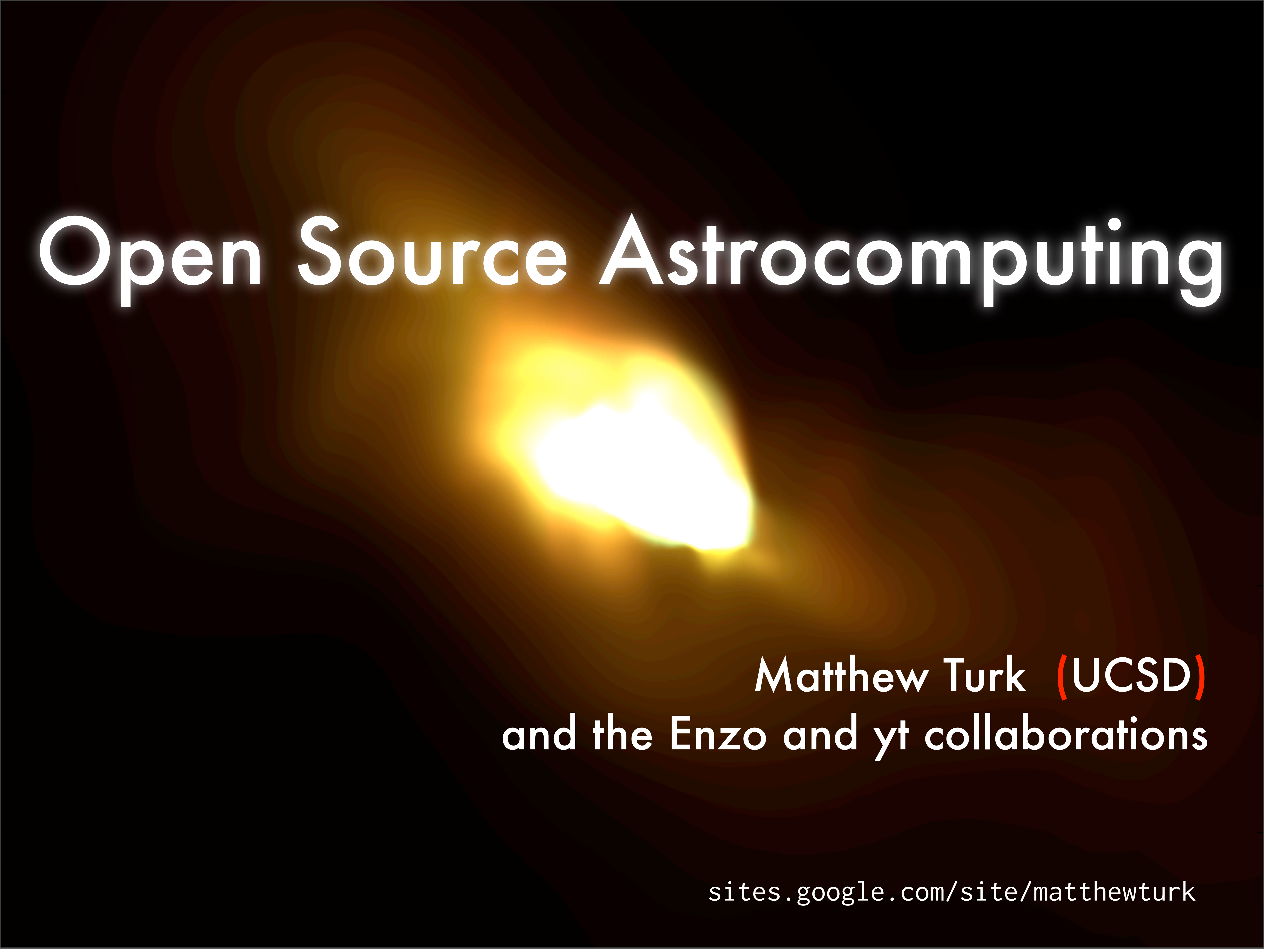


Open Source Astrocomputing

Matthew Turk (UCSD)
and the Enzo and yt collaborations

sites.google.com/site/matthewturk

“Future of Astrocomputing”

I wanted to present today about what I think, as a relatively new researcher, the future of Astrocomputing is going to be characterized by. Not highly scalable problems, not a rethinking of parallelism, not GPUs or databases or PGAS languages, but rather a sociological issue.

Reproducibility & Collaboration

The future of astrocomputing absolutely must be focused on the generation of sustainable mechanisms for reproducibility of results and collaboration between research groups. This will never be a completed goal; the idea of consolidation of astrophysical simulation codes is anathema to verification and validation of results. However, the means for participation for new researchers, for verification and validation, and for the broadening of participation in astrophysical computation will require a consistent focus on encouraging reproducibility and collaboration.

Open Source

And, to put it simply, the only feasible way to encourage reproducibility and collaboration is through the application of Open Source philosophy. (As a side note, in general I have a personal resistance to the usage of “Open Source” over the terminology “Free Software” -- however, for the purposes of this talk, I will concede the territory and utilize those words.)

The application of Open Source principles to astrophysical computation is more than just tossing up a tarball on a website and setting up a mailing list. It requires a rethinking of the mechanism for outreach and engagement of a community.

To that end, I would like to discuss two case studies: that of Enzo, an astrophysical simulation code, and that of yt, a code designed for the analysis and visualization of astrophysical data. However, before doing so, I would like to that the time to identify three common objections that I have heard raised about open source computing in scientific fields of study.

1.

Does Open Source remove my edge on the competition?

The first of these three objections is that of the competitive advantage. Does making available the ability to run simulations, particularly new and exciting types of simulations, prevent you from being competitive academically? Will other people -- the imagined vast, ravenous hordes of people watching every commit on a source code repository -- simply steal out your methods and code, and use it to their own advantage?

2.

What about issues of correctness?

The second speaks to an insecurity, one that I have heard expressed quite often, and one that I too have thought on occasion. Does providing the means of verification and validation of a piece of simulation code provide also the ammunition for others to discredit a model, publish a paper lambasting your work, or even simply identify flaws and marginalize your work.

3.

Do support structures encumber productivity?

And finally, “What about all the emails?” Does providing an open source code give license to everyone who downloads it to pester you endlessly? And, more specifically -- if the type of Open Source Methodology that you use is truly a mechanism for community engagement, rather than source code distribution, won’t it become unbearable to shepherd external users?

Lone Coder

Shared Source

Closed Collaboration

Open Source

Most open source scientific codes follow a standard trajectory: a single person working in isolation, who ends up sharing the source with some close collaborators, and then perhaps an ultimate open sourcing of the code to the public.

Lone Coder

Shared Source

Closed Collaboration

Open Source

Most open source scientific codes follow a standard trajectory: a single person working in isolation, who ends up sharing the source with some close collaborators, and then perhaps an ultimate open sourcing of the code to the public.

Lone Coder

Shared Source

Closed Collaboration

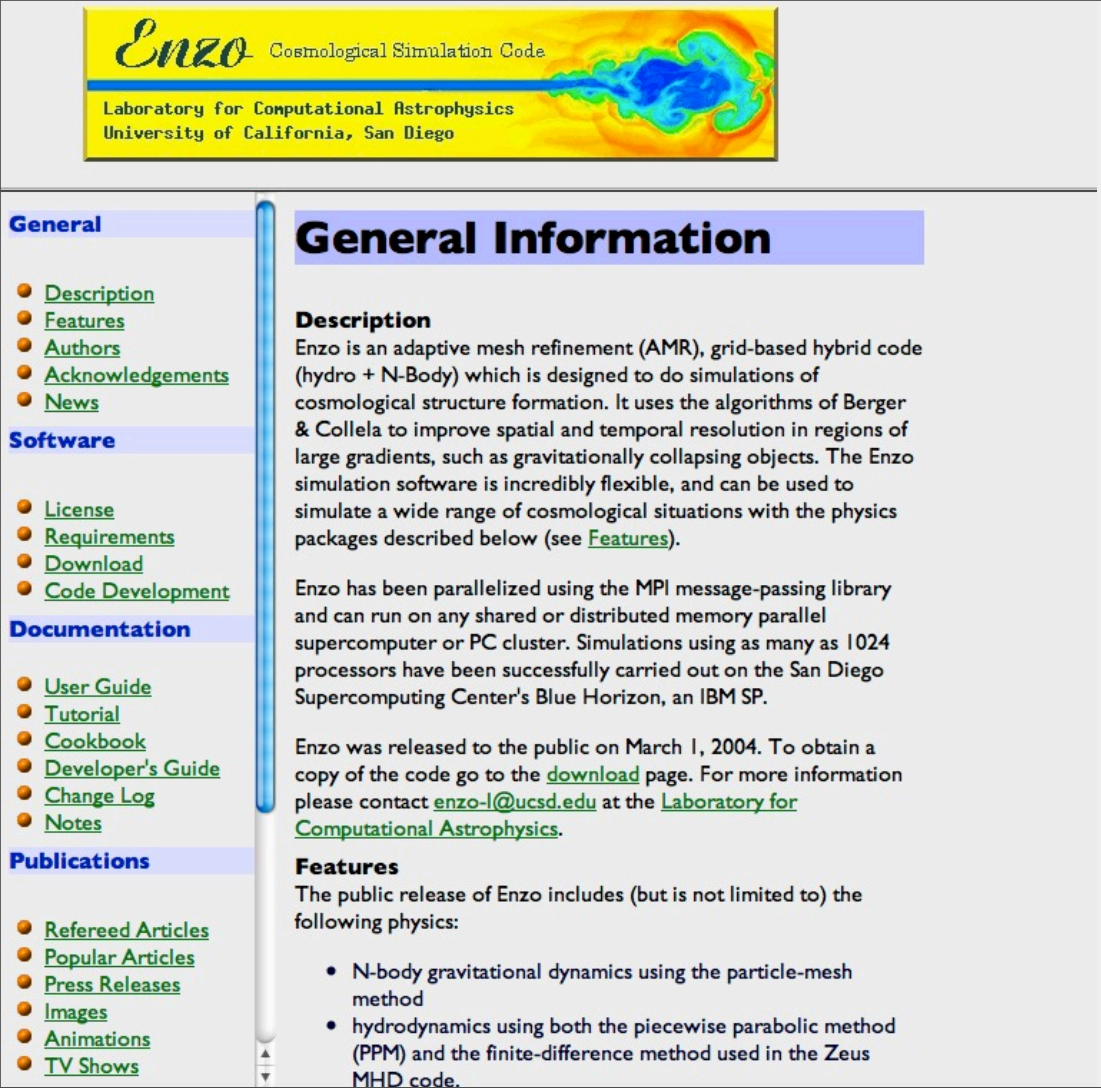
Open Source

I’ll discuss the process by which Enzo moved to Open Source, and how it has benefited from that process.

enzo

I'd like to first star by discussing the case study of Enzo. Enzo is an astrophysical simulation code, originally written by Greg Bryan, which has been stewarded by Mike Norman at the LCA for many years. Mike is a pioneer in developing open source codes, and without him the Enzo community would not be what it is today. But rather than starting with a discussion of where the Enzo code is today, I'd like to step back and take a look at how it got to be what it is.

When I was at Penn State in 2003, working with Tom Abel, I was handed a tarball called enzo.tar.gz.


Enzo Cosmological Simulation Code

Laboratory for Computational Astrophysics
University of California, San Diego

General

- Description
- Features
- Authors
- Acknowledgements
- News

Software

- License
- Requirements
- Download
- Code Development

Documentation

- User Guide
- Tutorial
- Cookbook
- Developer's Guide
- Change Log
- Notes

Publications

- Refereed Articles
- Popular Articles
- Press Releases
- Images
- Animations
- TV Shows

General Information

Description
Enzo is an adaptive mesh refinement (AMR), grid-based hybrid code (hydro + N-Body) which is designed to do simulations of cosmological structure formation. It uses the algorithms of Berger & Collela to improve spatial and temporal resolution in regions of large gradients, such as gravitationally collapsing objects. The Enzo simulation software is incredibly flexible, and can be used to simulate a wide range of cosmological situations with the physics packages described below (see Features).

Enzo has been parallelized using the MPI message-passing library and can run on any shared or distributed memory parallel supercomputer or PC cluster. Simulations using as many as 1024 processors have been successfully carried out on the San Diego Supercomputing Center's Blue Horizon, an IBM SP.

Enzo was released to the public on March 1, 2004. To obtain a copy of the code go to the download page. For more information please contact enzo-l@ucsd.edu at the Laboratory for Computational Astrophysics.

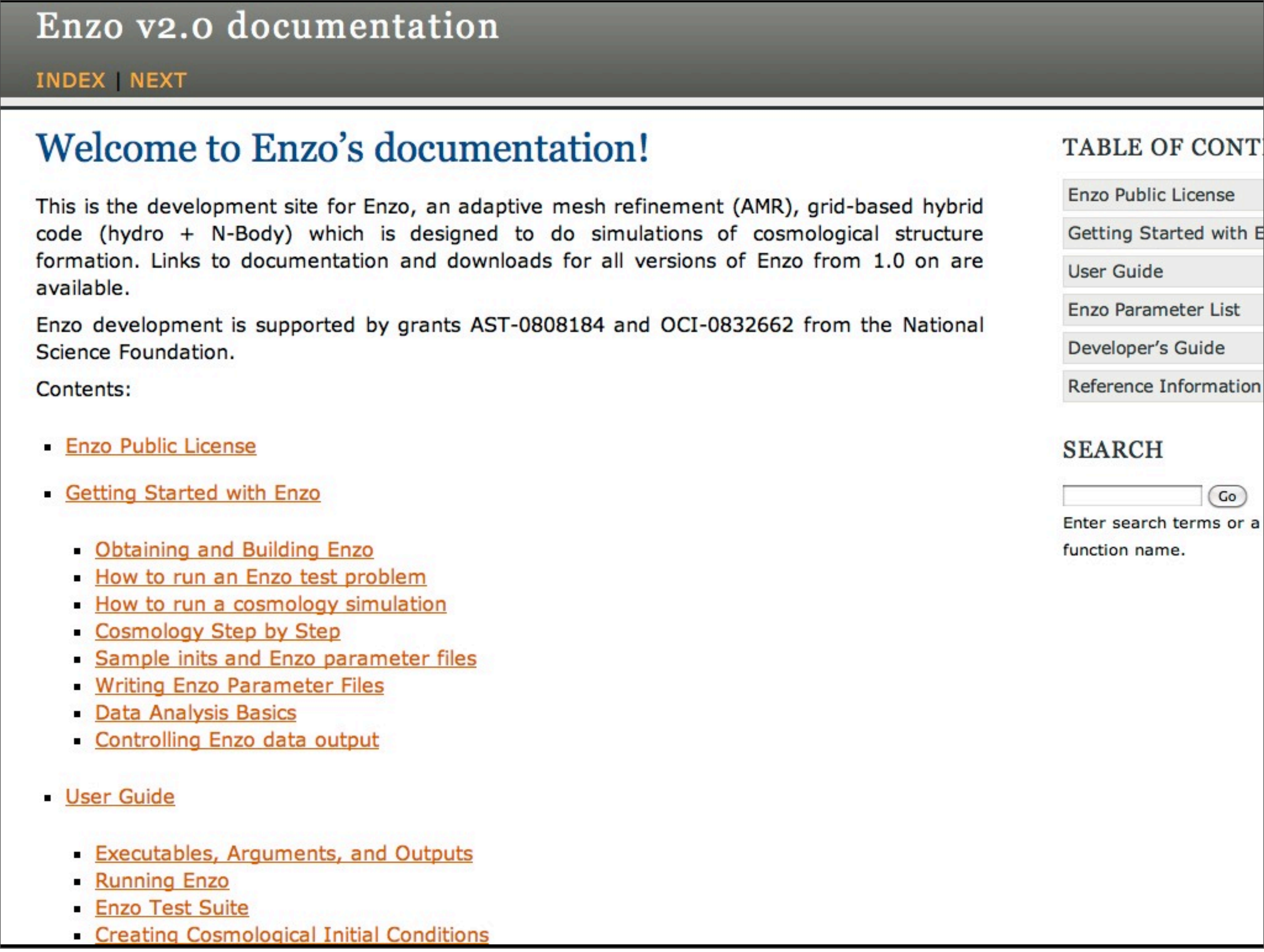
Features
The public release of Enzo includes (but is not limited to) the following physics:

- N-body gravitational dynamics using the particle-mesh method
- hydrodynamics using both the piecewise parabolic method (PPM) and the finite-difference method used in the Zeus MHD code.

That tarball was enormous. By that time, while Enzo was not yet publicly available, the manual was online, the cookbook was online, and the support structures for asking questions were in place -- thanks to Mike Norman, Greg Bryan, and Brian O'Shea.

But even so, I was not only new to Enzo, I was new to graduate school and new to simulations on the whole. I was good with computers, so that was in my favor, but it was still a large undertaking. Without the infrastructure that had been built around it, it would have been hopeless.

Back in the day, the manual consisted of a website.


Enzo v2.0 documentation

INDEX | NEXT

Welcome to Enzo's documentation!

This is the development site for Enzo, an adaptive mesh refinement (AMR), grid-based hybrid code (hydro + N-Body) which is designed to do simulations of cosmological structure formation. Links to documentation and downloads for all versions of Enzo from 1.0 on are available.

Enzo development is supported by grants AST-0808184 and OCI-0832662 from the National Science Foundation.

