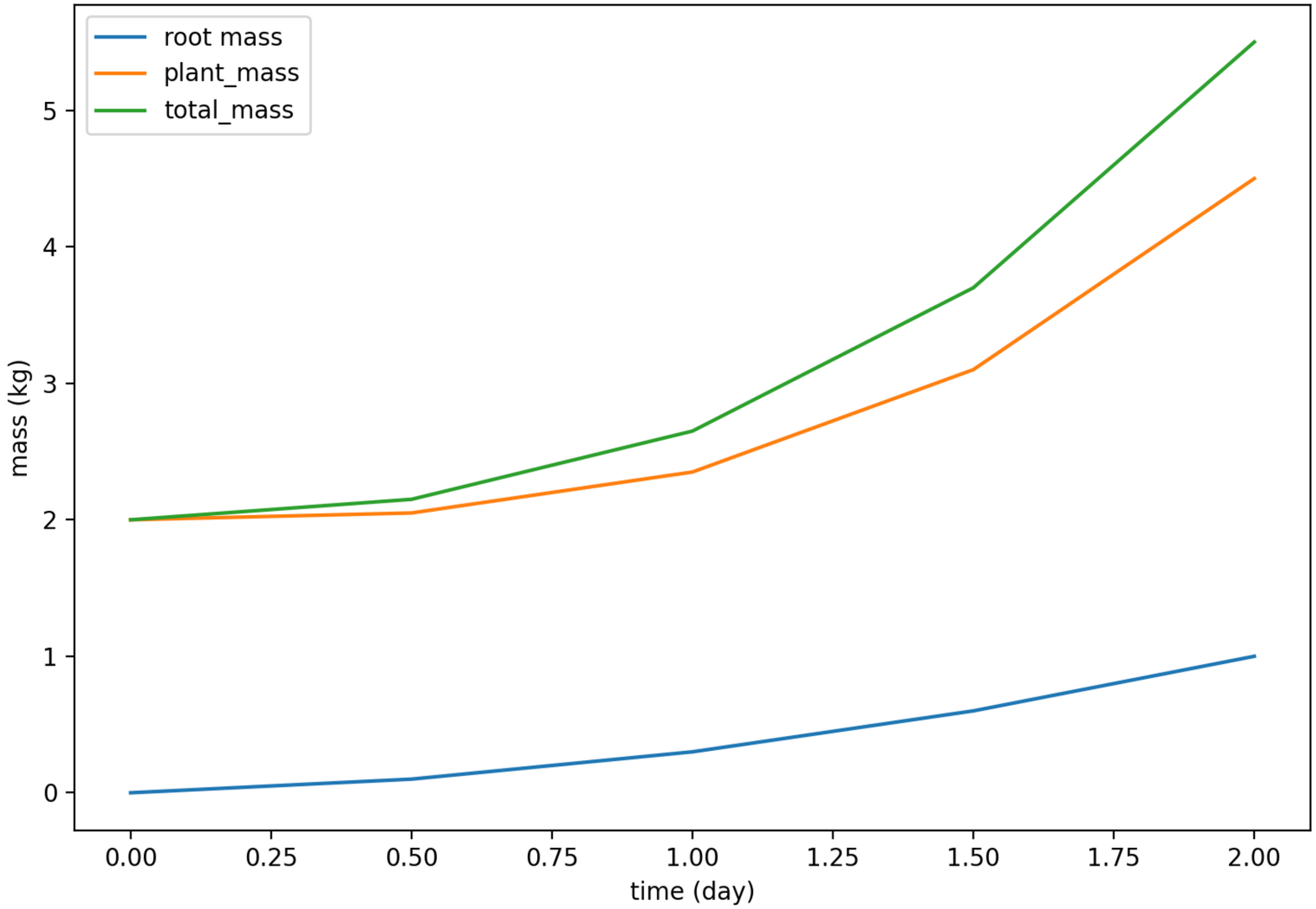


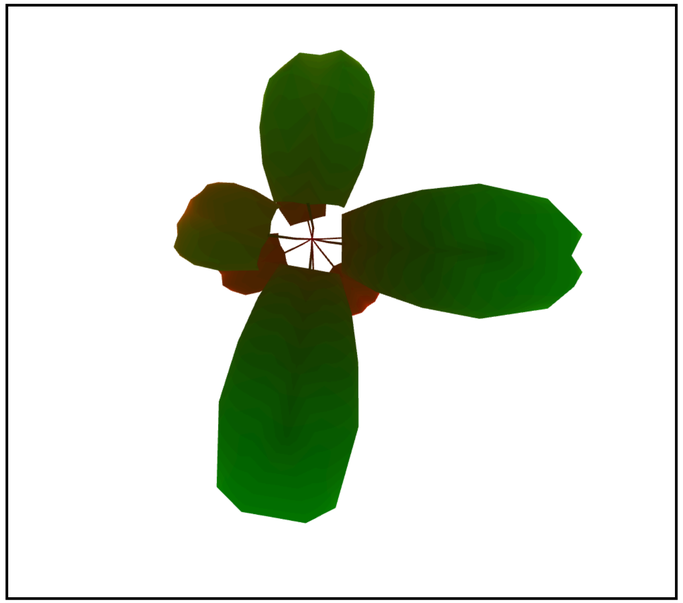
In [ ]:
```
run(['yamls/plant_v2.yml', 'yamls/light.yml', 'yamls/roots.yml'], production_run=True)
display_last_timestep(with_light=True, with_masses=True)
```

In [19]:
```
run(['yamls/plant_v2.yml', 'yamls/light.yml', 'yamls/roots.yml'], production_run=True)
display_last_timestep(with_light=True, with_masses=True)
```

```
YggRunner(runner): Starting I/O drivers and models on system Meagans-Air in namespace yggdr
YggRunner(runner): plant finished running.
End of input from temp_height.
YggRunner(runner): plant finished exiting.
YggRunner(runner): roots finished running.
No more messages from model process.
TimeSyncModelDriver(plant2root): returncode = 0
YggRunner(runner): roots finished exiting.
YggRunner(runner): light finished running.
YggRunner(runner): light finished exiting.
YggRunner(runner): plant2root finished running.
YggRunner(runner): plant2root finished exiting.
YggRunner(runner): All models completed
YggRunner(runner):                  init 0.000000
YggRunner(runner):          load drivers 0.117040
YggRunner(runner):         start drivers 0.350204
YggRunner(runner):             run models 47.102921
YggRunner(runner):        close channels 0.000142
YggRunner(runner):               clean up 0.072957
YggRunner(runner): ========================================
YggRunner(runner):                 Total 47.643264
```

