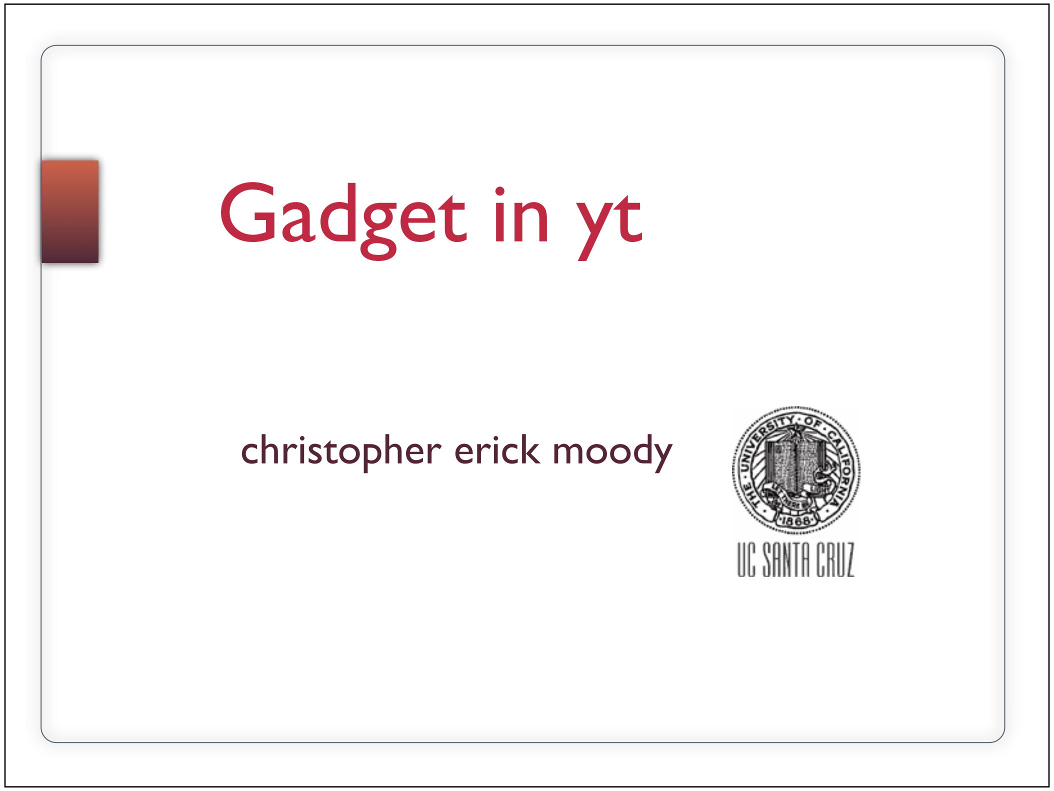
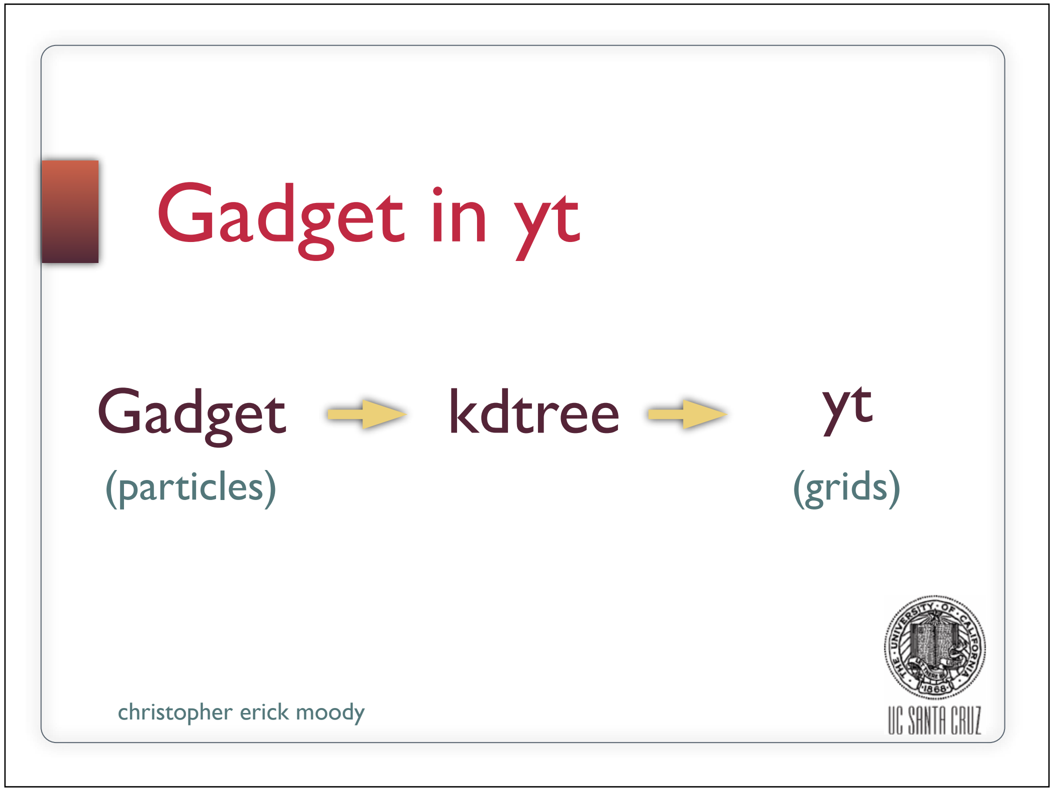
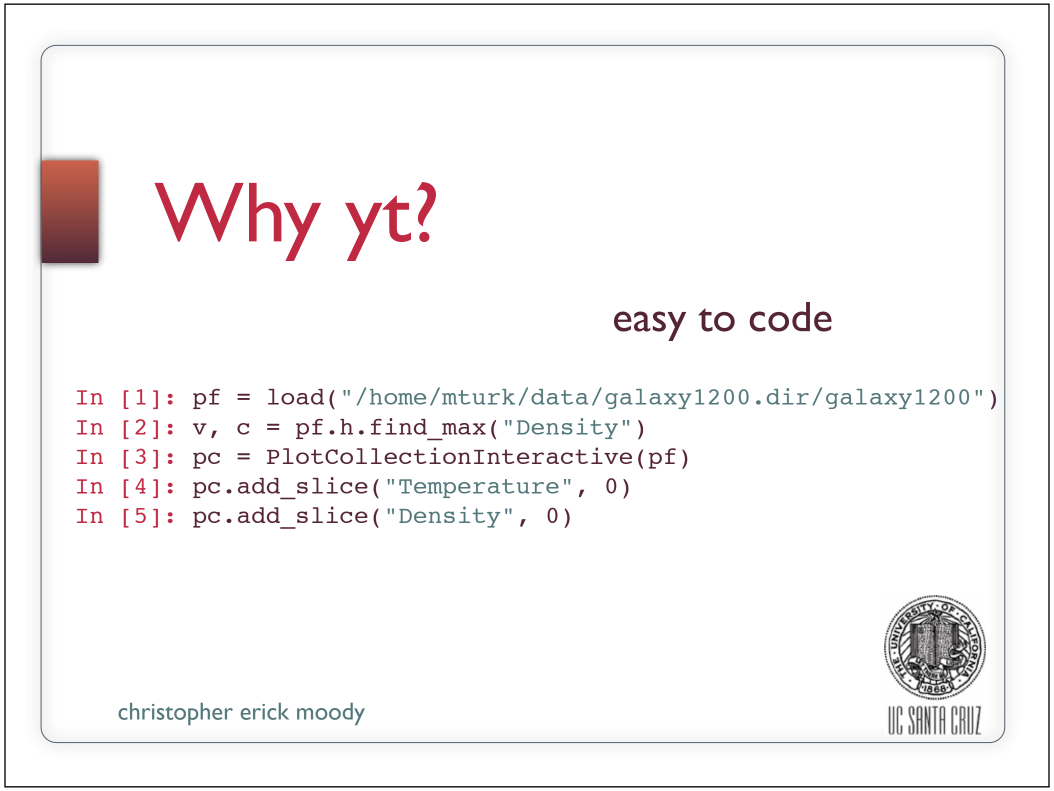


