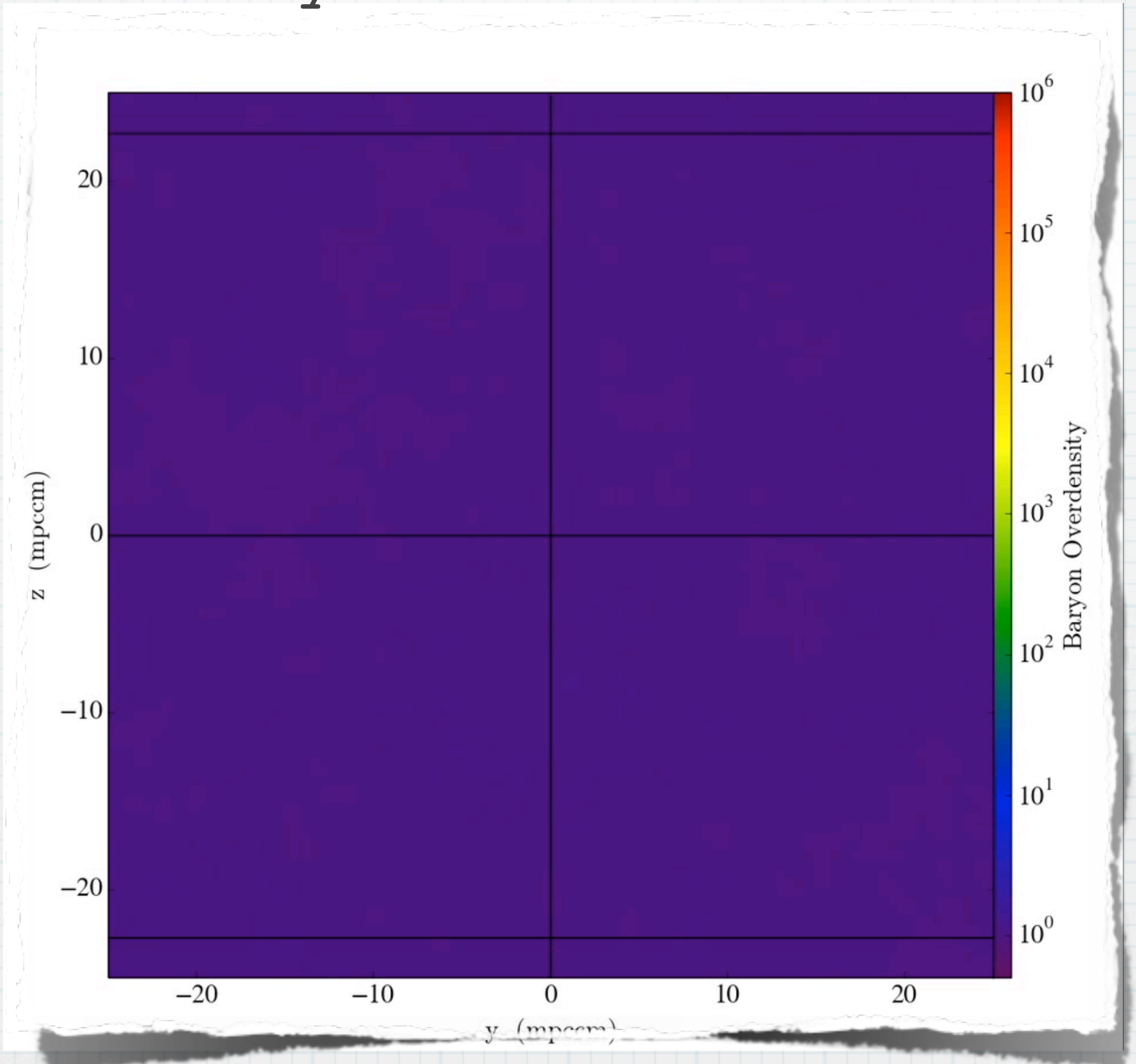
The yt Project

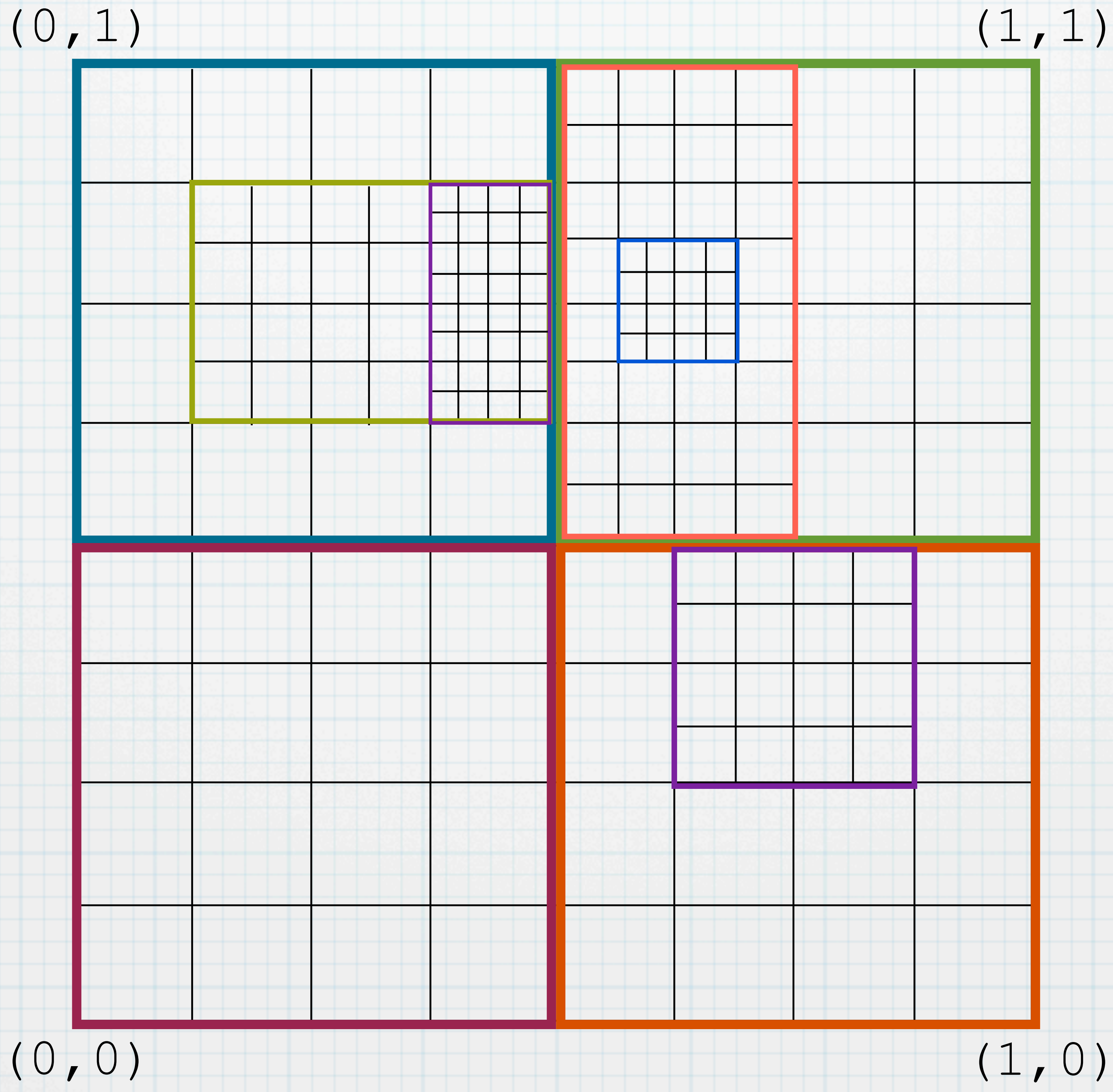
Britton Smith
Institute for Astronomy
University of Edinburgh
@aBrittonSmith

The yt Simulation Analysis Toolkit

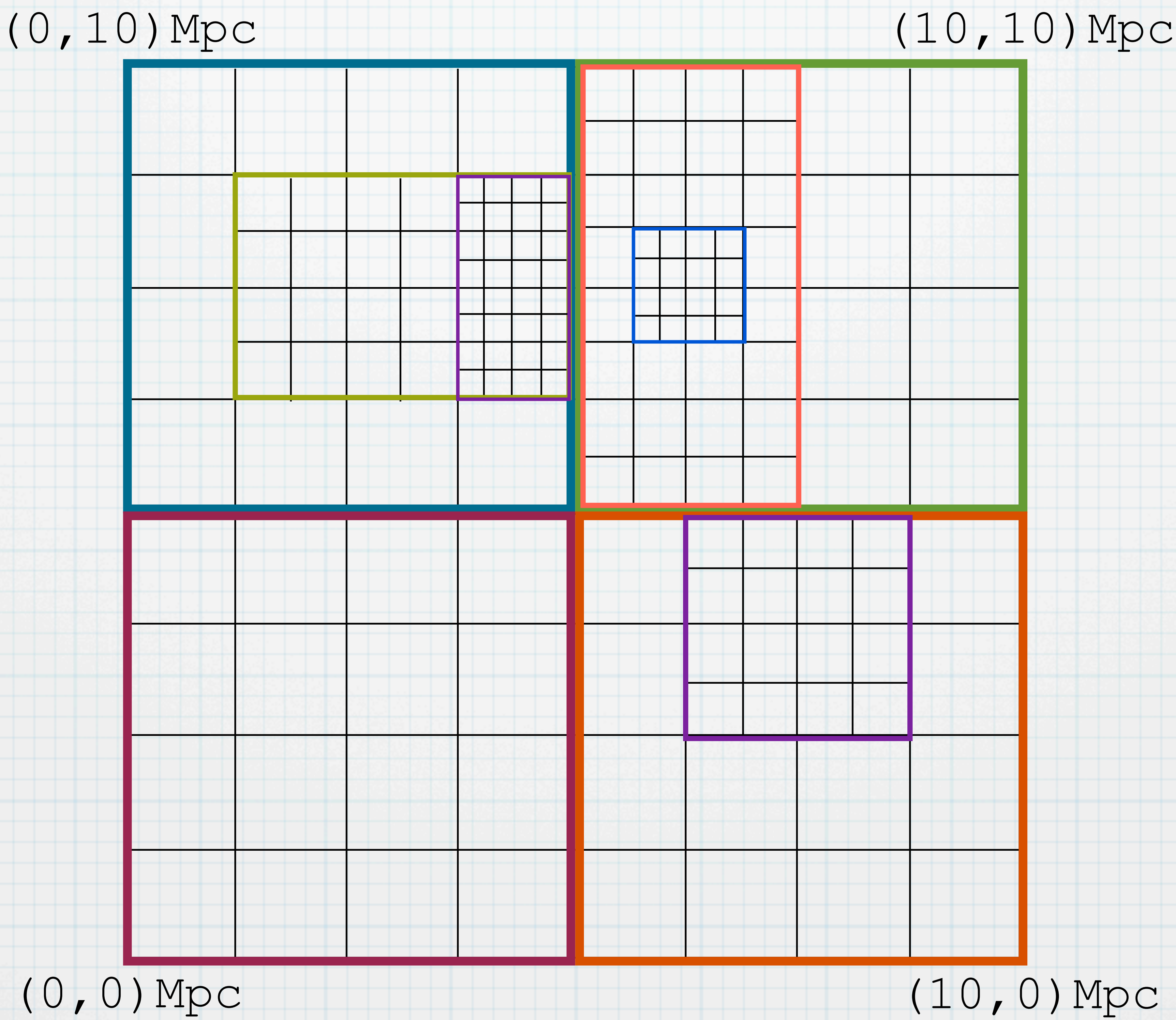
Problem 1: Simulation data is complex.
Problem 2: Everyone solves this differently.