Contents:

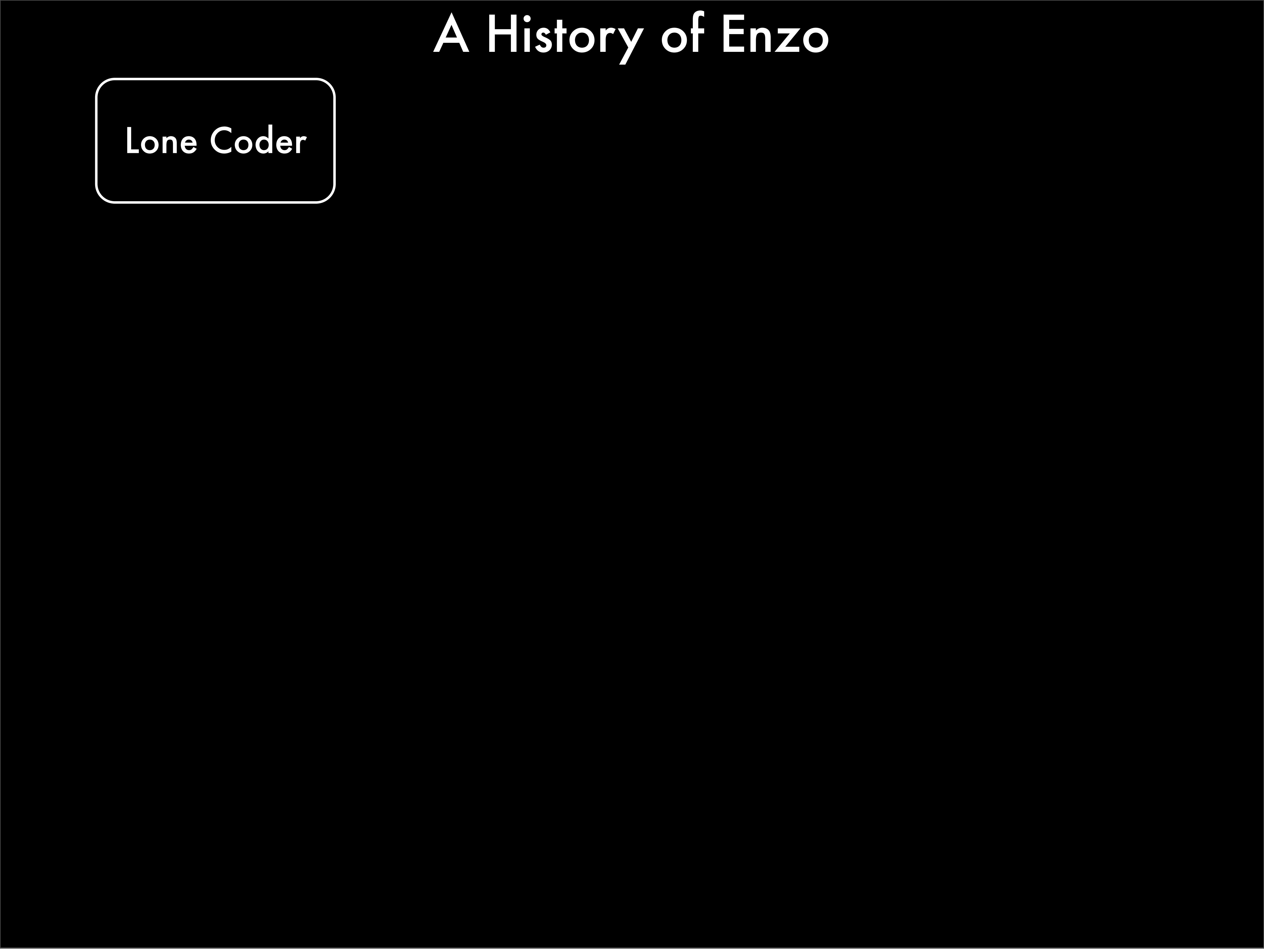
That's still true today! It's gotten a facelift, and a bunch of added content, but it's still a website that has information, pointers to other resources, and a guide to the source code.

A History of Enzo

Enzo, like many different codes, followed a standard track of development. The initial version was written by a lone coder: unlike, say, the FLASH code or CASTRO, Enzo was originally written by Greg Bryan by himself. This seems to be the rule more than the exception in astrophysical codes. At some point, Greg shared the code with others in his research group and his other collaborators, and then this became a closed collaboration. This code was then open sourced and made freely available to the public.

However, even though that's the main riverbed, there were several forks and tributaries that complicate the story.

The open source branch of Enzo was primarily characterized by two classes of individuals: those with repository access, and those without. While changes often flowed downstream from the developers, it was much rarer for there to be crosstalk downstream or, in particular, sharing of changes back with the primary developers. While bugfixes would occasionally make their way back up, major physics modules never did. In the closed collaboration, however, code sharing was common: even though the source control practices did not really lend themselves to this (it was usually a bunch of tarballs being passed around!) routines and physics modules were shared, and a great deal of crosstalk occurred.



A History of Enzo

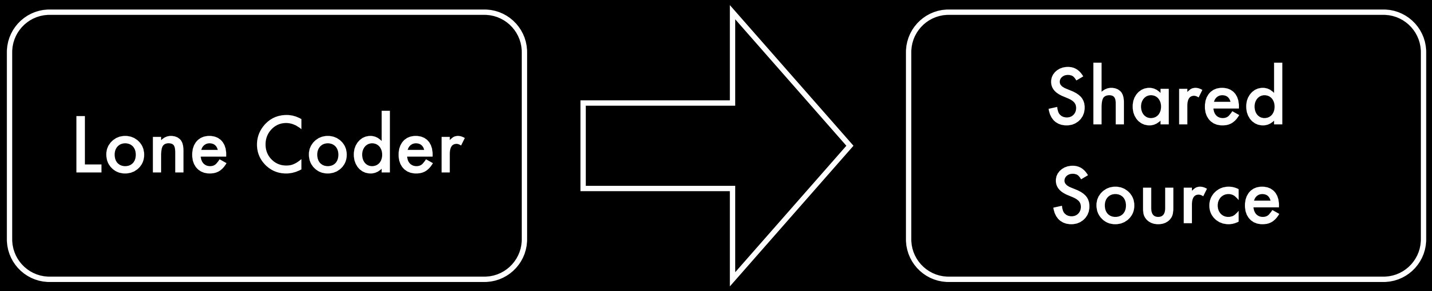
Lone Coder

Enzo, like many different codes, followed a standard track of development. The initial version was written by a lone coder: unlike, say, the FLASH code or CASTRO, Enzo was originally written by Greg Bryan by himself. This seems to be the rule more than the exception in astrophysical codes. At some point, Greg shared the code with others in his research group and his other collaborators, and then this became a closed collaboration. This code was then open sourced and made freely available to the public.

However, even though that's the main riverbed, there were several forks and tributaries that complicate the story.

The open source branch of Enzo was primarily characterized by two classes of individuals: those with repository access, and those without. While changes often flowed downstream from the developers, it was much rarer for there to be crosstalk downstream or, in particular, sharing of changes back with the primary developers. While bugfixes would occasionally make their way back up, major physics modules never did. In the closed collaboration, however, code sharing was common: even though the source control practices did not really lend themselves to this (it was usually a bunch of tarballs being passed around!) routines and physics modules were shared, and a great deal of crosstalk occurred.

A History of Enzo

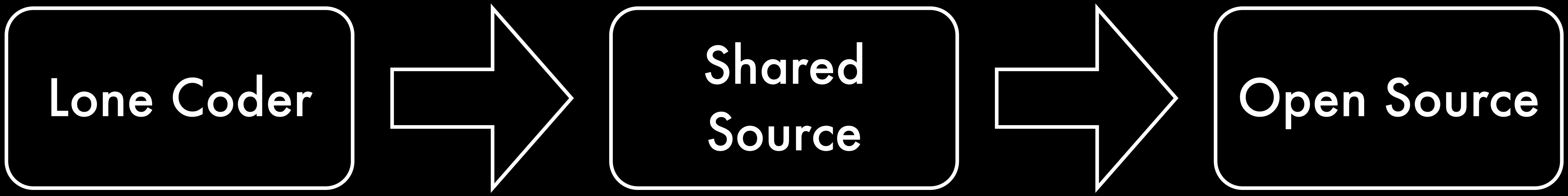


Enzo, like many different codes, followed a standard track of development. The initial version was written by a lone coder: unlike, say, the FLASH code or CASTRO, Enzo was originally written by Greg Bryan by himself. This seems to be the rule more than the exception in astrophysical codes. At some point, Greg shared the code with others in his research group and his other collaborators, and then this became a closed collaboration. This code was then open sourced and made freely available to the public.

However, even though that's the main riverbed, there were several forks and tributaries that complicate the story.

The open source branch of Enzo was primarily characterized by two classes of individuals: those with repository access, and those without. While changes often flowed downstream from the developers, it was much rarer for there to be crosstalk downstream or, in particular, sharing of changes back with the primary developers. While bugfixes would occasionally make their way back up, major physics modules never did. In the closed collaboration, however, code sharing was common: even though the source control practices did not really lend themselves to this (it was usually a bunch of tarballs being passed around!) routines and physics modules were shared, and a great deal of crosstalk occurred.

A History of Enzo

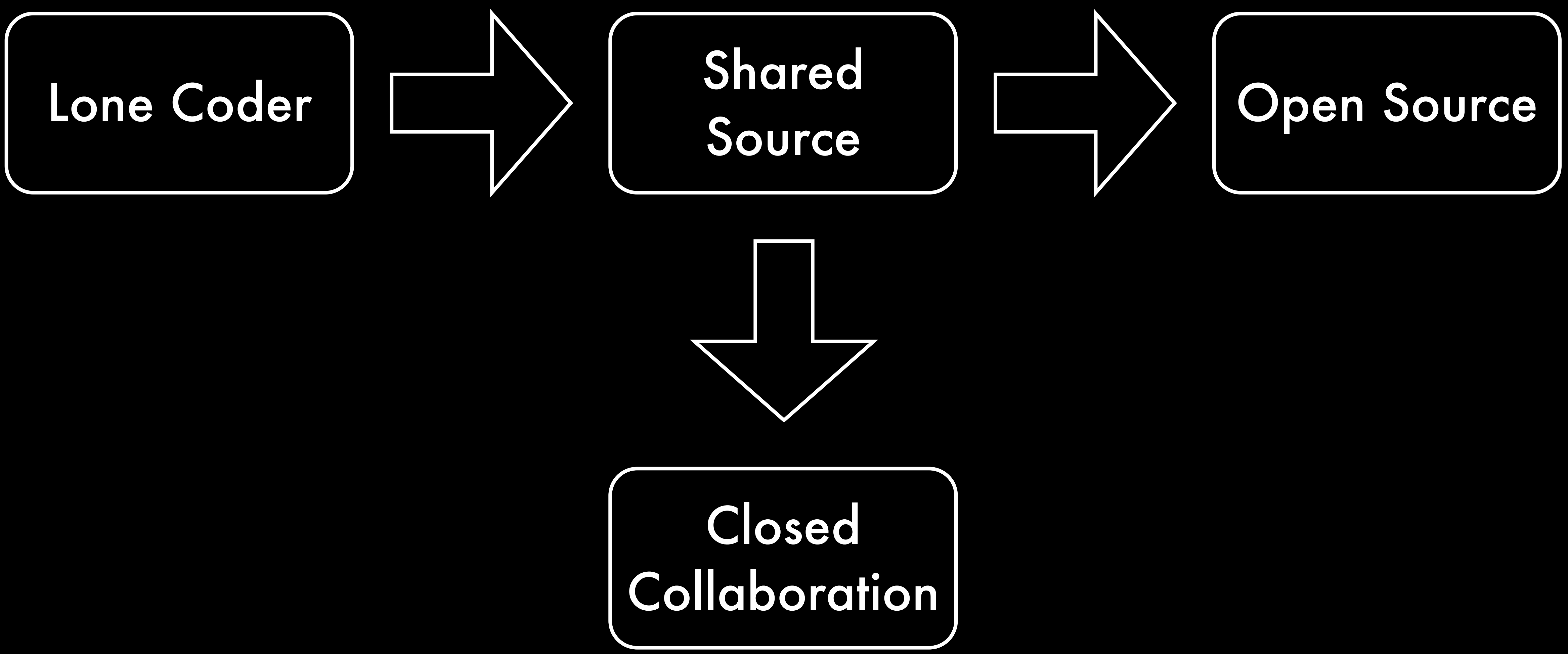


Enzo, like many different codes, followed a standard track of development. The initial version was written by a lone coder: unlike, say, the FLASH code or CASTRO, Enzo was originally written by Greg Bryan by himself. This seems to be the rule more than the exception in astrophysical codes. At some point, Greg shared the code with others in his research group and his other collaborators, and then this became a closed collaboration. This code was then open sourced and made freely available to the public.

However, even though that's the main riverbed, there were several forks and tributaries that complicate the story.

The open source branch of Enzo was primarily characterized by two classes of individuals: those with repository access, and those without. While changes often flowed downstream from the developers, it was much rarer for there to be crosstalk downstream or, in particular, sharing of changes back with the primary developers. While bugfixes would occasionally make their way back up, major physics modules never did. In the closed collaboration, however, code sharing was common: even though the source control practices did not really lend themselves to this (it was usually a bunch of tarballs being passed around!) routines and physics modules were shared, and a great deal of crosstalk occurred.

A History of Enzo

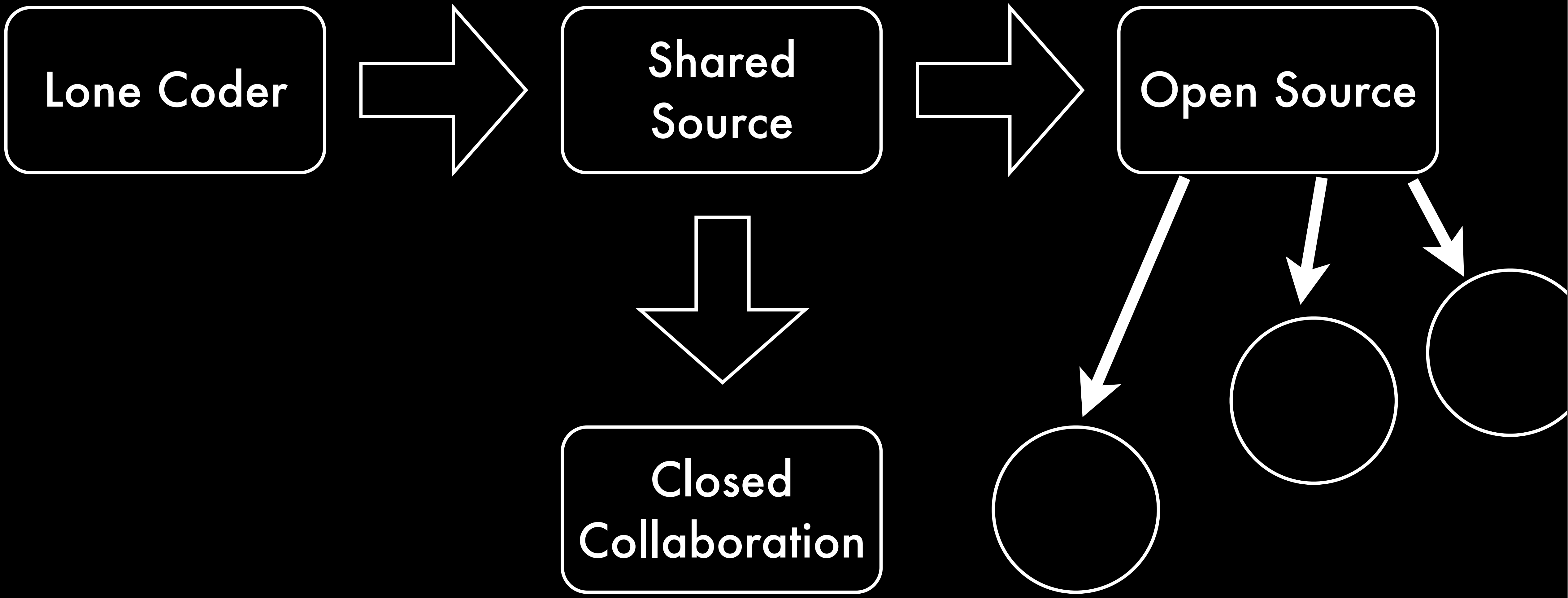


Enzo, like many different codes, followed a standard track of development. The initial version was written by a lone coder: unlike, say, the FLASH code or CASTRO, Enzo was originally written by Greg Bryan by himself. This seems to be the rule more than the exception in astrophysical codes. At some point, Greg shared the code with others in his research group and his other collaborators, and then this became a closed collaboration. This code was then open sourced and made freely available to the public.

However, even though that's the main riverbed, there were several forks and tributaries that complicate the story.

The open source branch of Enzo was primarily characterized by two classes of individuals: those with repository access, and those without. While changes often flowed downstream from the developers, it was much rarer for there to be crosstalk downstream or, in particular, sharing of changes back with the primary developers. While bugfixes would occasionally make their way back up, major physics modules never did. In the closed collaboration, however, code sharing was common: even though the source control practices did not really lend themselves to this (it was usually a bunch of tarballs being passed around!) routines and physics modules were shared, and a great deal of crosstalk occurred.