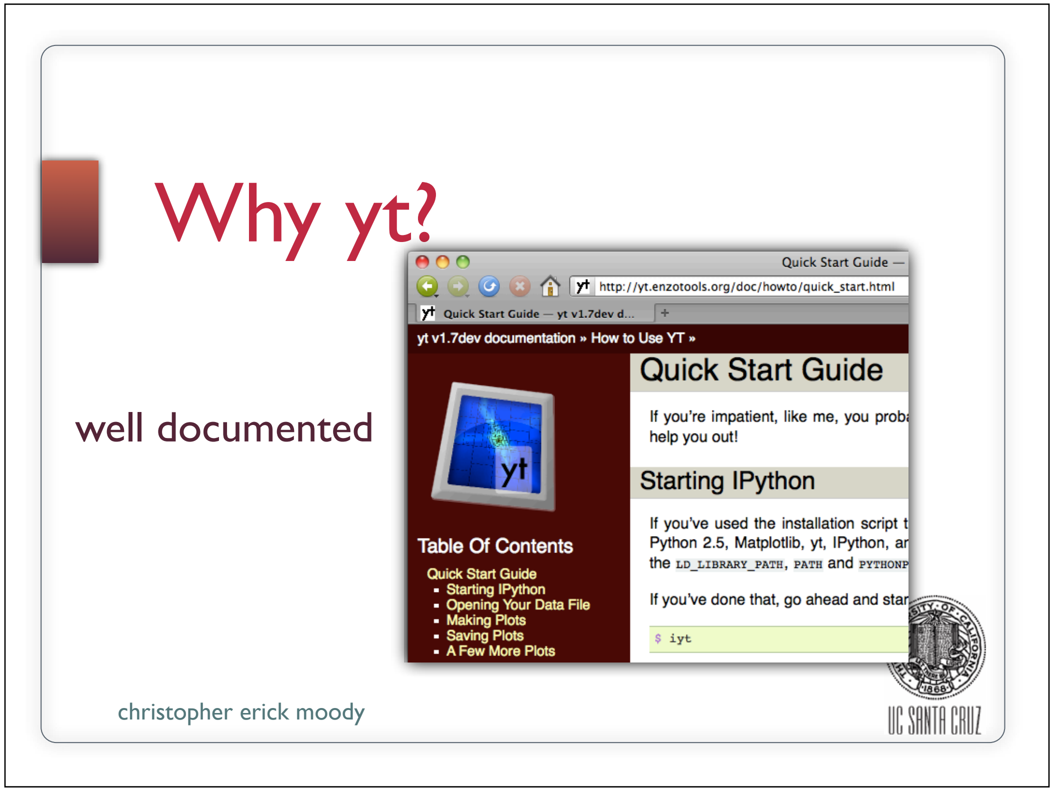
First of all, hello, and thank you for giving me the opp to speak
My name is chris moody and I’m a grad student here at uc santa cruz and I’ve been working with Joel for the last year and a half. Today I’m going to present work I’ve done the last week or so with matt turk.



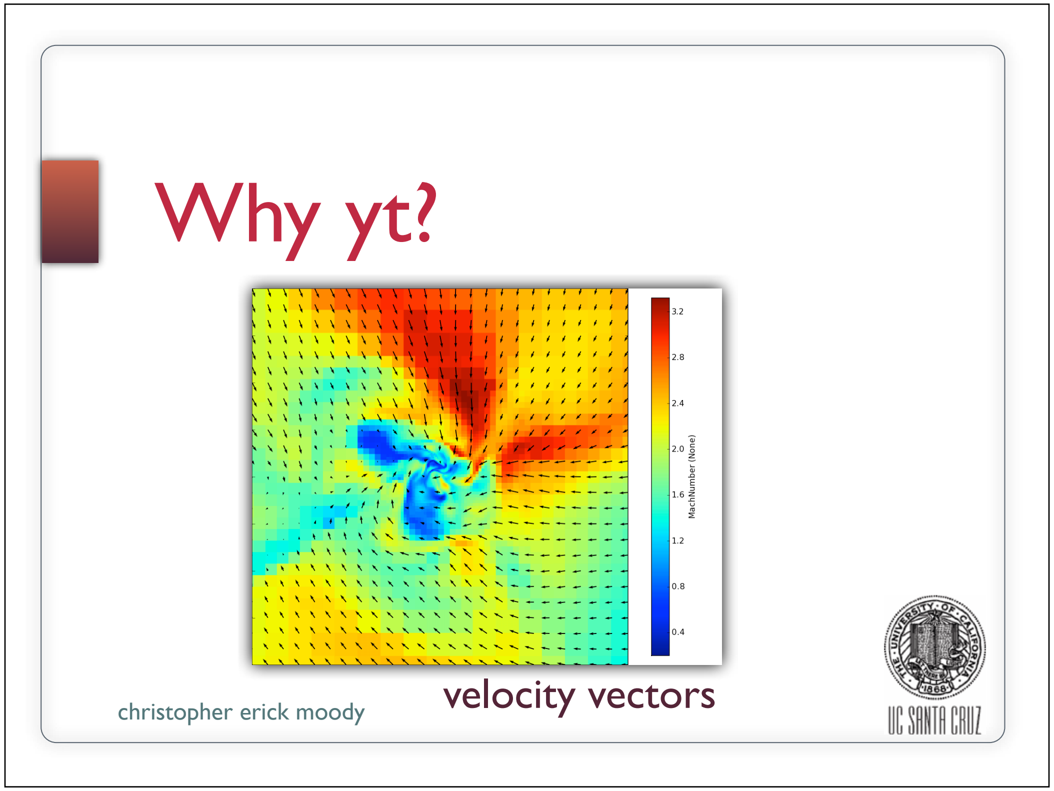
gadget as you know is a particle code while yt is built around enzo, and adaptive mesh code. The two, at least without some work, don't play that well together. But getting particles into yt shouldn't be difficult; the crux of the problem is getting particles onto a grid; and then we have to go backwards and given a grid, what particles exist in it? We have to do this quickly and efficiently. To go between the two, you need a kdtree, which you can think of as an octree. The tree divides a box of particles into smaller nested boxes.
And in the ten minutes I have right now, I'm going to try and convince you to use the my code.