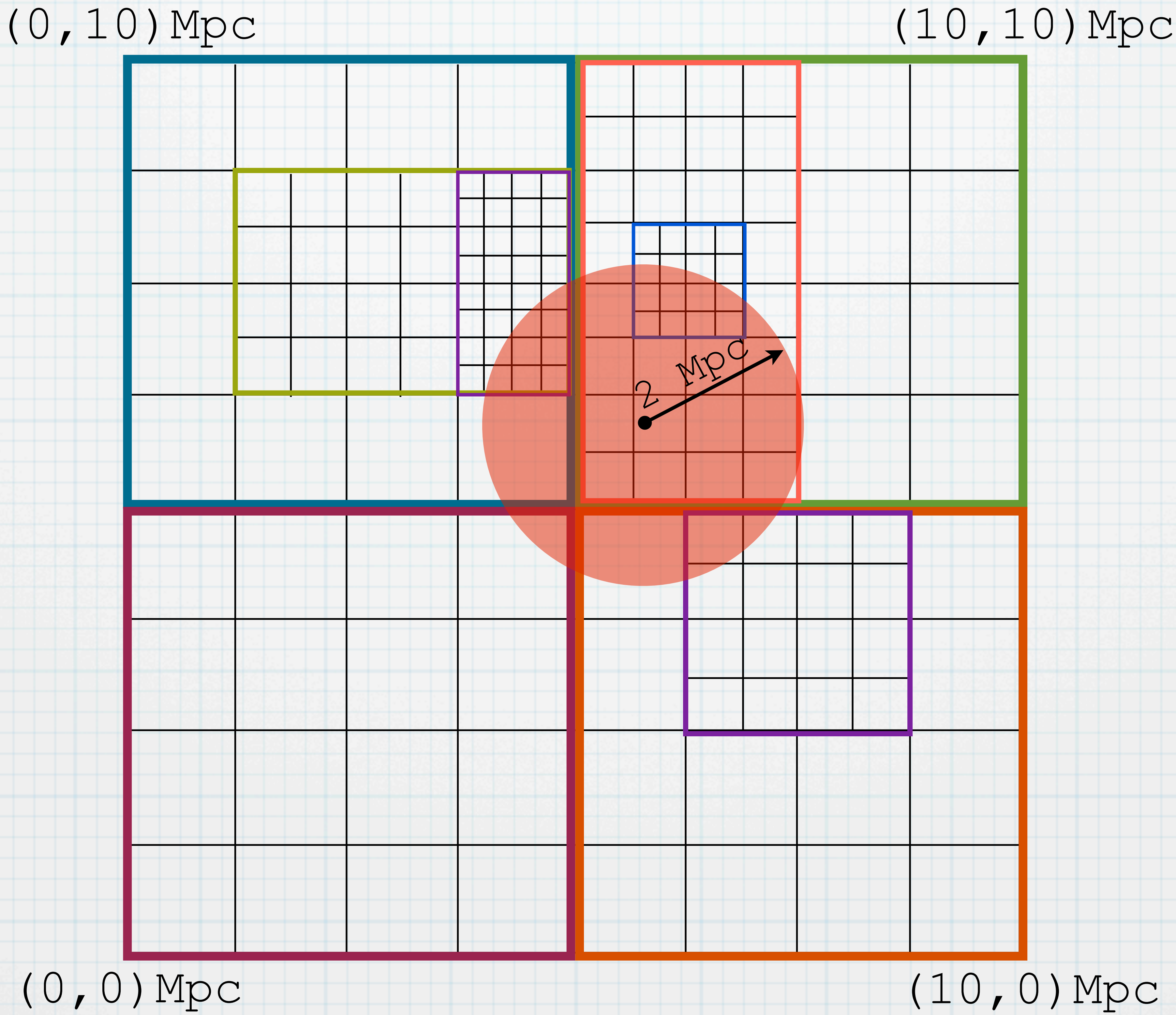
Data on disk has no physical meaning.



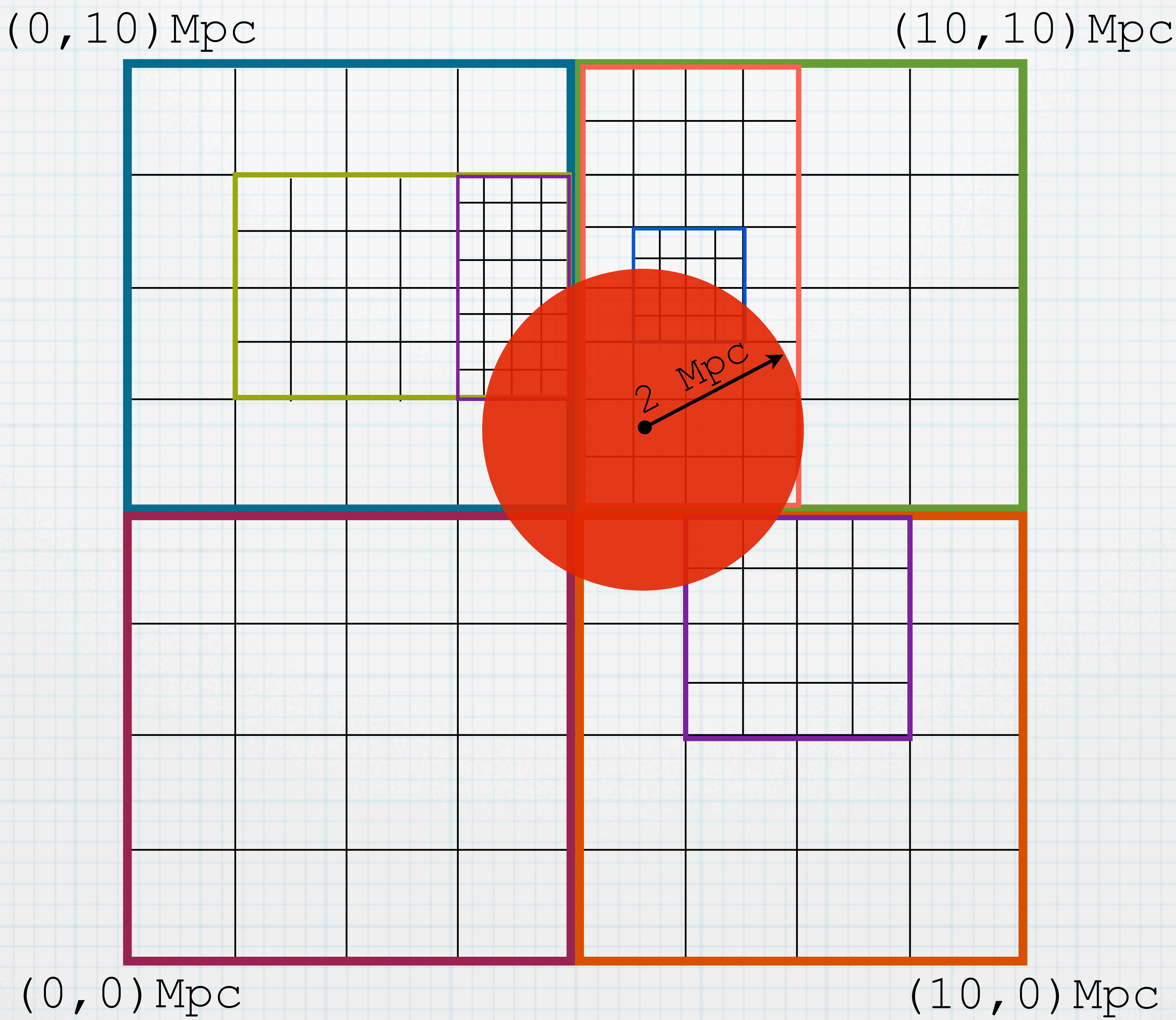
yt lets you think about physical objects



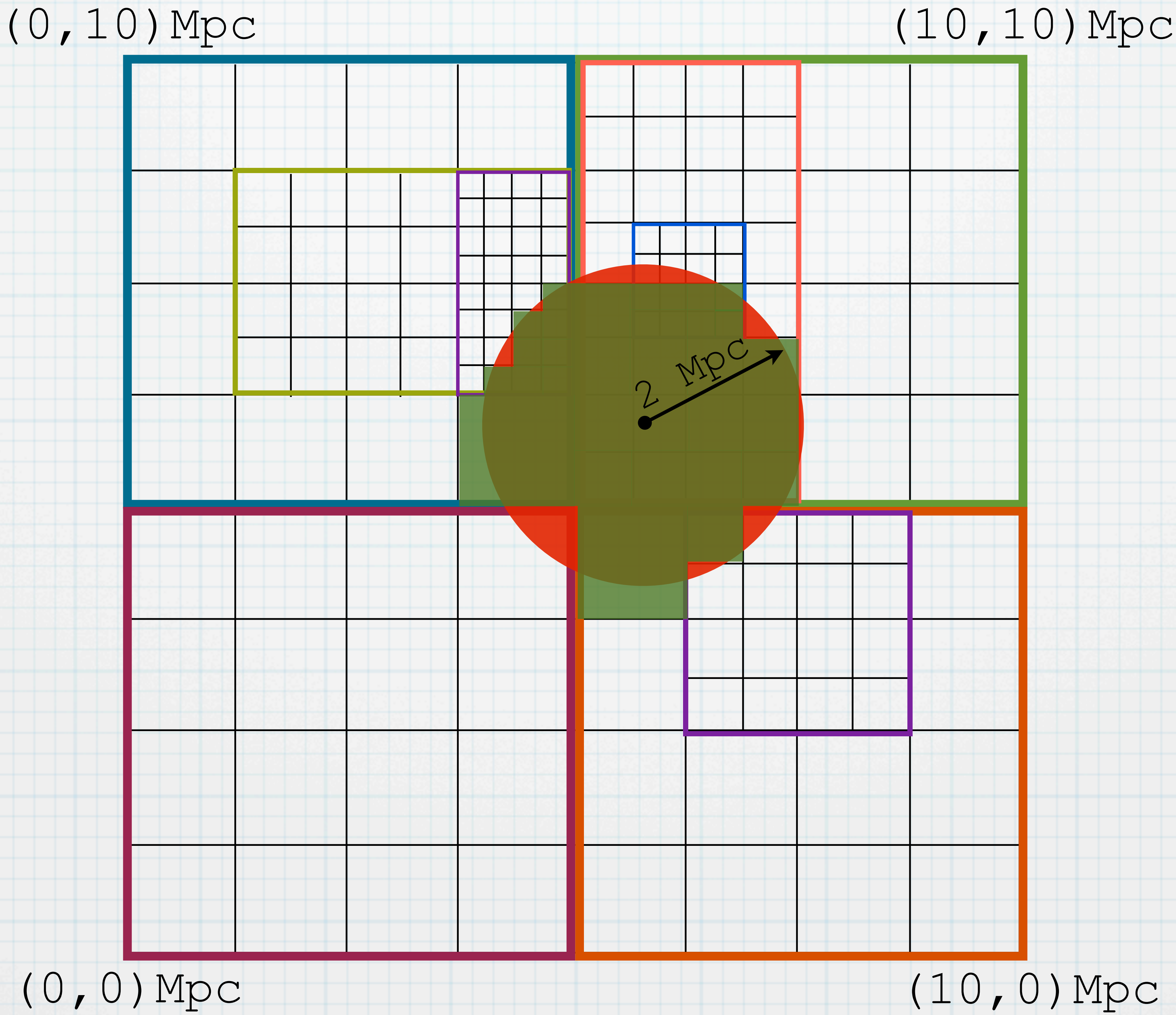
yt lets you think about physical objects



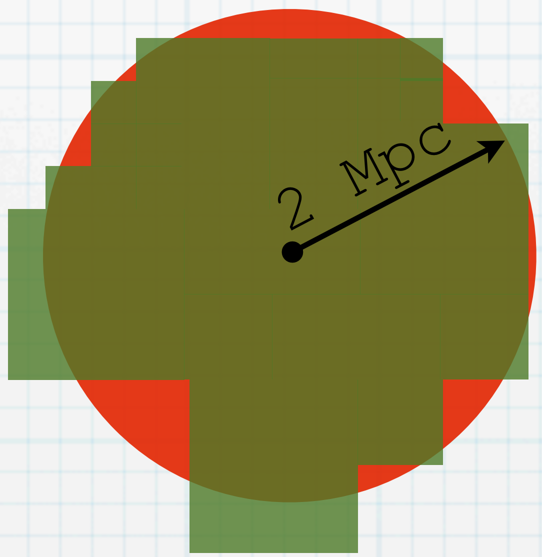
and forget what's underneath.



