Does J have a Place in Scientific Computing?

The View from Astronomy

by J. Patrick Harrington

J is a wonderful language, but it is known/used by relatively few programers.

There is always discussion of how we might expand our user base. And one part of that discussion is the question:

What is our competition ?

Since this discussion is generally rather data-free, I have taken a little survey of the astronomers at the University of Maryland, College Park campus.

A survey of 60 astronomers at U Md.

"When you program, what are the 3 languages you use most?"

Language	1st	2nd	3rd	total
IDL	18	9	5	95
C	12	11	7	77
Python	9	9	11	65
Fortran	7	11	7	57
Perl	2	3	5	19
C++	3	1	4	18
Matlab	3	1	3	17
R	2	3	0	14
Mathematica	0	1	5	7
J	1	0	0	4
IRAF	0	1	1	3
PHP	0	1	0	2
Java	0	0	1	1

total = 4*(1st)+2*(2nd)+(3rd)

(Responses from professors, researchers & grad students)

I was not surprised at 3 of the top 4 languages: astronomers write big compute-intensive programs in Fortran and C because they still produce compiled code that executes faster than other languages. And I was aware that Python is growing in our field.

But why is IDL number one?

Features of IDL (Interactive Data Language):

(1) it is interactive (as opposed to compiled)
(2) from its earliest days, has very good graphics
(3) it is an array based language (so important for the analysis of images !)
(4) it has a syntax which is rather fortran-like

J shares the first three of these attributes. In fact, the early documentation states that IDL was in part inspired by APL! But I have never been able to interest any astronomers in J. It's that last point: the familiarity of the fortran-like syntax. J just seems too strange.

Another key point: The adoption of the language by a "critical mass" of astronomers. I think its use really caught on when, before the launch of the Hubble Space Telescope in 1990, a team of NASA astronomers used IDL for the software to analyze HST images. This introduced hundreds of astronomers (including me) to IDL. And we have continued to use it for data analysis in subsequent space missions and orbiting astronomical satellites.

So the bar is pretty high for the adoption of a programming language in any of the sciences. Science is becoming ever more collaborative, so there is less opportunity for the lone J programmer to function outside a group.

I think J will always hold its own as the "secret weapon" of the few who have discovered it.

And we can always hope that some genius will write a "killer app" in J and bring this language to wide attention.

I've used J for over 20 yrs, but am no expert.
So I can't show this audience any interesting code.

I write big programs only occasionally, when I have a particular problem to solve. But J seems so natural and quick, that I rarely use anything else.

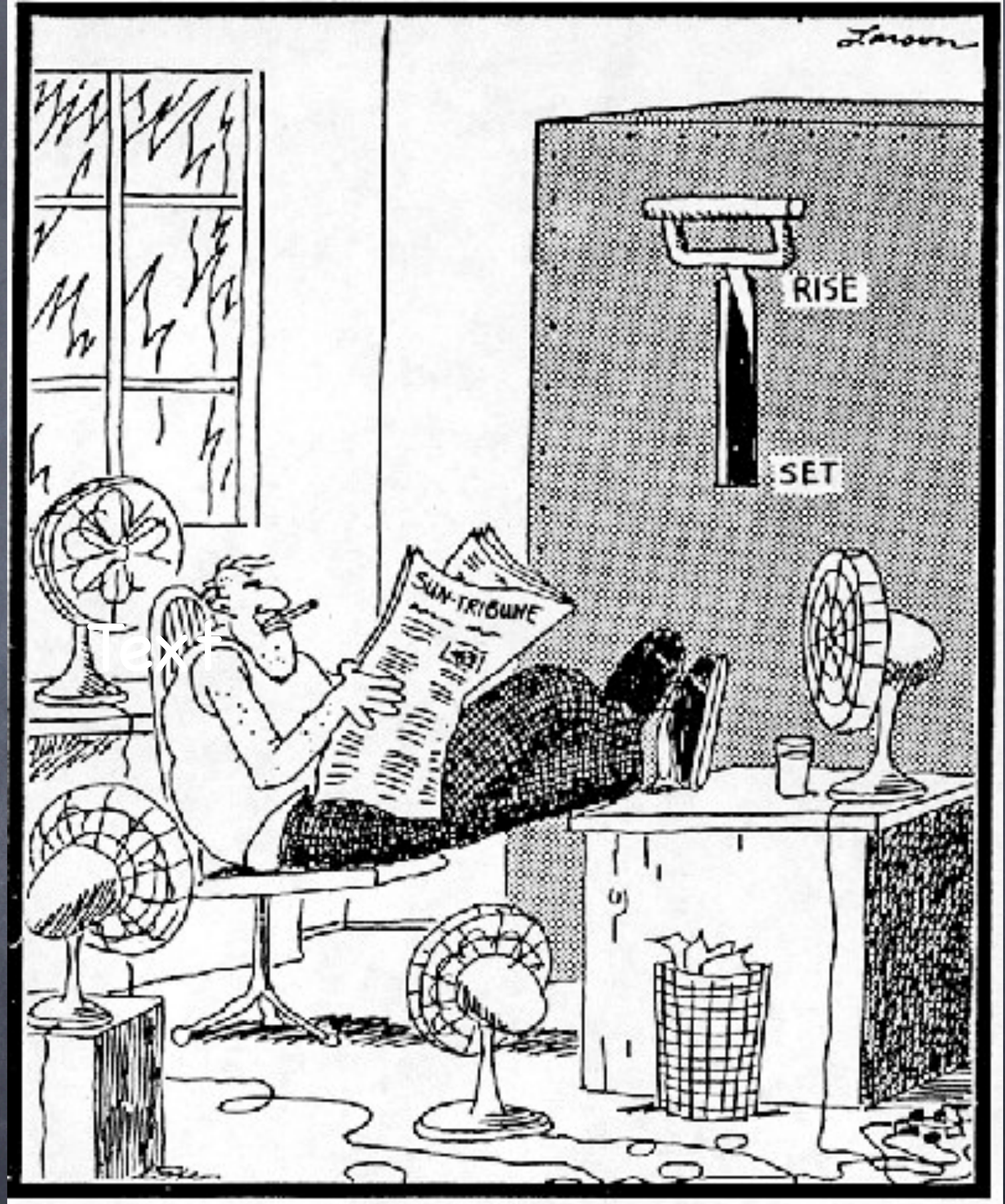
Since I can't really give you a J talk, I'm going to tell you about a piece of astronomical software that I hope some of you may find interesting.

Last semester I taught a course on stellar evolution for physics and astronomy seniors. I discovered that there is a wonderful, research-quality stellar evolution program that has just appeared: MESA: "Modules for Experiments in Stellar Astrophysics"

Have you ever wanted to look at all the details of what goes on inside a star as it evolves from a gas cloud to a white dwarf ? Or even better, a core collapse supernova?

Then MESA is the thing for you! You can do real astrophysics without all those years of grad school!

(too bad the source code is in fortran -- fortran 95!)



Inside the sun

The MESA home page at mesa.sourceforge.net

Open source code for stellar evolution that will run under linux or Mac OS X

(it runs fine on my MacBook Pro under OS 10.6.8)

MESA home

http://mesa.sourceforge.net/

Reader

Address Book ▾ WunderMap® Physics Documents NuVoc – J Wiki J local MARCS SAO/NASA ADS Bookmarks f... Harrington artprice.com...infor

MESA
Modules for Experiments in Stellar Astrophysics

MESA 1st Author
Bill Paxton

MESA Area Stewards
Asteroseismology
Rich Townsend
Binaries
Pablo Marchant Campos
Microphysics
Frank Timmes

MESA Council
Lars Bildsten
Aaron Dotter
Falk Herwig
Frank Timmes
Ed Brown
Rich Townsend
Matteo Cantiello

Hosted by Sourceforge
Source on GitHub
Generated by jekyll
Design by Andreas Viklund

MESA

Why a new 1D stellar evolution code?

The MESA Manifesto discusses the motivation for the MESA project, outlines a MESA code of conduct, and describes the establishment of a MESA Council. Before using MESA, you should read the **manifesto document**. Here's a brief extract of some of the key points

Stellar evolution calculations remain a basic tool of broad impact for astrophysics. New observations constantly test the models, even in 1D. The continued demand requires the construction of a general, modern stellar evolution code that combines the following advantages:

- **Openness**: anyone can download sources from the website.
- **Modularity**: independent modules for physics and for numerical algorithms; the parts can be used stand-alone.
- **Wide Applicability**: capable of calculating the evolution of stars in a wide range of environments.
- **Modern Techniques**: advanced AMR, fully coupled solution for composition and abundances, mass loss and gain, etc.
- **Comprehensive Microphysics**: up-to-date, wide-ranging, flexible, and independently useable microphysics modules.
- **Performance**: runs well on a personal computer and makes effective use of parallelism with multi-core architectures.

Users are encouraged to add to the capabilities of MESA, which will remain a community resource. However, use of MESA requires adherence to the MESA code of conduct:

- That all publications and presentations (research, educational, or outreach) deriving from the use of MESA acknowledge the MESA Instrument papers.
- That user modifications and additions are given back to the community.
- That users alert the MESA Council about their publications, either pre-release or at the time of publication.
- That users make available in a timely fashion (e.g., online at the MESA website) all information needed for others to recreate their MESA results -- "open know how" to match "open source."