But first I'll review why you should use yt at all. Matt Turk and Mike Norman have tried convince you coding in yt and enzo is an easy thing to.
Just look a this: 5 lines load in a dataset, find the peak density, and plot the temp and dens along a slice.
Furthermore, this little bit of code I have here is copied directly off of their...



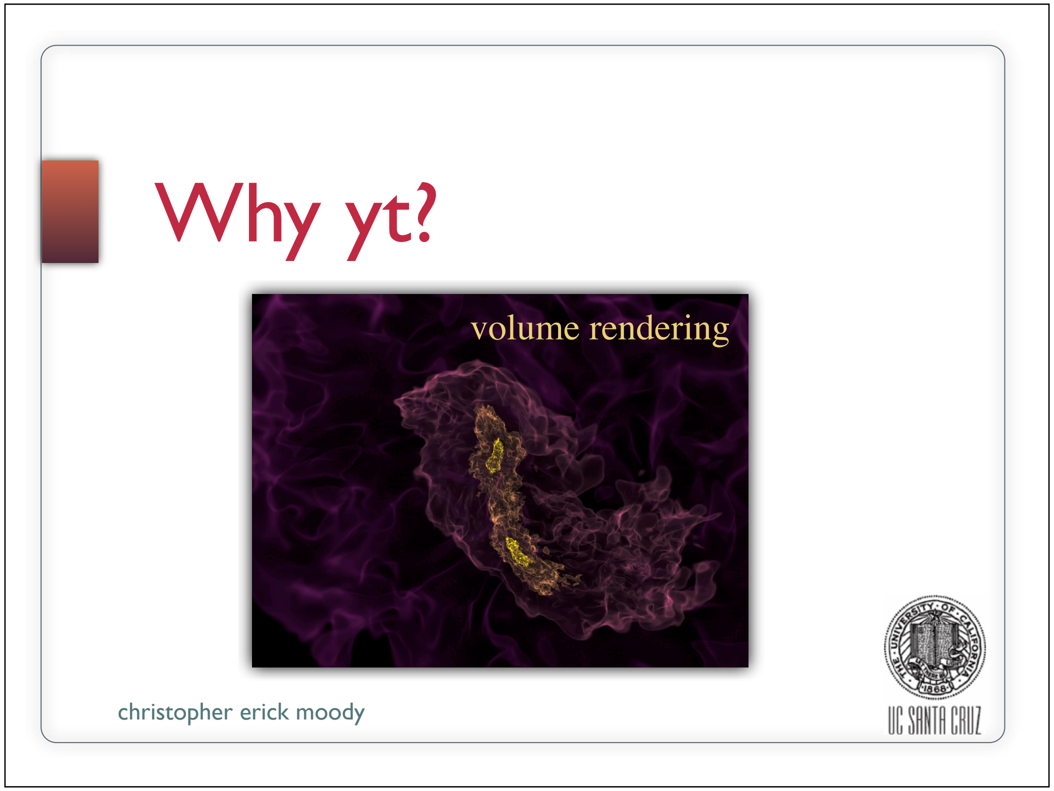
...well documented website.
Documentation and ease of use is the key to efficient research. They implicitly promise you a quick analysis cycle.



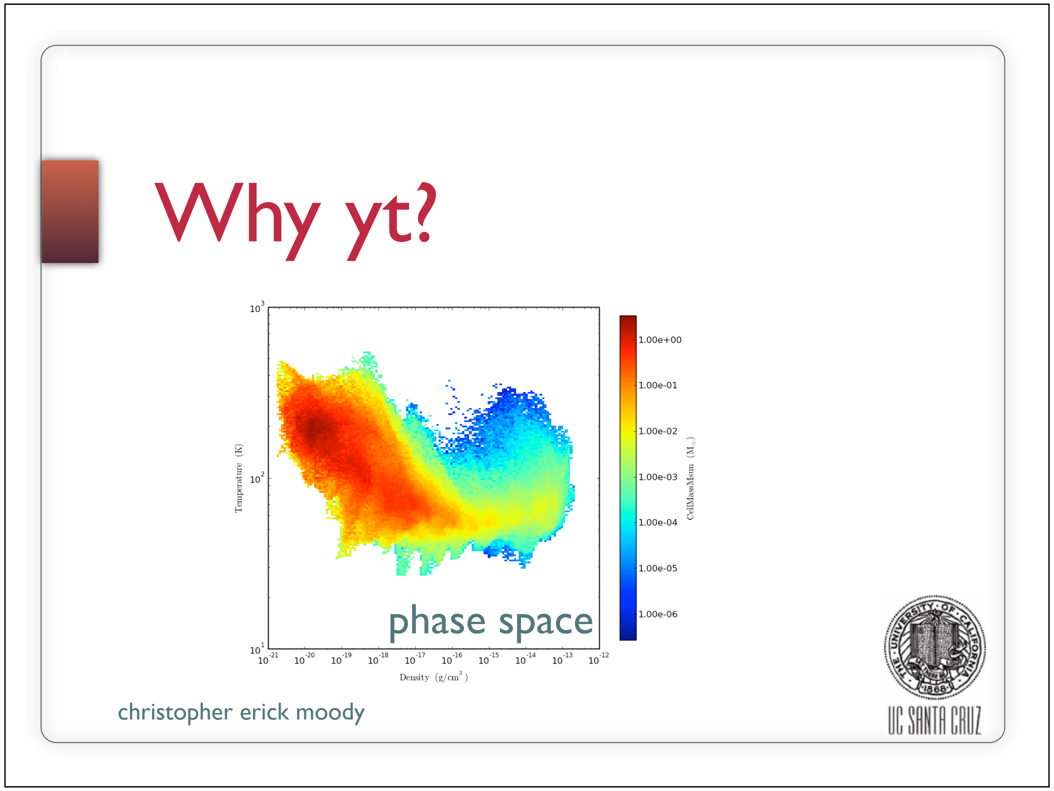
And those are good, utilitarian reasons to use yt. But I’m to tell you why yt is awesome.