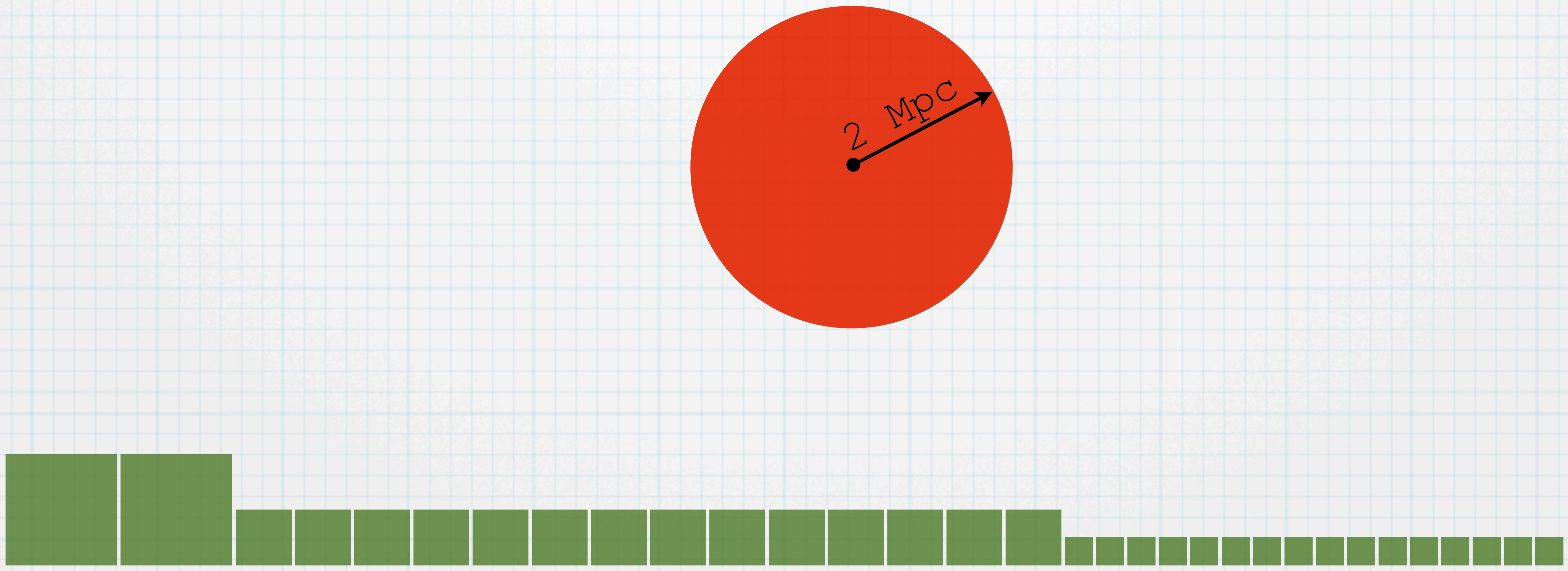
yt gives you the data you want



and only the data you want.

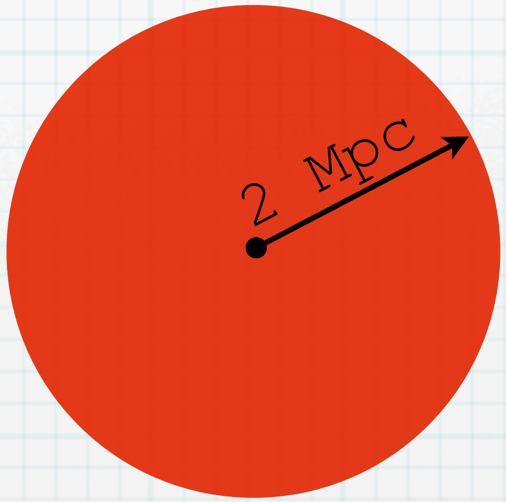


You can do whatever you want with it.



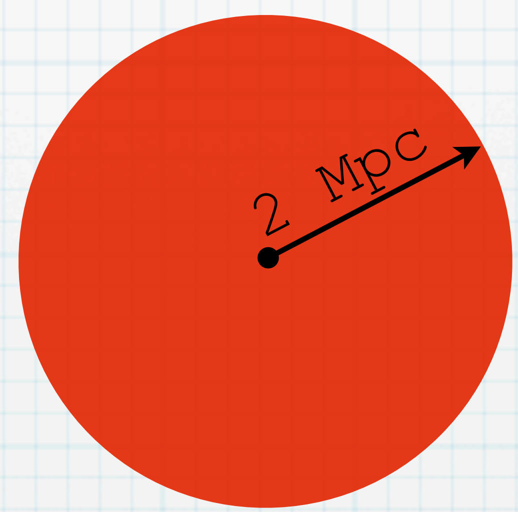
You can do whatever you want with it.

```python
import yt
ds = yt.load("DD0252/DD0252")
sp = ds.sphere(center,(2, “Mpc”))
```



You can do whatever you want with it.

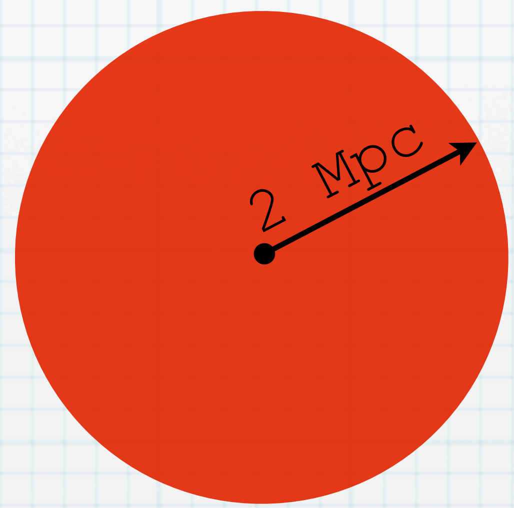
```
import yt
ds = yt.load("DD0252/DD0252")
sp = ds.sphere(center,(2, “Mpc”))
```



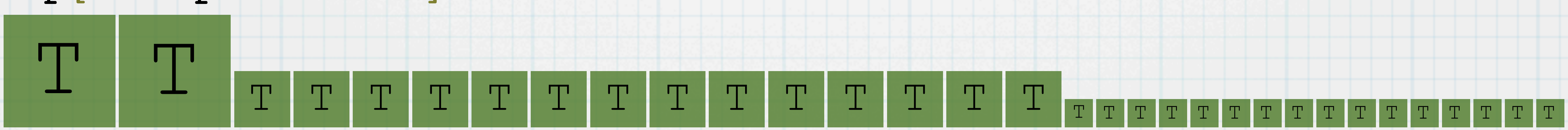

```
sp[“density”]
```

You can do whatever you want with it.

```python
import yt
ds = yt.load("DD0252/DD0252")
sp = ds.sphere(center,(2, “Mpc”))
```

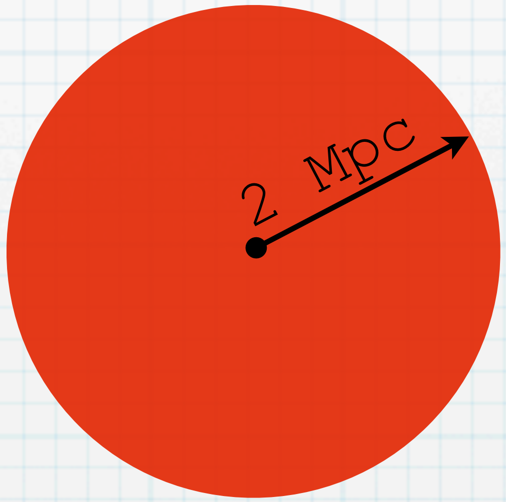



```python
sp[“temperature”]
```

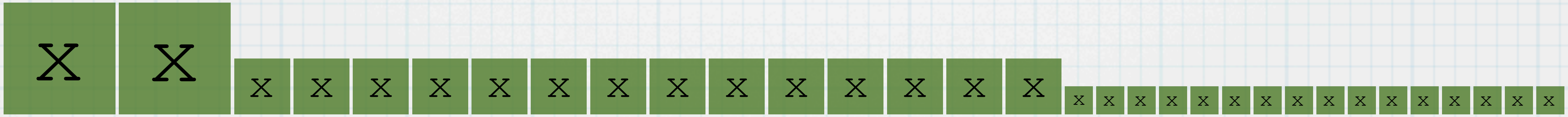


Spatial information is not lost.

```python
import yt
ds = yt.load("DD0252/DD0252")
sp = ds.sphere(center,(2, “Mpc”))
```

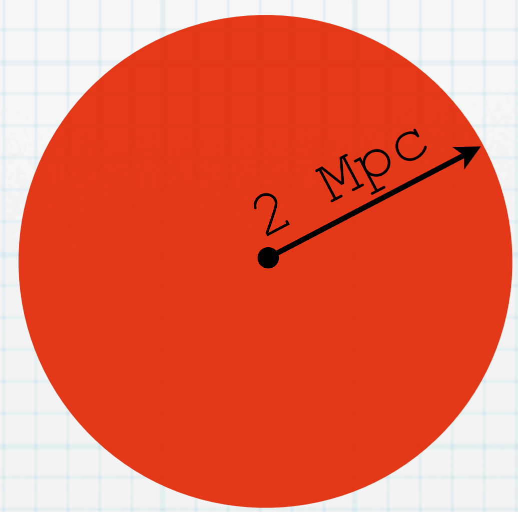



```python
sp[“x”]
```

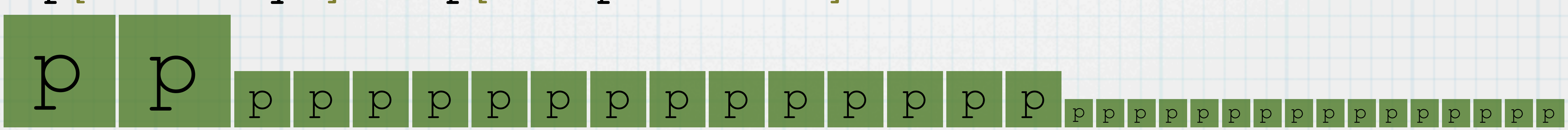


Data containers give fields as NumPy arrays.

```python
import yt
ds = yt.load("DD0252/DD0252")
sp = ds.sphere(center,(2, “Mpc”))
```

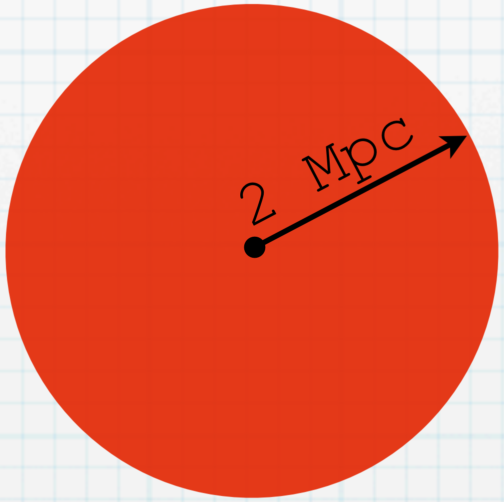



```python
sp[“density”] * sp[“temperature”]
```



Symbolic units and unit conversion.