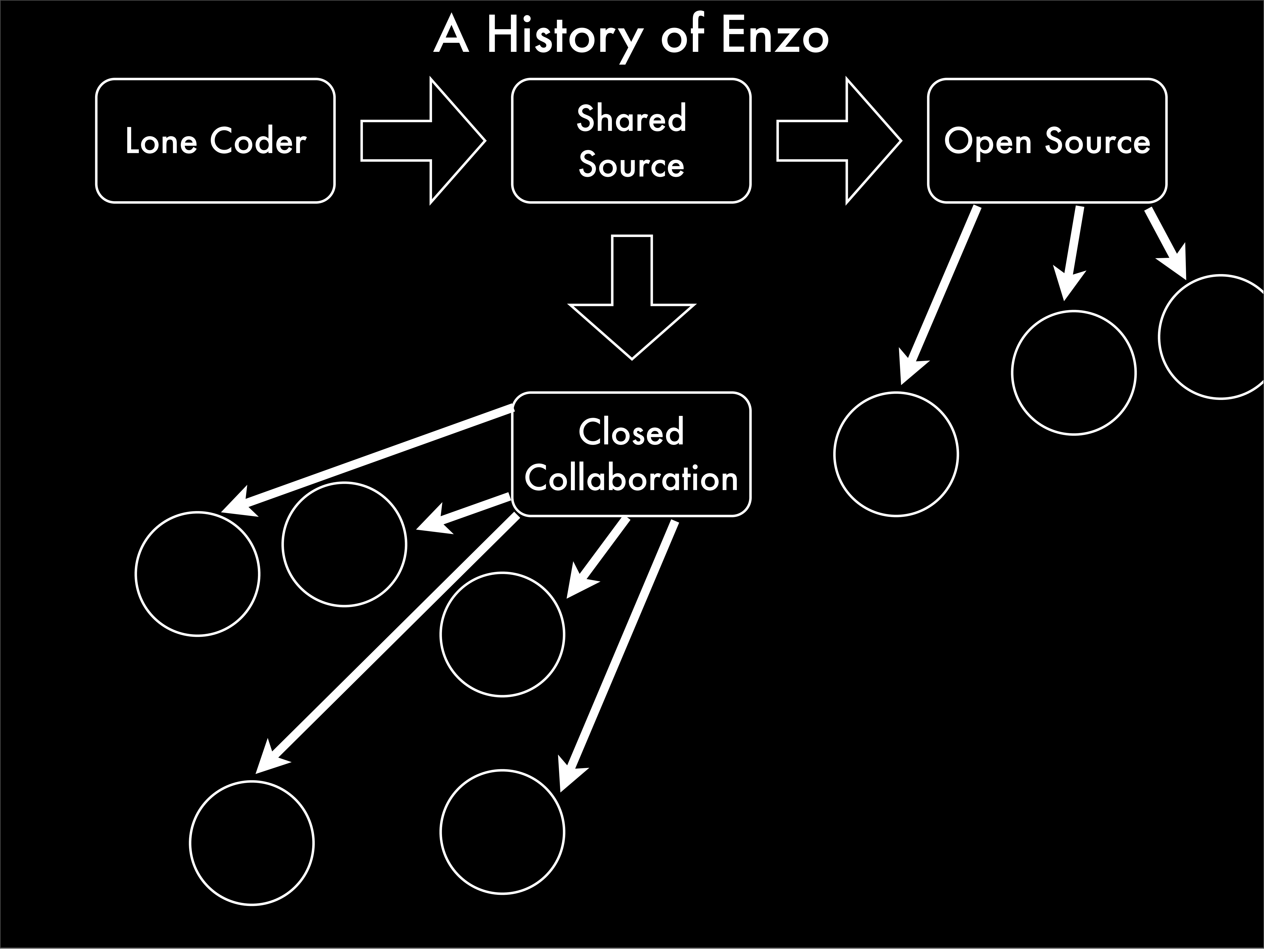
A History of Enzo



Enzo, like many different codes, followed a standard track of development. The initial version was written by a lone coder: unlike, say, the FLASH code or CASTRO, Enzo was originally written by Greg Bryan by himself. This seems to be the rule more than the exception in astrophysical codes. At some point, Greg shared the code with others in his research group and his other collaborators, and then this became a closed collaboration. This code was then open sourced and made freely available to the public.

However, even though that's the main riverbed, there were several forks and tributaries that complicate the story.

The open source branch of Enzo was primarily characterized by two classes of individuals: those with repository access, and those without. While changes often flowed downstream from the developers, it was much rarer for there to be crosstalk downstream or, in particular, sharing of changes back with the primary developers. While bugfixes would occasionally make their way back up, major physics modules never did. In the closed collaboration, however, code sharing was common: even though the source control practices did not really lend themselves to this (it was usually a bunch of tarballs being passed around!) routines and physics modules were shared, and a great deal of crosstalk occurred.

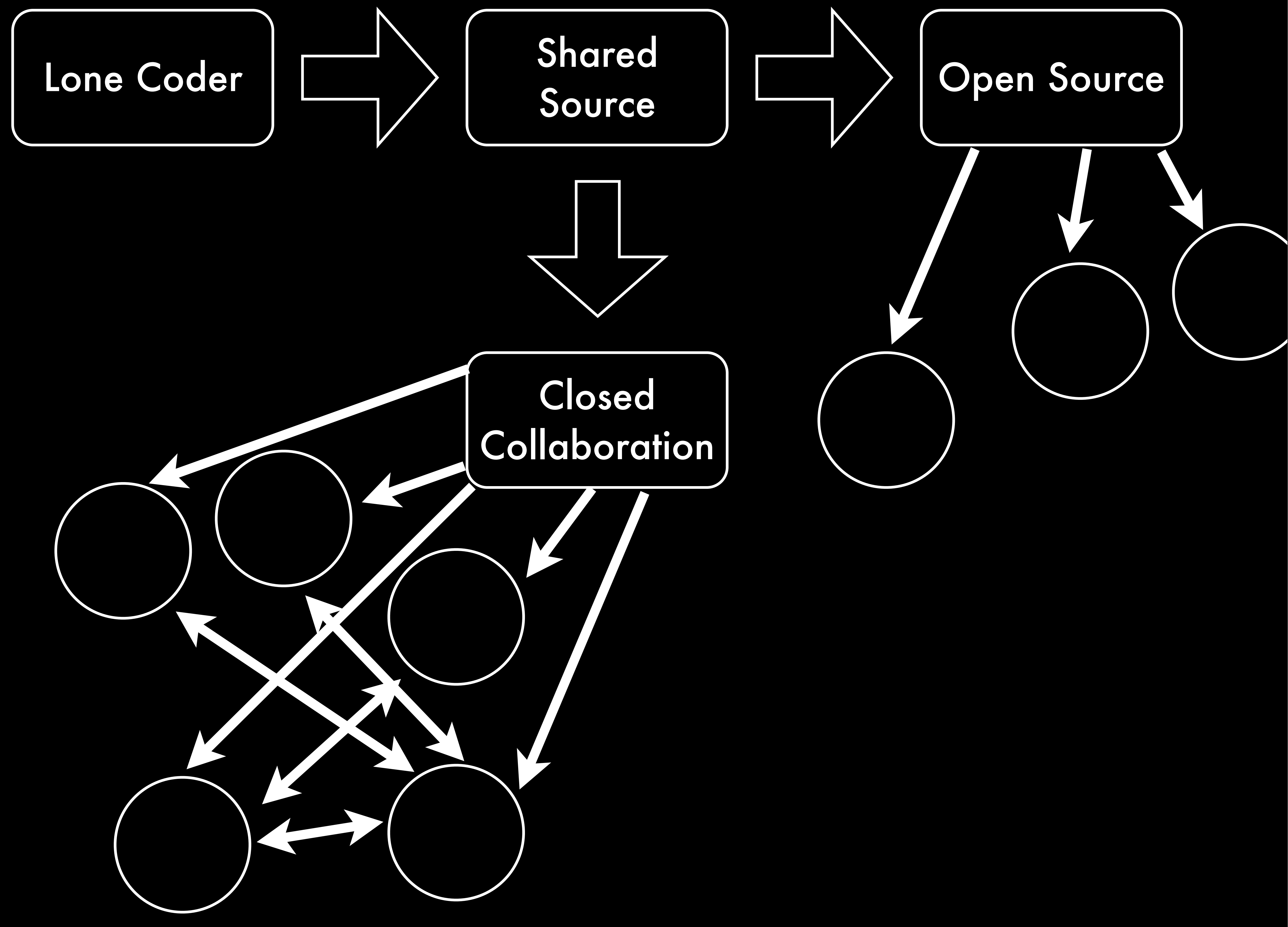


Enzo, like many different codes, followed a standard track of development. The initial version was written by a lone coder: unlike, say, the FLASH code or CASTRO, Enzo was originally written by Greg Bryan by himself. This seems to be the rule more than the exception in astrophysical codes. At some point, Greg shared the code with others in his research group and his other collaborators, and then this became a closed collaboration. This code was then open sourced and made freely available to the public.

However, even though that's the main riverbed, there were several forks and tributaries that complicate the story.

The open source branch of Enzo was primarily characterized by two classes of individuals: those with repository access, and those without. While changes often flowed downstream from the developers, it was much rarer for there to be crosstalk downstream or, in particular, sharing of changes back with the primary developers. While bugfixes would occasionally make their way back up, major physics modules never did. In the closed collaboration, however, code sharing was common: even though the source control practices did not really lend themselves to this (it was usually a bunch of tarballs being passed around!) routines and physics modules were shared, and a great deal of crosstalk occurred.

A History of Enzo



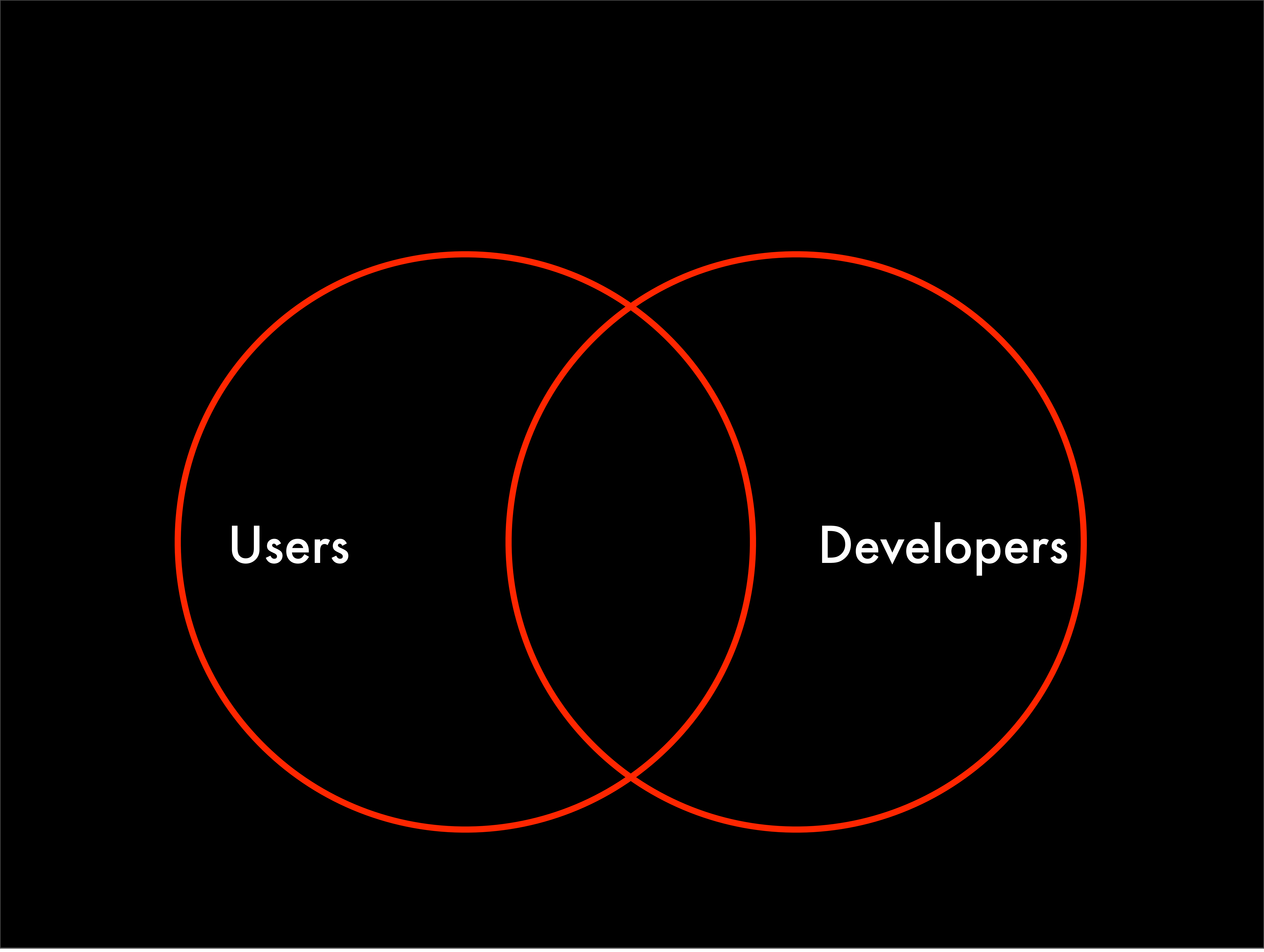
Enzo, like many different codes, followed a standard track of development. The initial version was written by a lone coder: unlike, say, the FLASH code or CASTRO, Enzo was originally written by Greg Bryan by himself. This seems to be the rule more than the exception in astrophysical codes. At some point, Greg shared the code with others in his research group and his other collaborators, and then this became a closed collaboration. This code was then open sourced and made freely available to the public.

However, even though that's the main riverbed, there were several forks and tributaries that complicate the story.

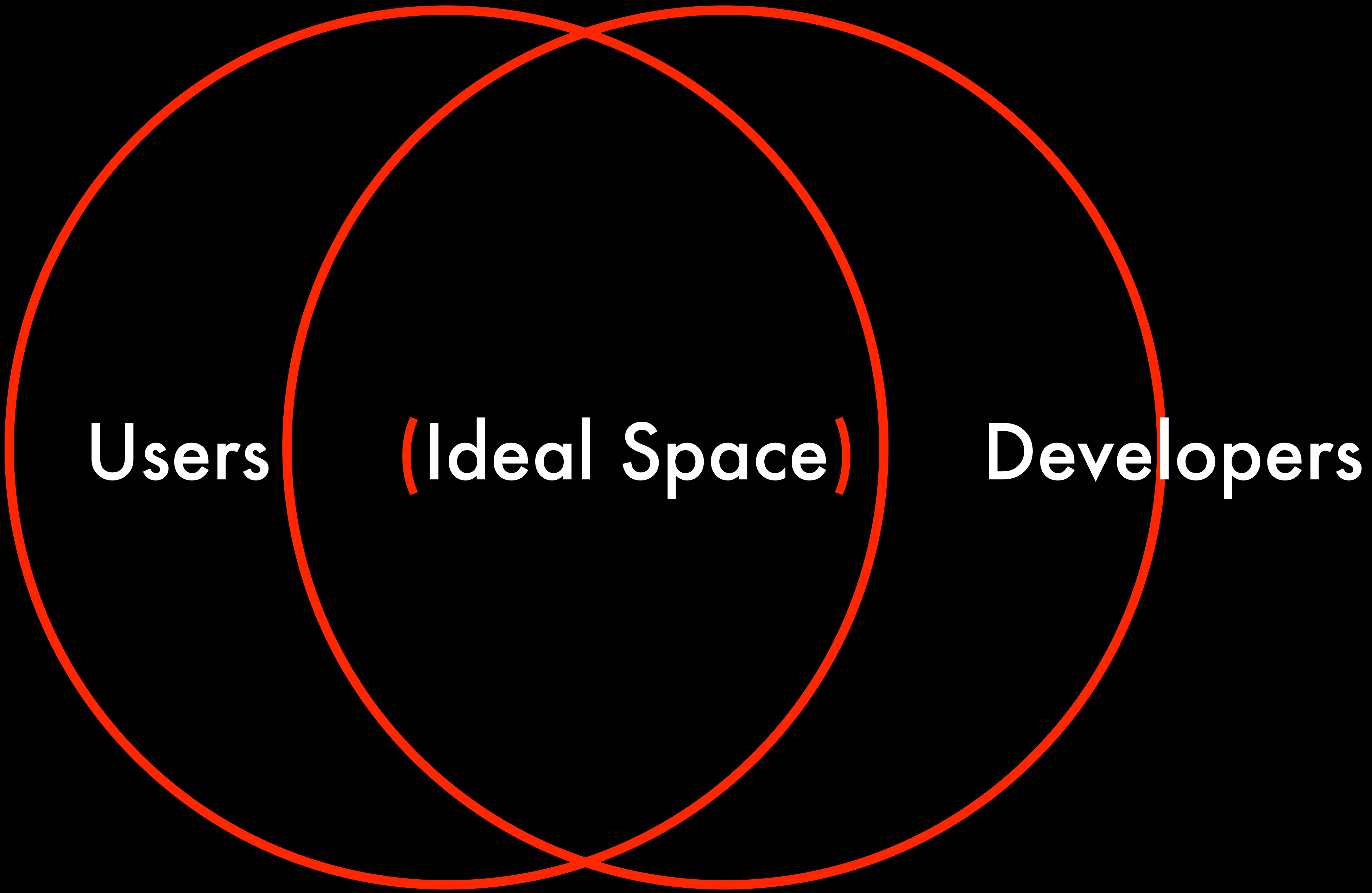
The open source branch of Enzo was primarily characterized by two classes of individuals: those with repository access, and those without. While changes often flowed downstream from the developers, it was much rarer for there to be crosstalk downstream or, in particular, sharing of changes back with the primary developers. While bugfixes would occasionally make their way back up, major physics modules never did. In the closed collaboration, however, code sharing was common: even though the source control practices did not really lend themselves to this (it was usually a bunch of tarballs being passed around!) routines and physics modules were shared, and a great deal of crosstalk occurred.