In [19]:
```
run(['yamls/plant_v2.yml', 'yamls/light.yml', 'yamls/roots.yml'], production_run=True)
display_last_timestep(with_light=True, with_masses=True)
```





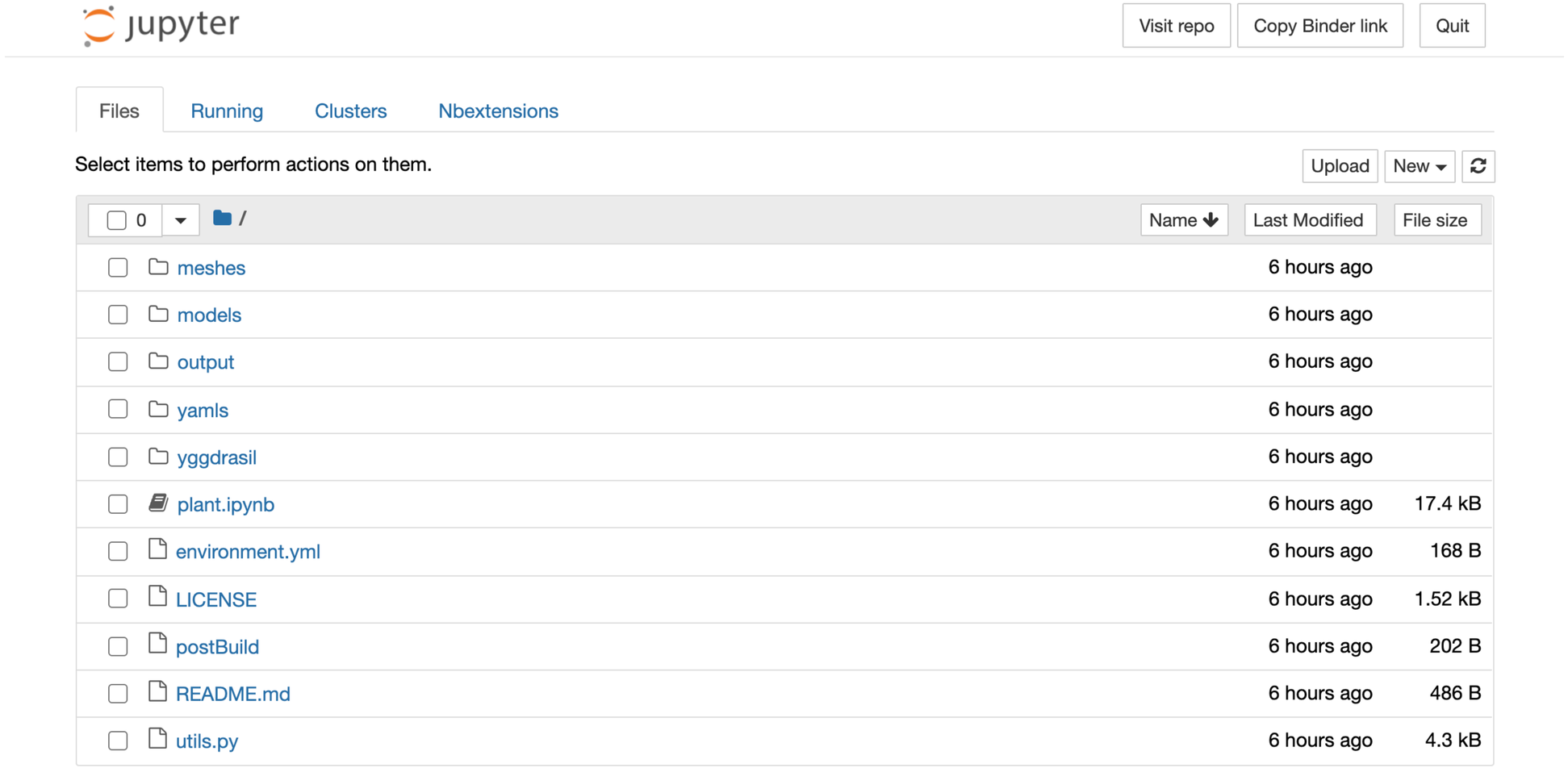
SOME NOTES

COMMAND LINE INTERFACE (CLI)