Velocity map with a superimposed vector field - all inside a collapsing and rotating star forming region.

Not my research. Note the variable size grid. Lower resolution out here, higher on the inside. Trying to write code or compute quantities that live on an adaptive mesh like this is more difficult than doing the same thing with particles. But I'm glad I won't have to because of yt. But yt can do more than just 2d plots...

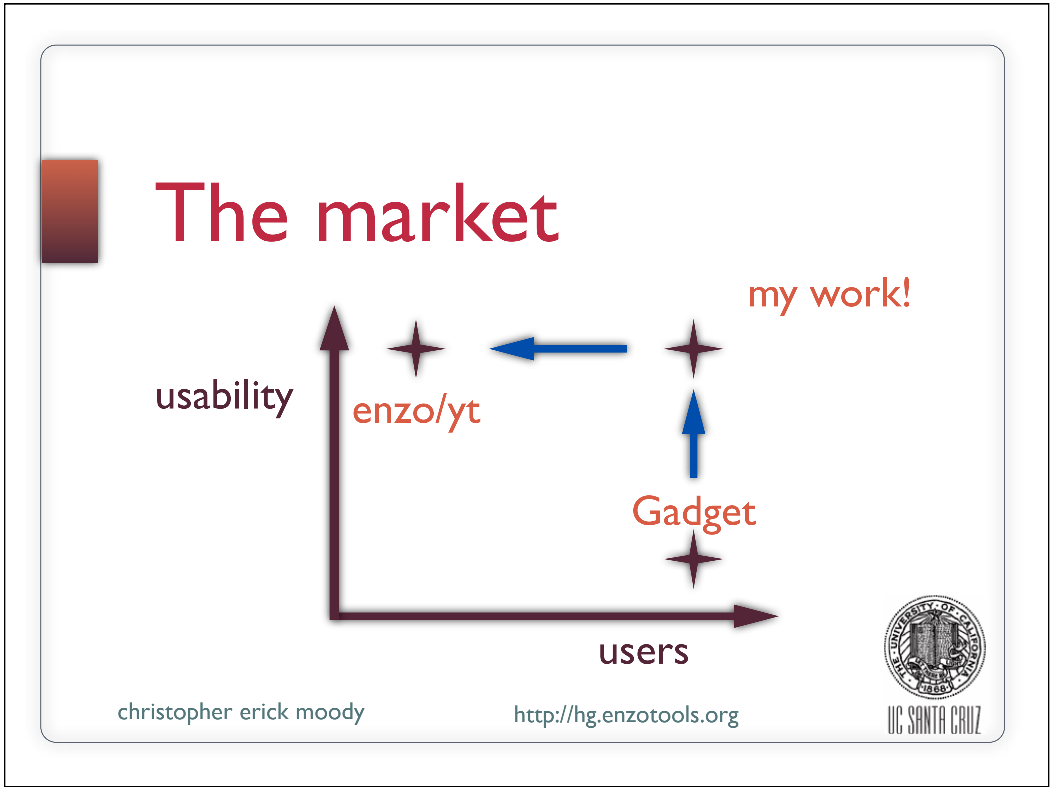


Volume rendering Pop III SF halo - you can visually identify the two fragmented clumps. The gaming industry has made a big business out of plotting this kind of thing very quickly - but for us, it's here for free. (SLOW!) One of the advantages of working in this field is that easily translates into spectacular imagery. Now I wouldn't dare call us artists, but I think this beauty can really capture our imagination.

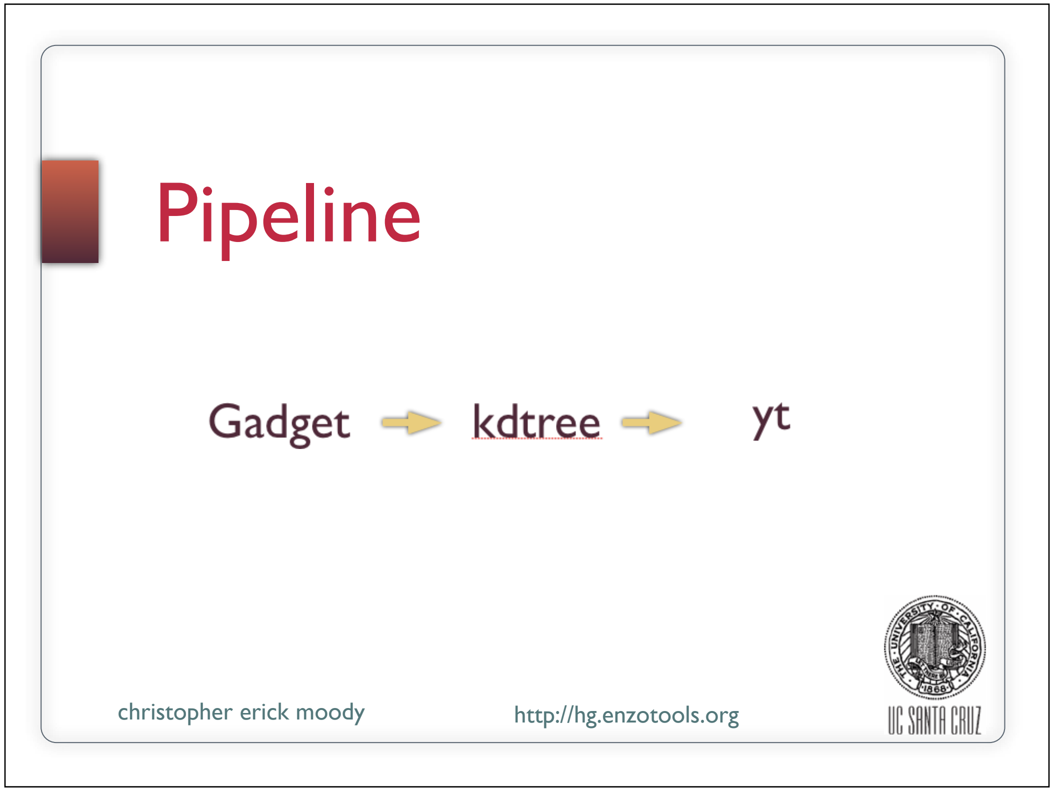


Of course not everything can be pretty - so all we ask for then is simplicity. Here is a phase space diagram showing the distribution of mass in the Density-Temperature plane in a collapsing pre-stellar cloud. A complex plot like this is still easily rendered in yt. Moral of the story is that it’s a pleasure to analyze data in yt.