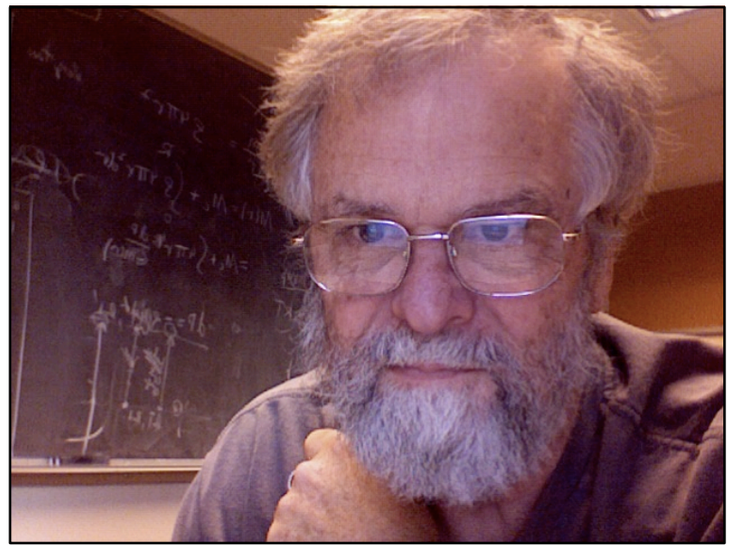
Latest News

- 14 Jul 2014 » **Installation Screencast**
- 01 Jul 2014 » **MESA Release Zip files**
- 08 Jun 2014 » **Release 6596**
- 24 Mar 2014 » **Release 6208**
- 21 Mar 2014 » **New MESA Website**
- 21 Mar 2014 » **Release 6188**
- 13 Mar 2014 » **New MESA SDK Version**
- 28 Feb 2014 » **Release 6022**
- 03 Feb 2014 » **Summer School 2014**
- 05 Jan 2014 » **Release 5819**

MESA

Modules for Experiments in Stellar Astrophysics

MESA is a state-of-the-art, modular, open source suite for stellar evolution



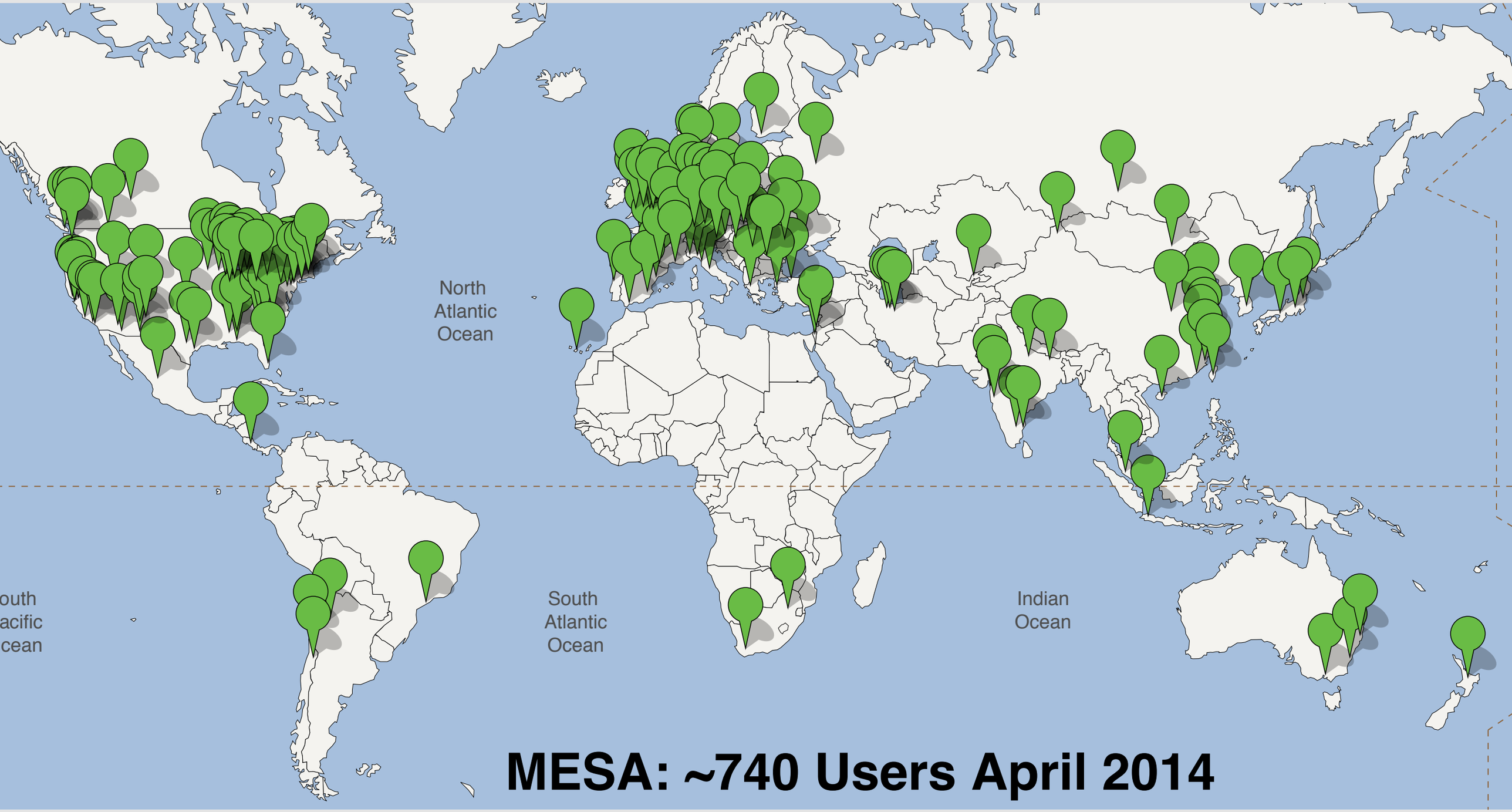
Bill Paxton, father of MESA

- MESA Stellar Evolution Code: mesa.sourceforge.net
- MESA Instrument Papers (Paxton et al. 2011, 2013)

MESA

- **Openness**: anyone can download sources from the website.
- **Modularity**: independent modules for physics and for numerical algorithms; the parts can be used stand-alone.
- **Wide Applicability**: capable of calculating the evolution of stars in a wide range of environments.
- **Modern Techniques**: advanced AMR, fully coupled solution for composition and abundances, mass loss and gain, etc.
- **Comprehensive Microphysics**: up-to-date, wide-ranging, flexible, and independently useable microphysics modules.
- **Performance**: runs well on a personal computer and makes effective use of parallelism with multi-core architectures.