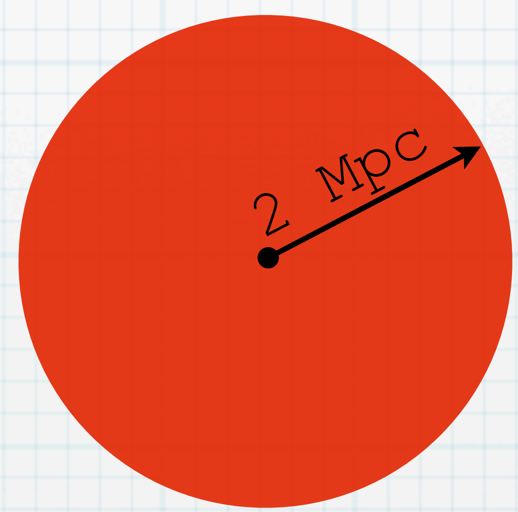
```
import yt
ds = yt.load("DD0252/DD0252")
sp = ds.sphere(center,(2, “Mpc”))
```



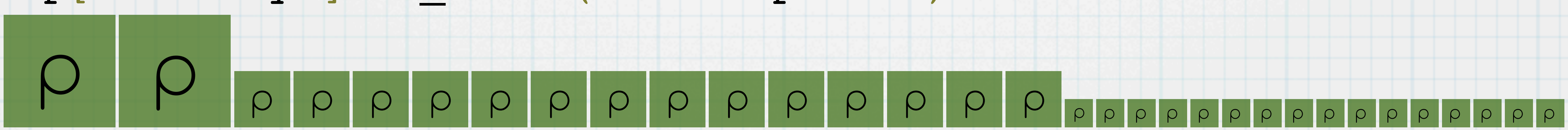

```
sp[“density”].in_units(“g/cm**3”)
```

Symbolic units and unit conversion.

```
import yt
ds = yt.load("DD0252/DD0252")
sp = ds.sphere(center,(2, “Mpc”))
```

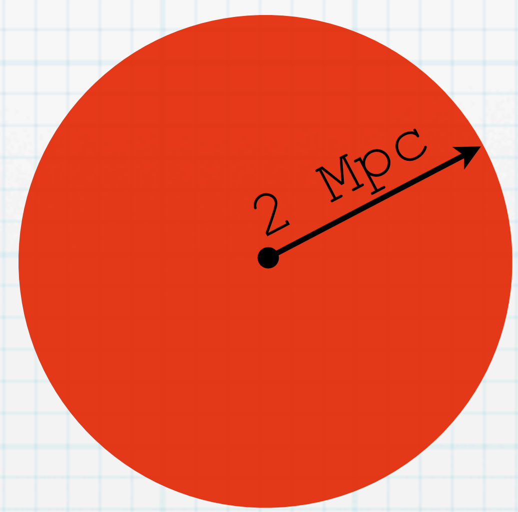



```
sp[“density”].in_units(“Msun/kpc**3”)
```

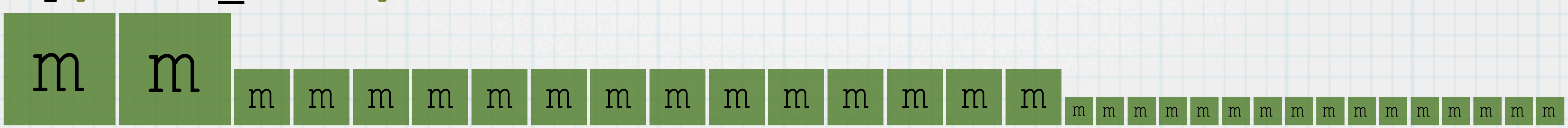


Derived quantities turn fields into single values.

```
import yt
ds = yt.load("DD0252/DD0252")
sp = ds.sphere(center,(2, “Mpc”))
```

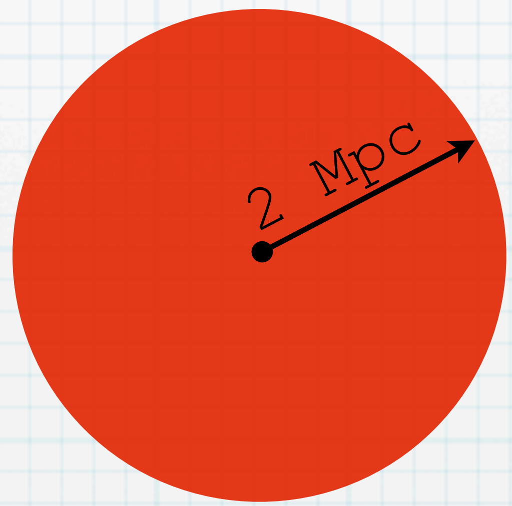



```
sp[“cell_mass”]
```



Derived quantities turn fields into single values.

```python
import yt
ds = yt.load("DD0252/DD0252")
sp = ds.sphere(center,(2, “Mpc”))
```




```python
sp.quantities.total_quantity(“cell_mass”)
```

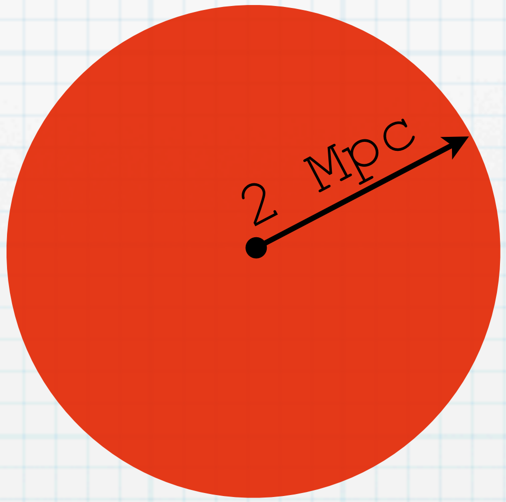
$m + m + m + m + m + m + m + m + m + m + m + m + m +$

$m + m + m + m + m + m + m + m + m + m + m + m + m + m + m + m + m +$

$m + m = M$

Derived quantities turn fields into single values.

```
import yt
ds = yt.load("DD0252/DD0252")
sp = ds.sphere(center,(2, “Mpc”))
```

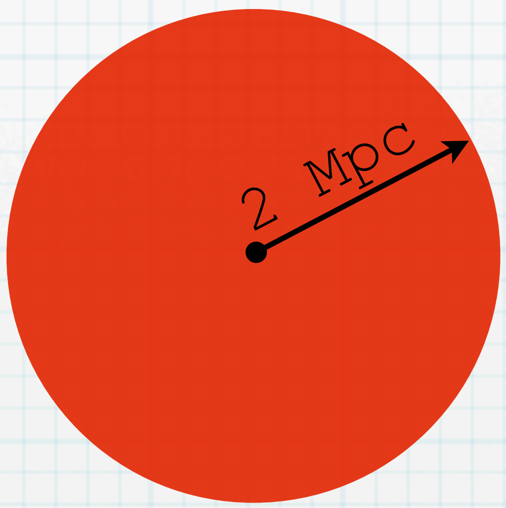



```
sp.quantities.total_quantity(“cell_mass”)
```

$$M=\sum m_i$$

Derived quantities turn fields into single values.

```python
import yt
ds = yt.load("DD0252/DD0252")
sp = ds.sphere(center,(2, “Mpc”))
```

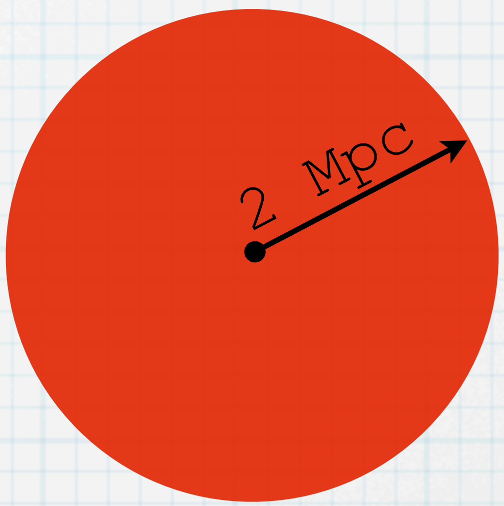



```python
sp.quantities.spin_parameter()
```

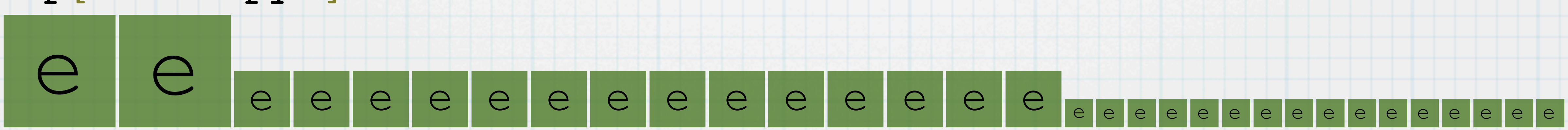
$$M=\frac{\sum L_i \,|\sum E_i{}^{1/2}\,|}{G\sum m_i{}^{5/2}}$$

Creating new fields is easy.

```
from yt.utilities.physical_constants \
    import kb

def my_field(field, data):
    return kb * data[“temperature”] * \
      data[“number_density”]**(-2./3)

ds.add_field(”entropy”,
             function=my_field,
             units=”keV*cm**2”)
```

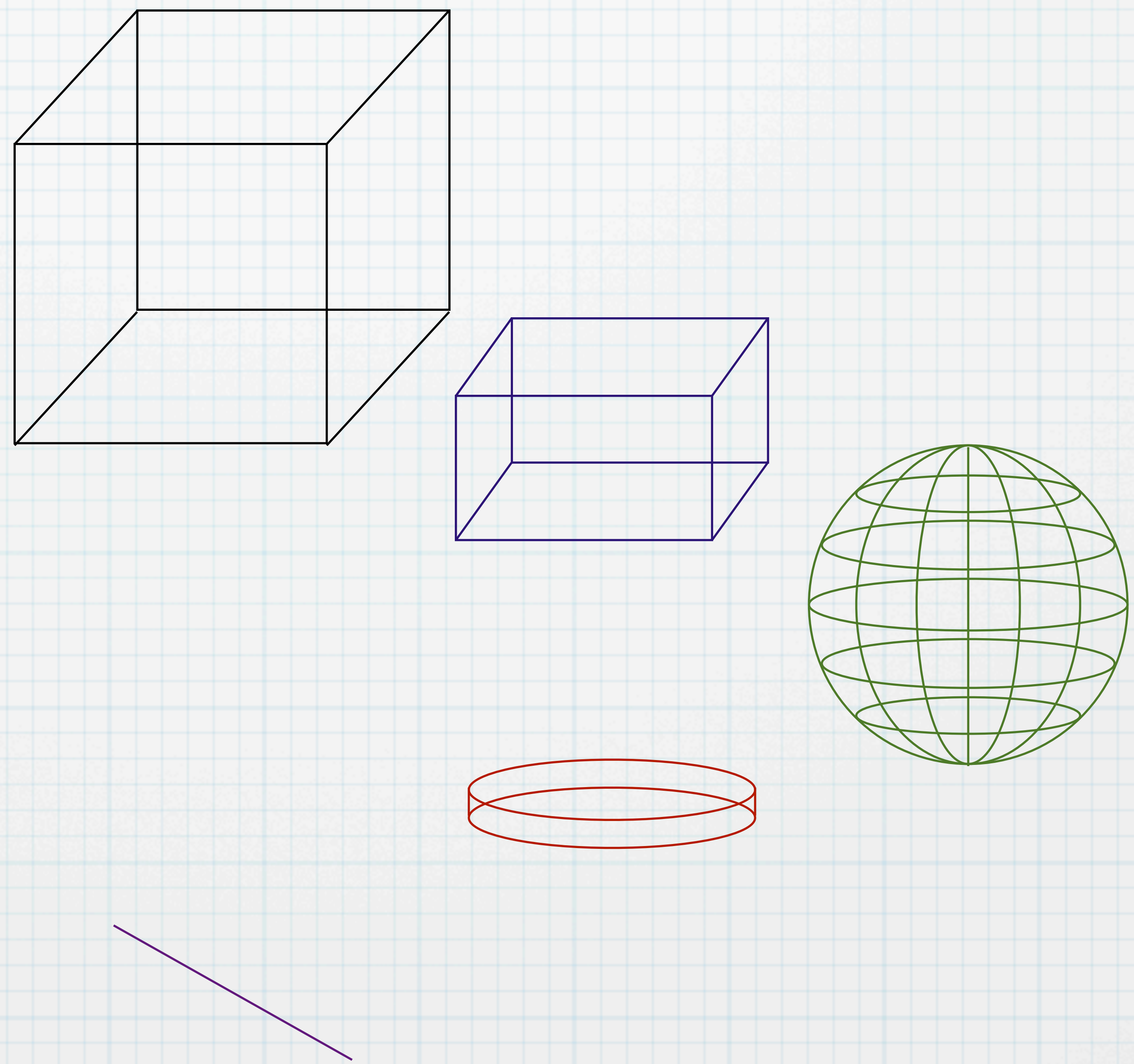



```
sp[“entropy”]
```



Data Containers

- ☑ All Data
- ☑ Region
- ☑ Sphere
- ☑ Disk
- ☑ Ray



Run it in serial.

```
import yt


# science here
ds = yt.load(...)
```

```
$ python my_script.py
```

Run it in parallel.

```
import yt
yt.enable_parallelism()

# science here
ds = yt.load(...)
```

```
$ mpirun -np 4 python my_script.py
```