Run yggdrasil integration from the command line w/o opening Python

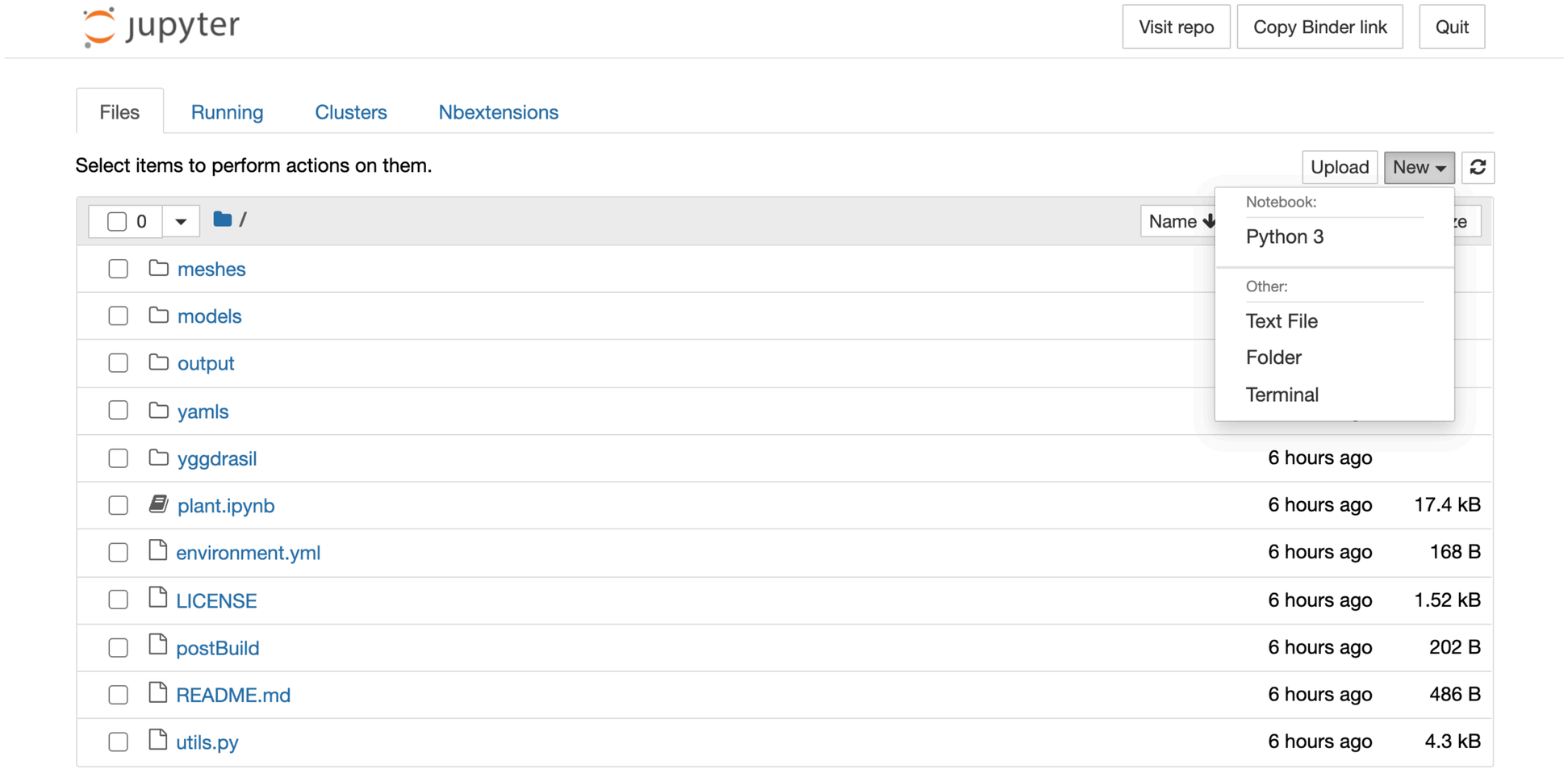
COMMAND LINE INTERFACE (CLI)

Run yggdrasil integration from the command line w/o opening Python



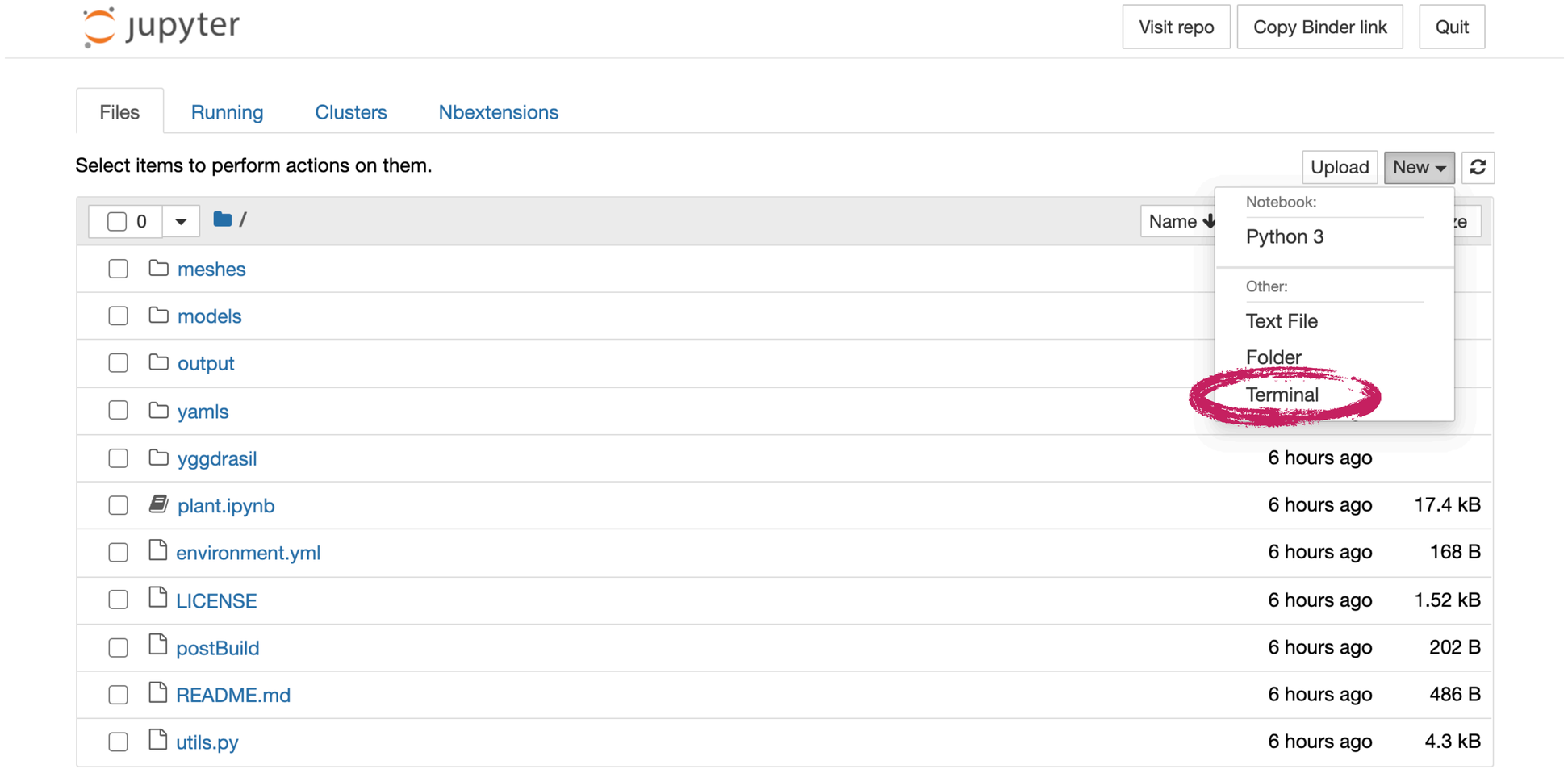
COMMAND LINE INTERFACE (CLI)