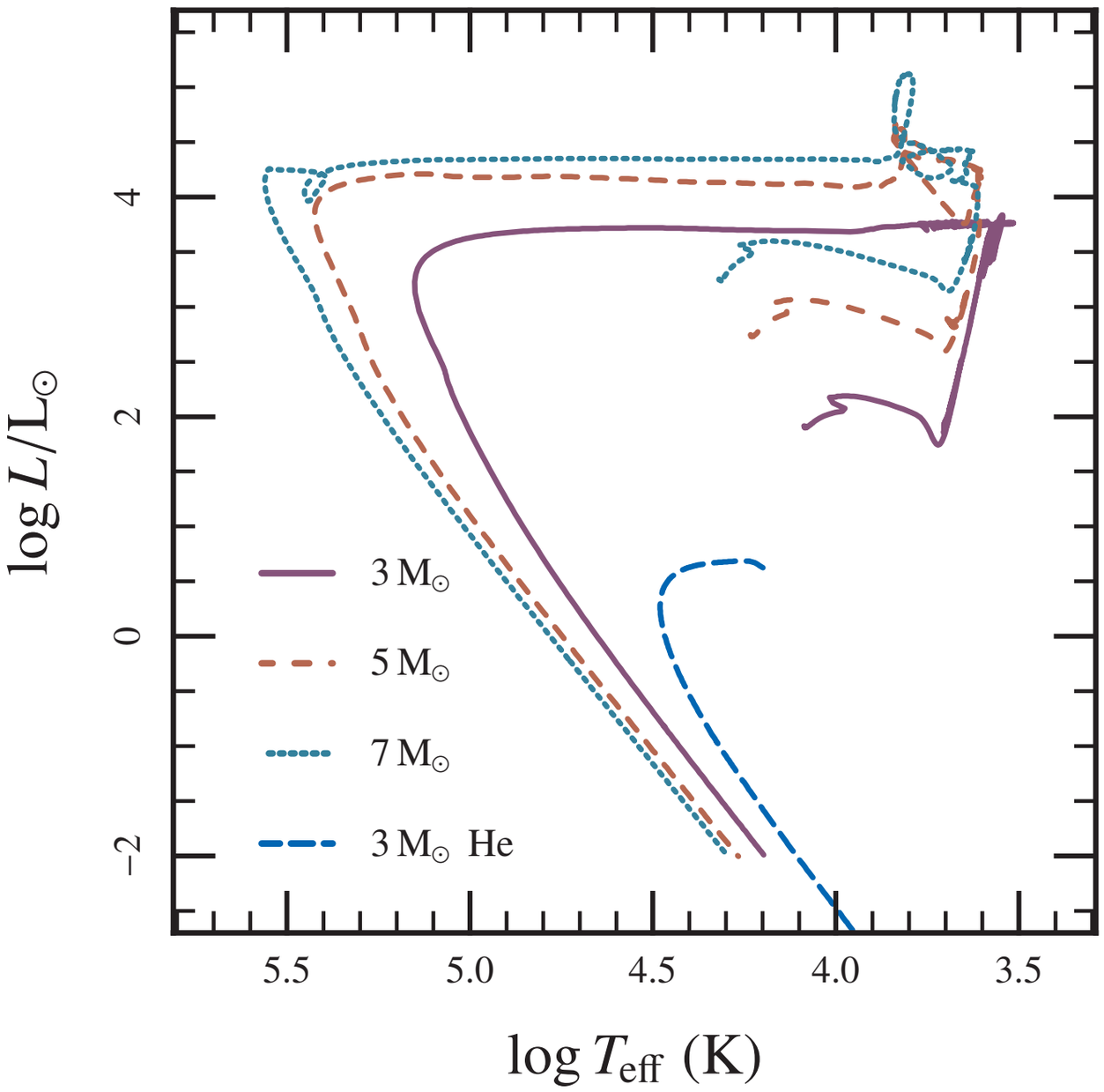
MESA

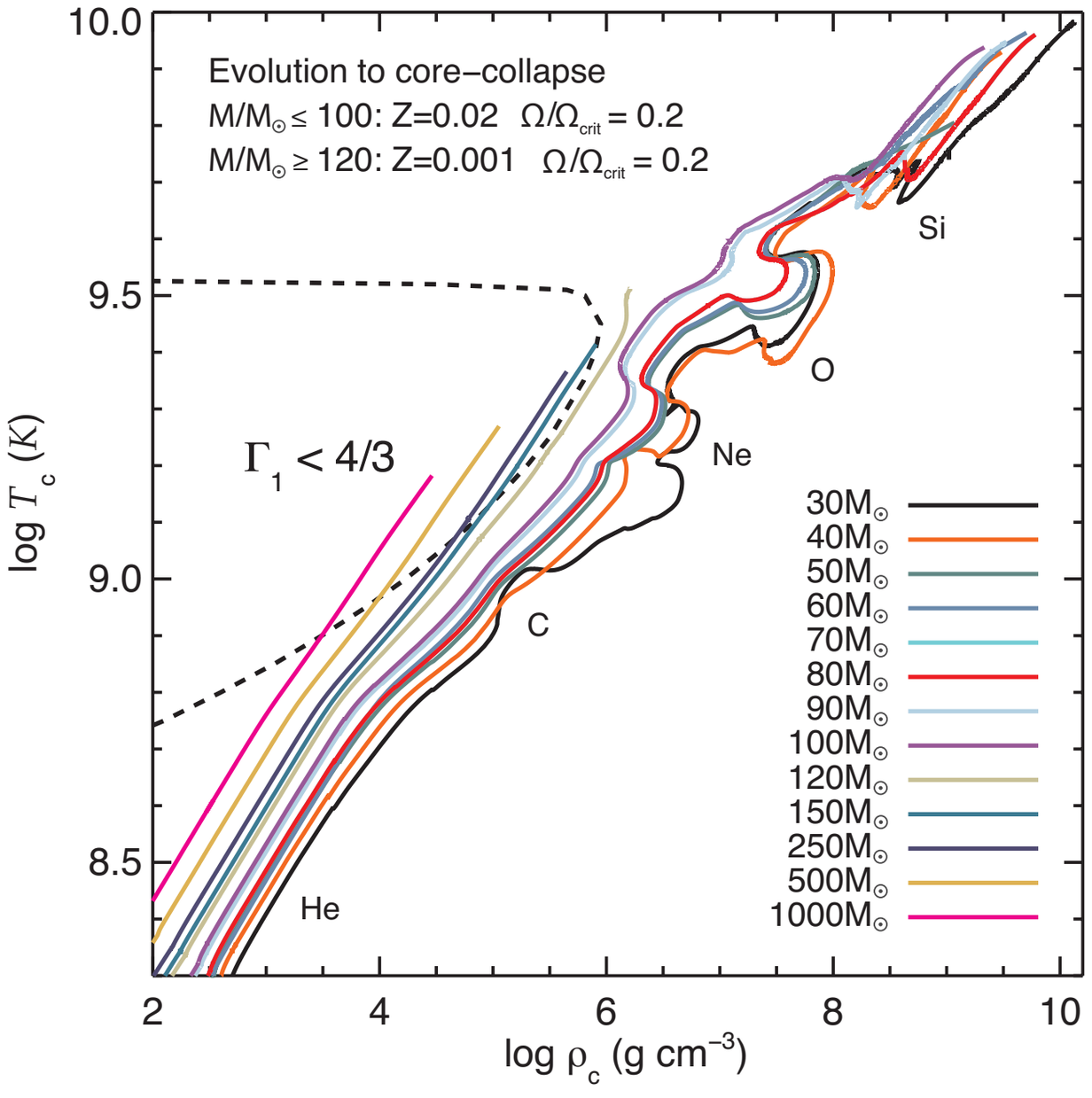


MESA: ~740 Users April 2014

Single stars with MESA

- Uninterrupted evolution to WD and core-collapse

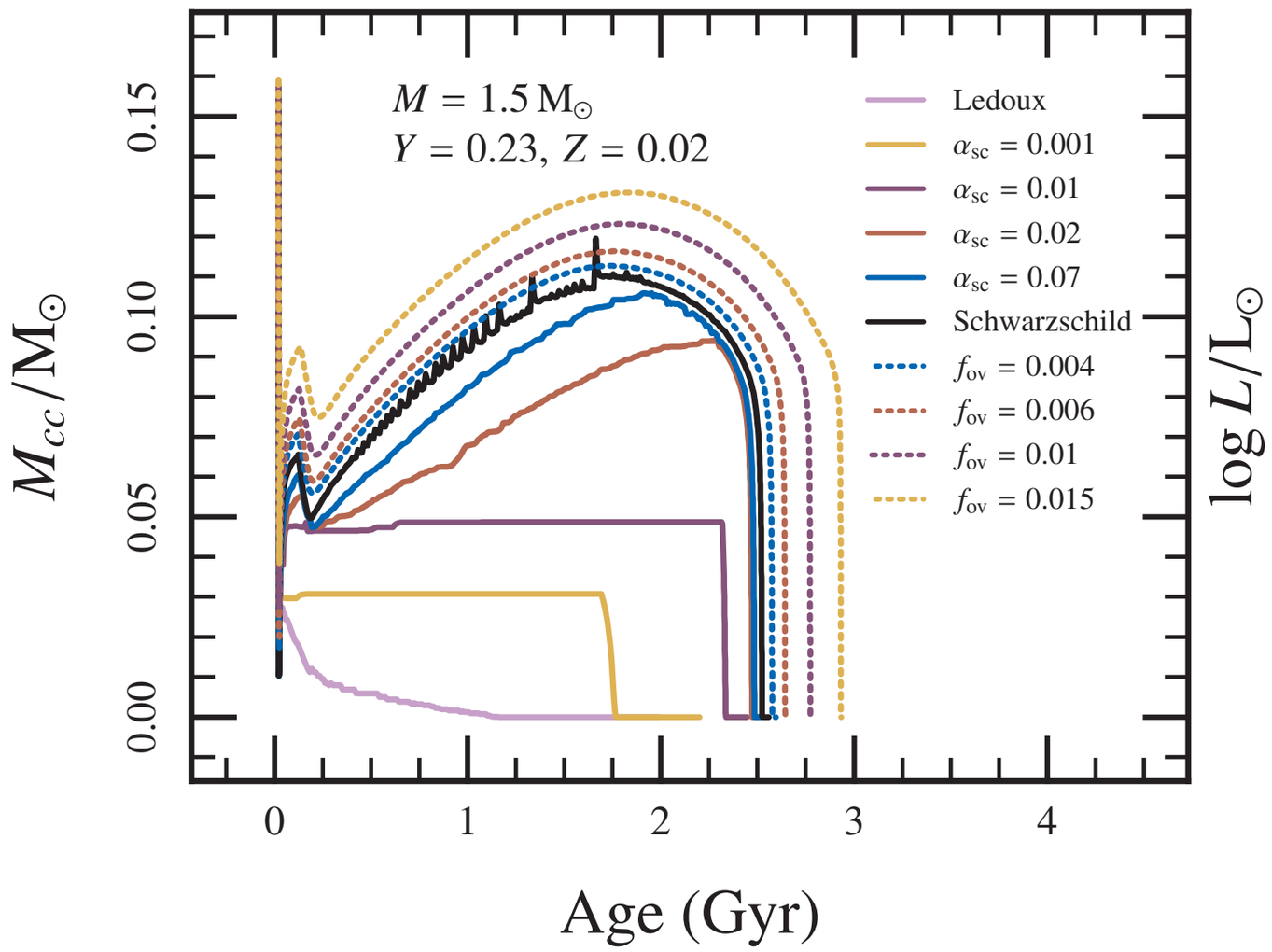