Your Analysis

```python
import numpy as np



my_objects = [...]
for object in my_objects:
    # one cpu for all tasks
```

```
$ python my_script.py
```

Your Analysis in Parallel

```
import numpy as np
import yt
yt.enable_parallelism()

my_objects = [...]
for object in yt.parallel_objects(my_objects):
    # one task per cpu
```

```
$ mpirun -np 4 python my_script.py
```

Your Analysis in Parallel

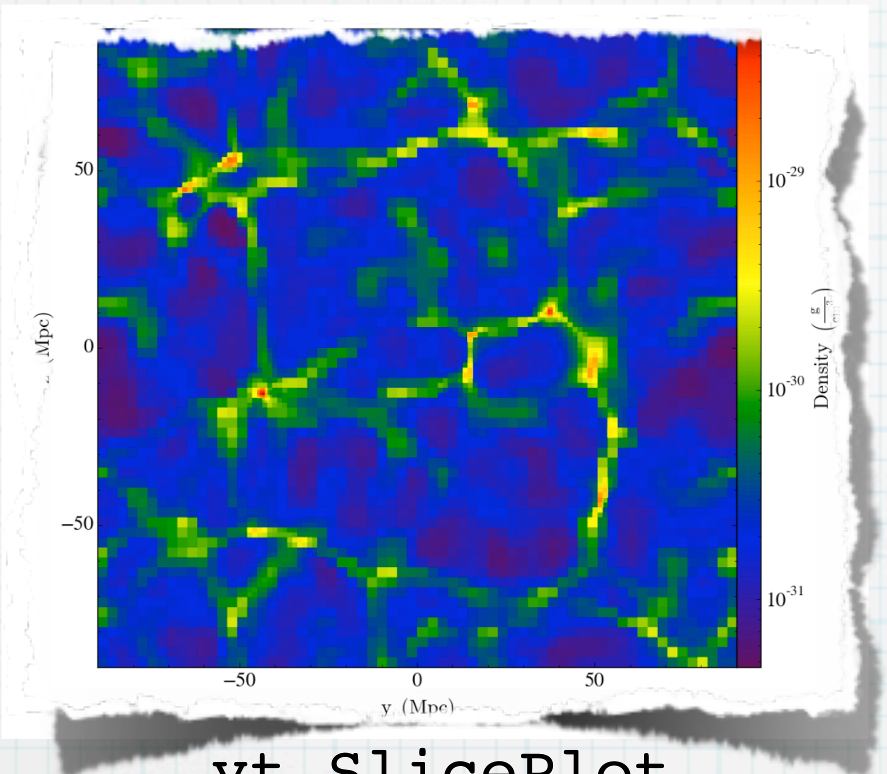
```python
import numpy as np
import yt
yt.enable_parallelism()

my_objects = [...]
for object in yt.parallel_objects(my_objects):
    # one task per cpu
```

```
$ mpirun -np 4 python my_script.py
```

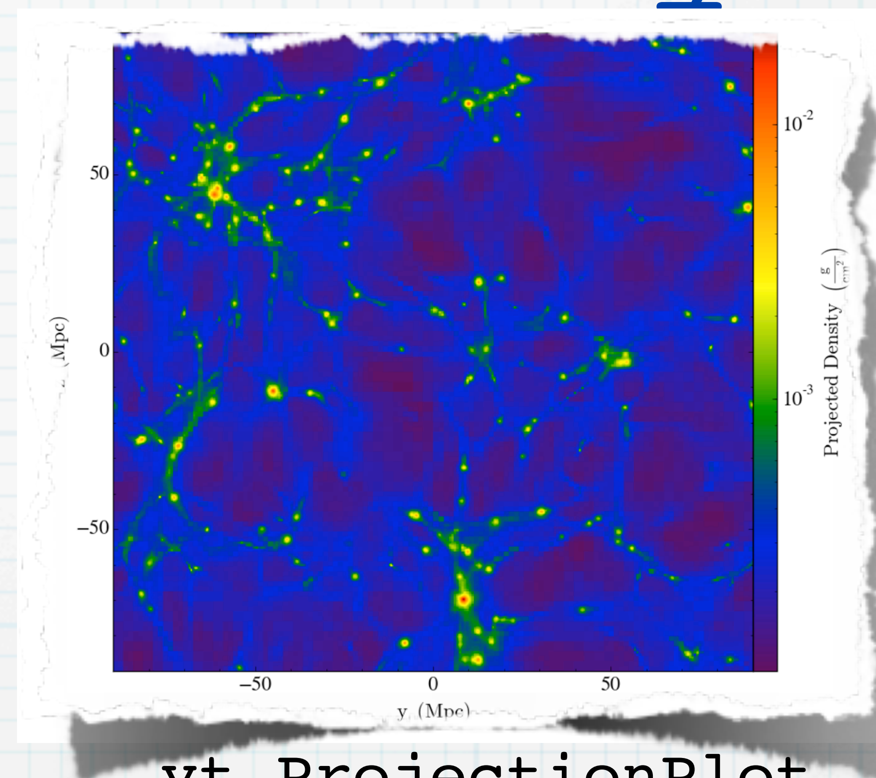
- ☑ store and collate results
- ☑ nested parallel loops
- ☑ task queues

(data exploration)

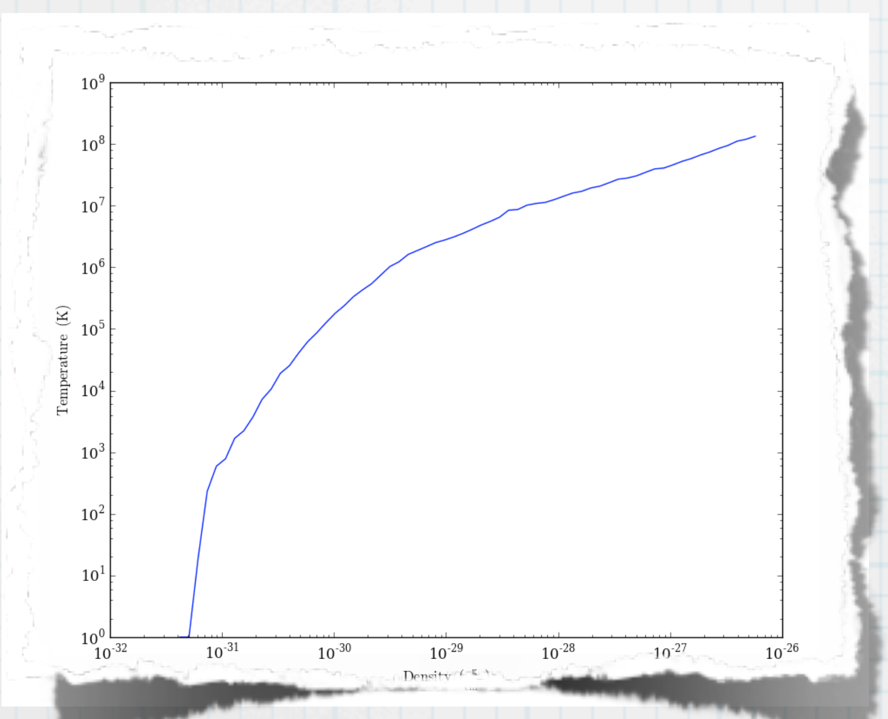
One-Button Analysis



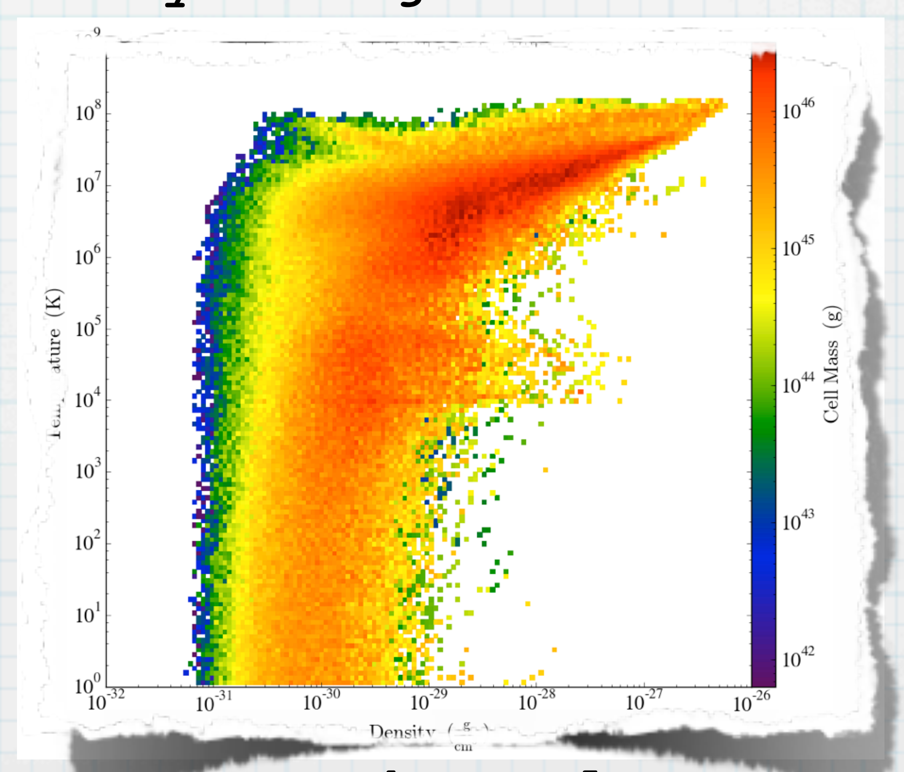
yt.SlicePlot



yt.ProjectionPlot



yt.ProfilePlot



yt.PhasePlot

(tell them here that yt does volume rendering, too)

Supported Codes

Simulation Codes

ART Athena
Boxlib Chombo
Enzo Flash
Eagle
OWLS
Gadget GDF
Piernik RAMSES
Tipsy

Halo Finders

FoF
HOP
Rockstar

Other Formats

array data
fits

Image: Devin Silvia

(tell them here that yt does volume rendering, too)