Run yggdrasil integration from the command line w/o opening Python



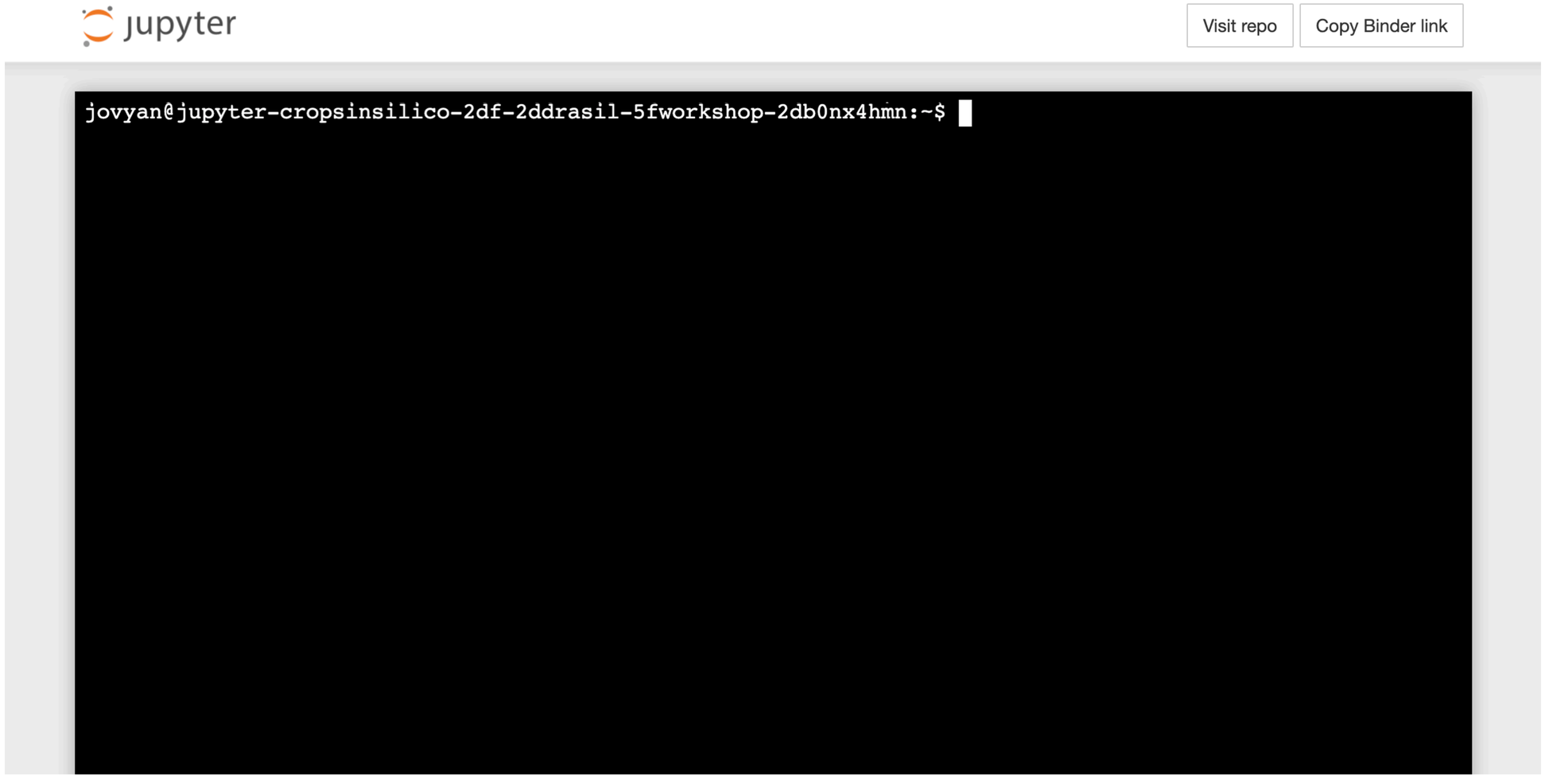
COMMAND LINE INTERFACE (CLI)