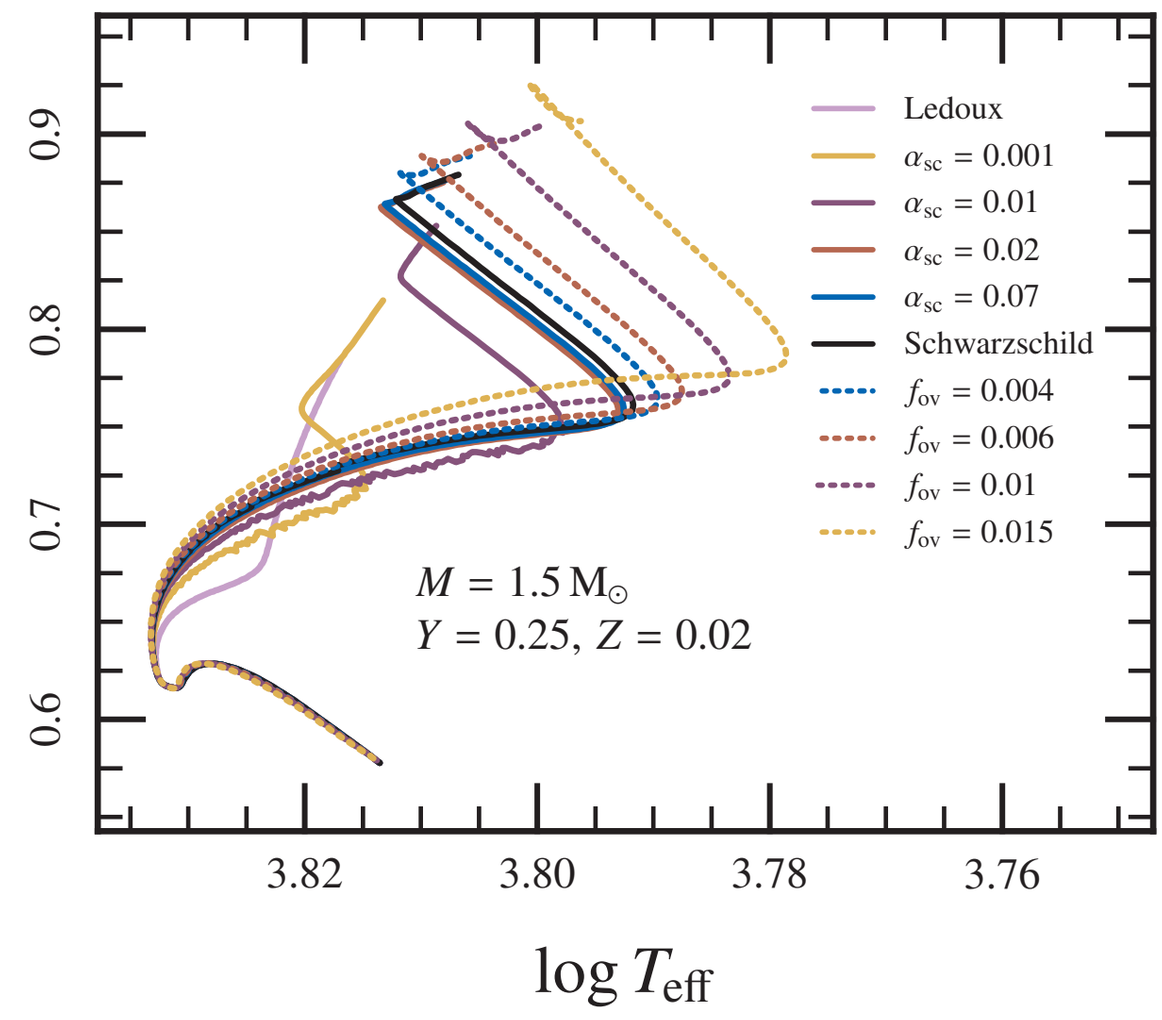


(Paxton+ 2013)

Single stars with MESA

- Mass loss/gain, Schwarzschild/Ledoux, overshooting, double diffusion, gravitational settling, radiative levitation...