An odd thing happened: the "Open Source" version of Enzo embodied closed source practices more than the closed source version!

What this really means is that, even though the Open Source branch of Enzo did all the heavy lifting in terms of community and documentation, the Closed Collaboration was where the really experimental features were getting implemented and passed around. Furthermore, the Open Source branch was being hobbled in terms of pulling contributions back in, with a path toward making them public.

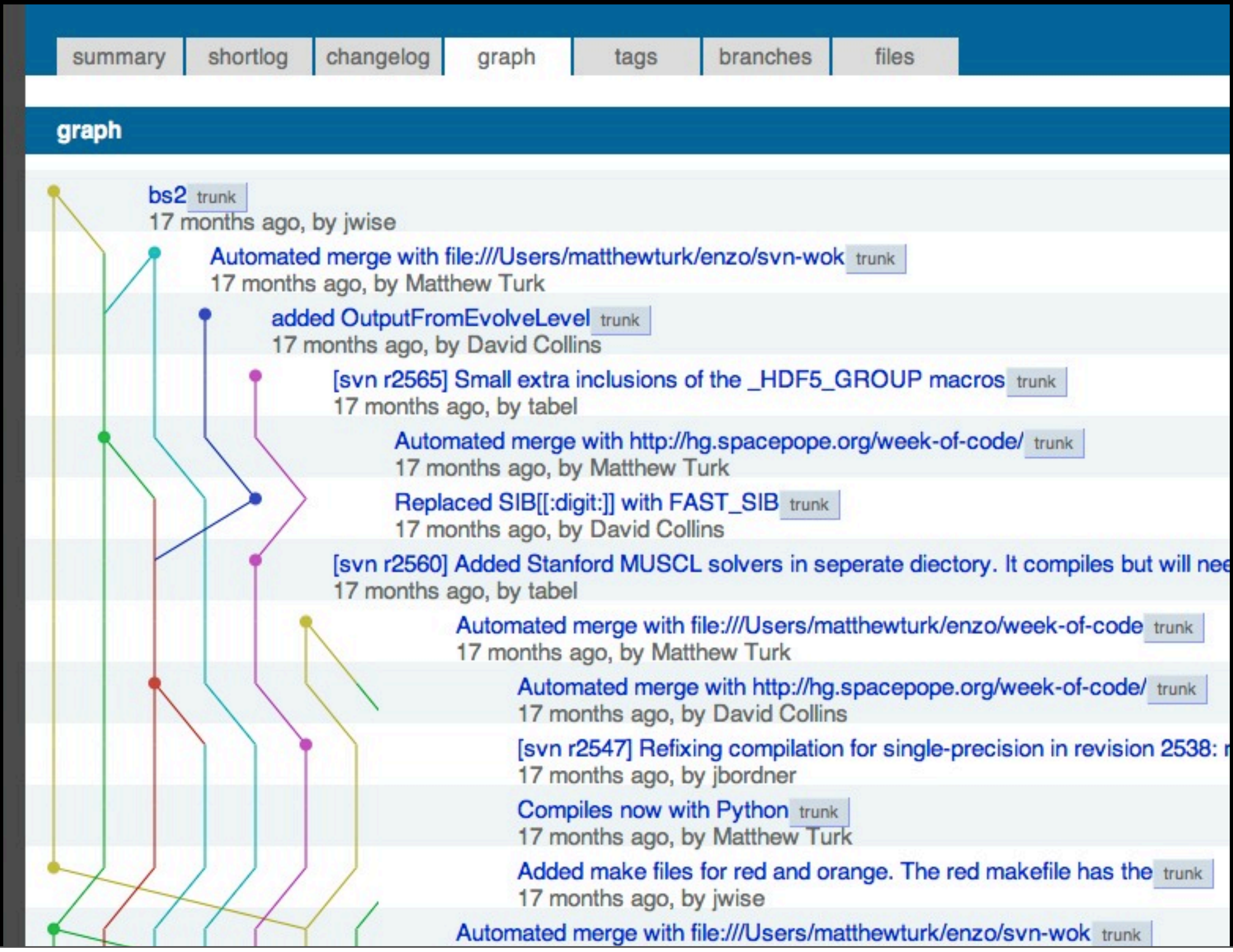


This led to a natural segregation between the “users” and the “developers.” The transition between the two was difficult, and this led to fragmentation of the code base and a difficulty in collaboration.



But what we really want is to remove this distance: while not every user will ever be a developer, and not every developer will be a user (but most will!) we want to increase the overlap between the two groups.

In 2009 and 2010, the primary development groups were re-unified.



bs2 trunk
17 months ago, by jwise

Automated merge with file:///Users/matthewturk/enzo/svn-wok trunk
17 months ago, by Matthew Turk

added OutputFromEvolveLevel trunk
17 months ago, by David Collins

[svn r2565] Small extra inclusions of the _HDF5_GROUP macros trunk
17 months ago, by tabel

Automated merge with http://hg.spacepope.org/week-of-code/ trunk
17 months ago, by Matthew Turk

Replaced SIB[[:digit:]] with FAST_SIB trunk
17 months ago, by David Collins

[svn r2560] Added Stanford MUSCL solvers in seperate diectory. It compiles but will nee
17 months ago, by tabel

Automated merge with file:///Users/matthewturk/enzo/week-of-code trunk
17 months ago, by Matthew Turk

Automated merge with http://hg.spacepope.org/week-of-code/ trunk
17 months ago, by David Collins

[svn r2547] Refixing compilation for single-precision in revision 2538: r
17 months ago, by jbordner

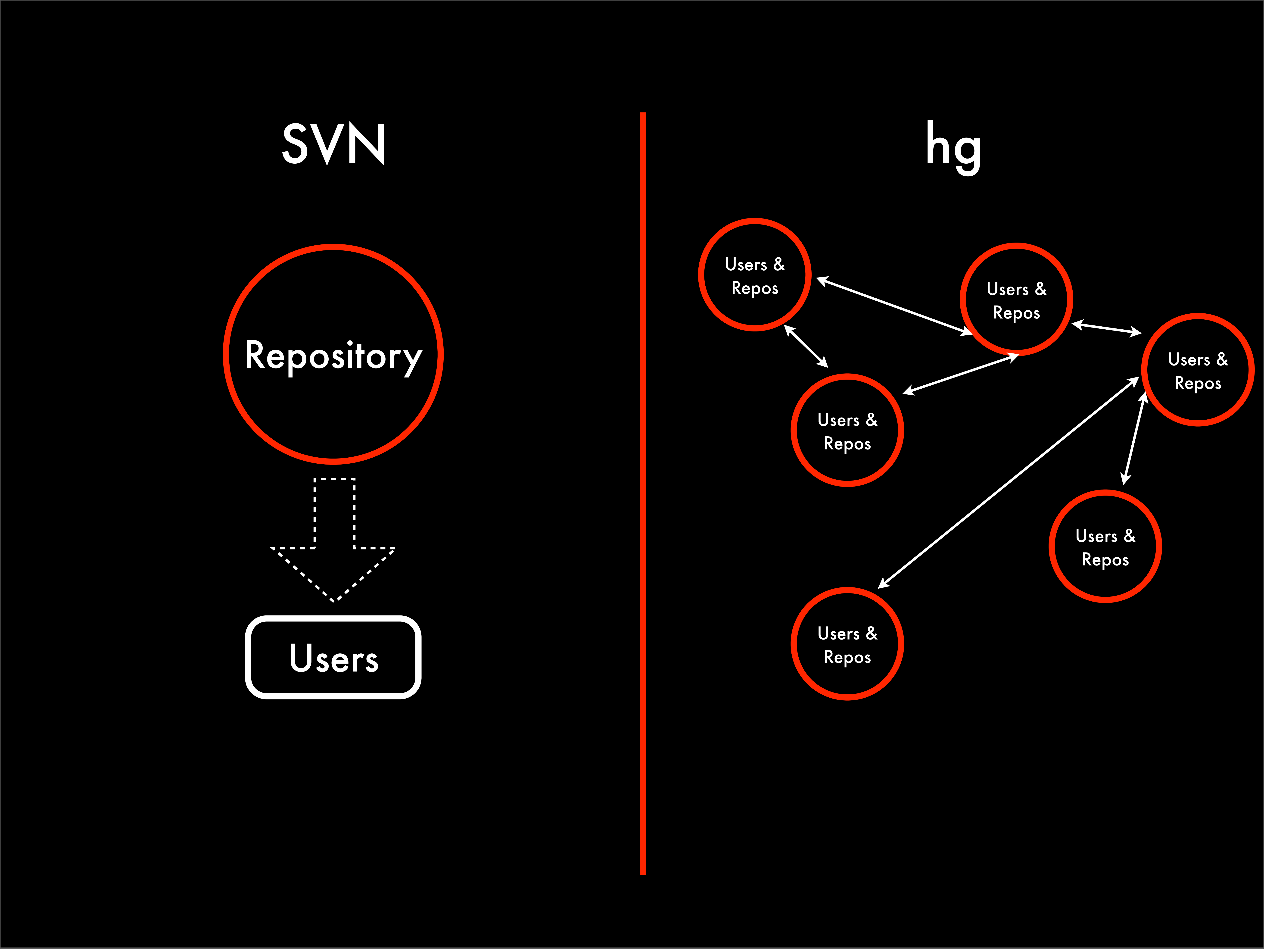
Compiles now with Python trunk
17 months ago, by Matthew Turk

Added make files for red and orange. The red makefile has the trunk
17 months ago, by jwise

Automated merge with file:///Users/matthewturk/enzo/svn-wok trunk

And so in 2009 and 2010, the Enzo house was put back in order. Through conscious effort, we have attempted to expand the ability of developers to collaborate, to discuss problems, to work together, and to share source code. We have held two developer workshops, with at least one (and probably two) to be held in 2011, that are open to whoever wants to attend. We have transitioned to a distributed version control system (mercurial) and we have attempted to democratize development.

Shown here is a screenshot from the first of our developer workshops, the “week of code.” During several bursts of intense activity, we have consolidated development into a shared repository and made this repository accessible to everyone in the general public. Through a combination of community outreach and technological developments, we have transformed Enzo into a massively participatory project.



Rather than a single centralized repository, we have moved to the model of Distributed Version Control: many repositories, everywhere, and each changeset is a fully-unique object. This enables crosstalk, as well as encouraging sharing of changes and local (unshared, even!) versioning.

Users no longer have to be beatified to keep track of their own modifications to the source.

The process since then has led to an important and unavoidable conclusion:

The conceptual separation of “users” from “developers” in Astrocomputing is actively harmful.

By describing astrocomputing codes in the terminology of users and developers, it creates a false distinction between those allowed to inspect, modify, extend a code and those who are expected to accept it unquestioningly. This stigmatizes the development of additional models, and furthermore, impedes the sharing of modules between users.

(For greater discussion of this, see work by Jono Bacon, Karl Fogel, Ben Collins-Sussman, and so on. The idea of “Highly Open Participation” needs to be extended to Astrocomputing.)

“Developers”

Every item on this list is part of the due diligence of using an astrophysical code.

“Developers”

Inspection and verification

Every item on this list is part of the due diligence of using an astrophysical code.

"Developers"

Inspection and verification
Tracking modifications

Every item on this list is part of the due diligence of using an astrophysical code.

"Developers"

Inspection and verification
Tracking modifications
Sharing information

Every item on this list is part of the due diligence of using an astrophysical code.

“Developers”

Inspection and verification
Tracking modifications
Sharing information
Adding functionality

Every item on this list is part of the due diligence of using an astrophysical code.

“Developers”

Inspection and verification
Tracking modifications
Sharing information
Adding functionality

All are necessary characteristics of the scientific process as a whole.

Every item on this list is part of the due diligence of using an astrophysical code.

“Users”

Codes cannot and should not be black boxes. As simulators, we have an intuitive understanding of what works, what doesn’t work, what the code can tell us and what it cannot. This is something we should not take for granted, and something we should not suggest others shy away from, either. It has been said that the easier a code is to use, the easier it can be used to do Bad Science. Accepting this as simply the status quo will stymie scientific growth.

By creating this false barrier, biases against simulating as a mechanism for promoting understanding will grow as well. “Development? Isn’t that the domain of the code monkey?”

"Users"

Uncritical acceptance of code...?

Codes cannot and should not be black boxes. As simulators, we have an intuitive understanding of what works, what doesn't work, what the code can tell us and what it cannot. This is something we should not take for granted, and something we should not suggest others shy away from, either. It has been said that the easier a code is to use, the easier it can be used to do Bad Science. Accepting this as simply the status quo will stymie scientific growth.

By creating this false barrier, biases against simulating as a mechanism for promoting understanding will grow as well. "Development? Isn't that the domain of the code monkey?"

“Users”

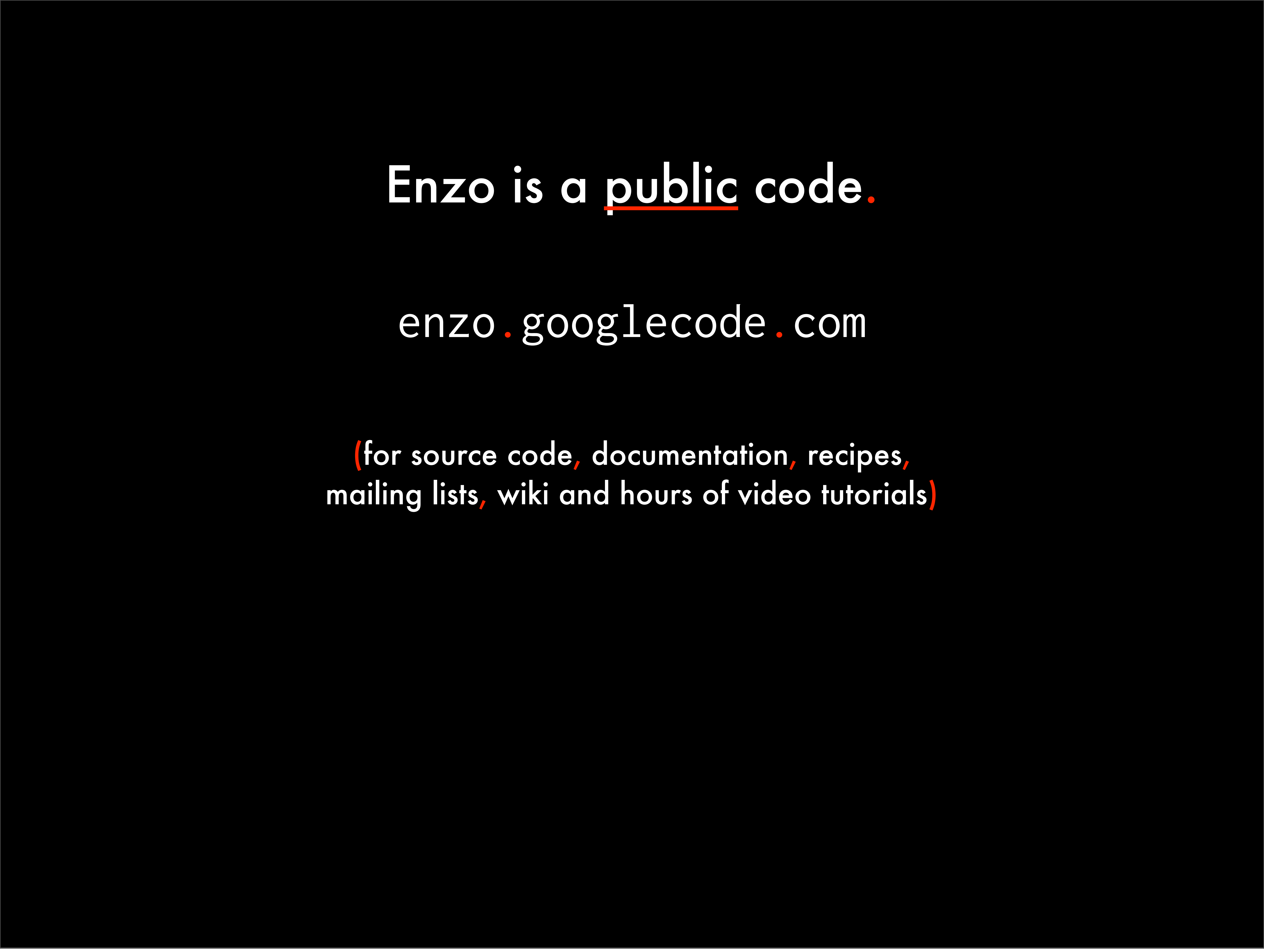
Uncritical acceptance of code...?

“These are people we give the code to that don’t care how it works.”

(an actual quotation!)

Codes cannot and should not be black boxes. As simulators, we have an intuitive understanding of what works, what doesn’t work, what the code can tell us and what it cannot. This is something we should not take for granted, and something we should not suggest others shy away from, either. It has been said that the easier a code is to use, the easier it can be used to do Bad Science. Accepting this as simply the status quo will stymie scientific growth.

By creating this false barrier, biases against simulating as a mechanism for promoting understanding will grow as well. “Development? Isn’t that the domain of the code monkey?”