Run yggdrasil integration from the command line w/o opening Python



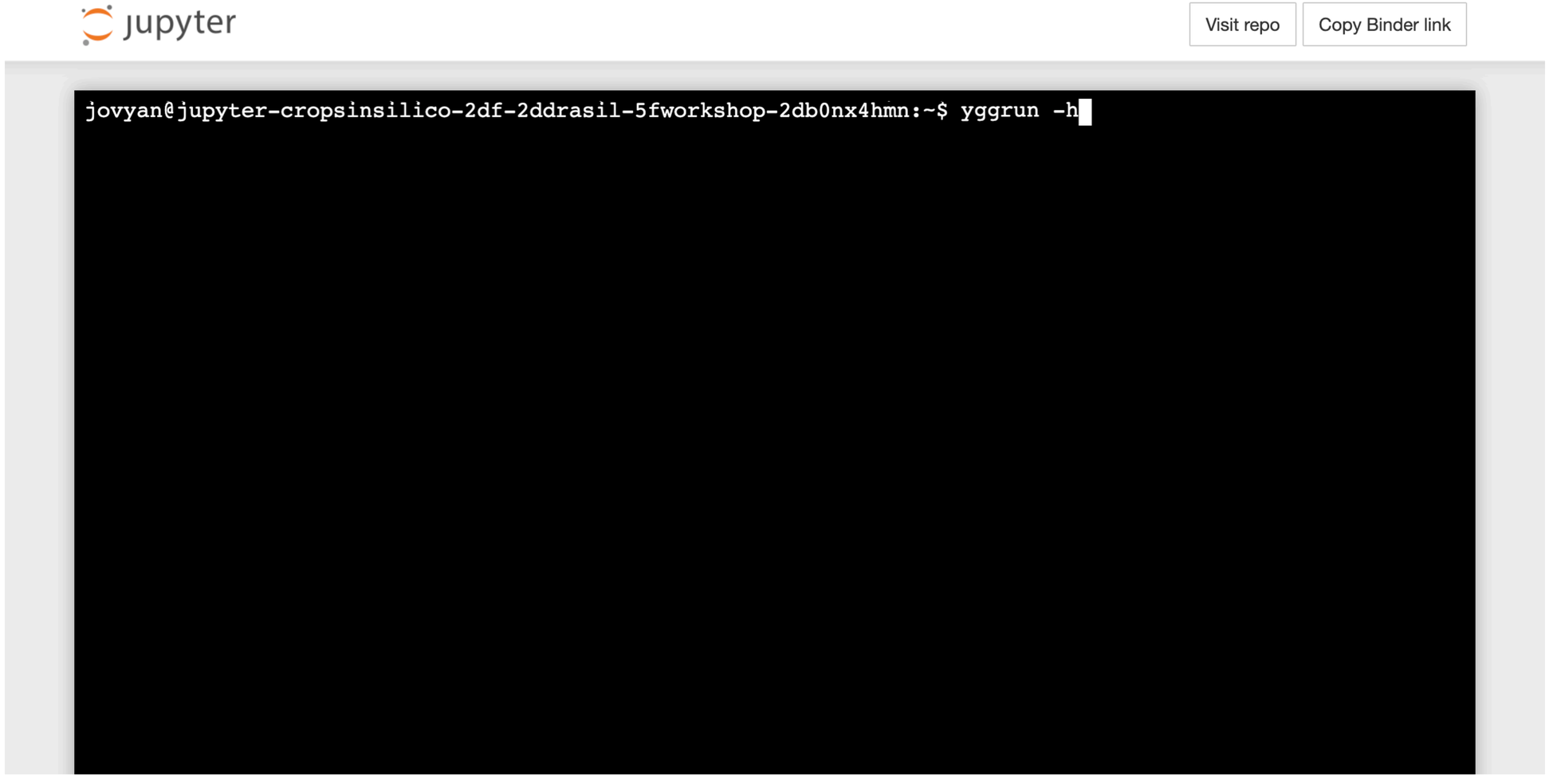
COMMAND LINE INTERFACE (CLI)

Run yggdrasil integration from the command line w/o opening Python



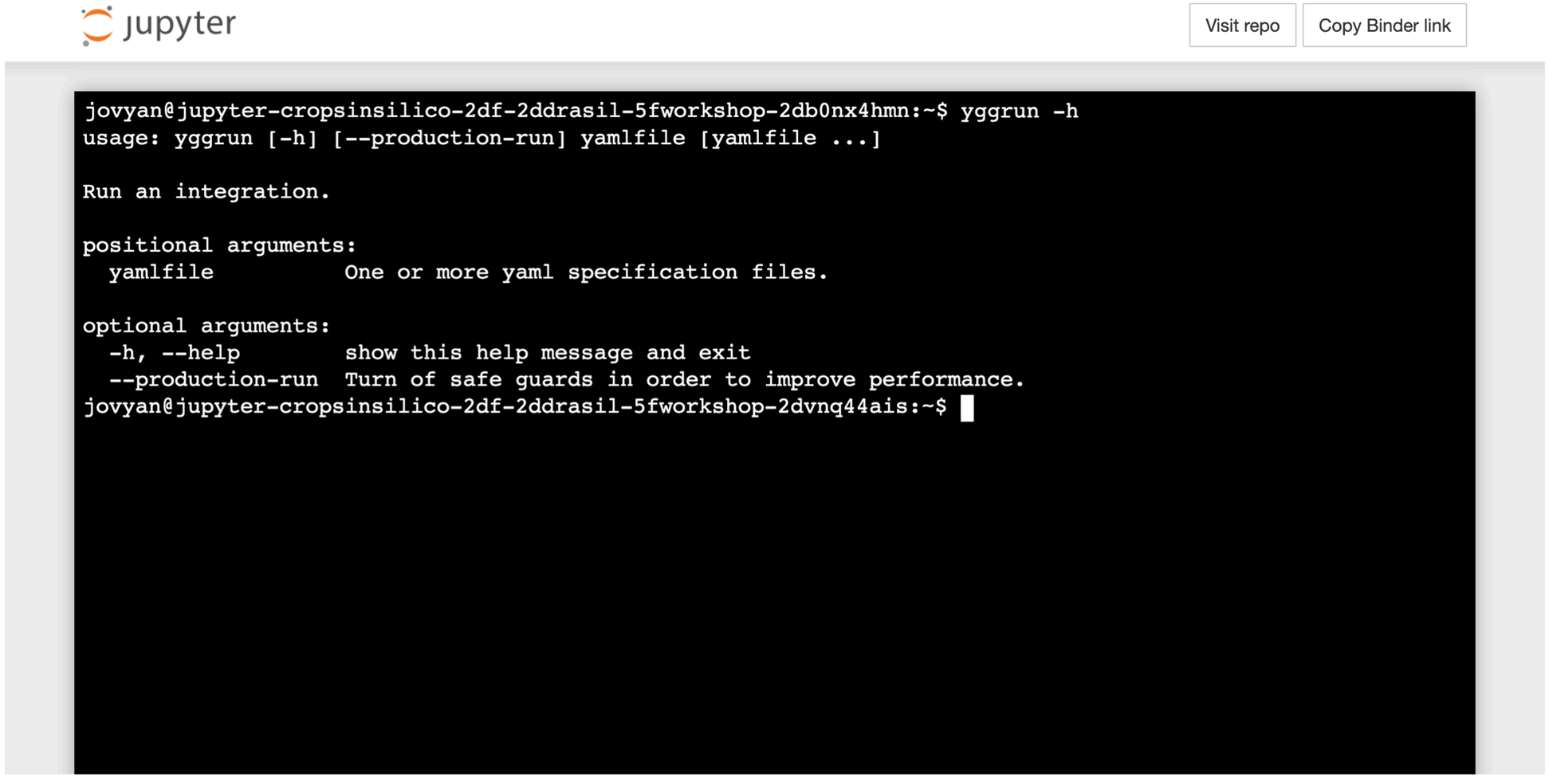
COMMAND LINE INTERFACE (CLI)