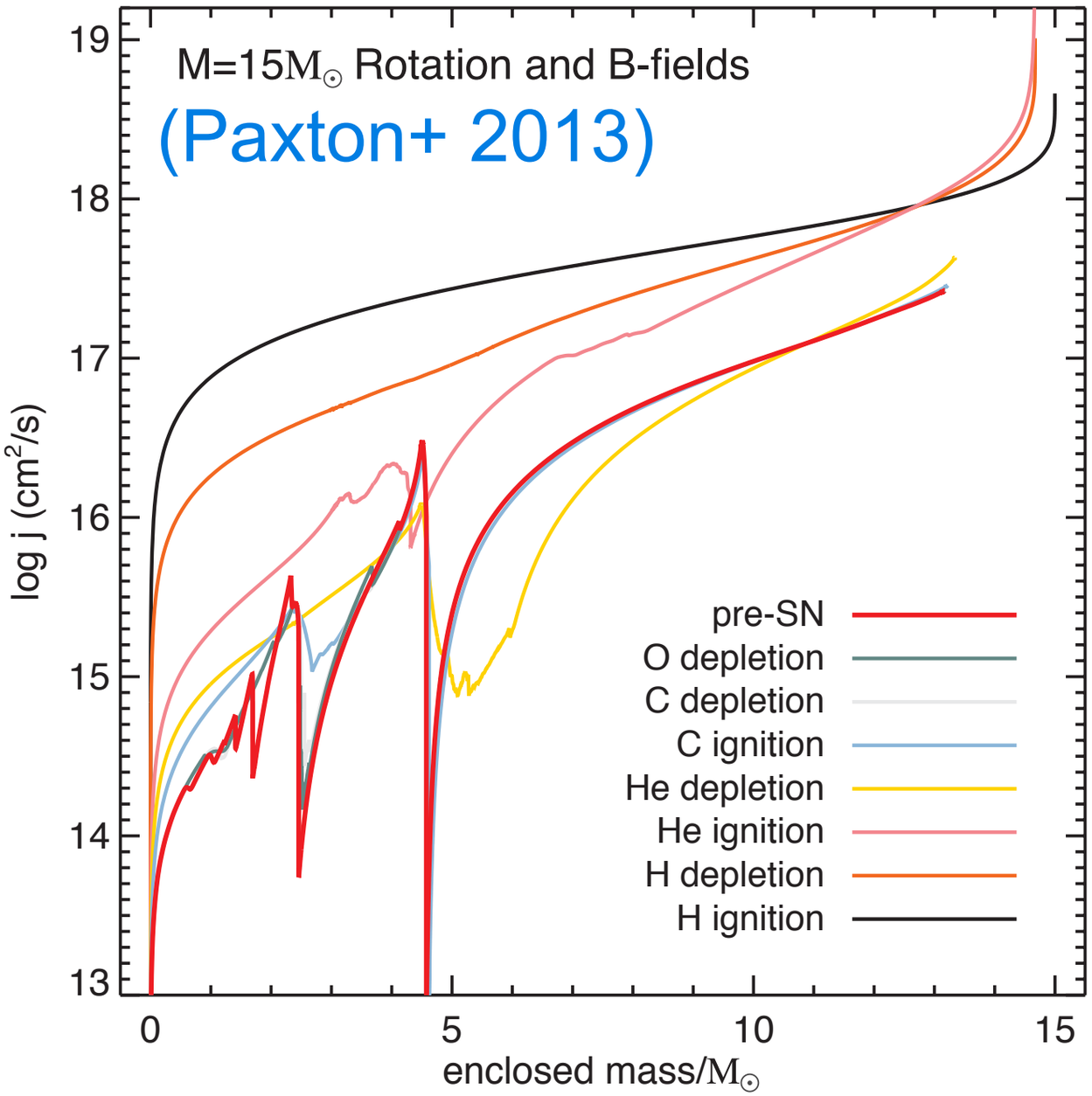


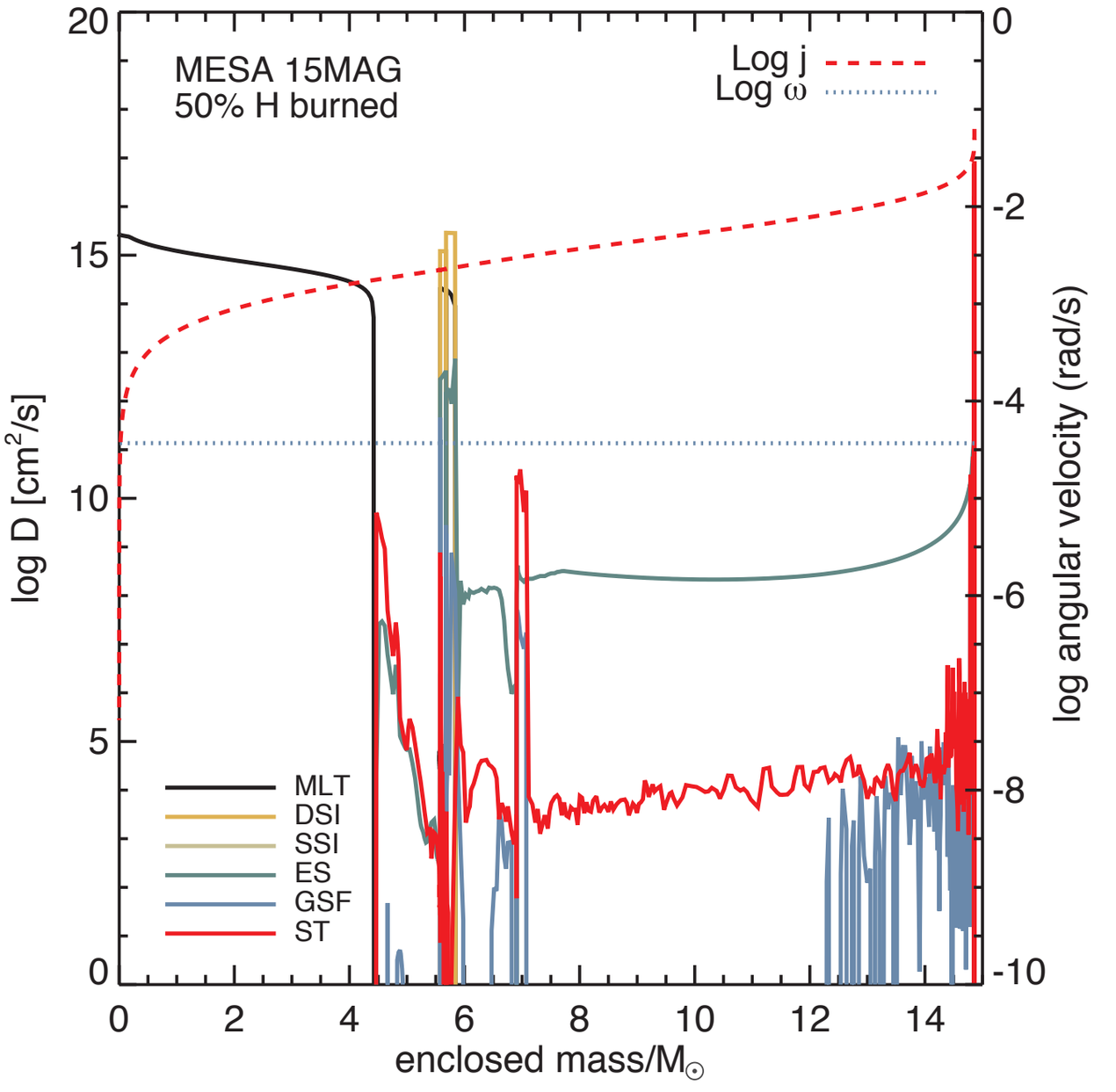


(Paxton+ 2013)

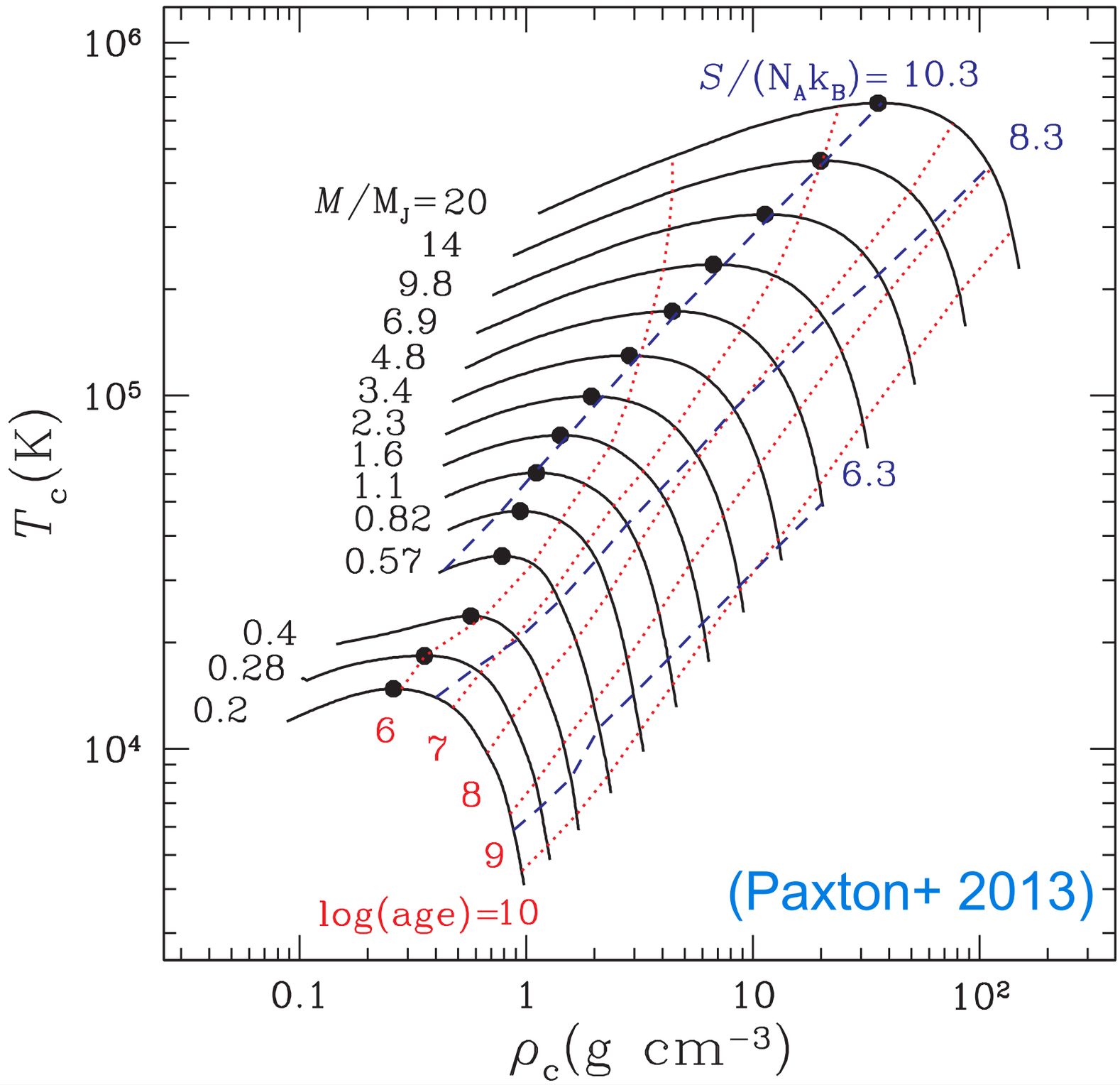
Single stars with MESA

- Includes the physics of **rotation** (in a diffusion approximation) and of dynamo generated **magnetic fields** in radiative zones





Giant Planets with MESA



The mtx module provides an interface to linear algebra routines for matrix manipulation. A large set of BLAS and LAPACK routines are included, but the mtx module can easily be modified to accept these routines from other linear algebra packages, e.g., GotoBLAS[10] or the Intel Math Kernel Library[11] (MKL). Sparse matrix operations are supported, including a subset of the SPARSKIT sparse matrix iterative solver[12] and an interface to the Intel version of the PARDISO sparse matrix direct solver. The routines in num make use of these matrix routines.

Modules interp_1d and interp_2d deal with one-dimensional and two-dimensional interpolation, respectively. One-dimensional interpolation is carried out using either a piecewise monotonic cubic method (Huynh 1993; Suresh & Huynh 1997) or a monotonicity-preserving method (Steffen 1990). Compared with the piecewise monotonic method, the monotonicity-preserving method is stricter and does not allow an interpolated value to range outside of the given values (Steffen 1990). Module interp_2d includes parts of the PSPLINE package[13] and routines by both Akima (1996) and Renka (1999) for bivariate interpolation and surface fitting on a grid or with a scattered set of data points. Both single- and double-precision versions of the two-dimensional interpolation routines are provided.

Module num provides a variety of solvers for stiff and non-stiff systems of ordinary differential equations (ODEs) and a Newton–Raphson solver for multidimensional, nonlinear root finding. The family of ODE solvers is derived from the routines of Hairer & Wanner (1996). The non-stiff ODE class are explicit Runge–Kutta integrators of orders 5 and 8 with dense output, automatic stepsize control, and optional monitoring for stiffness. The stiff ODE solvers are linearly implicit Runge–Kutta, with second-, third-, and fourth-order versions and two implicit extrapolation integrators of variable order: either midpoint or Euler. All integrators support dense, banded, or sparse matrix routines, analytic or numerical difference Jacobians, explicit or implicit ODE systems, dense output, and automatic stepsize control.