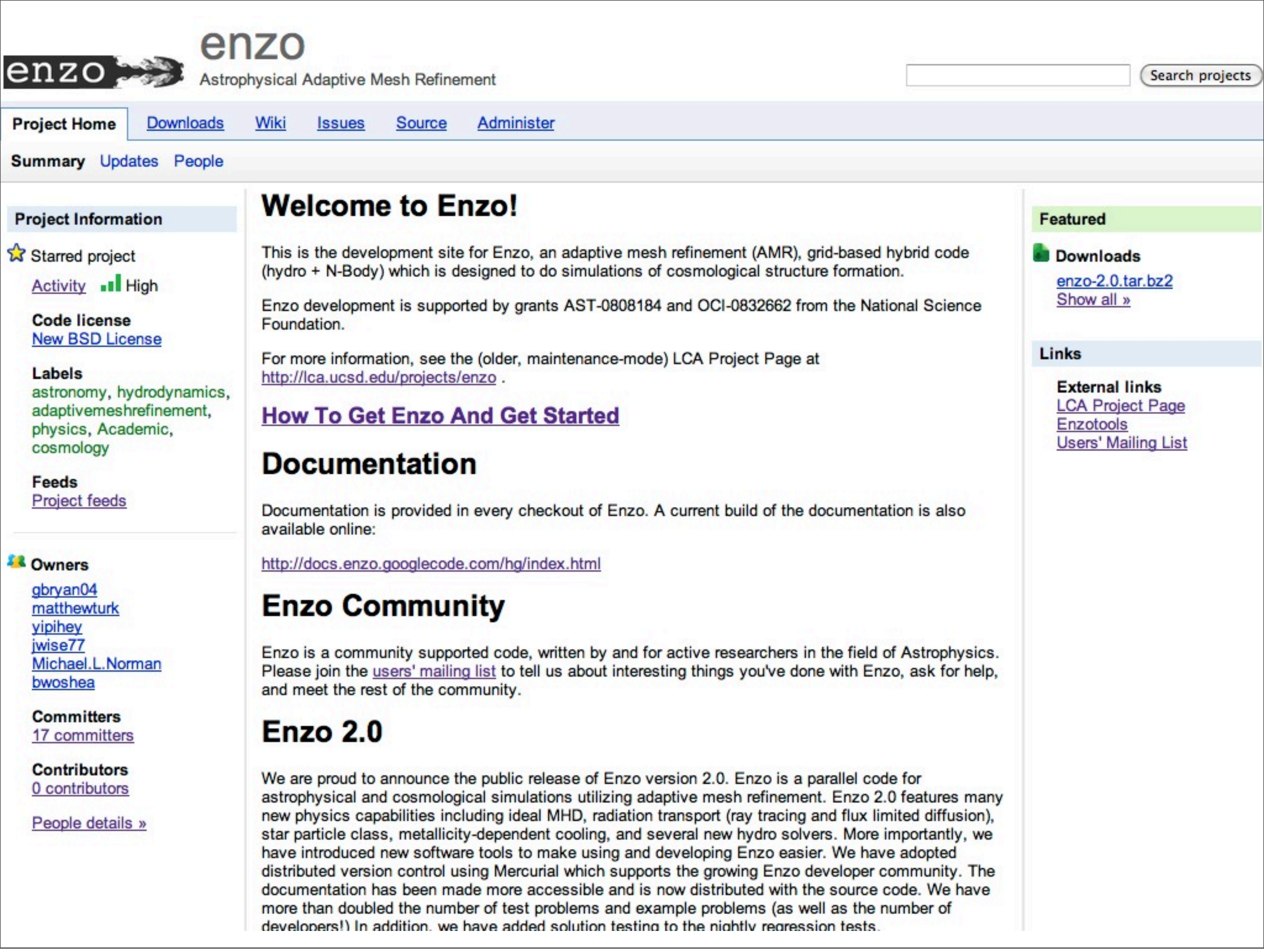


Enzo is a public code.

enzo.googlecode.com

(for source code, documentation, recipes,
mailing lists, wiki and hours of video tutorials)

The entire Enzo community is accessible from this website: not just the source code, but tutorials, documentation, examples, mailing lists, and so on and so forth.



Project Information

Starred project
Activity High

Code license
New BSD License

Labels
astronomy, hydrodynamics, adaptivemeshrefinement, physics, Academic, cosmology

Feeds
Project feeds

Owners
gbryan04
matthewturk
yipihey
jwise77
Michael.L.Norman
bwoshea

Committers
17 committers

Contributors
0 contributors

People details »

Welcome to Enzo!

This is the development site for Enzo, an adaptive mesh refinement (AMR), grid-based hybrid code (hydro + N-Body) which is designed to do simulations of cosmological structure formation.

Enzo development is supported by grants AST-0808184 and OCI-0832662 from the National Science Foundation.

For more information, see the (older, maintenance-mode) LCA Project Page at http://lca.ucsd.edu/projects/enzo .

How To Get Enzo And Get Started

Documentation

Documentation is provided in every checkout of Enzo. A current build of the documentation is also available online:

http://docs.enzo.googlecode.com/hg/index.html

Enzo Community

Enzo is a community supported code, written by and for active researchers in the field of Astrophysics. Please join the users' mailing list to tell us about interesting things you've done with Enzo, ask for help, and meet the rest of the community.

Enzo 2.0

We are proud to announce the public release of Enzo version 2.0. Enzo is a parallel code for astrophysical and cosmological simulations utilizing adaptive mesh refinement. Enzo 2.0 features many new physics capabilities including ideal MHD, radiation transport (ray tracing and flux limited diffusion), star particle class, metallicity-dependent cooling, and several new hydro solvers. More importantly, we have introduced new software tools to make using and developing Enzo easier. We have adopted distributed version control using Mercurial which supports the growing Enzo developer community. The documentation has been made more accessible and is now distributed with the source code. We have more than doubled the number of test problems and example problems (as well as the number of developers!) In addition, we have added solution testing to the nightly regression tests.

Featured

Downloads
enzo-2.0.tar.bz2
Show all »

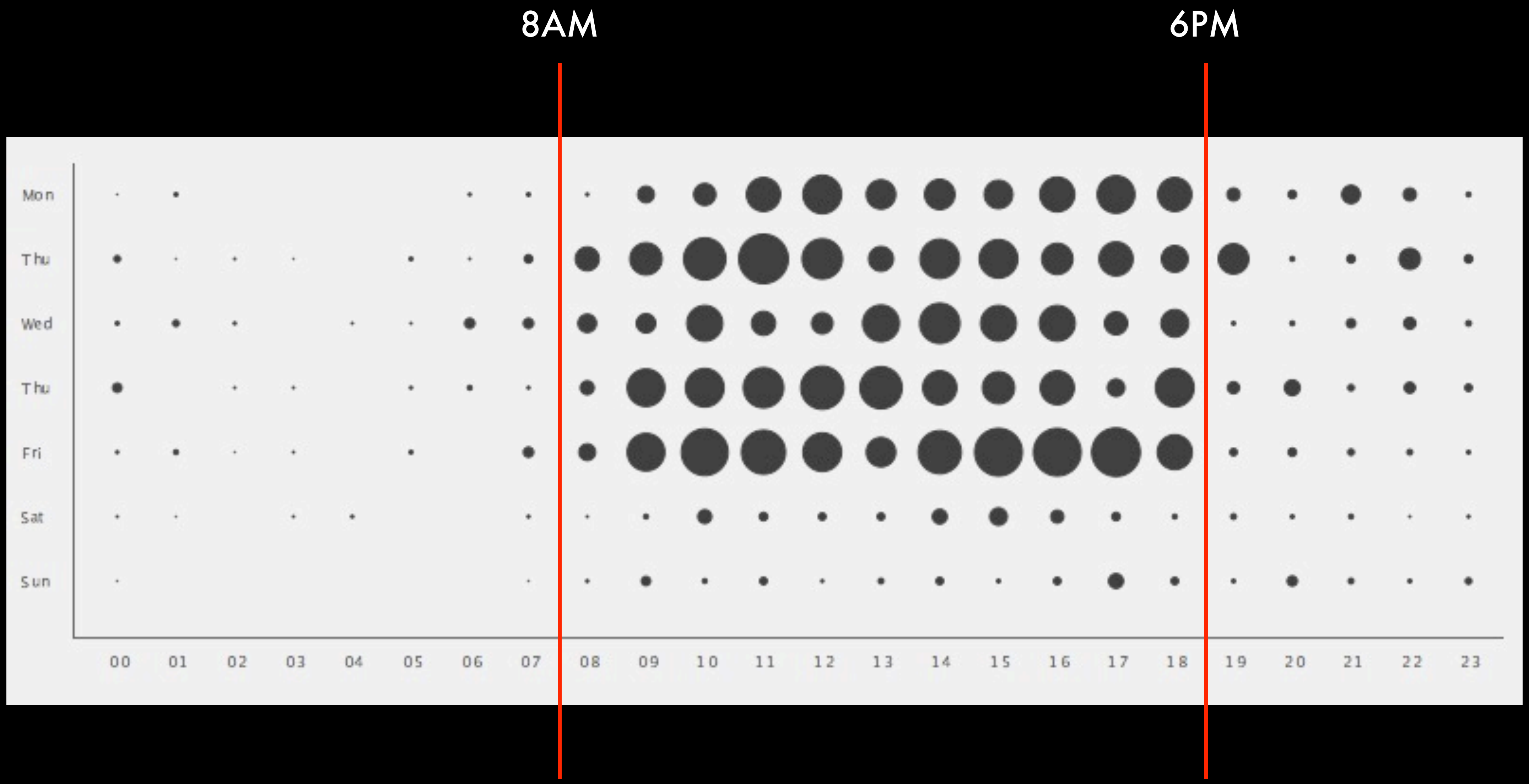
Links

External links
LCA Project Page
Enzotools
Users' Mailing List

Nearly all of Enzo has been written by working scientists.

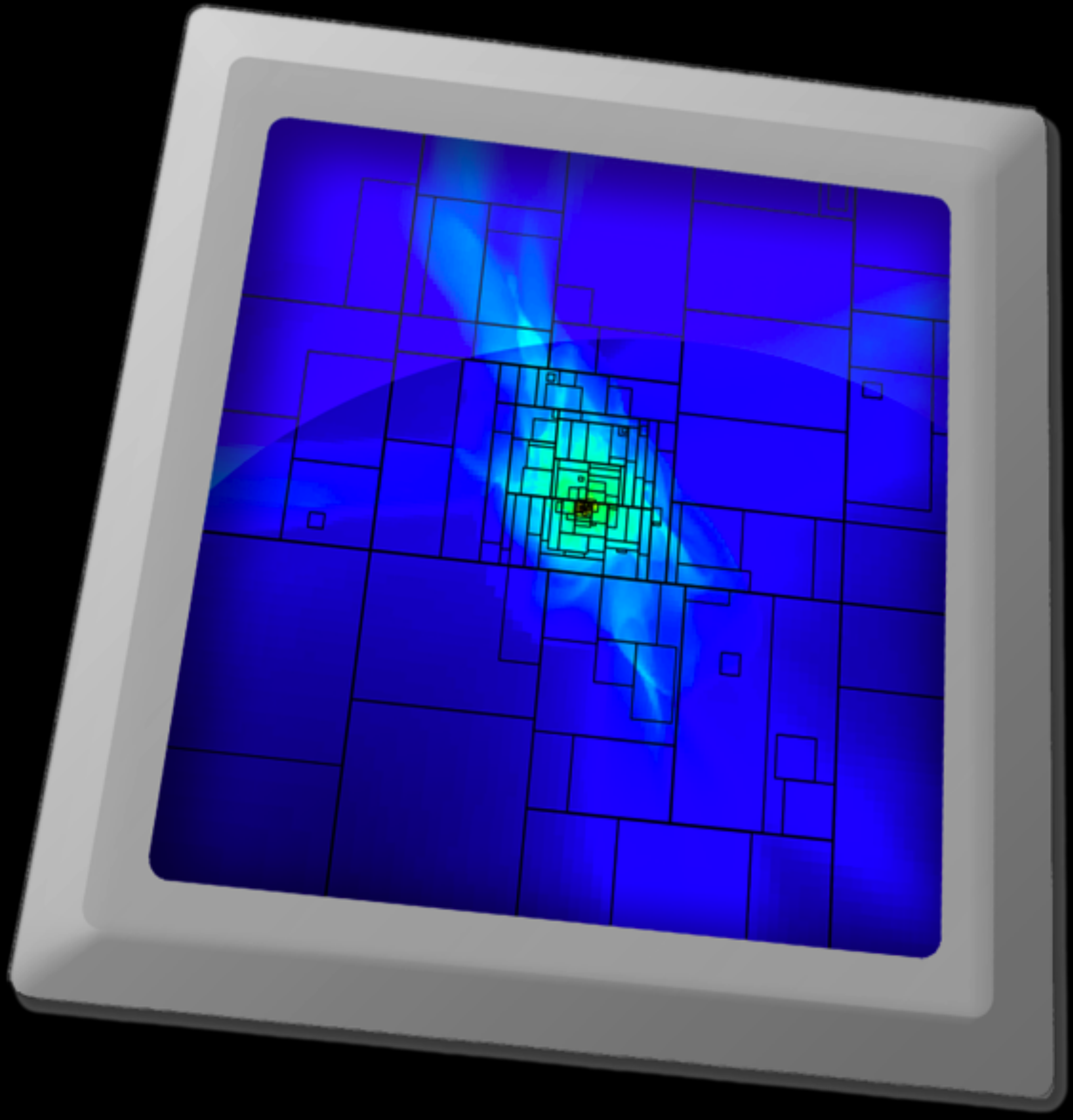
This is an important point, and one that should be emphasized. The development of Enzo has been driven by the pragmatic needs of working scientists. This development has accelerated since the highly-participatory shift in its development. This includes things like ray tracing, chemistry, parallelism improvements, utilization of accelerators, threading, inline analysis, magnetic fields, streaming IO, star particle enhancements, cooling models, and on and on.

260,000 lines of code
C, C++, Fortran and (a little) CUDA
>30 contributors
Contributors from 15 institutions



Enzo is still mostly an 8-6 code, but there are commits at every time of day. It has nearly doubled in size over the last eighteen months.

We hope to engage more members of the community to contribute changes, fixes, and so on. With this new found energy, we also intend to go along routes that I think no one saw before: this Spring we will begin the push to Enzo 3.0, where the accumulated technical debt of the last 15 years will be addressed.



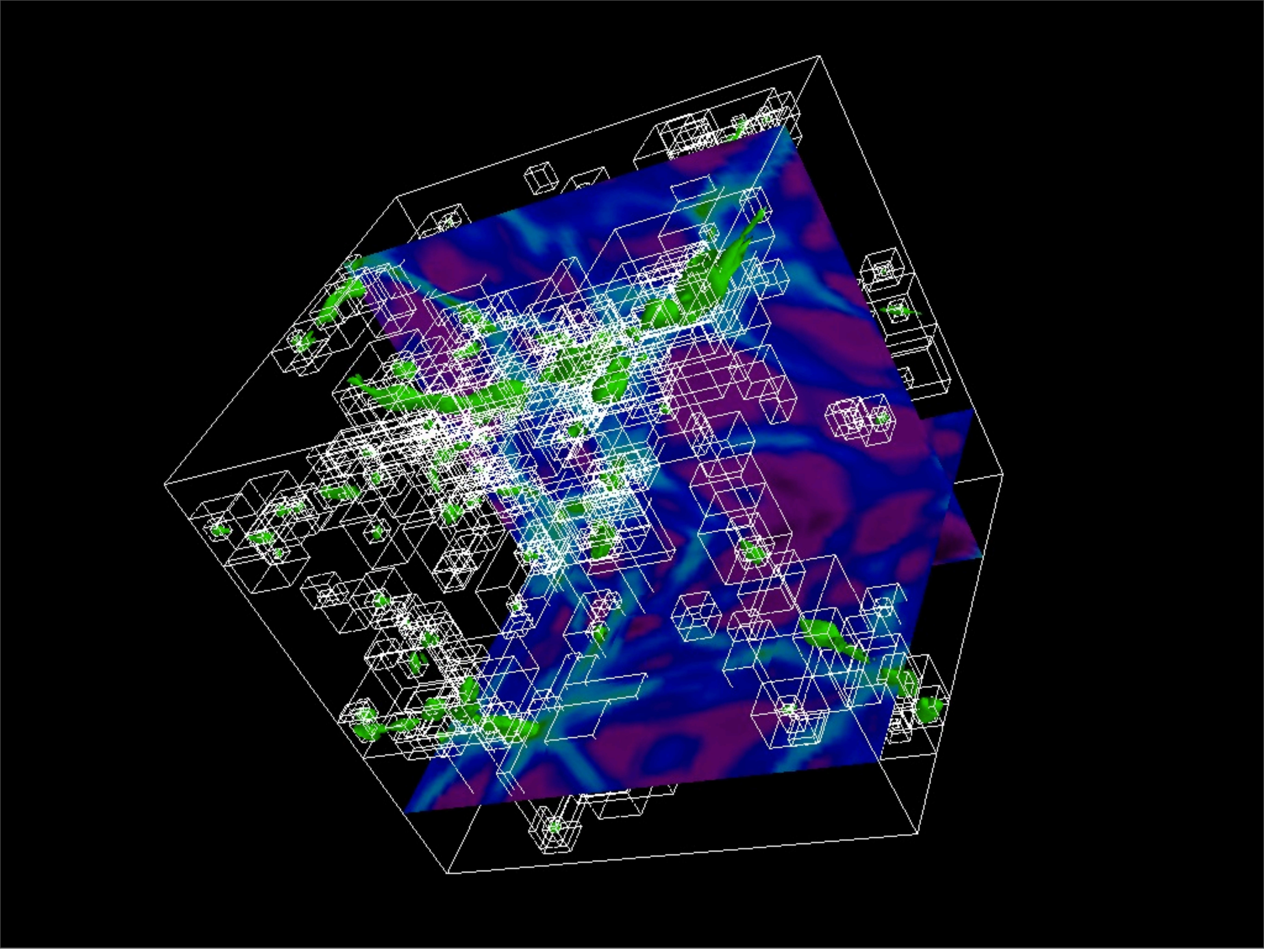
yt

astro-ph/1011.3514
yt.enzotools.org

Now I'm going to transition to talking more about a project I'm the lead developer on, yt. I started yt at Stanford with my advisor Tom Abel, when he and John Wise and I were sitting around talking to the lead developer of a visualization package called HippoDraw. It started out as a slicer and a phase plot creator, and now it's moved into being a parallel analysis and visualization package that can handle many different tasks.