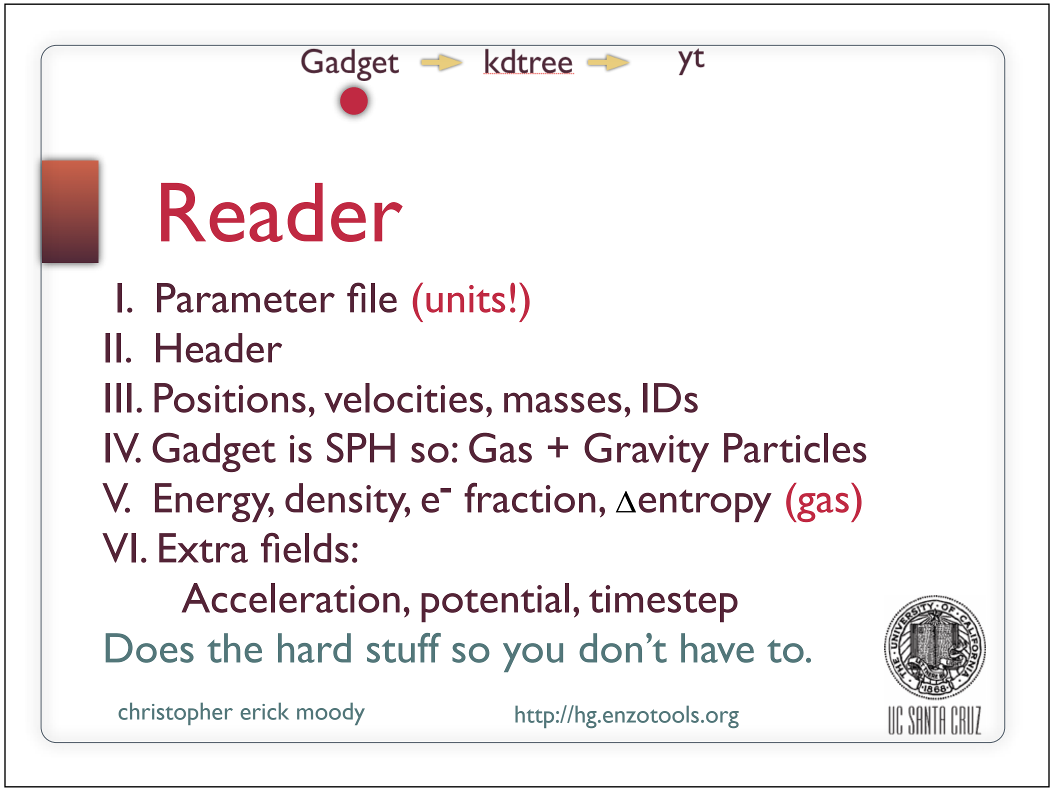
On the other side of the fence we have Gadget. We all know about it. It ran famous simulations like Millennium run in 2005… And this wouldn't really be giving a legitimate talk at a computational astrophysics school if I didn't include at least one mesmerizing, zoom-in movie. So here you go. The code paper for Gadget has over 845+ citations on it. Enzo has just over 60. There are caveats to just comparing the number of citations, but I like think of it as a first guess to popularity. Point is, that there are a lot of Gadget snapshots out there.



Two years ago when I first started working with Joel, he had a list of projects for me. We went through each one and graded them in three respects. Interest, impact, and for the lack of a better word, easiness. Interest is pretty self-explanatory - do you have an emotional connection with your work? If a lot of folks can use your work, or if your theory is rooted in obs, the better the impact. And if you have the resources you need - it's certainly easier. So I'm used to making these kinds of plots. Gadget as I said before has a lot of users - but yt/enzo makes up for it in usability. Gadget as far as I know, doesn't the analysis suite enzo does. But this is perfect for me.
From my work in gadget I had already written code to read in snapshots, makes octrees, and on top of all that it was already in python - perfect for yt! I was in the right place in the right time, with the right resources to make an impact. So with some luck, my code combines the userbase of Gadget with the yt toolset. That way we can all have the best of both worlds.

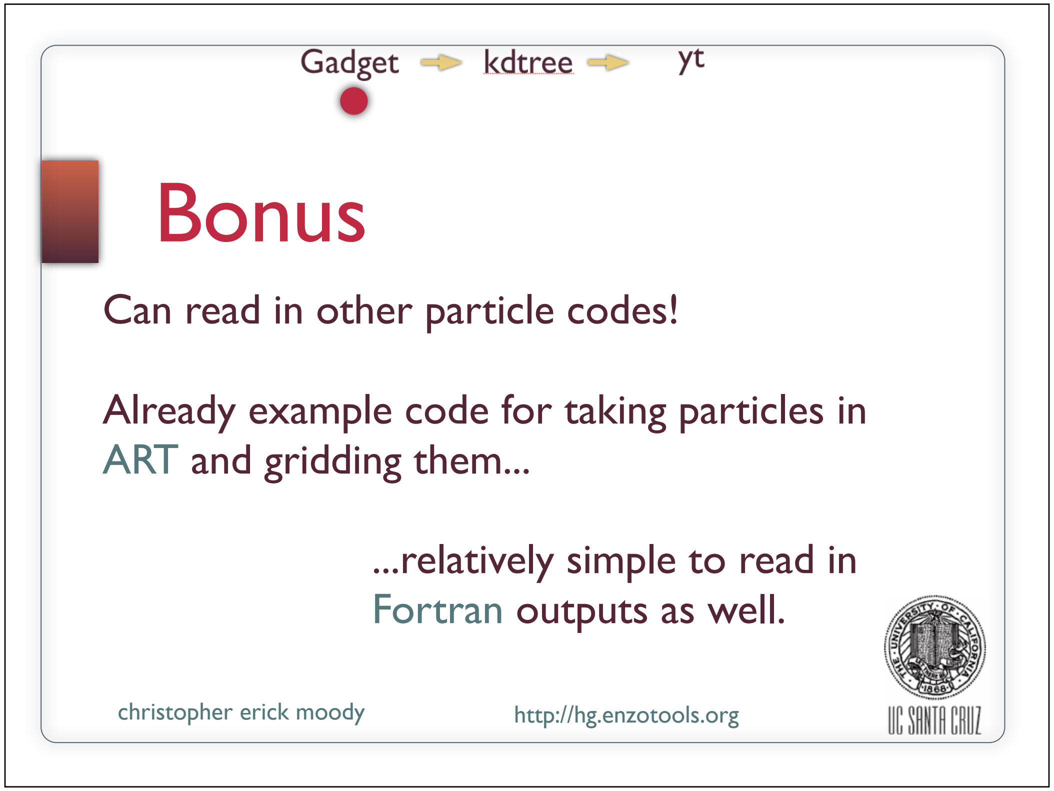


Let’s go back to 1st slide, where I show my program’s pipeline
Let’s keep this in mind as we go step through the program