Stellar evolution requires a mix of numerical methods to solve the simultaneous differential equations of stellar structure and nucleosynthesis.

The Newton–Raphson solver for multidimensional, nonlinear root finding supports square, banded, and sparse matrices and analytic or automatic numerical differencing for the Jacobians. It has the ability to reuse Jacobians and employs a line search method to give improved convergence. The multidimensional Newton–Raphson solver is used by MESA star to solve highly nonlinear systems of differential-algebraic equations with tens of thousands of variables (see Section 6). The structure of the Newton–Raphson solver is derived from Lesaffre's version of the Eggleton stellar evolution code (Eggleton 1971; Pols et al. 1995; Lesaffre et al. 2006) and some details of the implementation will be described in Section 6.

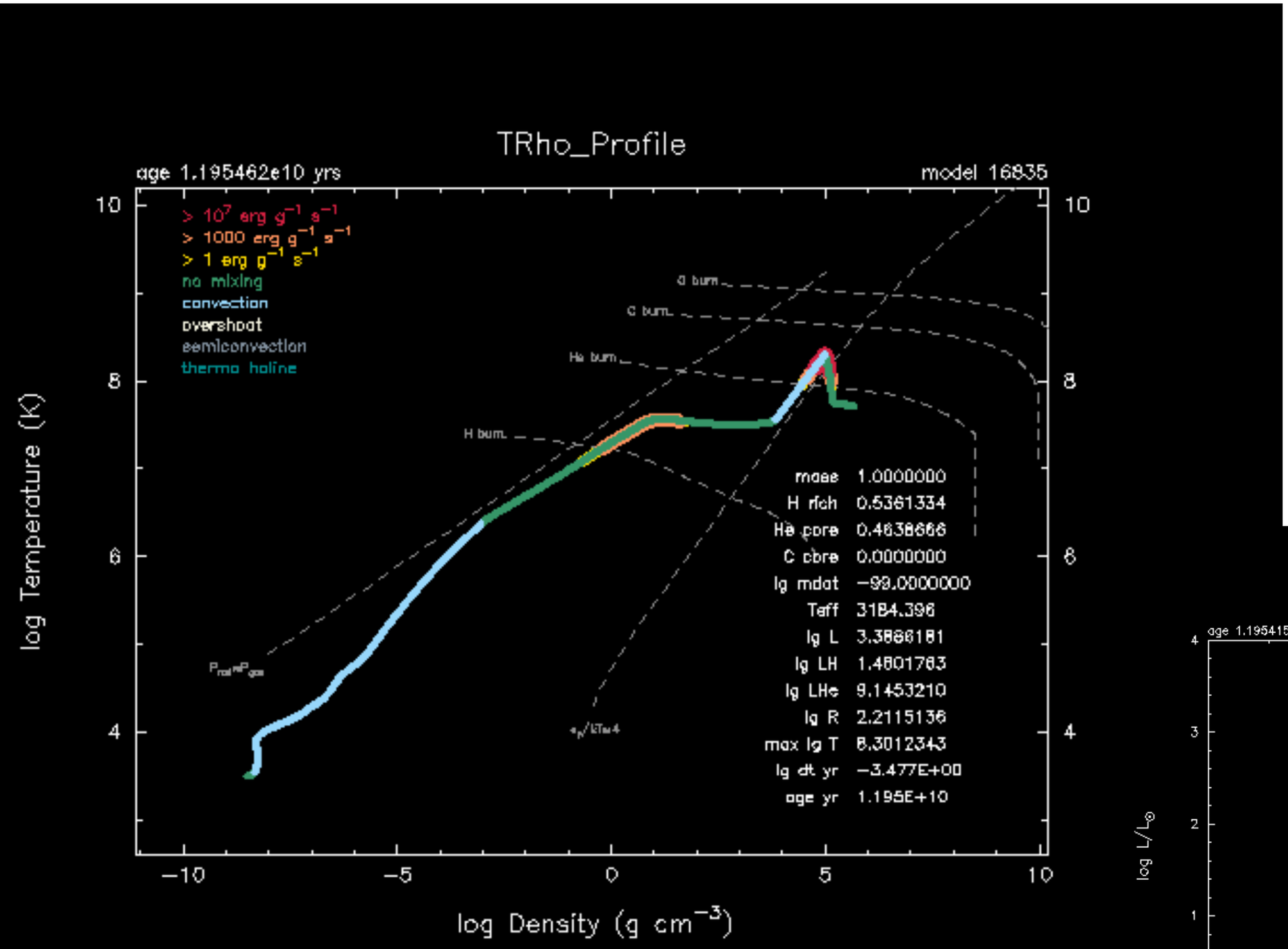
The alert module provides a framework for reporting messages, including errors, to the terminal. The utils module provides a number of functions for checking if a variable has been assigned a bad value (e.g., NaN or infinity) and tracking Fortran I/O unit numbers in use. It also provides subroutines for basic file I/O and for allocating arrays of different types and dimensions, including a Fortran implementation of a hash tree that is used by the stellar evolution module to update the model mesh. Programs and scripts that are used for testing that each module has compiled correctly are stored in utils.

The above is from the defining paper by Paxton et al., Astrophysical J. Supplement, 192 (2011).

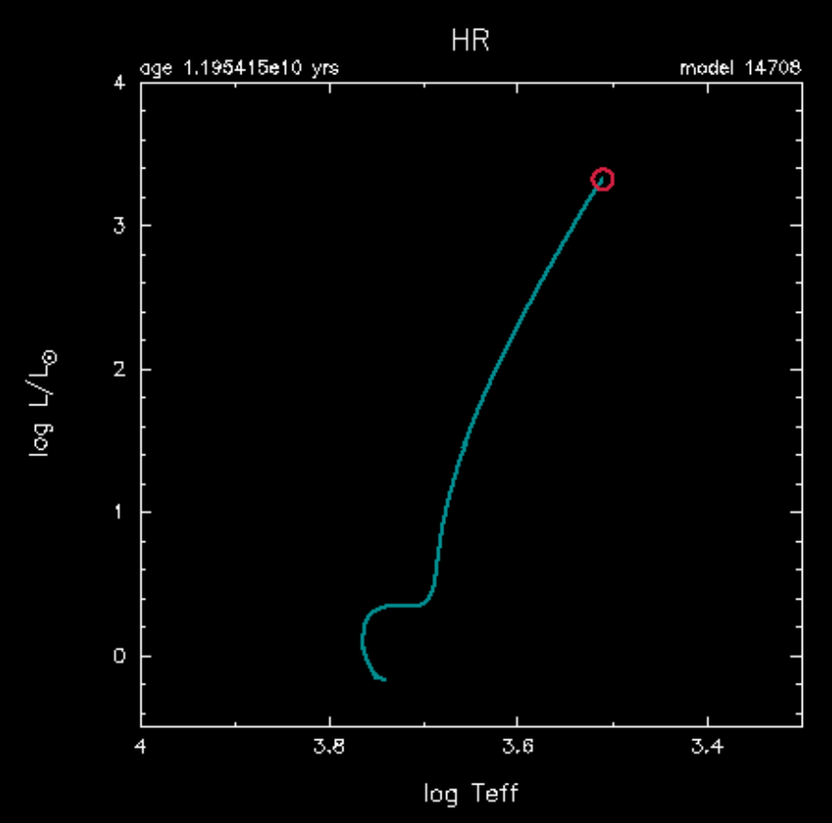
When you run MESA, it shows a number of interesting plots: the changing temperature and luminosity of the evolving star (the H-R diagram); the temperature and density of each of the thousand or so shells from the surface to the center; where nuclear "burning" is taking place, etc. The plots are displayed with "pgplot".

When the program stops, the plots disappear, but big ASCII files have been dumped with the details of every 50th model.

Two of the plots generated as the star evolves.