Run yggdrasil integration from the command line w/o opening Python



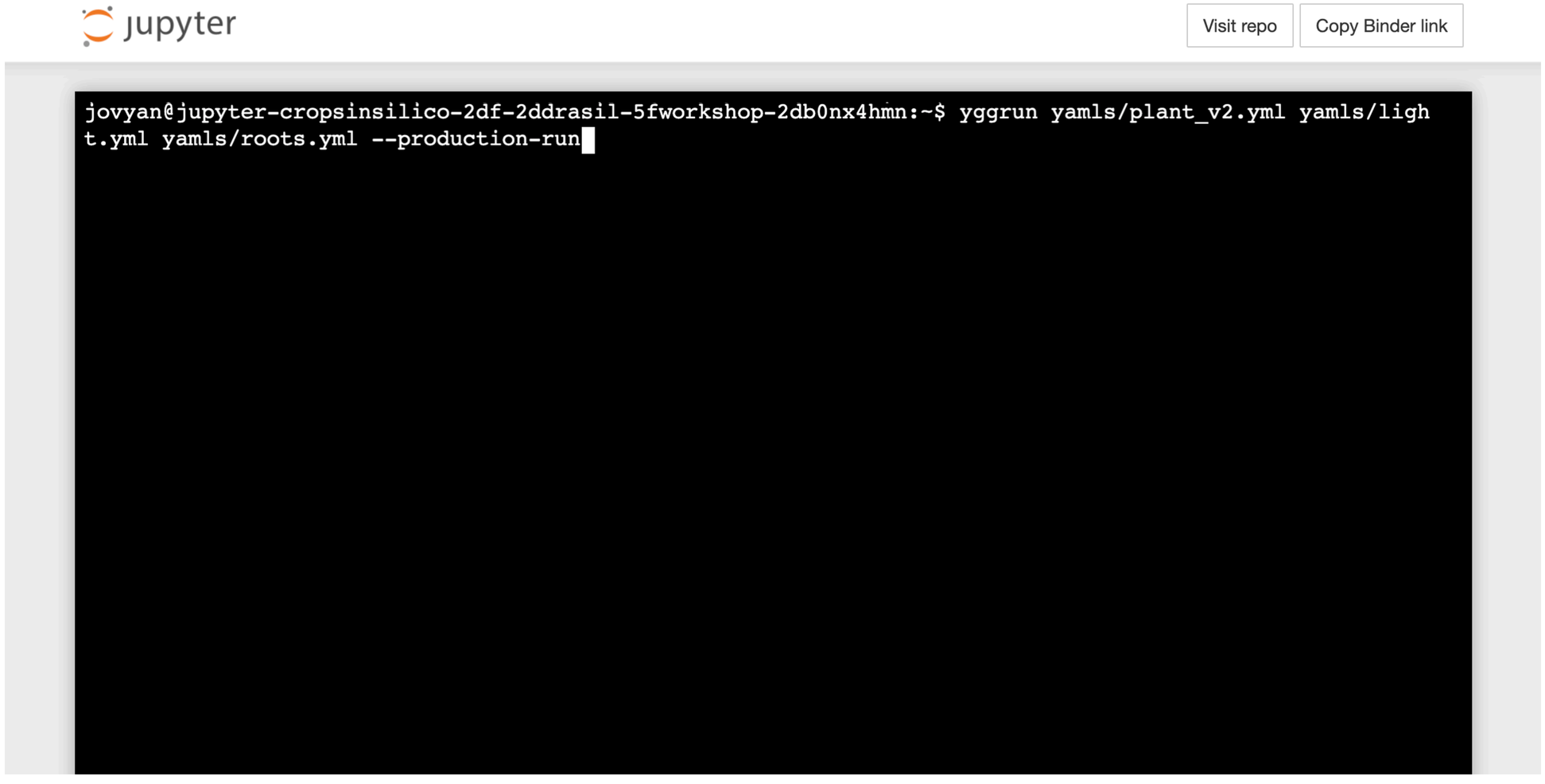
COMMAND LINE INTERFACE (CLI)

Run yggdrasil integration from the command line w/o opening Python



COMMAND LINE INTERFACE (CLI)

Run yggdrasil integration from the command line w/o opening Python



COMMAND LINE INTERFACE (CLI)

Run yggdrasil integration from the command line w/o opening Python

jupyter

Visit repo | Copy Binder link

```
jovyan@jupyter-cropsinsilico-2df-2ddrasil-5fworkshop-2db0nx4hmn:~$ yggrun yamls/plant_v2.yml yamls/ligh
t.yml yamls/roots.yml --production-run
INFO:runner.startDrivers[350]:YggRunner(runner): Starting I/O drivers and models on system jupyter-crop
sinsilico-2df-2ddrasil-5fworkshop-2db0nx4hmn in namespace yggdrasil with rank 0
INFO:runner.waitModels[399]:YggRunner(runner): plant finished running.
End of input from temp_height.
INFO:runner.waitModels[405]:YggRunner(runner): plant finished exiting.
INFO:runner.waitModels[399]:YggRunner(runner): roots finished running.
timesync server: End of input.
No more messages from model process.
INFO:DSLModelDriver.after_loop[123]:TimeSyncModelDriver(plant2root): returncode = 0
INFO:runner.waitModels[405]:YggRunner(runner): roots finished exiting.
INFO:runner.waitModels[399]:YggRunner(runner): light finished running.
INFO:runner.waitModels[405]:YggRunner(runner): light finished exiting.
INFO:runner.waitModels[399]:YggRunner(runner): plant2root finished running.
INFO:runner.waitModels[405]:YggRunner(runner): plant2root finished exiting.
INFO:runner.waitModels[418]:YggRunner(runner): All models completed
INFO:runner.run[194]:YggRunner(runner):                 init    0.000001
INFO:runner.run[194]:YggRunner(runner):         load drivers    0.366398
INFO:runner.run[194]:YggRunner(runner):        start drivers    0.803702
INFO:runner.run[194]:YggRunner(runner):            run models    55.033687
INFO:runner.run[194]:YggRunner(runner):       close channels    0.000216
INFO:runner.run[194]:YggRunner(runner):              clean up    0.247878
INFO:runner.run[196]:YggRunner(runner): ========================================
INFO:runner.run[197]:YggRunner(runner):                Total    56.451883
```