How do we analyze?

But before we speak much more about what yt does and how it’s developed, I want to take a moment to ask you: what is the *right* way to analyze astrophysical data?



I've chosen this image, of a galaxy cluster simulation, to demonstrate the fundamental disconnect in astrophysical simulations. As astronomers, our primary concern is with galaxies, and stars, and clusters. But as simulators, we're stuck looking at particles, grids, cells, and so on. Here you could see both of these aspects -- and while they are in some sense related, a grid is a poor substitute for a galaxy.

yt has been designed to address physical, not computational, entities.

The entire focus of yt has been on abstracting out the simulational aspects wherever possible. This means that rather than loading up AMR grids and analyzing them, you load up the simulation and then address halos, or spheres, or disks -- and then yt handles selecting the data, performing operations on it, and so on. This enables a new workflow to be designed that focuses on the underlying physics of the calculation.

For instance, the process of selecting halos, calculating their angular momentum and looking at phase plots of their entropy can be done in only a handful of lines, none of which have any knowledge about the underlying simulation data format. Despite that, yt still makes accessible the underlying simulation objects, but they are not *required* to analyze data.

Enzo, Orion, CASTRO, FLASH

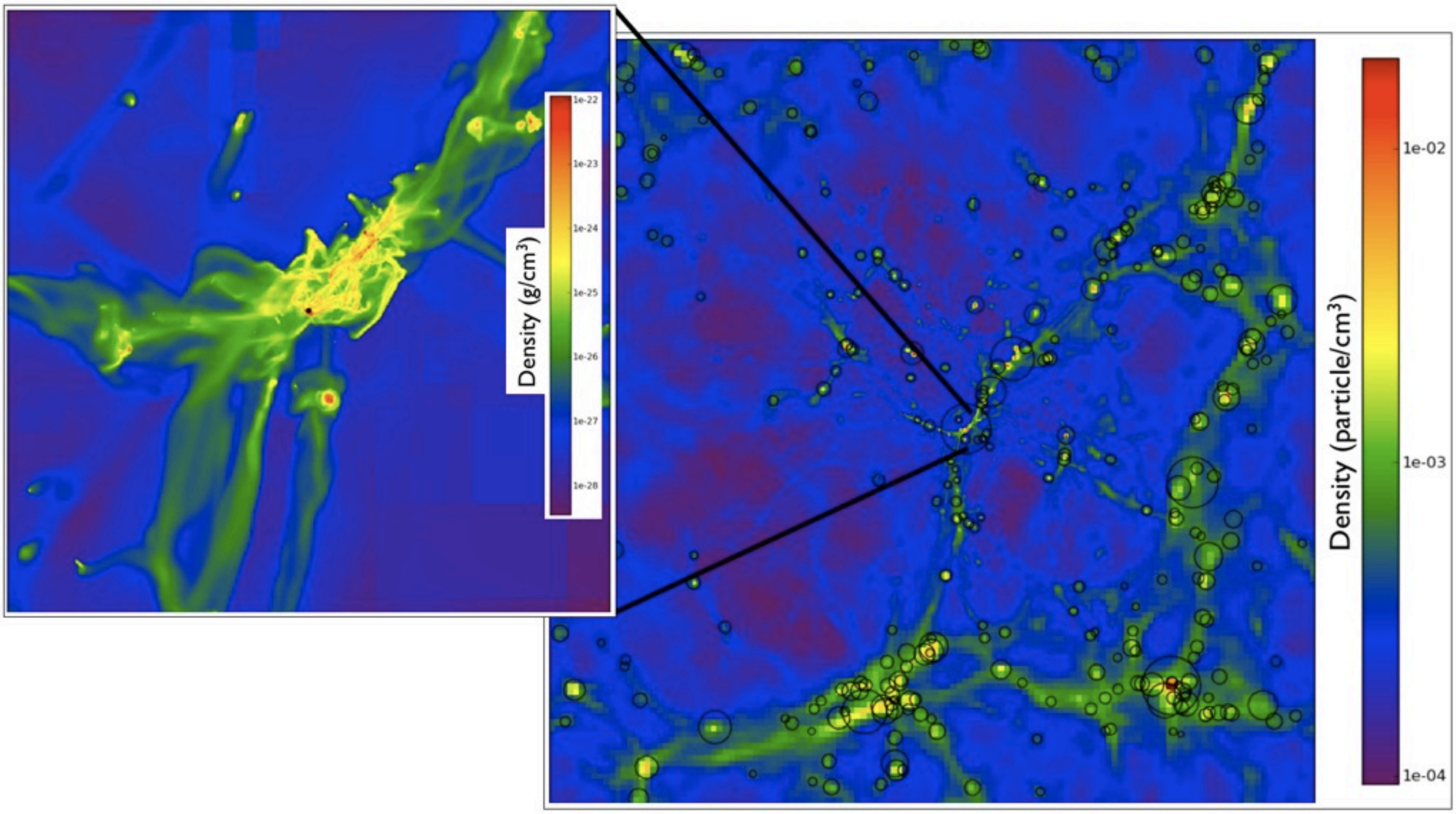
Chombo, Tiger, ART, RAMSES

(...and more to come?)

yt's design is also neutral to the underlying code. We currently support Enzo, Orion, CASTRO and FLASH nearly fully, with somewhat limited support for Chombo, Tiger, ART and RAMSES. We hope to continue this trend by extending to additional codes, but also to continue to improve support for the existing codes. We can only do this in the context of a desire from external users, however.

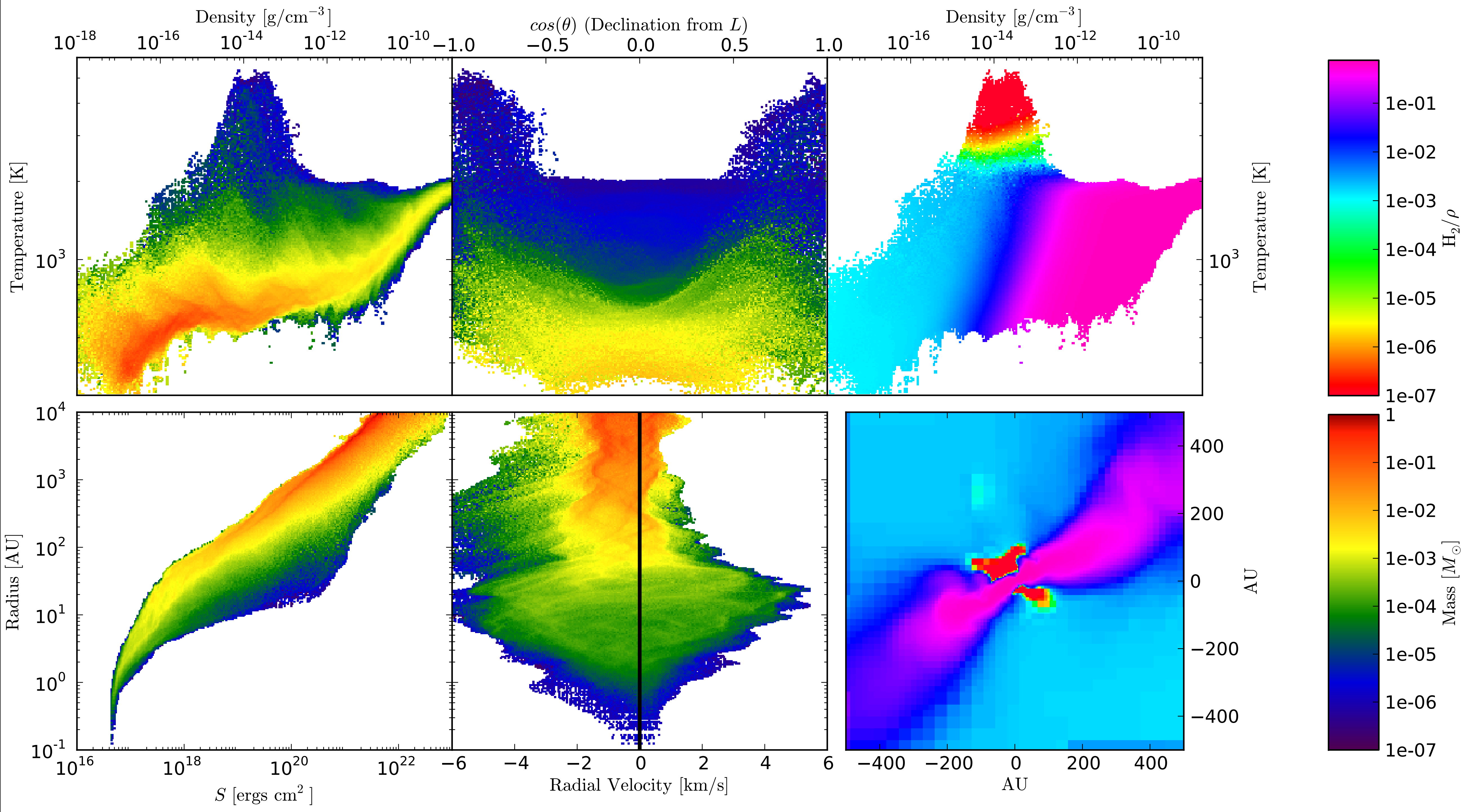
The generalization effort was started by Jeff Oishi, but it has benefited from help from Oliver Hahn, Stella Offner, John ZuHone, and Chris Moody, along with several others.

yt has many, many features. I'm not going to list them here, but they are in the paper, and I encourage you to investigate. We're targeting Blue Waters for in situ and post processing of data.



(Kim, Wise, Alvarez and Abel)

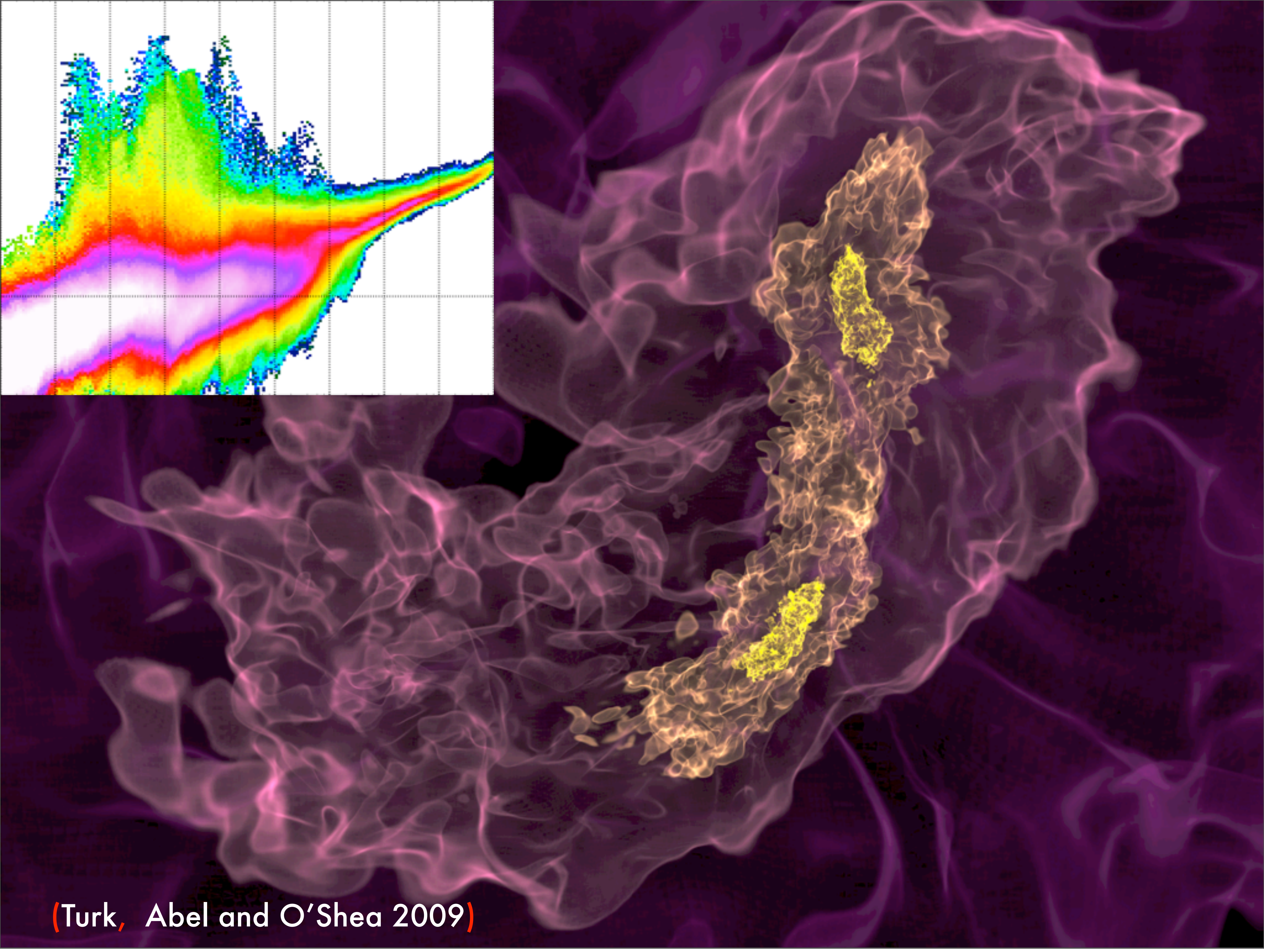
One of the simplest sets of tasks is to visualize 2D representations of datasets -- these can be line integrals (projections), slices, and even oblique slices. This image, provided by Ji-hoon Kim, shows a galaxy formation simulation wherein halos have been overplotted, using the HOP halo finder, provided by yt. yt also provides friend-of-friend and a completely ground-up parallel version of HOP, created and implemented by Stephen Skory.



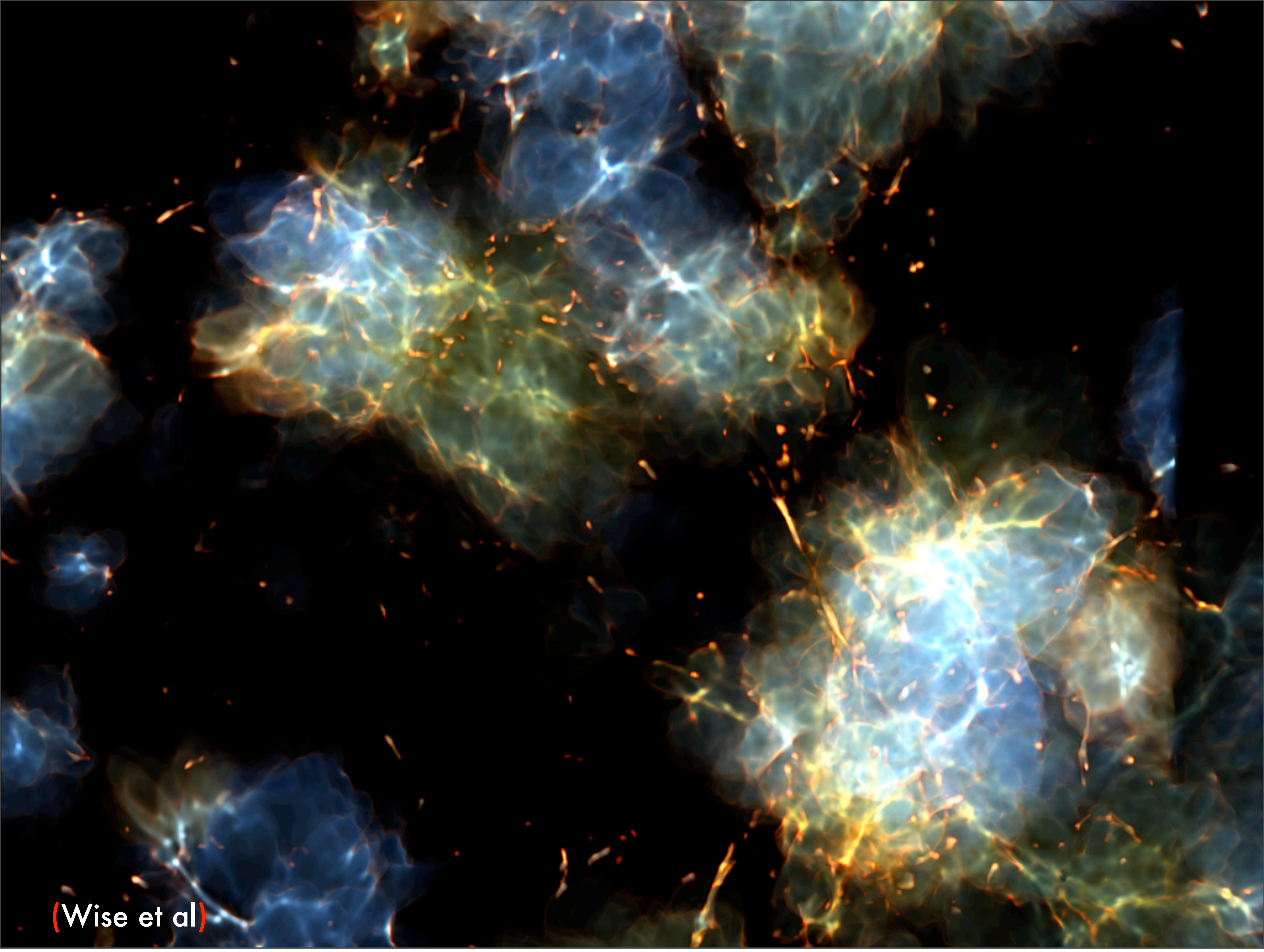
(Turk, Norman and Abel 2010)

This is a slide from my own research. yt provides the ability to select data and then plot various components. On this slide are a number of interesting applications of the data manipulation capabilities of yt -- specifically, we see multi-variate plots of a Population III star forming region. The upper left is a standard mass distribution as a function of density and temperature. In the top middle, I've replotted it so that you can see the mass distribution as a function of temperature and declination; all of the hot gas is confined to the polar regions, as you can see. The upper right is the average molecular hydrogen fraction. On the bottom are plots of the entropy, the inward velocity, and an image plot of the molecular hydrogen fraction along the y axis.