Supported Codes

Bolatto et al. 2013

Simulation

ART
Boxlib
Enzo
Eagle
OWLS
Gadget
Piernik
Tipsy

lo Finders

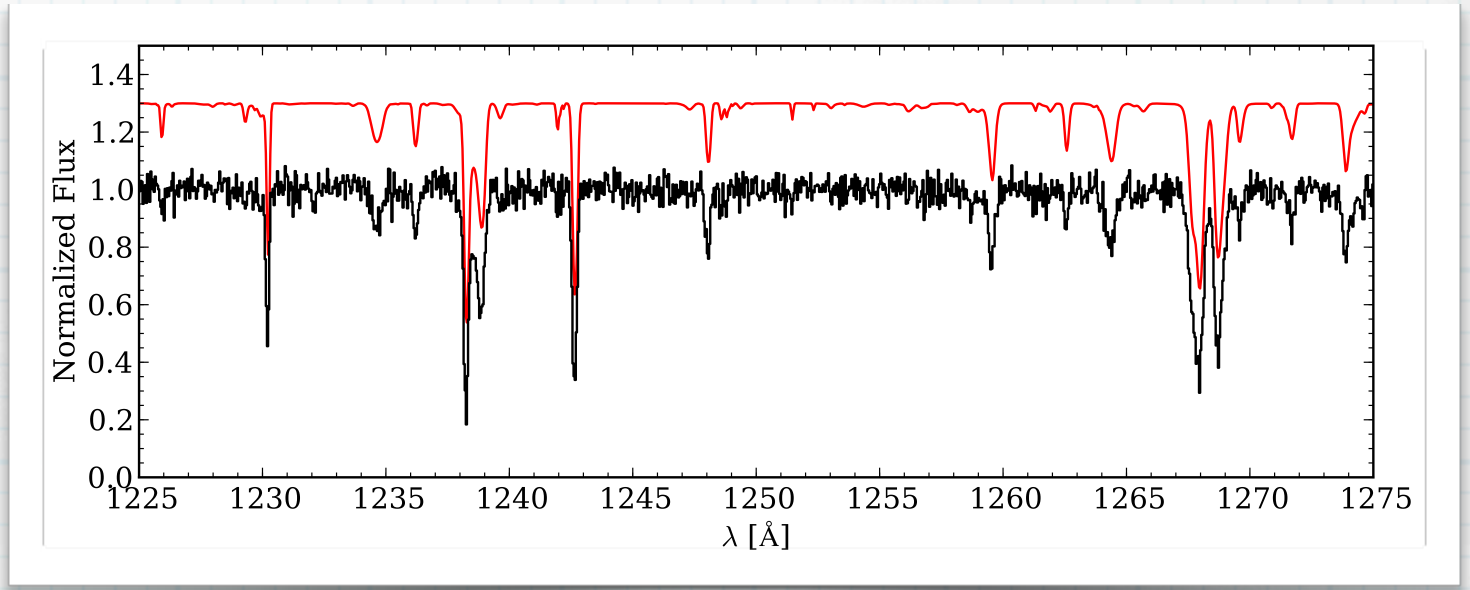
FoF
HOP
Rockstar

er Formats

rray data
fits

Image: Devin Silvia

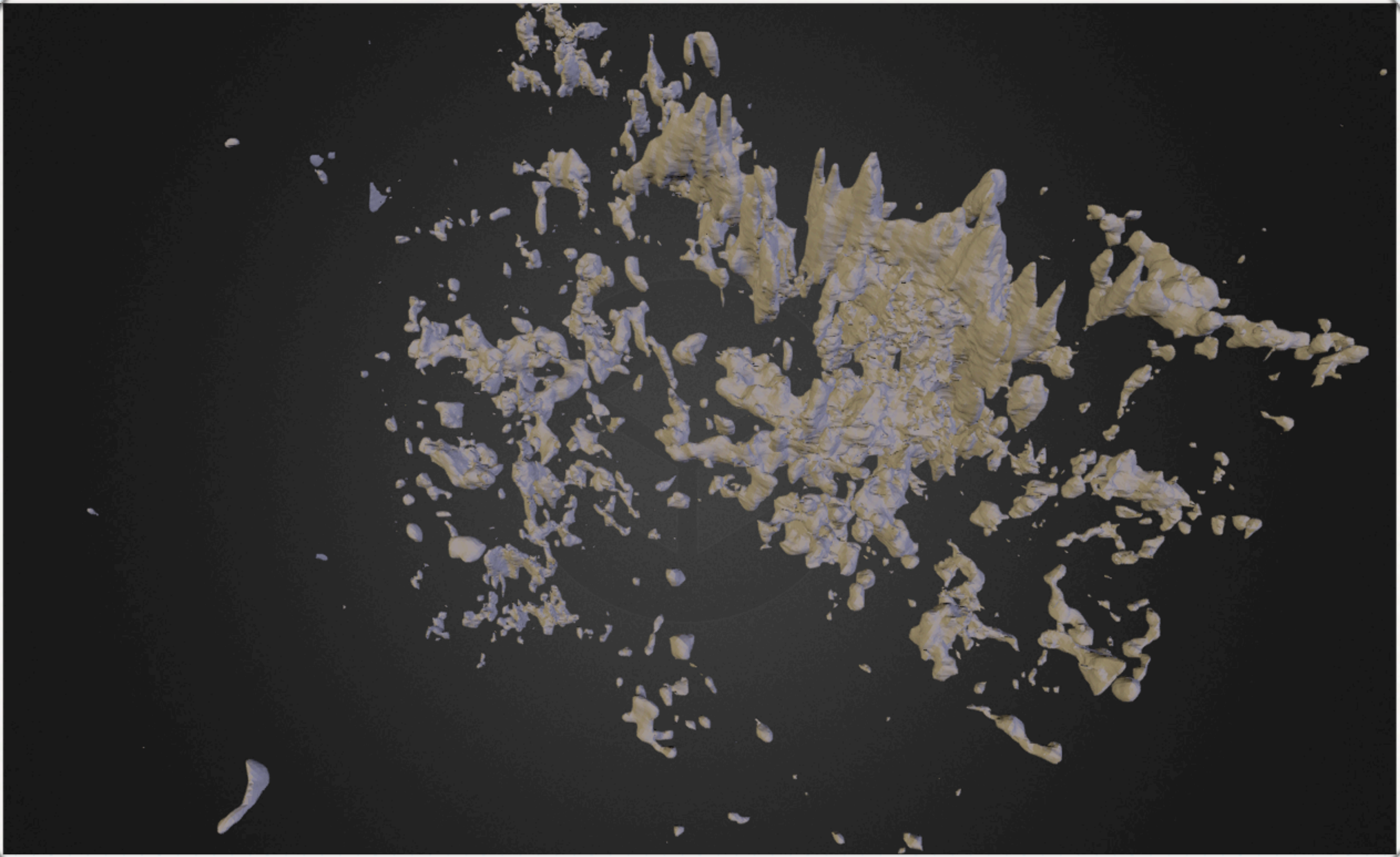
Applications
(analysis modules)



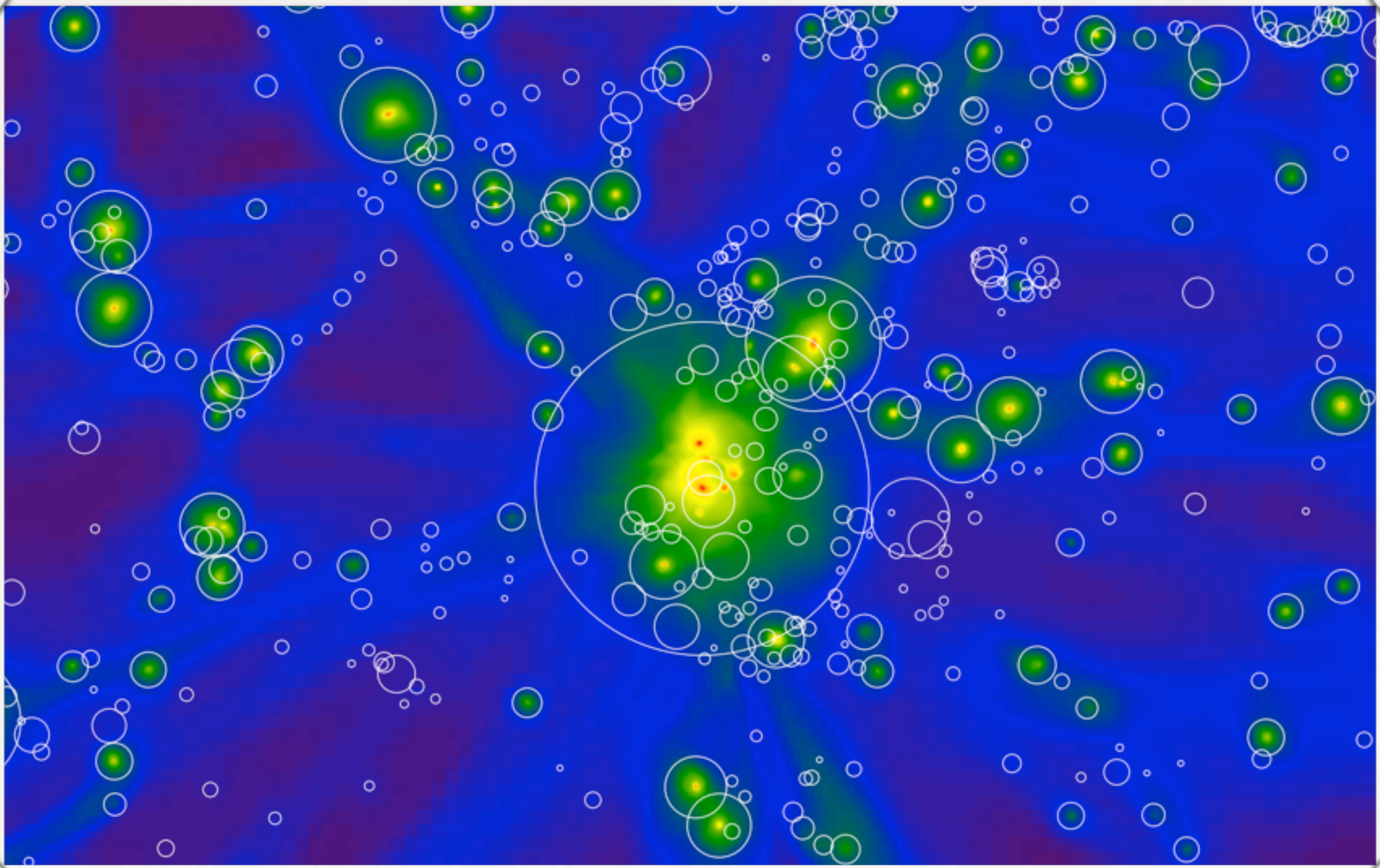
- ☑ absorption spectra
- ☑ contouring/clump finding
- ☑ emission maps
- ☑ export to renderers
- ☑ light cones
- ☑ mock SZ
- ☑ mock X-ray observatories
- ☑ particle trajectories
- ☑ PPV fits cubes

(we are very interested in synthetic observation)

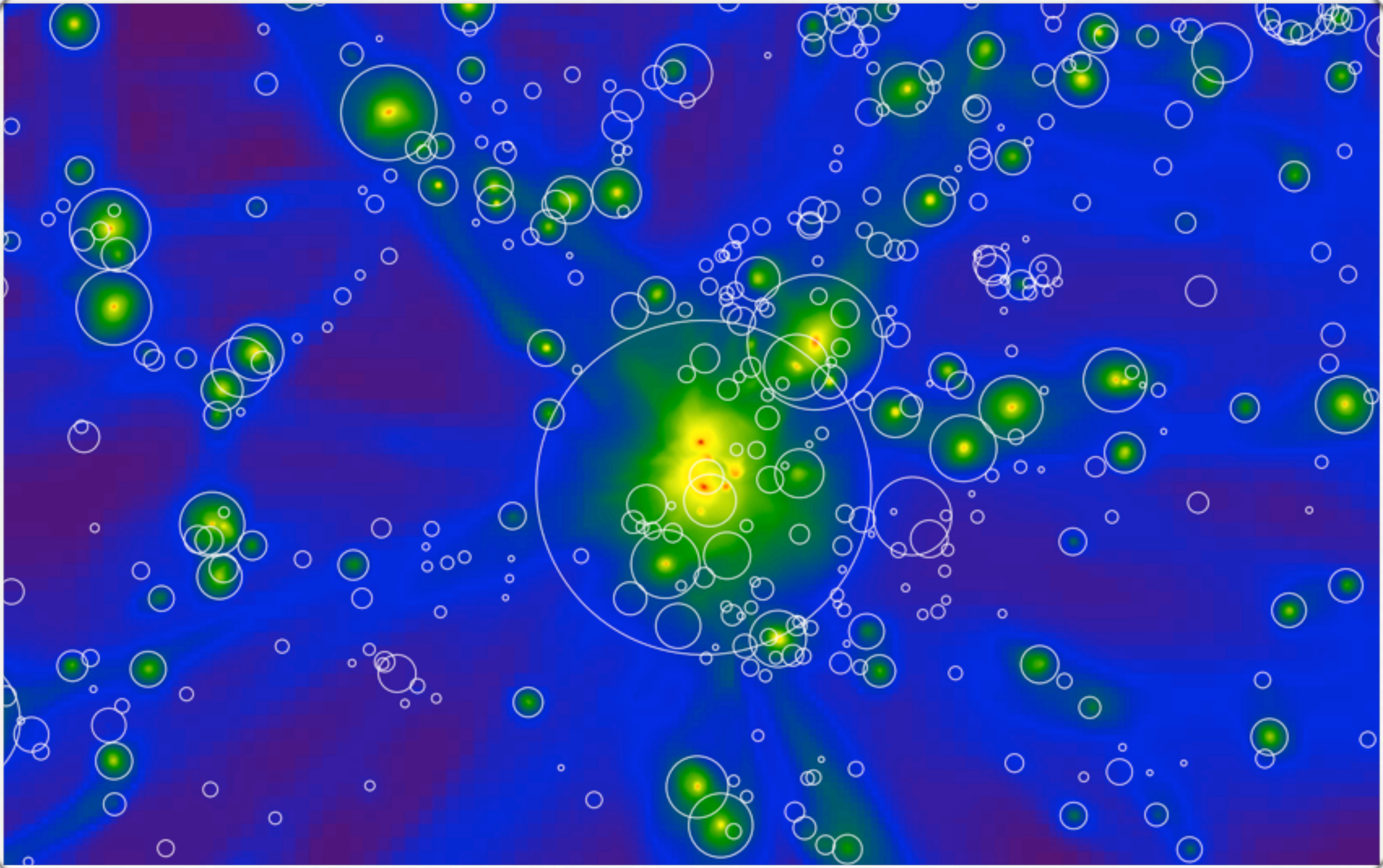
Adam Ginsberg Slide



Halos
(Britton's favorite)