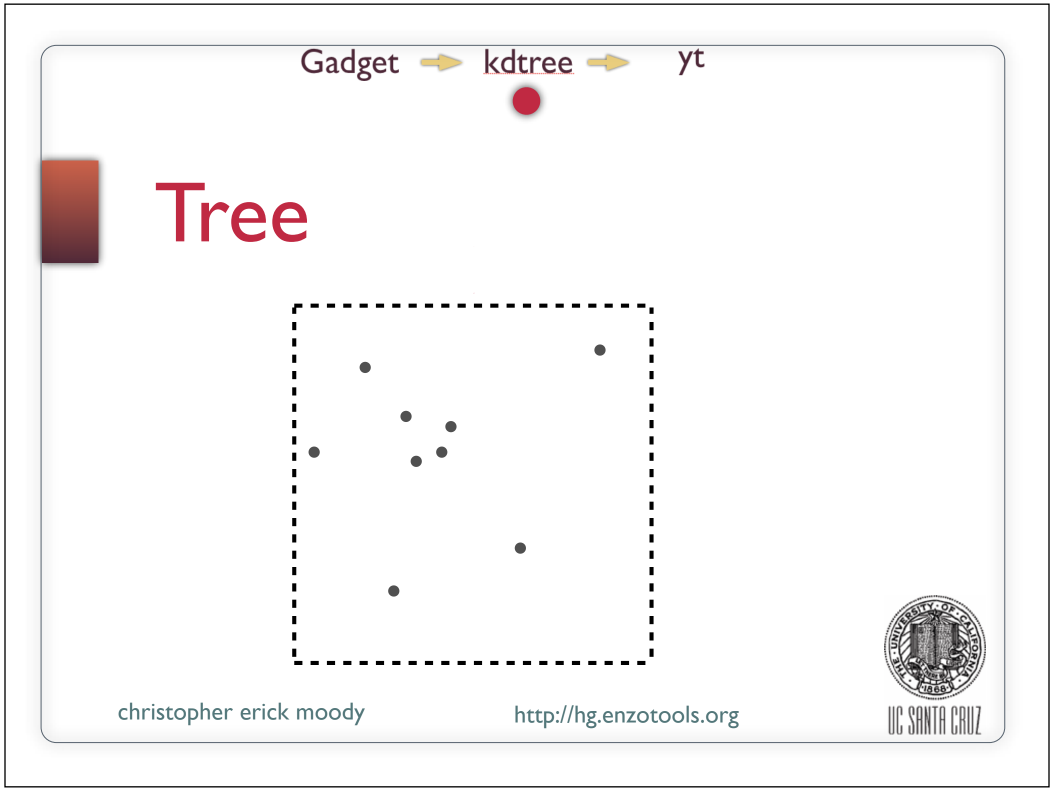


The reader has to be prepared for any kind of gadget snapshot. It tries to read as much information as it can. That means reading in the parameter file, which typically contains the simulations parameters like the softening length. It will read in units – if you are like me, automatic detection of units will save you a few hours of headaches. Now, the header of each snapshot has other flags necessary for reading the rest of the file – such as how many gas or how many dark matter particles there are. Whether cooling is included or not.

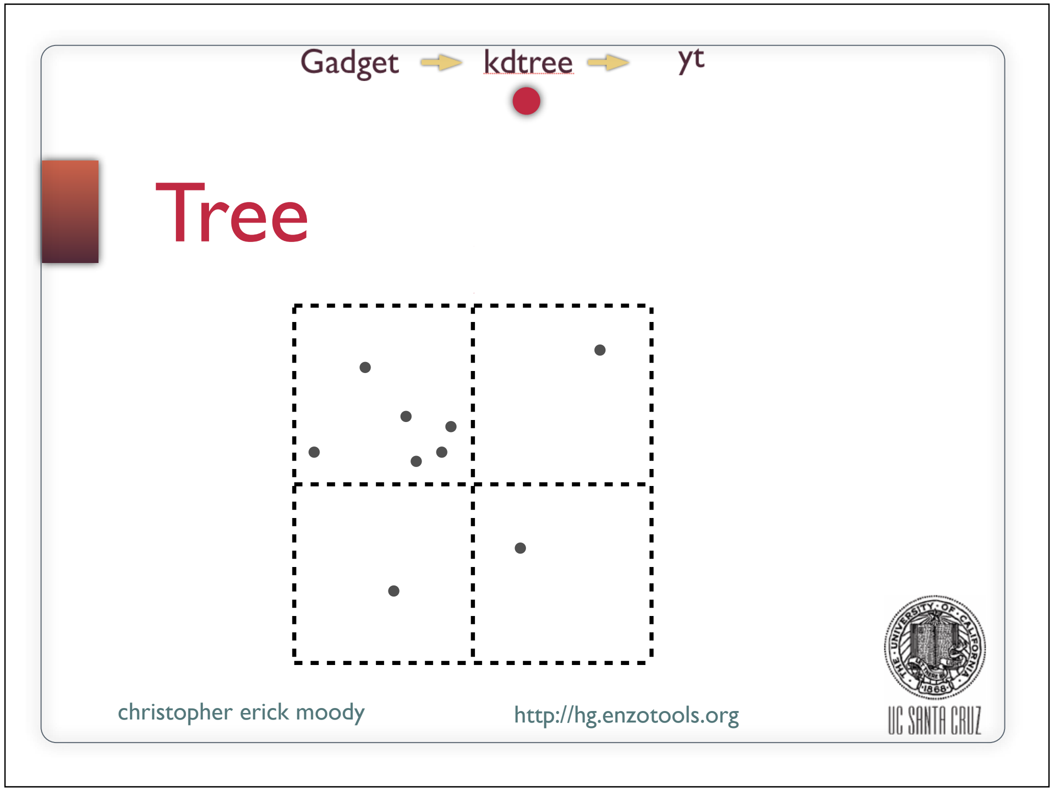
A given snapshot might have any combination of these extra fields, and it's difficult to know ahead of time. The point is it does the hard stuff for you so you don't have to.