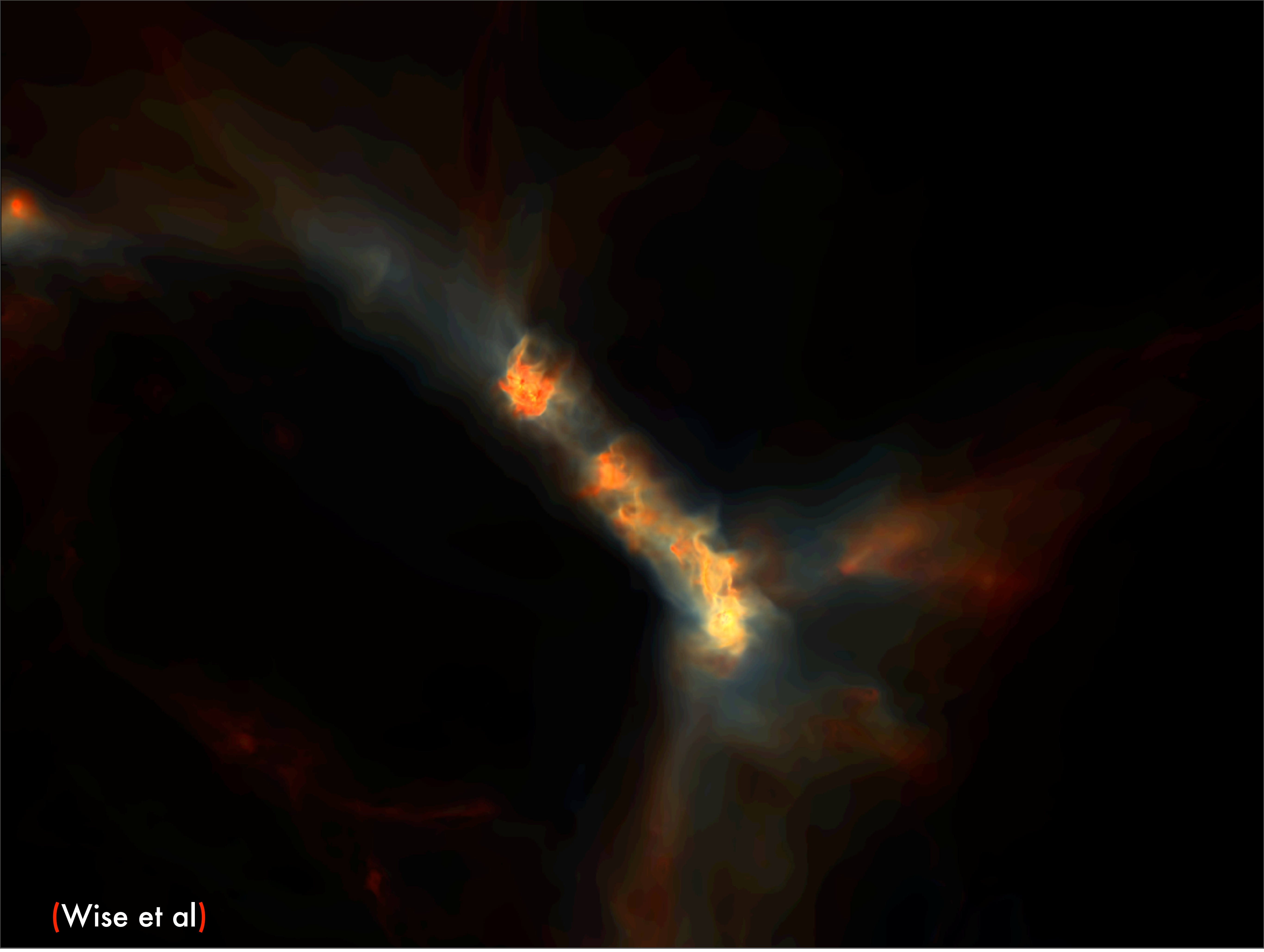
As you can see, not only can we do visualizations of the data with yt, but also interesting quantitative visualizations in non-spatial variables.



One of the things I'm most proud of is the volume rendering capabilities of yt, which can be motivated by physical characteristics in the simulation. I used yt to visualize this primordial star forming region, which we showed in a 2009 paper was able to fragment and form two Population III stars. I'm particularly proud of this visualization: I set the camera to point down the angular momentum system of the two clumps, selected isocontours that drew out chemical instabilities, and then included in the upper left a phase plot showing the different kinks in the equation of state of the gas.

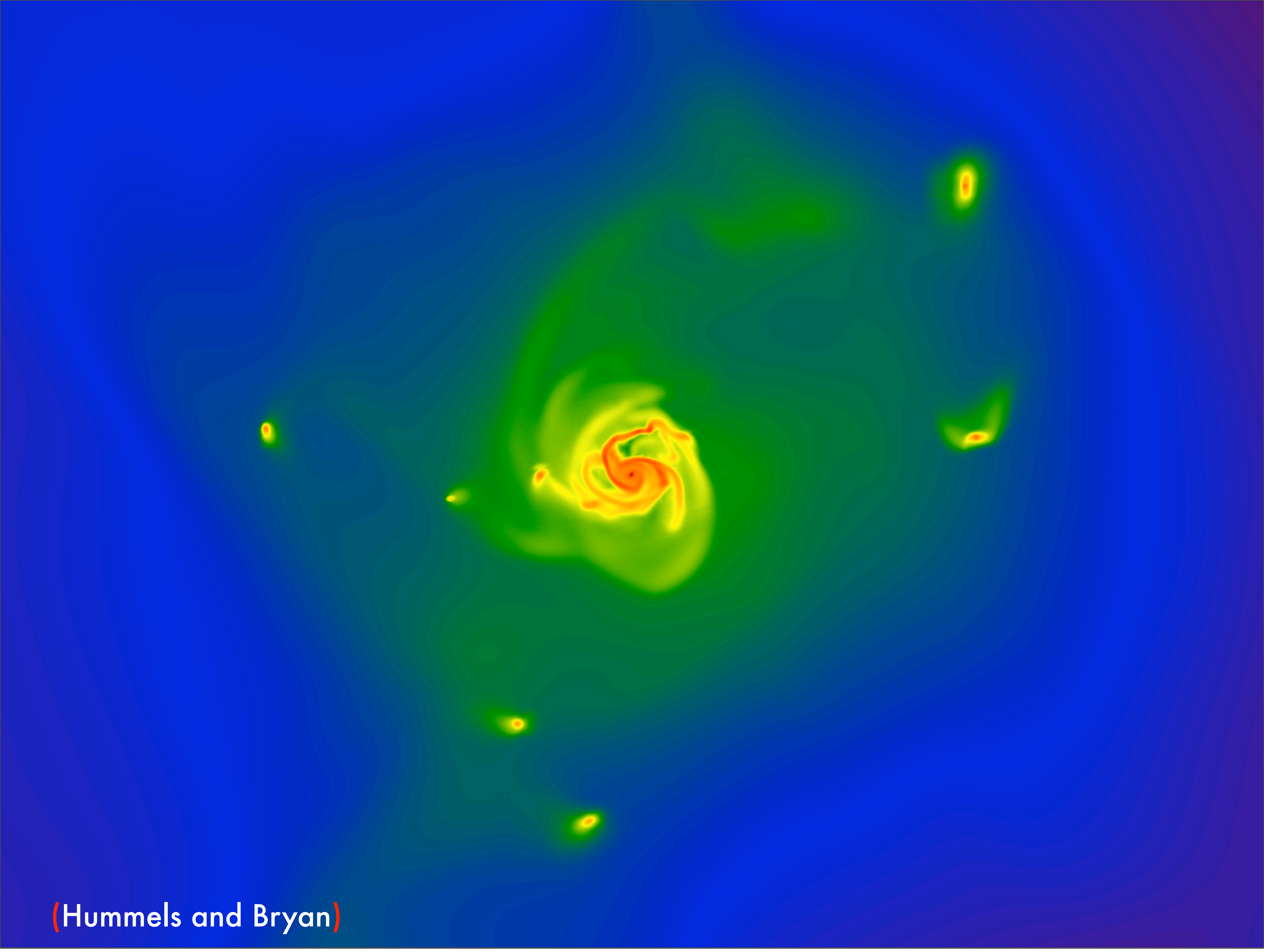


This is a reionization simulation by John Wise, which features 10^6 Msun dark matter resolution in a 12.5 Mpc/h (comoving) box. This image was generated at z=8.5, and it uses both Planck-spectrum emission and approximate scattering to visualize what this region may look like. The yt volume renderer solves the (non-scattering) radiation transfer equation, and so we are able to mock up simple simulated observations with it.



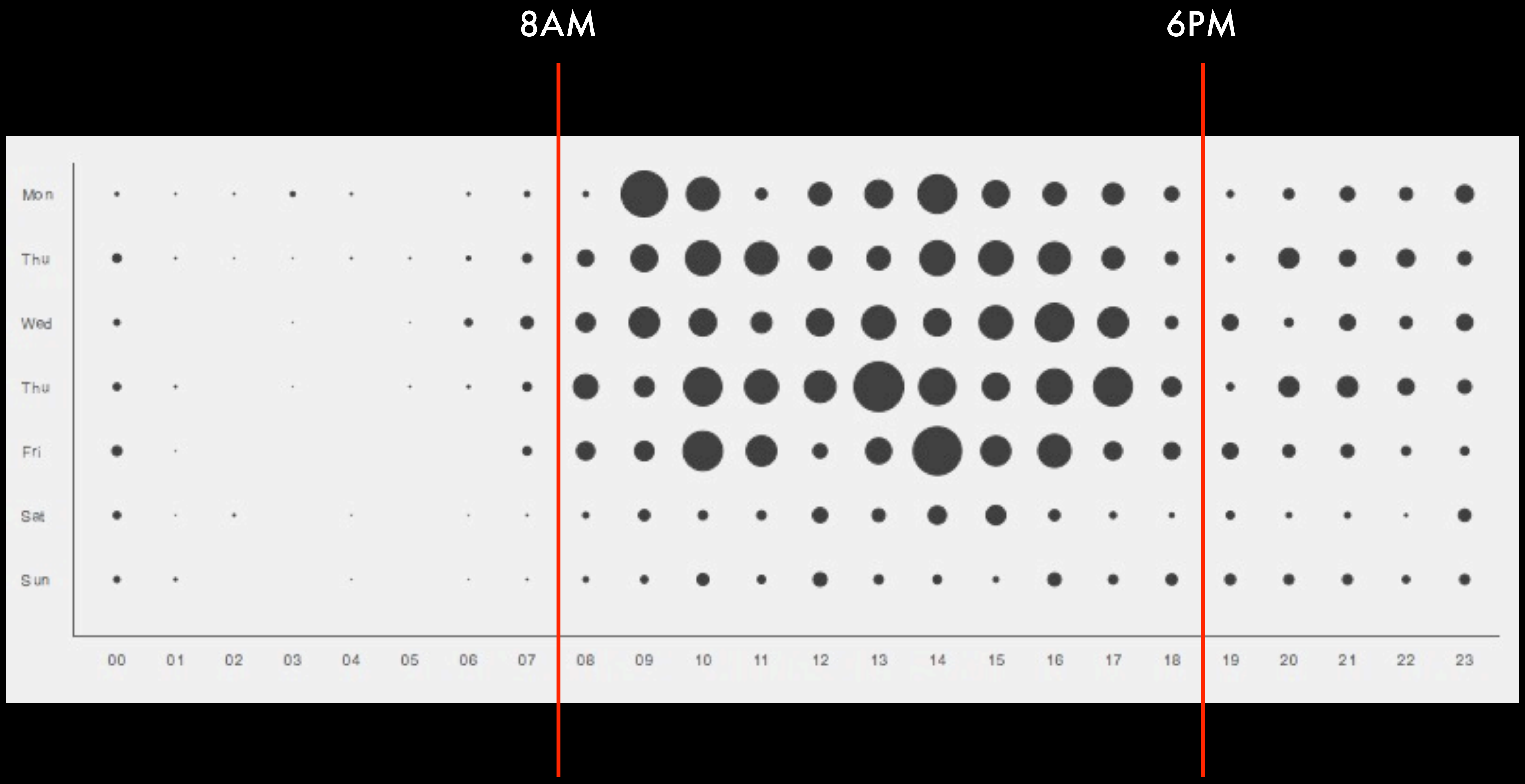
This is another image from John Wise, of a set of dwarf galaxies at z=17. Each cloud is about 10^7 Msun. The simulation includes radiation feedback from PopIII stars and their remnant black holes.

By making the ability to volume render beautiful, narrative images available to working scientists, the yt project is explicitly attempting to improve access to outreach-quality visualization. We hope to pursue outreach visualization as well as scientific visualization, and we would like to explore collaborations with Planetaria and outreach coordinators. yt visualizations have won DOE awards, and several have been featured at the Adler planetarium.



Of course, while volume rendering can be used to create pretty pictures, it can also be used to construct quantitative analysis. I've used it in my own work to calculate 2D Toomre Q parameters, but here we see an image by Cameron Hummels of a galactic disk in one of his simulations. To visualize the formation of galaxies, Cameron uses the yt halo finder to identify halos, calculate their angular momentum, and then align the volume rendering camera along that angular momentum vector. He then calculates the line integral at every pixel in an image, which returns a column density at oblique angles.

71,000 lines of code
Python, Cython, C
12 contributors (60+ users)
Contributors from 7 institutions



yt, like enzo, is a mostly workday project. We’re still growing, and there are a number of places it could be improved or extended, but it’s an energized community.

yt is open source, but unlike Enzo
it has not reached a critical mass.

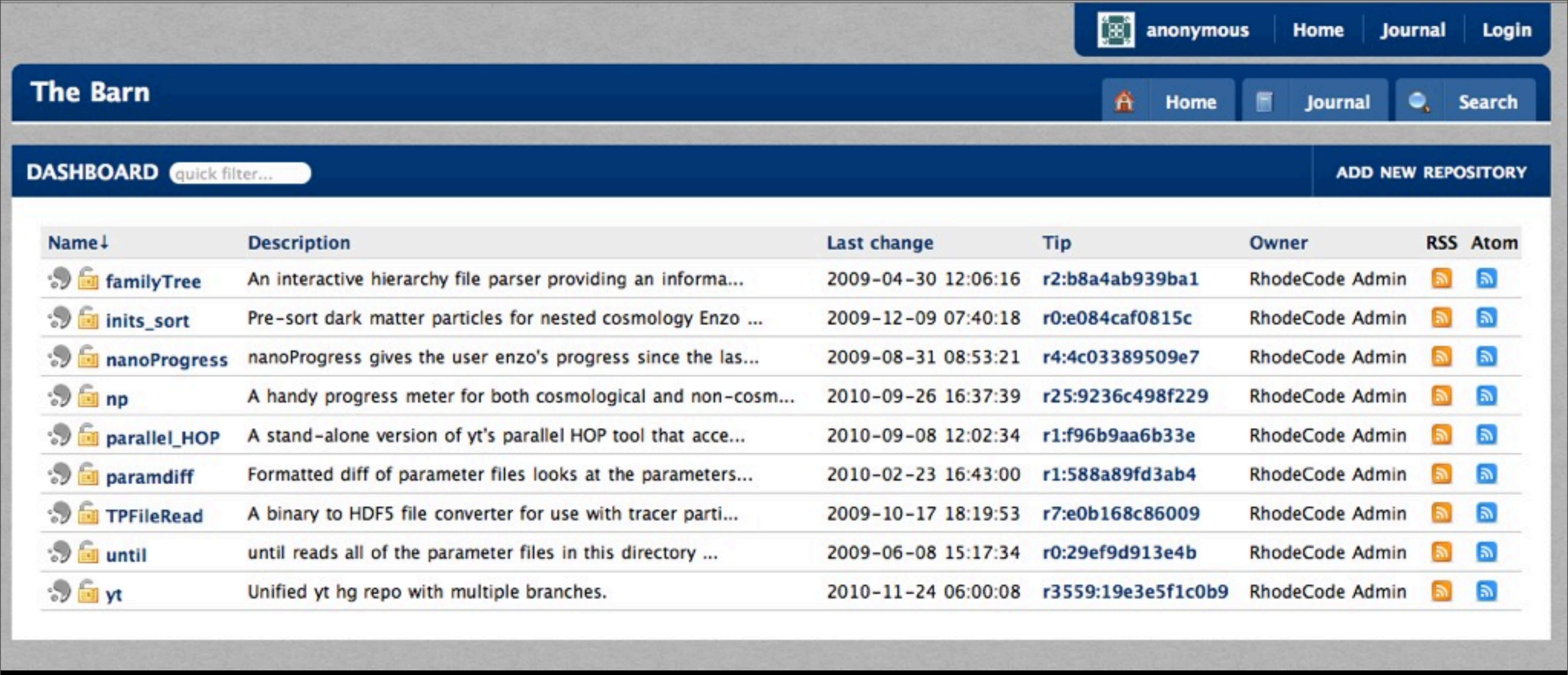
We're trying to engage the public, but there are a number of factors that seem to stymie collaboration on this type of project. I think that what it comes down to ...