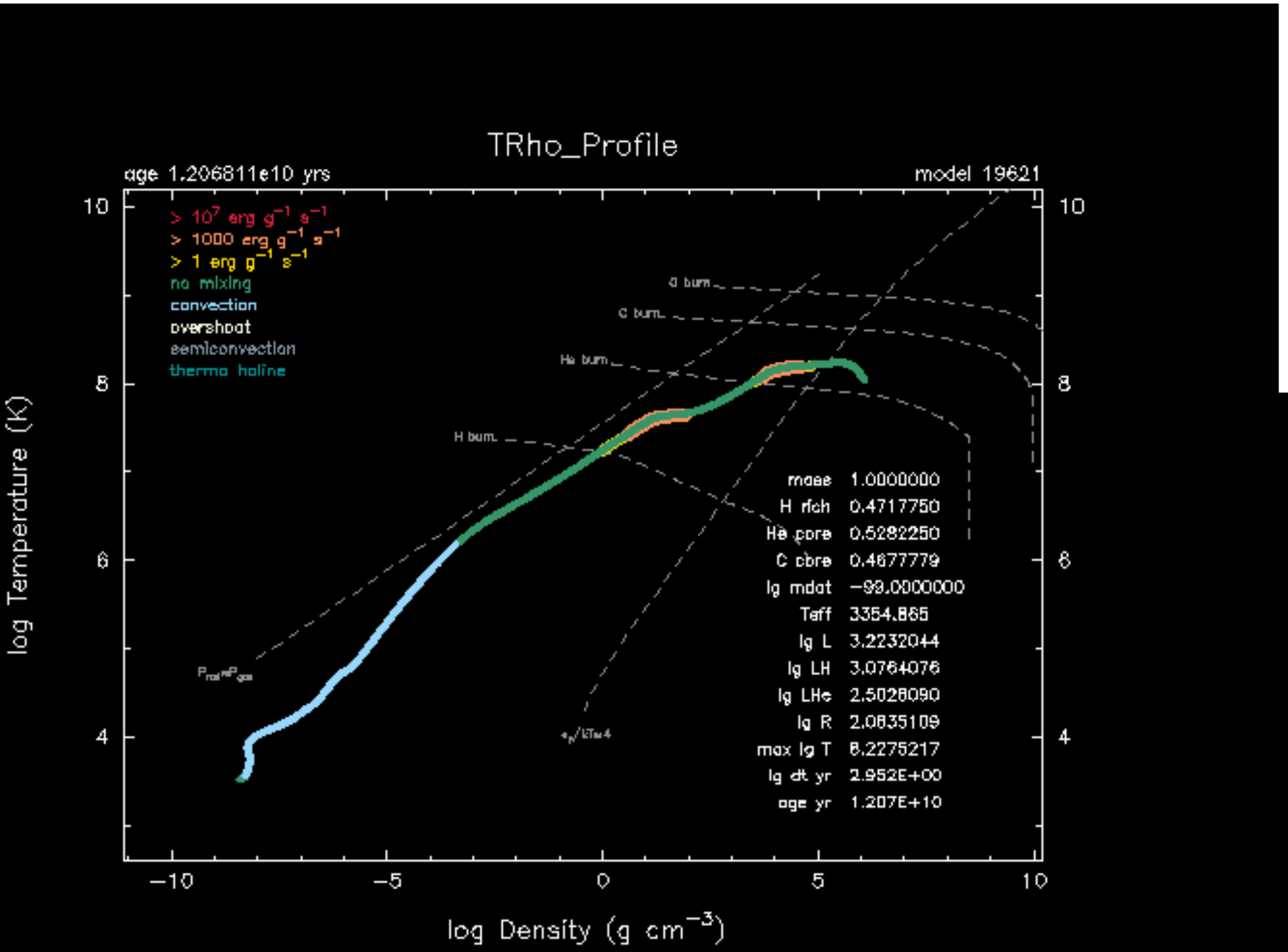
Track in the H-R diagram
↓

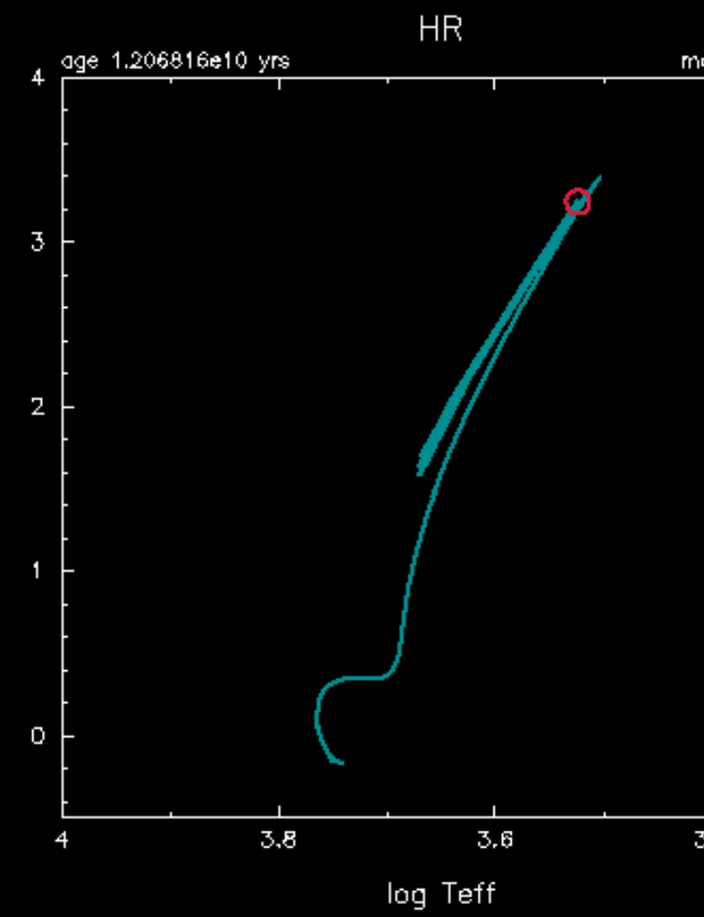


The He-flash in a solar-mass star

(The time step between models is only 3 days.)

Later in this star's evolution an inert carbon core appears ...





Double shell burning (H & He) on the asymptotic giant branch (AGB).

To really look into what the star has been doing, you want to look into these LOG files & plot all sorts of quantities. Folks have written tools to do this using Python, IDL, Mathematica, Matlab, etc.

Naturally, I use J for this sort of analysis; the code to read in all 90 variables is trivial and is a good example of how J makes life simple.

A bit of J code to read the MESA output for analysis.

```
NB. For reading the MESA LOGS/profilexxx.data files:
NB. assigns values to all the tabulated column names.
NB. Usage: jread 30    (reads "LOGS/profile30.data")
NB. ------------------------------------------
PPP=: '/Users/jph/mesa/mysun/LOGS/'
require 'plot'
jread=: monad define
A=. LF cut fread PPP,'profile',(":y),'.data'
g_col=: cut >1{A
(g_col)=: _&". >cut >2{A ⇐ Here's where some global variables are assigned
print 'Mass';star_mass;' Model No.';model_number;' Age';star_age;
      ' Number of zones in model';num_zones
m=. star_mass
print 'T_eff';Teff;' Luminosity';photosphere_L;' Radius';photosphere_r;
      ' Z';initial_z
print 'H fraction';(star_mass_h1%m);' He fraction';(star_mass_he4%m);
      ' C fraction';(star_mass_c12%m)
NB.------------------------------------------
head=: cut >4{A          NB. column heads
erase'y'   NB. because y is one of the column names!
(head)=: |: _&". >5}.A  ⇐ Here's where the variables are assigned
T6=: 10^(_6+logT)
print 'He core';(he_core_mass%m);' C core';(c_core_mass%m);
      '  Central density';10^(:logRho
' Maximum & central temperature';(({. \:~T6),({:T6));'million K'
)
```

Some of the 90 variables saved for each shell:

```
   names 0
abar                        burn_ar                 burn_c
burn_ca                     burn_cr                 burn_fe
burn_mg                     burn_min1               burn_min2
burn_n                      burn_na                 burn_ne
burn_o                      burn_s                  burn_si
burn_ti                     c12                     c12_c12
c12_o16                     c_core_mass             center_c12
center_eta                  center_h1               center_he3
center_he4                  center_n14              center_ne20
center_o16                  cno                     conv_vel_div_csound
csound                      d_lnepsnuc_dlnT         d_lnepsnuc_dlnd
days                        dq_ratio                dynamic_time
entropy                     eps_grav                eps_nuc
eta                         fe_core_mass            free_e
g_col                       global                  gradT
gradT_sub_grada             grada                   gradr
h1                          he3                     he4
he_core_mass                head                    initial_mass
initial_z                   kh_timescale            logP
logPgas                     logR                    logRho
logT                        log_D_mix               log_abs_eps_grav_dm_div_L
log_abs_eps_nuc             log_conv_vel            log_mlt_D_mix
log_mlt_Gamma               log_opacity             log_q
logdq                       logtau                  logxm
logxq                       luminosity              mass
mg24                        mixing_type             mlt_mixing_length
mlt_mixing_type             model_number            mu
n14                         ne20                    net_energy
neutron_rich_core_mass      non_nuc_neu             nuc_timescale
num_zones                   o16                     o16_o16
o_core_mass                 other                   pgas_div_ptotal
photo                       photosphere_L           photosphere_r
pnhe4                       power_h_burn            power_he_burn
```

```
jread 88
```