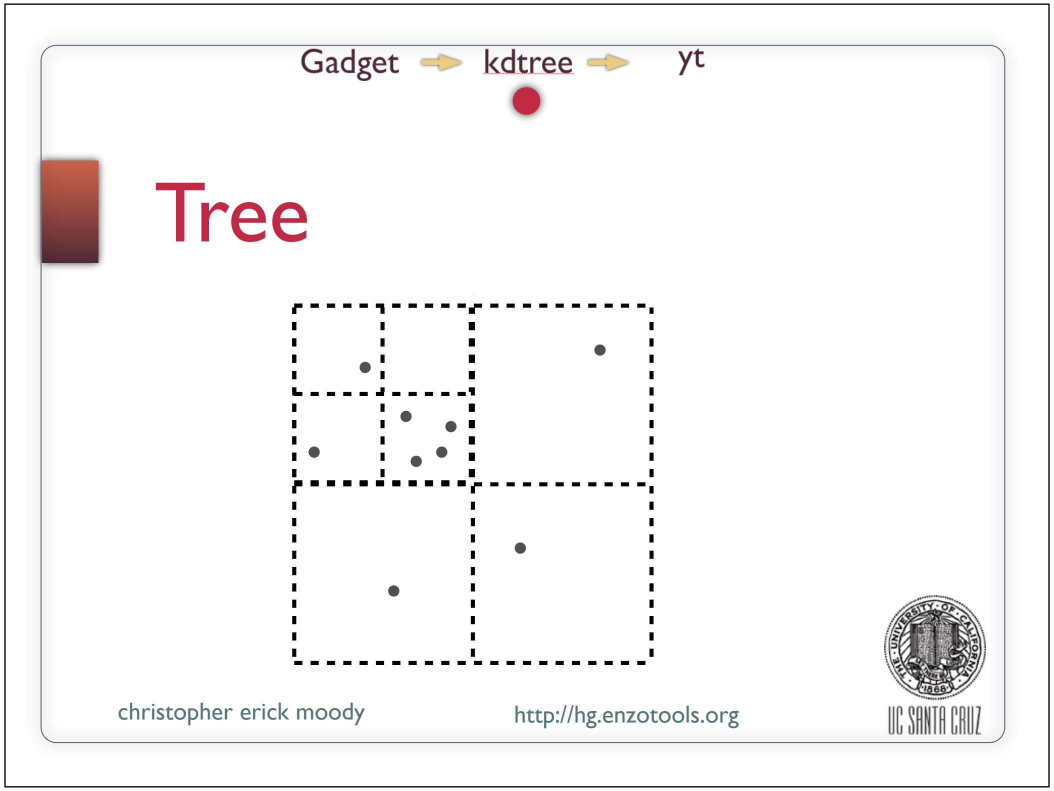
Can read in other particle codes! and run them through the same procedure.It’s not limited to just gadget. I’ve been looking at Daniel Ceverino’s simulations, which are based off of ART code, and looking at just the particles. Because they are particles, they should be simple to push through the same pipeline. And I’ve already got some example code for reading in this binary files up on the website as well.



Step two, after having read in the particles, we have to apply a grid.

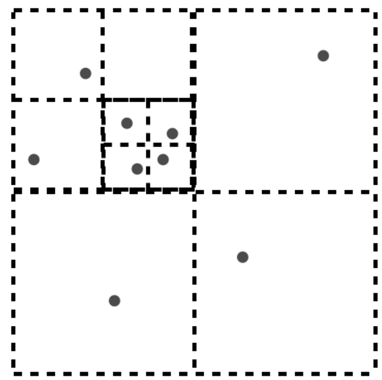


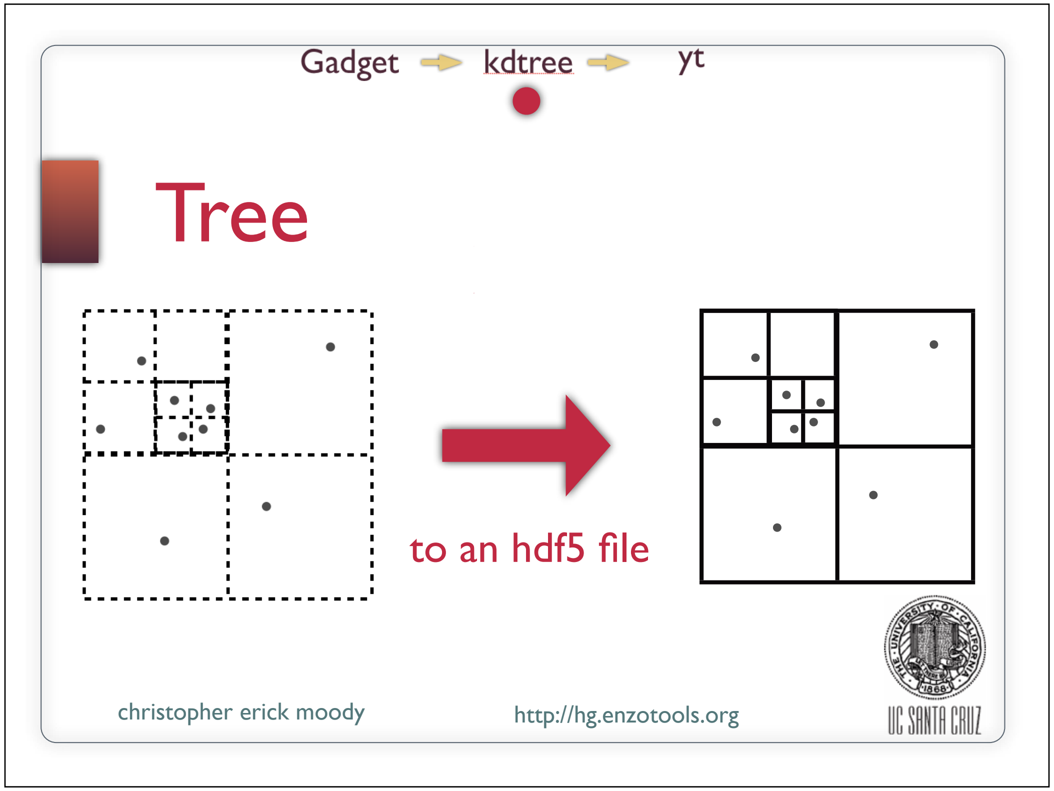
We do so recursively, subdividing the position space into smaller ...