Enzo is like a car.

...is that Enzo is like a car. You assemble it, you drive it, and you tinker with it...

yt is like a toolbox.

... but yt is a tool that’s taken as given. If it works for you, you use it, and you’d prefer not to think about it. Developing a new feature in yt does not necessarily lead to a new publication. And that’s okay! And another issue is that we have a long way to go for making accessible easy participation of small things, like analysis scripts and field definitions.



Name↓	Description	Last change	Tip	Owner	RSS	Atom
familyTree	An interactive hierarchy file parser providing an informa...	2009-04-30 12:06:16	r2:b8a4ab939ba1	RhodeCode Admin		
inits_sort	Pre-sort dark matter particles for nested cosmology Enzo ...	2009-12-09 07:40:18	r0:e084caf0815c	RhodeCode Admin		
nanoProgress	nanoProgress gives the user enzo's progress since the las...	2009-08-31 08:53:21	r4:4c03389509e7	RhodeCode Admin		
np	A handy progress meter for both cosmological and non-cosm...	2010-09-26 16:37:39	r25:9236c498f229	RhodeCode Admin		
parallel_HOP	A stand-alone version of yt's parallel HOP tool that acce...	2010-09-08 12:02:34	r1:f96b9aa6b33e	RhodeCode Admin		
paramdiff	Formatted diff of parameter files looks at the parameters...	2010-02-23 16:43:00	r1:588a89fd3ab4	RhodeCode Admin		
TPFileRead	A binary to HDF5 file converter for use with tracer parti...	2009-10-17 18:19:53	r7:e0b168c86009	RhodeCode Admin		
until	until reads all of the parameter files in this directory ...	2009-06-08 15:17:34	r0:29ef9d913e4b	RhodeCode Admin		
yt	Unified yt hg repo with multiple branches.	2010-11-24 06:00:08	r3559:19e3e5f1c0b9	RhodeCode Admin		

...we're working to enable greater participation.

We hope that in the future we'll be able to provide a bitbucket-like interface, enabling simple contributions and simple repositories for people to share minor modifications. Unlike Enzo, where often physics modules have to touch the code in a number of places, yt has a substantial number of possible enhancements that could be very self-contained. Not only that, but the reproducibility of papers will be greatly enhanced by having a location to store analysis scripts. We hope to bring this online in early 2011.

1.

Does Open Source remove my edge on the competition?

1.

If anything, Open Source increases global competitiveness.

By expanding the community, and treating it as a community of developers, then the entire conversation changes. It’s no longer “What am I giving away” but rather, “What is being shared?” Simply giving away code is the wrong solution to the problem of open source: the correct solution is to construct a community of users and developers, and to shepherd that community toward a participatory atmosphere.

2.

What about issues of correctness?

2.

Science is incremental. Accuracy is double-edged.

Science is an incremental process. We build not only on the work of past researchers, but also their misunderstandings and mistakes. Without an inspectable piece of simulation code, a set of results should be viewed as unreproducible. Not only should inspectable code be required for reproducibility, but for verifiability of the results. Inaccurate results that are never corrected will do more harm than good, as time continues. Algorithms are not enough: implementations are necessary.

For more on this, see the works of Cameron Neylon, Titus Brown, Randall J. LeVeque, and so on.

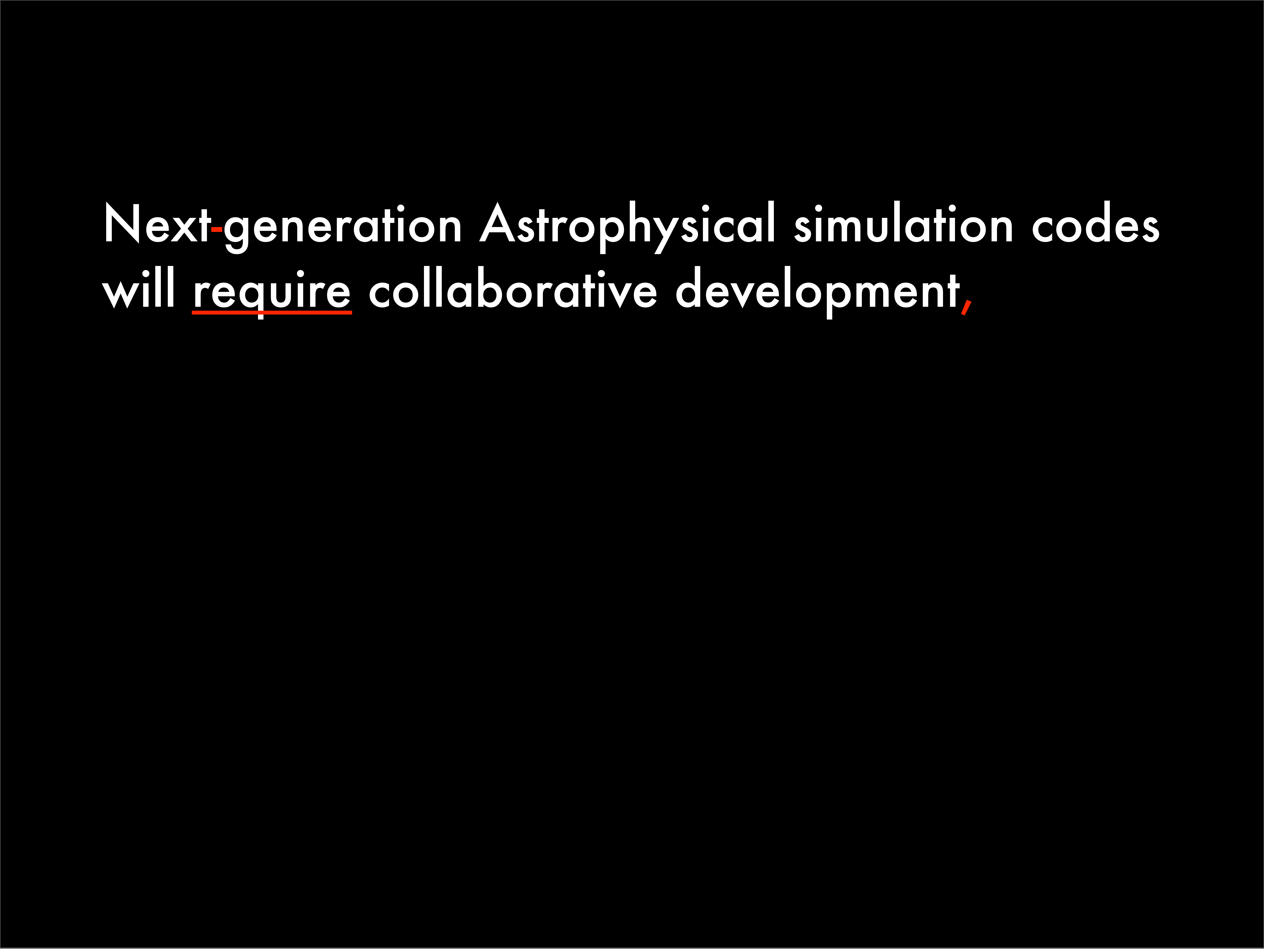
3.

Do support structures encumber productivity?

3.

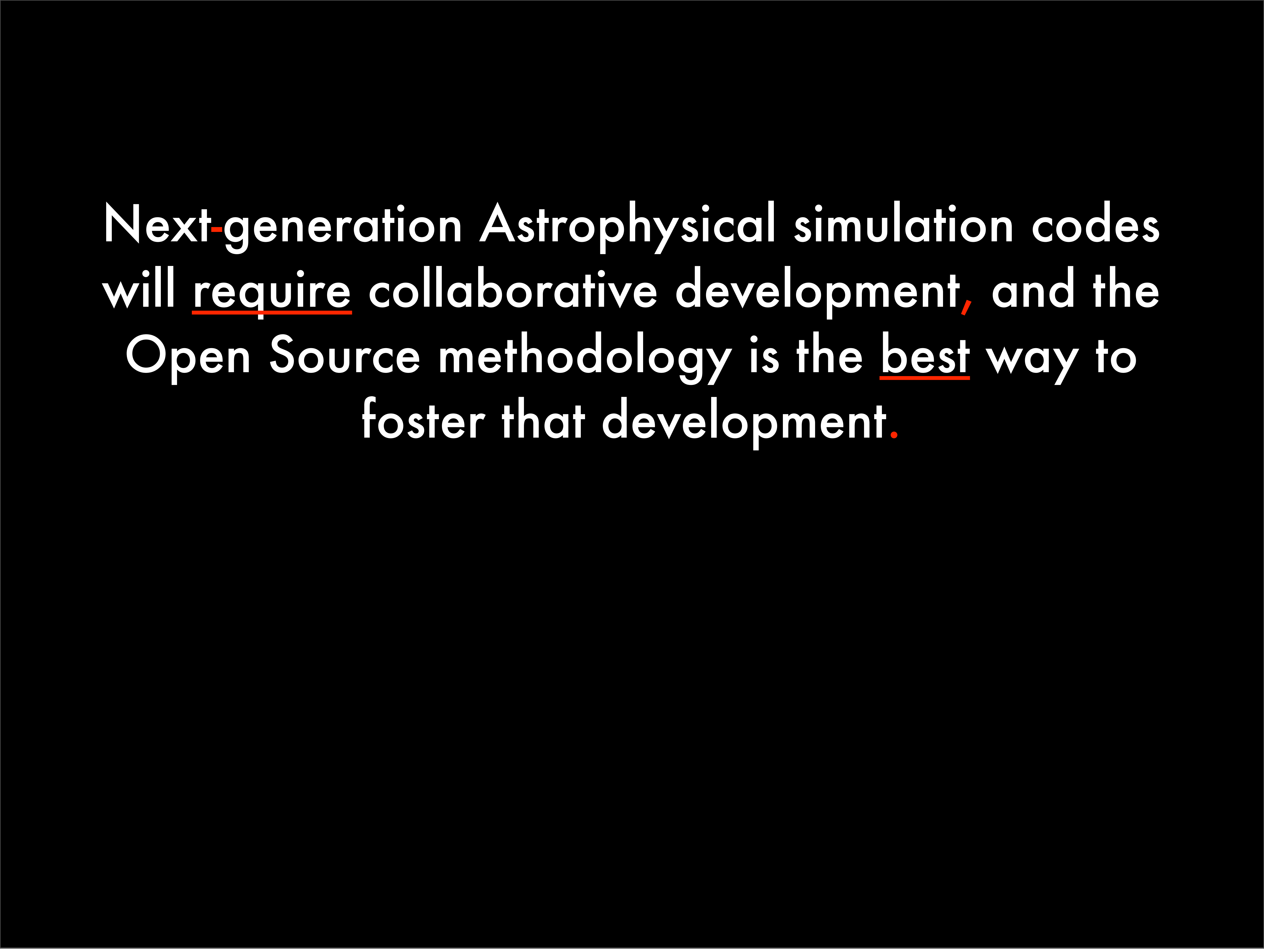
No.

By developing a community of developers and users, the burden is spread around. People become more interested in helping, in contributing, and answering questions. Helping others is analogous to training the next generation of collaborators and researchers.



Next-generation Astrophysical simulation codes will require collaborative development,

In past years, it was common for graduate students or individual researchers to be intimately familiar with -- or even the sole author of! -- astrophysical simulation codes. Modern challenges presented by multiphysics simulations on complex computing platforms, however, will require a collaborative methodology. A single point of failure is no longer acceptable, nor even attainable.

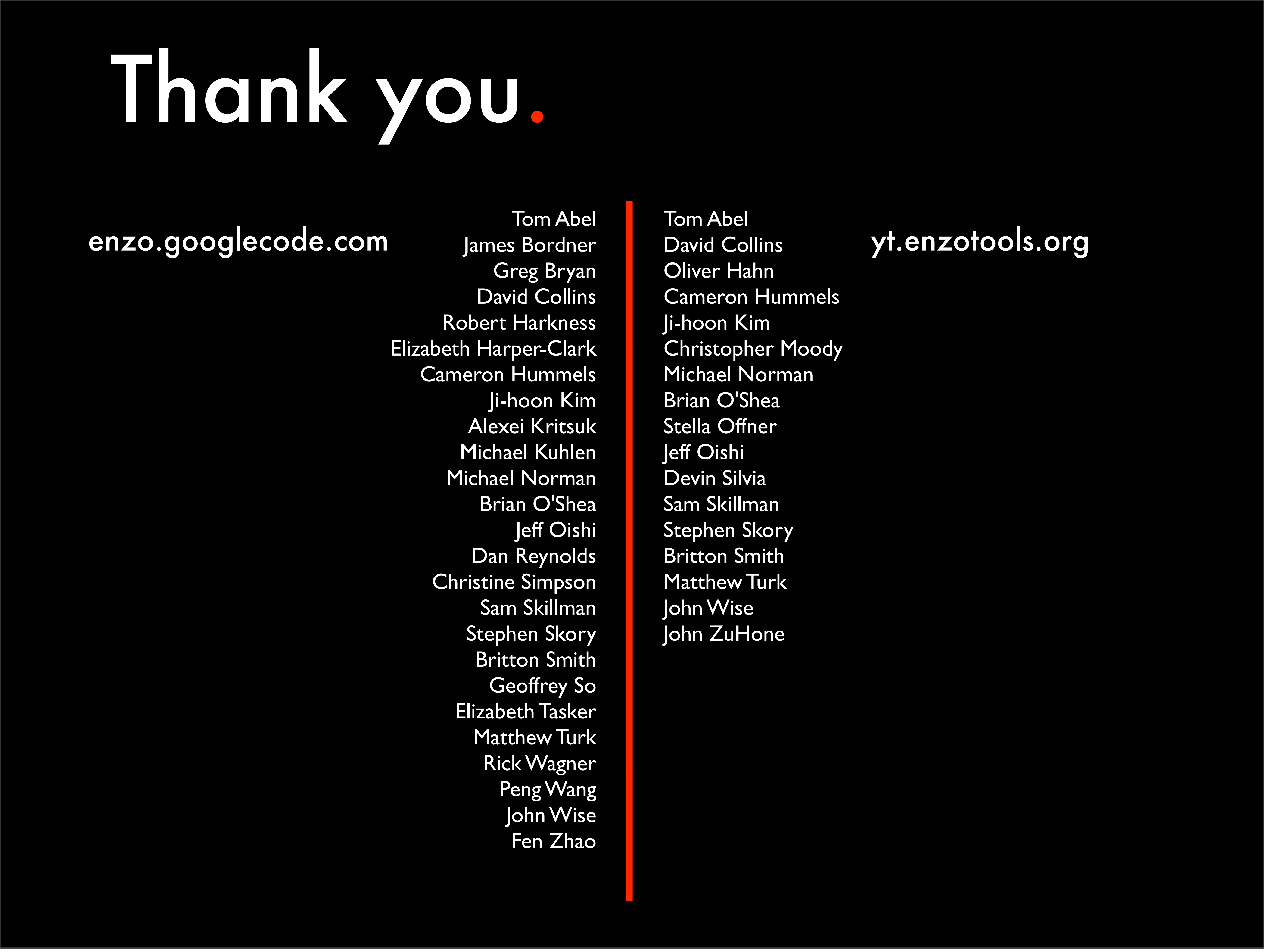


Next-generation Astrophysical simulation codes will require collaborative development, and the Open Source methodology is the best way to foster that development.

The only way to create sustainable mechanism for development is through open source methodologies. This is more than simply putting up tarballs, or distributing source code. It will require collaboration, community development, education, and a willingness to participate. For a relatively small, highly-competitive field like computational astrophysics, this may be a challenge. But it is necessary.

A final thought: who is the “we” in the room?

It’s very common to hear the construction, “If we are to get to exascale...” “If we want to use an exascale machine...” “If we don’t utilize many cores ...” and so on. It’s still not clear to me who this refers to. Is there, in fact, an Astrophysics Simulation community? Or is it really a collection of fiefdoms? And is that a status quo “we” can live with?



Thank you.

enzo.googlecode.com

Tom Abel
James Bordner
Greg Bryan
David Collins
Robert Harkness
Elizabeth Harper-Clark
Cameron Hummels
Ji-hoon Kim
Alexei Kritsuk
Michael Kuhlen
Michael Norman
Brian O'Shea
Jeff Oishi
Dan Reynolds
Christine Simpson
Sam Skillman
Stephen Skory
Britton Smith
Geoffrey So
Elizabeth Tasker
Matthew Turk
Rick Wagner
Peng Wang
John Wise
Fen Zhao

yt.enzotools.org

Tom Abel
David Collins
Oliver Hahn
Cameron Hummels
Ji-hoon Kim
Christopher Moody
Michael Norman
Brian O'Shea
Stella Offner
Jeff Oishi
Devin Silvia
Sam Skillman
Stephen Skory
Britton Smith
Matthew Turk
John Wise
John ZuHone

This is an incomplete listing of the people who have contributed or encouraged development on either of these projects. We have also enjoyed the support of a number of funding agencies, including the Department of Energy, the National Science Foundation, and the University of California HIPACC.