Mass	14.4797	Model No.	5550	Age	1.20416e7	Number of zones in model	1410

T_eff	3526.05	Luminosity	67449.7	Radius	696.904	Z	0.02

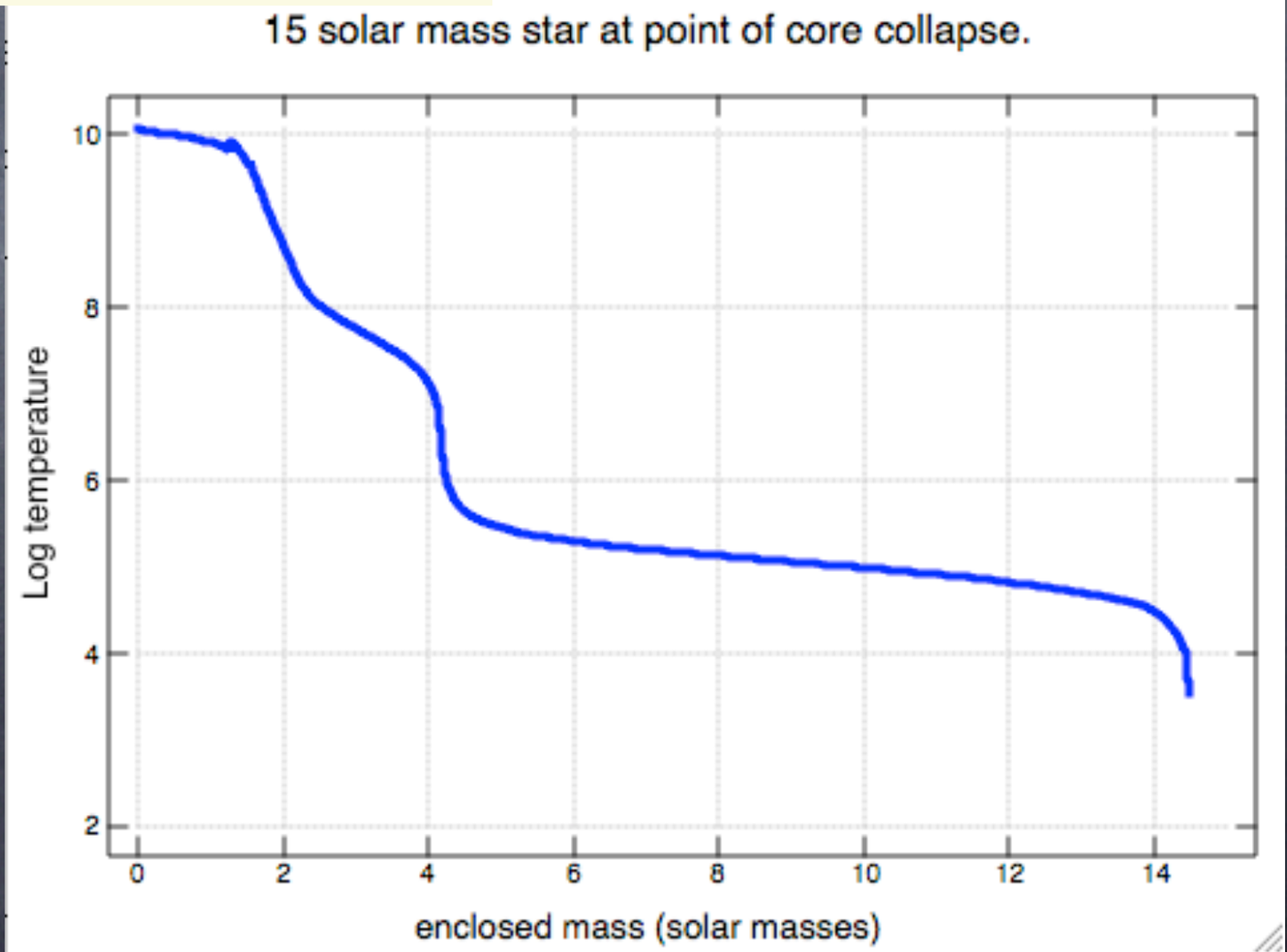
H fraction	0.475537	He fraction	0.356994	C fraction	0.00708972

He core	0.283537	C core	0.14429	Central density	1.50783e10

Maximum & central temperature	11558.2 11558.2	million K

```
ps=. 'pensize 3; title 15 solar mass star at point of core collapse.'
ps=. ps,';xcaption enclosed mass (solar masses);ycaption Log temperature'
ps plot mass; logT
```

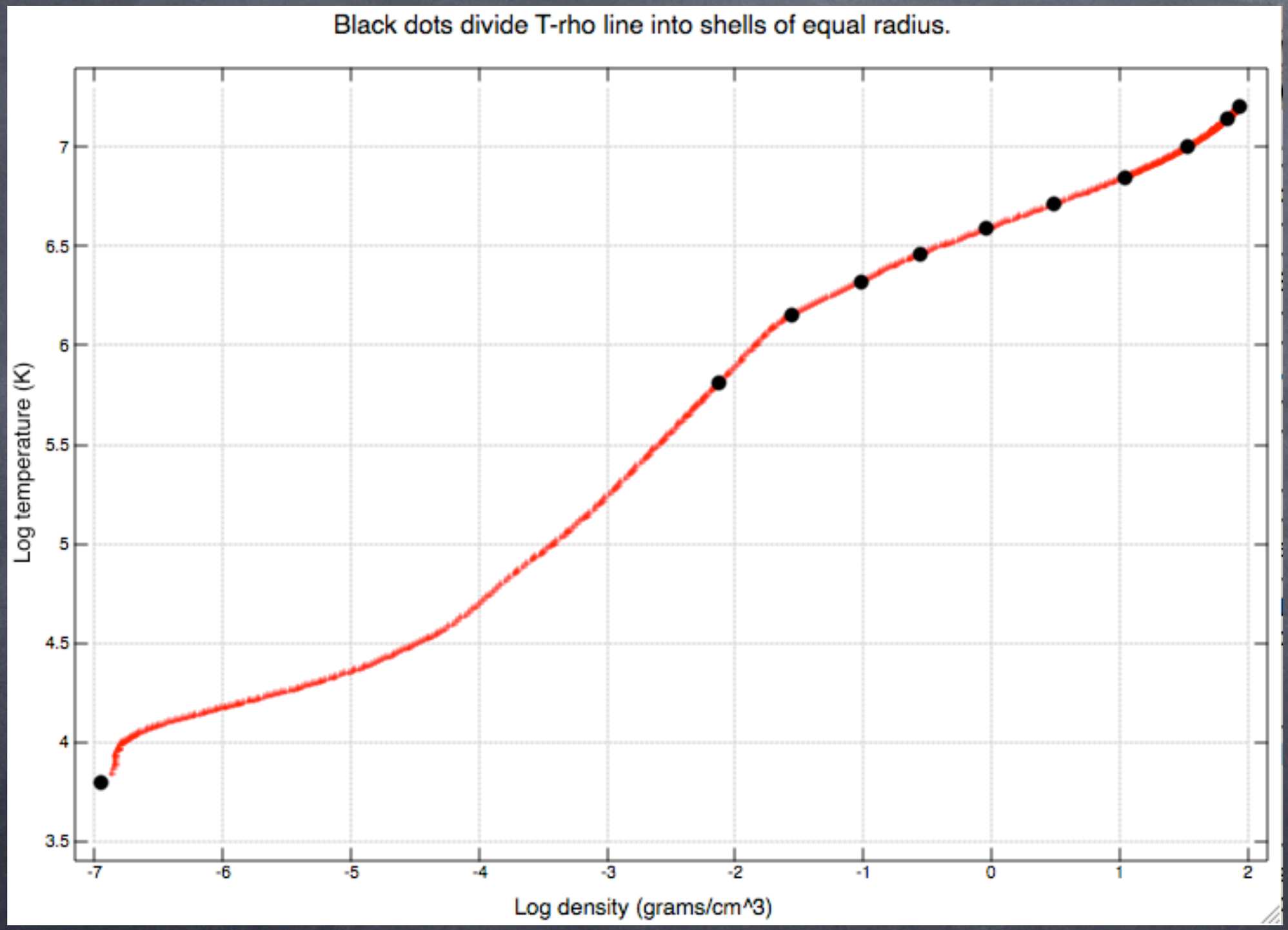
Reading the snapshot of a model and plotting the run of temperature.



An example: In the plot of temperature vs. density, we may want to see in what zones (either in mass or in radius) these changes are occurring. This verb "Trho" places dots on the T-ρ curve marking points that divide the star into shells of equal mass or of equal radius.

```
NB. Plot log T vs log rho for all zones and put dots
NB. at points of equal mass or radius. E.g., to put
NB. dots dividing line into 10 regions of equal mass
NB. or radius: mass Trho 10  -or- radius Trho 10
Trho=: dyad define
f=. ({.x)*int01 y                    NB. division frets
kk=. ,I.({."1 /:~"1 a)=(a=.|f-/x)
pd 'reset'
pd 'title Black dots divide T-rho line into shells of equal radius.'
pd 'xcaption Log density (grams/cm^3)'
pd 'ycaption Log temperature (K)'
pd 'pensize 3;color red'
pd logRho;logT
pd 'type dot;pensize 4'
pd 'color black'
pd (kk{logRho);(kk{logT)
pd 'show'
)
int01=: i.@>: % ]
```

The plot produced by "radius Trho 10" :



Here we show the T-rho plot of a young solar-type star. Note the large drop in temperature and density in the outer 10% of the star's radius.

J is a wonderful language for quick analysis and experimentation with observational data.

What I have tried to illustrate here is that J is also ideal for working with the data produced by large, sophisticated codes. These codes, dealing with areas such as stellar evolution or stellar atmospheres, will continue to be written in Fortran or C, because they are limited by the speed of the computers available. But they produce large volumes of numerical output, and the post-processing of the output has become a vital and complex part of research. And this post-processing is another place where J can play an important role.