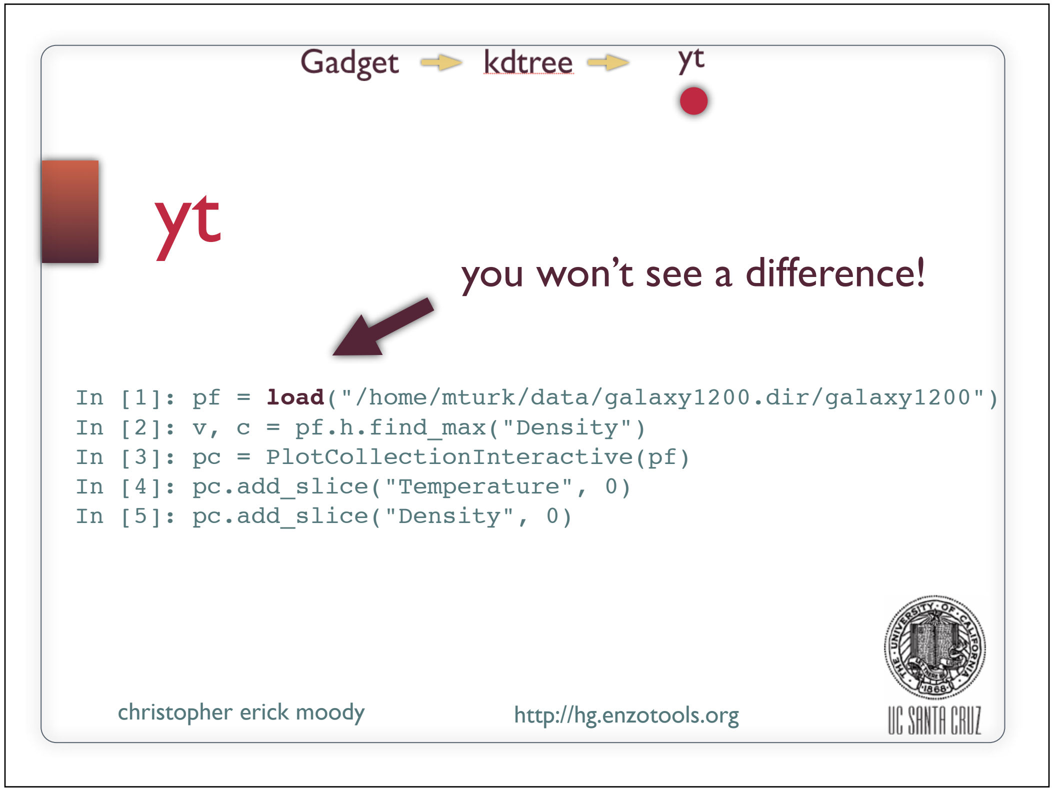
...and smaller boxes.

Tree

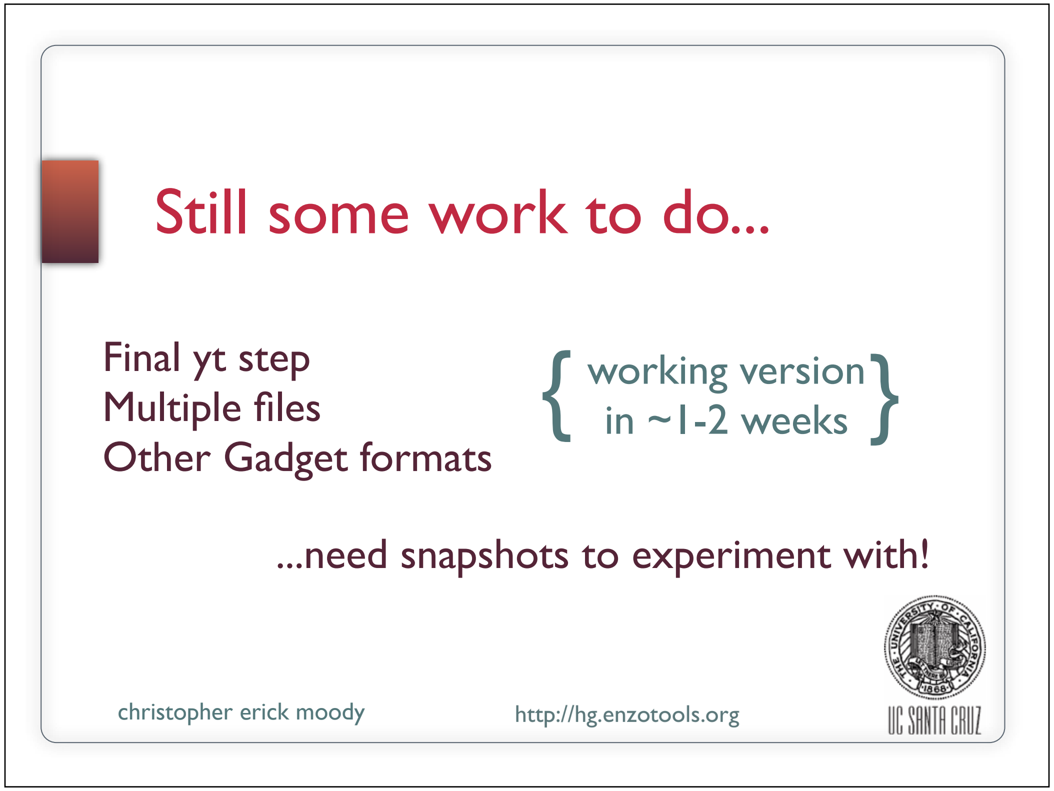




When we're done, we save the data to an hdf5 file for portability. yt is designed for efficient use of memory; that means it reads only the boxes that it needs. Otherwise we'd have to store the entire snapshot in memory, which is expensive and wasteful for large simulations. So my code rearranges the data so that on the disk they are stored in a spatially logical order. So if yt tries to calculate the where the peak density is, it might want a particular region of data, such as these boxes. Before it would have had to load the entire thing in, but now it knows immediately where this information is stored – and it doesn't have to load the entire snapshot. It is efficient, and speedy.



Now back to the second slide I showed you.
You'll run my code externally – just a single command – and it will translate your snapshot to something yt can read. All of the technology is behind the scenes, all inside this one load statement. yt is designed around simplicity, and so is my add-on. And the beauty is that you won't notice a difference.
Now that hopefully I've peaked your interest to use my code...



...I can tell you all the caveats. Matt and I are working on the final importing to yt step. It's not finished yet. Another feature I hope to get running soon is multiple gadget files. Gadget snapshots can be written in parallel, which for massive simulations means that a single output is split up into many files. That's not supported right now either, but it will be soon. Gadget also has two other, less popular snapshot formats, and there's no support for them at the moment either. With some luck, we can get a preliminary working version up in the next week or two. Also, while I'm up here, if anyone has snapshots that aren't proprietary and you don't mind parting with, I could really use them to test my code against.

I'm hoping I've convinced you to use it. You can get it here, at hg.enzotools.org for the low, low cost of your firstborn child. Nah, just kidding it's free. Once again my name is chris moody, thank you.