"PRODUCTION RUN" FLAG

Speeds up execution by skipping type checking and input validation
(only used once integration has been debugged)

```
In [ ]: run(['yamls/plant_v2.yml', 'yamls/light.yml', 'yamls/roots.yml'], production_run=True)
        display_last_timestep(with_light=True)
        plot_mass()
```

jupyter

Visit repo | Copy Binder link

```
jovyan@jupyter-cropsinsilico-2df-2ddrasil-5fworkshop-2db0nx4hmn:~$ yggrun yamls/plant_v2.yml yamls/ligh
t.yml yamls/roots.yml --production-run
INFO:runner.startDrivers[350]:YggRunner(runner): Starting I/O drivers and models on system jupyter-crop
sinsilico-2df-2ddrasil-5fworkshop-2db0nx4hmn in namespace yggdrasil with rank 0
INFO:runner.waitModels[399]:YggRunner(runner): plant finished running.
End of input from temp_height.
INFO:runner.waitModels[405]:YggRunner(runner): plant finished exiting.
INFO:runner.waitModels[399]:YggRunner(runner): roots finished running.
timesync server: End of input.
No more messages from model process.
INFO:DSLModelDriver.after_loop[123]:TimeSyncModelDriver(plant2root): returncode = 0
INFO:runner.waitModels[405]:YggRunner(runner): roots finished exiting.
```

"PRODUCTION RUN" FLAG

Speeds up execution by skipping type checking and input validation
(only used once integration has been debugged)

```
In [ ]: run(['yamls/plant_v2.yml', 'yamls/light.yml', 'yamls/roots.yml'], production_run=True)
        display_last_timestep(with_light=True)
        plot_mass()
```

jupyter

Visit repo | Copy Binder link

```
jovyan@jupyter-cropsinsilico-2df-2ddrasil-5fworkshop-2db0nx4hmn:~$ yggrun yamls/plant_v2.yml yamls/ligh
t.yml yamls/roots.yml --production-run
INFO:runner.startDrivers[...]:YggRunner(runner): Starting I/O drivers and models on system jupyter-crop
sinsilico-2df-2ddrasil-5fworkshop-2db0nx4hmn in namespace yggdrasil with rank 0
INFO:runner.waitModels[399]:YggRunner(runner): plant finished running.
End of input from temp_height.
INFO:runner.waitModels[405]:YggRunner(runner): plant finished exiting.
INFO:runner.waitModels[399]:YggRunner(runner): roots finished running.
timesync server: End of input.
No more messages from model process.
INFO:DSLModelDriver.after_loop[123]:TimeSyncModelDriver(plant2root): returncode = 0
INFO:runner.waitModels[405]:YggRunner(runner): roots finished exiting.
```

SPLITTING "CALLS"

Client/timesync "calls" can be split up into their component send & receive

In [ ]:
```
from models import plant_v2_split
print_python_source_diff(plant_v2.run, plant_v2_split.run)
```

SPLITTING "CALLS"

Client/timesync "calls" can be split up into their component send & receive

In [20]:
```
from models import plant_v2_split
print_python_source_diff(plant_v2.run, plant_v2_split.run)
```

```
15: +         # Perform send portion of call to synchronize data for time step
16: +         # with the root model
17:           root_state = {'mass': mass}
    -         flag, root_state = plant2root.call(t, root_state)
    ?             ------------                                ^^^^

18: +         flag = plant2root.send(t, root_state)
    ?                                                   ^^^^

19:           if not flag:
20:               raise Exception("Error performing time-step synchronization "
21:                               "with root model.")
```

SPLITTING "CALLS"

Client/timesync "calls" can be split up into their component send & receive

In [20]:
```
from models import plant_v2_split
print_python_source_diff(plant_v2.run, plant_v2_split.run)
```

```
23:             # Get light data by calling light model
24:             flag, light = light_rpc.call(mesh.vertices[:, 2], t)
25:             if not flag:
26:                 raise Exception("Error calling light model")
    -
27: +
28: +           # Perform receive portion of call to synchronize data for time ste
29: +           # with the root model
30: +           flag, root_state = plant2root.recv()
31: +           if not flag:
32: +               raise Exception("Error performing recv for time-step "
33: +                               "synchronization with root model.")
34: +           mass = root_state['mass']
35: +
```

SPLITTING "CALLS"

Client/timesync "calls" can be split up into their component send & receive

In [ ]:
```
print_yaml_diff('yamls/plant_v2.yml', 'yamls/plant_v2_split.yml')
run(['yamls/plant_v2_split.yml', 'yamls/light.yml', 'yamls/roots.yml'], production_run=True)
display_last_timestep()
```

SPLITTING "CALLS"

Client/timesync "calls" can be split up into their component send & receive

In [21]:
```
print_yaml_diff('yamls/plant_v2.yml', 'yamls/plant_v2_split.yml')
run(['yamls/plant_v2_split.yml', 'yamls/light.yml', 'yamls/roots.yml'], production_run=True)
display_last_timestep()
```

```
 1:   model:
 2:     name: plant
 3:     language: python
    -   args: [../models/plant_v2.py, 0.0, 48.0, 6.0]
 4: +   args: [../models/plant_v2_split.py, 0.0, 48.0, 6.0]
    ?                                                 ++++++


 5:     client_of: light
 6:     timesync: plant2root
 7:     outputs:
 8:       - name: light
 9:         default_file:
10:           name: ../output/light_%03d.pkl
11:           filetype: pickle
12:           is_series: True
```

SPLITTING "CALLS"

Client/timesync "calls" can be split up into their component send & receive

```
In [21]: print_yaml_diff('yamls/plant_v2.yml', 'yamls/plant_v2_split.yml')
         run(['yamls/plant_v2_split.yml', 'yamls/light.yml', 'yamls/roots.yml'], production_run=True)
         display_last_timestep()
```

```
 1:   model:
 2:     name: plant
 3:     language: python
    -   args: [../models/plant_v2.py, 0.0, 48.0, 6.0]
 4: +   args: [../models/plant_v2_split.py, 0.0, 48.0, 6.0]
    ?                                                  ++++++

 5:     client_of: light
 6:     timesync: plant2root
 7:     outputs:
 8:       - name: light
 9:         default_file:
10:           name: ../output/light_%03d.pkl
11:           filetype: pickle
12:           is_series: True
```