Data Containers



Halo Catalogs

Halo finder outputs are loadable datasets...

```
import yt

ds = yt.load("rockstar_halos/halos_64.0.bin")
ad = ds.all_data()

print ad["virial_radius"]
[ 556.86444092  452.42440796  215.99993896   93.48892212
  484.66888428  81.67009735   81.67009735] kpccm/h
```

- ☑ FOF
- ☑ Hop
- ☑ Rockstar

Halo Analysis

Make your own halo analysis pipeline with the **HaloCatalog.**

```
from yt.analysis_modules.halo_analysis.api import \
    HaloCatalog

data_ds  = yt.load("DD0064/DD0064")
halos_ds = yt.load("rockstar_halos/halos_64.0.bin")

hc = HaloCatalog(data_ds=data_ds, halos_ds=halos_ds)
```

Add:

- ☑ Callbacks
- ☑ Filters
- ☑ Quantities

Halo Analysis

Callbacks analyze, alter, or save data from a single halo.

```
hc = HaloCatalog(data_ds=data_ds, halos_ds=halos_ds)

hc.add_callback("sphere")

hc.add_callback("profile", "radius",
                ["overdensity", "matter_mass"])
```

```
def halo_sphere(halo, ...):
    "Create a sphere data container."

    dds = halo.halo_catalog.data_ds
    sphere = dds.sphere(center, radius)
    halo.data_object = sphere

add_callback("sphere", halo_sphere)
```

Halo Analysis

Callbacks analyze, alter, or save data from a single halo.

```
hc = HaloCatalog(data_ds=data_ds, halos_ds=halos_ds)

hc.add_callback("sphere")

hc.add_callback("profile", "radius",
                ["overdensity", "matter_mass"])

hc.add_callback("virial_quantities",
                ["radius", "matter_mass"])
```

Halo Analysis

Filters return True or False to keep or remove halos from the catalog.

```
hc.add_filter("quantity_value",
              "matter_mass_200", ">", 1e12, "Msun")

hc.add_filter("random")
```

```
def fifty_fifty(halo):
    "Filter halos by a quantity."

    return np.random.random() > 0.5

add_filter("random", fifty_fifty)
```

Halo Analysis

Quantities return a value or values associated with a halo property.

```
hc.add_quantity("spin_parameter")
```

```
def spin_parameter(halo):
    "Halo spin parameter."

    object = halo.data_object
    return object.quantities.spin_parameter()

add_quantity("spin_parameter", spin_parameter)
```

Halo Analysis

Quantities are accessible later in the pipeline and are saved at the end.

```
hc.add_callback("print_spin")
```

```
def print_spin(halo):
    "Print the spin parameter."

    print halo.quantities["spin_parameter"]

add_callback("print_spin", print_spin)
```

Halo Analysis

All actions are performed in order on each halo.

```
hc.create()
```

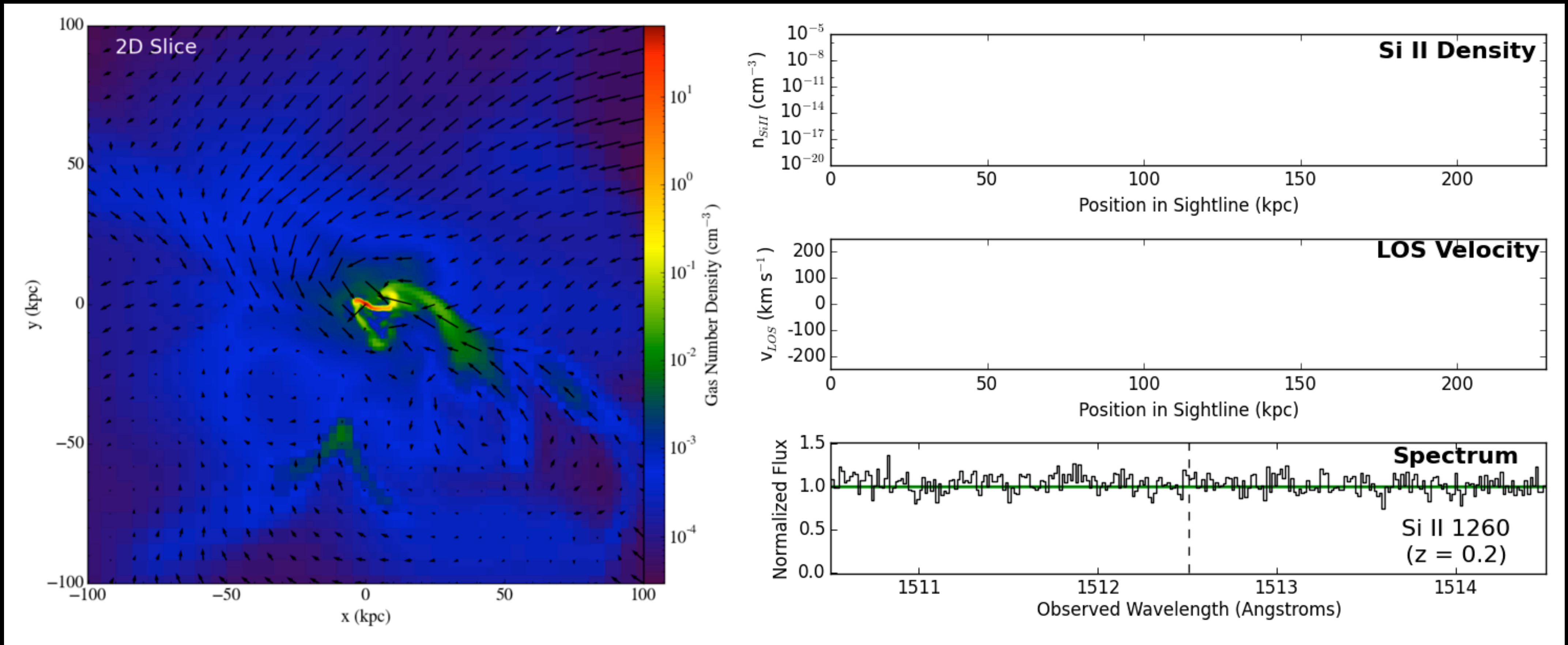
Halo Analysis

HaloCatalogs are loadable datasets...

```
ds = yt.load("catalog_0064/catalog_0064.0.h5")

ad = ds.h.all_data()

print ad["stellar_mass"].in_units("g")
[  3.45200213e+44   0.00000000e+00   1.33784869e+45
   6.18495540e+44   0.00000000e+00   9.65736532e+44] g
```

Spin-offs

Trident: a synthetic spectral generation utility built on yt

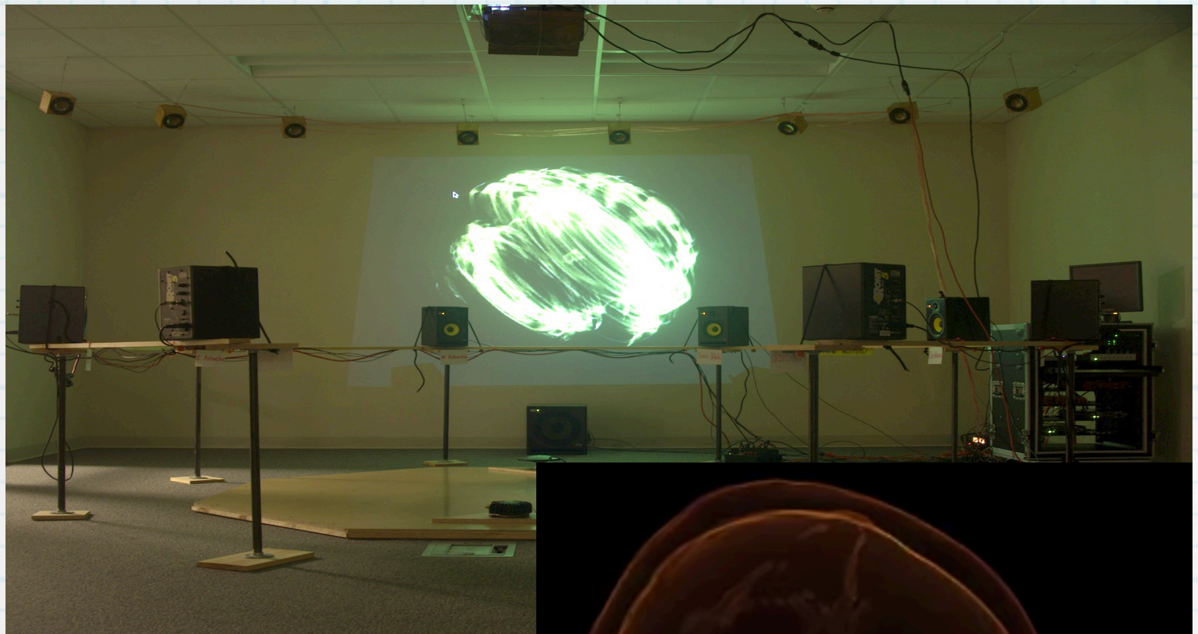


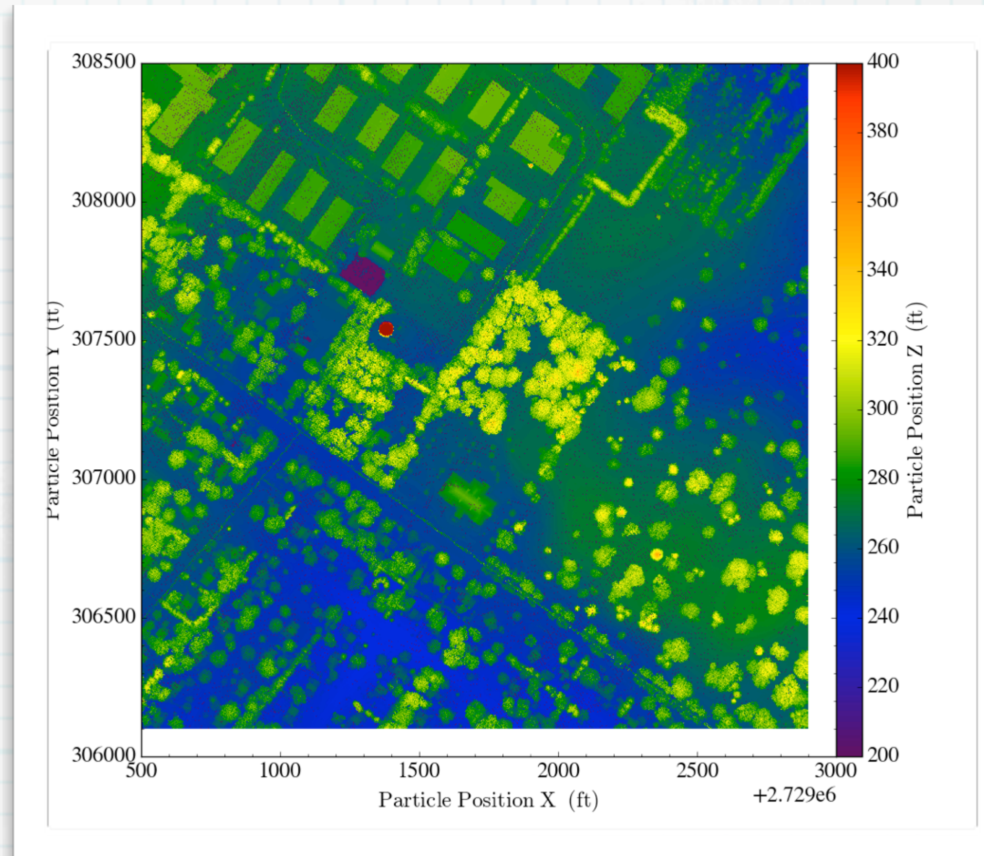
Cameron Hummels (Arizona)
Devin Silvia (Michigan State)
Britton Smith (Edinburgh)