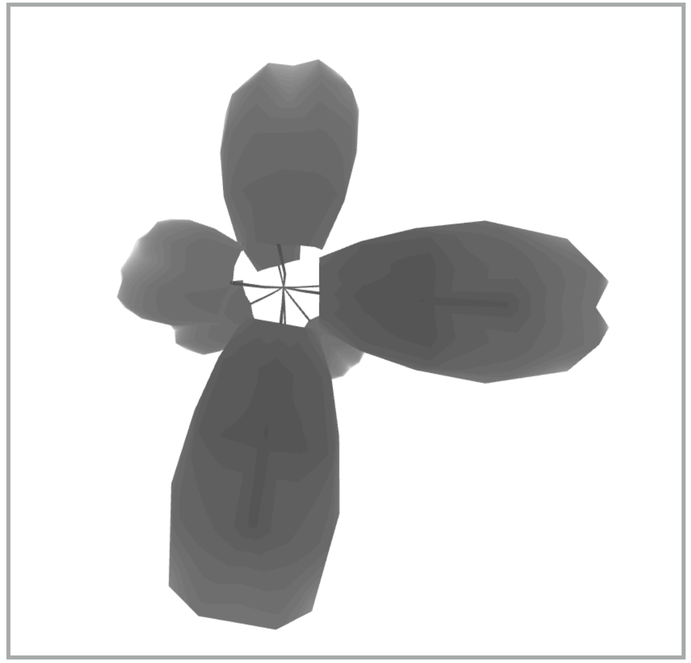
YAML Stays the Same

SPLITTING "CALLS"

Client/timesync "calls" can be split up into their component send & receive

In [21]:
```
print_yaml_diff('yamls/plant_v2.yml', 'yamls/plant_v2_split.yml')
run(['yamls/plant_v2_split.yml', 'yamls/light.yml', 'yamls/roots.yml'], production_run=True)
display_last_timestep()
```

```
YggRunner(runner): Starting I/O drivers and models on system Meagans-Air in namespace yggdrasil with rank 0
YggRunner(runner): plant finished running.
End of input from temp_height.
YggRunner(runner): plant finished exiting.
YggRunner(runner): roots finished running.
No more messages from model process.
TimeSyncModelDriver(plant2root): returncode = 0
YggRunner(runner): roots finished exiting.
YggRunner(runner): light finished running.
YggRunner(runner): light finished exiting.
YggRunner(runner): plant2root finished running.
YggRunner(runner): plant2root finished exiting.
YggRunner(runner): All models completed
YggRunner(runner):                  init 0.000000
YggRunner(runner):          load drivers 0.059761
YggRunner(runner):         start drivers 0.348838
YggRunner(runner):             run models 42.793313
YggRunner(runner):        close channels 0.000163
YggRunner(runner):              clean up 0.104158
YggRunner(runner): ========================================
YggRunner(runner):                 Total 43.306233
```



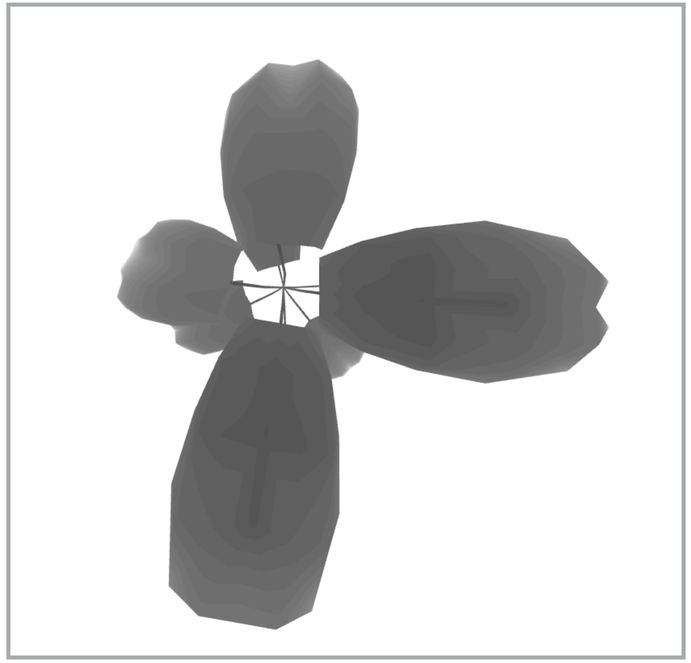
SPLITTING "CALLS"

Client/timesync "calls" can be split up into their component send & receive

```
In [21]: print_yaml_diff('yamls/plant_v2.yml', 'yamls/plant_v2_split.yml')
         run(['yamls/plant_v2_split.yml', 'yamls/light.yml', 'yamls/roots.yml'], production_run=True)
         display_last_timestep()
```

```
YggRunner(runner): Starting I/O drivers and models on system Meagans-Air in namespace yggdrasil with rank 0
YggRunner(runner): plant finished running.
End of input from temp_height.
YggRunner(runner): plant finished exiting.
YggRunner(runner): roots finished running.
No more messages from model process.
TimeSyncModelDriver(plant2root): returncode = 0
YggRunner(runner): roots finished exiting.
YggRunner(runner): light finished running.
YggRunner(runner): light finished exiting.
YggRunner(runner): plant2root finished running.
YggRunner(runner): plant2root finished exiting.
YggRunner(runner): All models completed
YggRunner(runner):                  init 0.000000
YggRunner(runner):          load drivers 0.059761
YggRunner(runner):         start drivers 0.348838
YggRunner(runner):             run models 42.793313
YggRunner(runner):        close channels 0.000163
YggRunner(runner):              clean up 0.104158
YggRunner(runner): ========================================
YggRunner(runner):                 Total 43.306233
```

Output stays the same, but integration is a bit faster



QUESTIONS?

Github: https://github.com/cropsinsilico/yggdrasil

Docs: https://cropsinsilico.github.io/yggdrasil/

Paper: https://doi.org/10.1093/insilicoplants/diz001

Project Website: http://cropsinsilico.org/