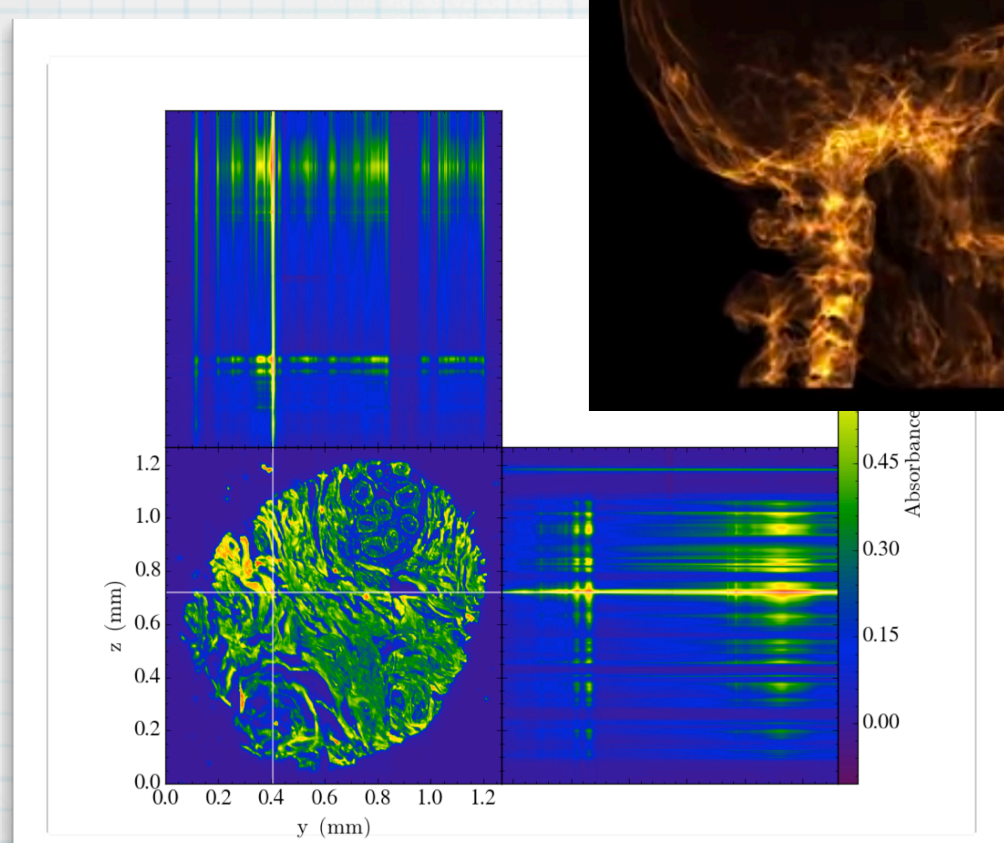
http://trident-project.org/

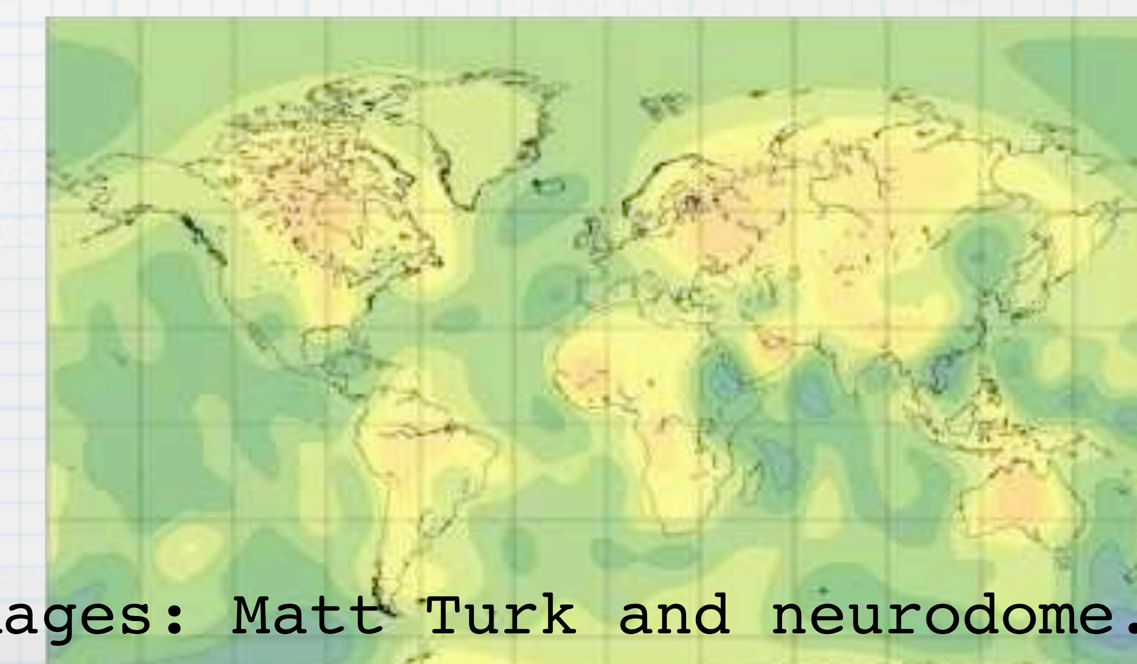
(yt is a good place to build something interdisciplinary)

Not Astronomy









Images: Matt Turk and neurodome.org

Coming Soon

(maybe with your help)

- ☑ improved scalability for particle data
- ☑ unstructured mesh
- ☑ new and better volume renderer
- ☑ support for more formats
- ☑ more connections to observation

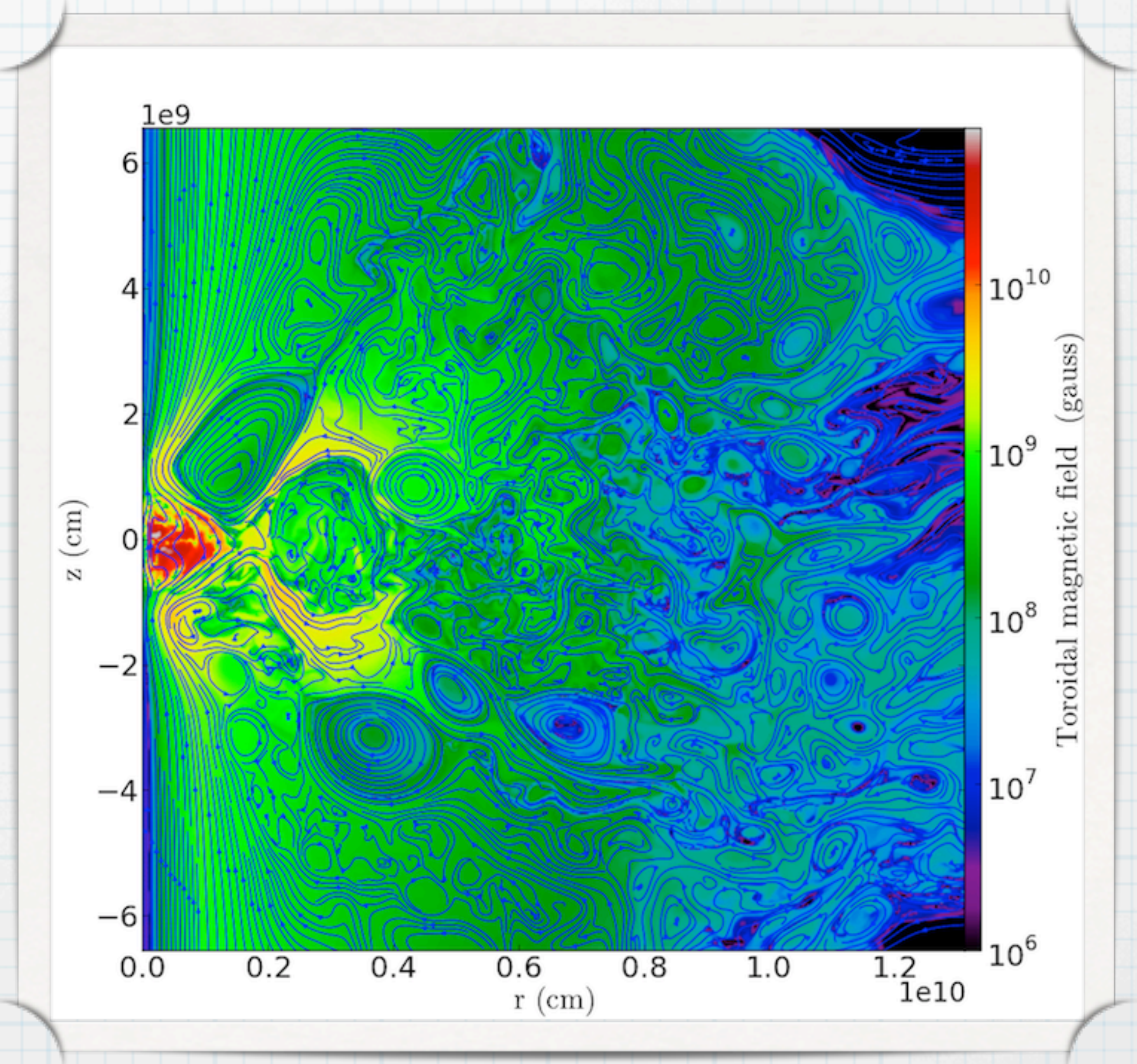


Image: Suoqing Ji

Code and Community

- yt-project.org
  bitbucket.org/yt_analysis/yt
- 16,573 commits made by 94 contributors since February 2007
- ~44% by Matt Turk, but many heavily invested developers
- 307 on yt-users
  91 on yt-dev
- #yt on irc.freenode.net
- Funding from NSF and Gordon and Betty Moore Foundation

Credit

Turk, M.J., Smith, B.D., Oishi, J.S., Skory, S., Skillman, S.W., Abel, T., Norman, M.L, 2011, **yt: A Multi-code Analysis Toolkit for Astrophysical Simulation Data**, ApJS, 192, 9

- 168 citations
- Inadequate representation of community

yt-3 paper coming soon...

Credit

ut Docs Community Develop Gallery Project Members Quick Links

Project Members

In September of 2014, a discussion on the yt-dev mailing list about project governance resulted in the development of a YTEP on the topic of project governance. As an outcome of that, the community decided to establish a "membership" process, whereby individuals who had contributed in a significant way to the project were recognized and identified as members.

Kenza Arraki

Member since 2014. Kenza has been a yt project developer since 2013. Her main contributions have been to the ART frontend and she is currently the ART code liason.

23 members for life
and counting

Hilary Egan

Member since 2014. Hilary began developing yt in 2013. She created and maintains the absorption spectrum fitting tool. She has also been involved in developing the new halo analysis framework.

Thank You

yt-project.org

yt-project.org/gallery.html

yt-project.org/data

Tom Abel
Gabriel Altay
Kenza Arraki
Elliott Biondo
Alex Bogert
Pengfei Chen
David Collins
Brian Crosby
Andrew Cunningham
Hilary Egan
John Forbes
Sam Geen
Nathan Goldbaum
William Gray
Eric Hallman
Markus Haider
Cameron Hummels

Christian Karch
Benjamin Keller
Ji-hoon Kim
Steffen Klemer
Kacper Kowalik
Mark Krumholz
Michael Kuhlen
Eve Lee
Sam Leitner
Yuan Li
Chris Malone
Josh Moloney
Chris Moody
Stuart Mumford
Andrew Myers
Jill Naiman

Desika Narayanan
Kaylea Nelson
Jeff Oishi
Jean-Claude Passy
John Regan
Sherwood Richers
Mark Richardson
Thomas Robitaille
Anna Rosen
Douglas Rudd
Anthony Scopatz
Noel Scudder
Devin Silvia
Sam Skillman
Stephen Skory
Aaron Smith

Britton Smith
Geoffrey So
Casey Stark
Antoine Strugarek
Ji Suoqing
Elizabeth Tasker
Benjamin Thompson
Stephanie Tonnesen
Matthew Turk
Miguel de Val-Borro
Rick Wagner
Mike Warren
Andrew Wetzel
John Wise
Mike Zingale
John ZuHone