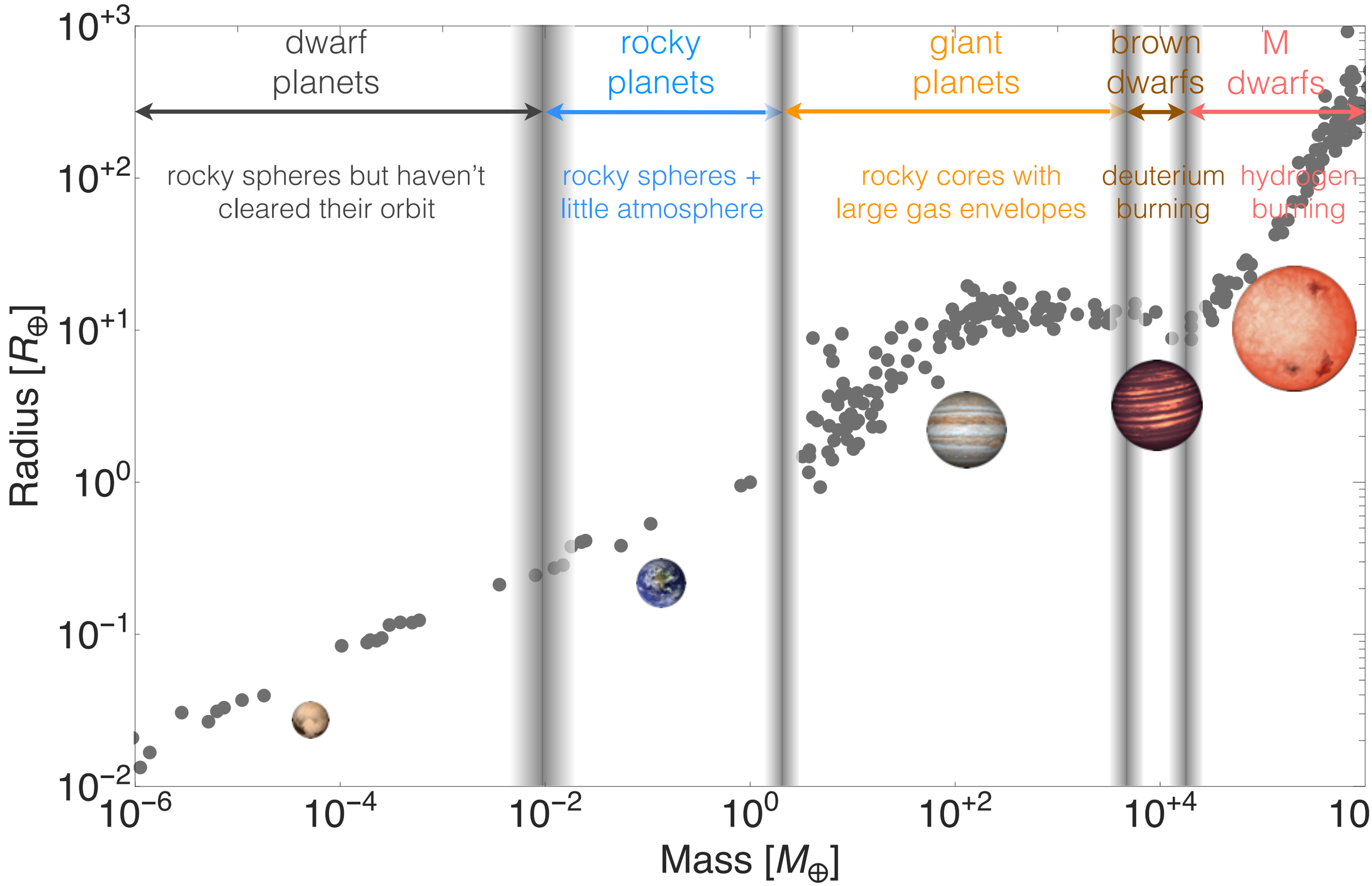
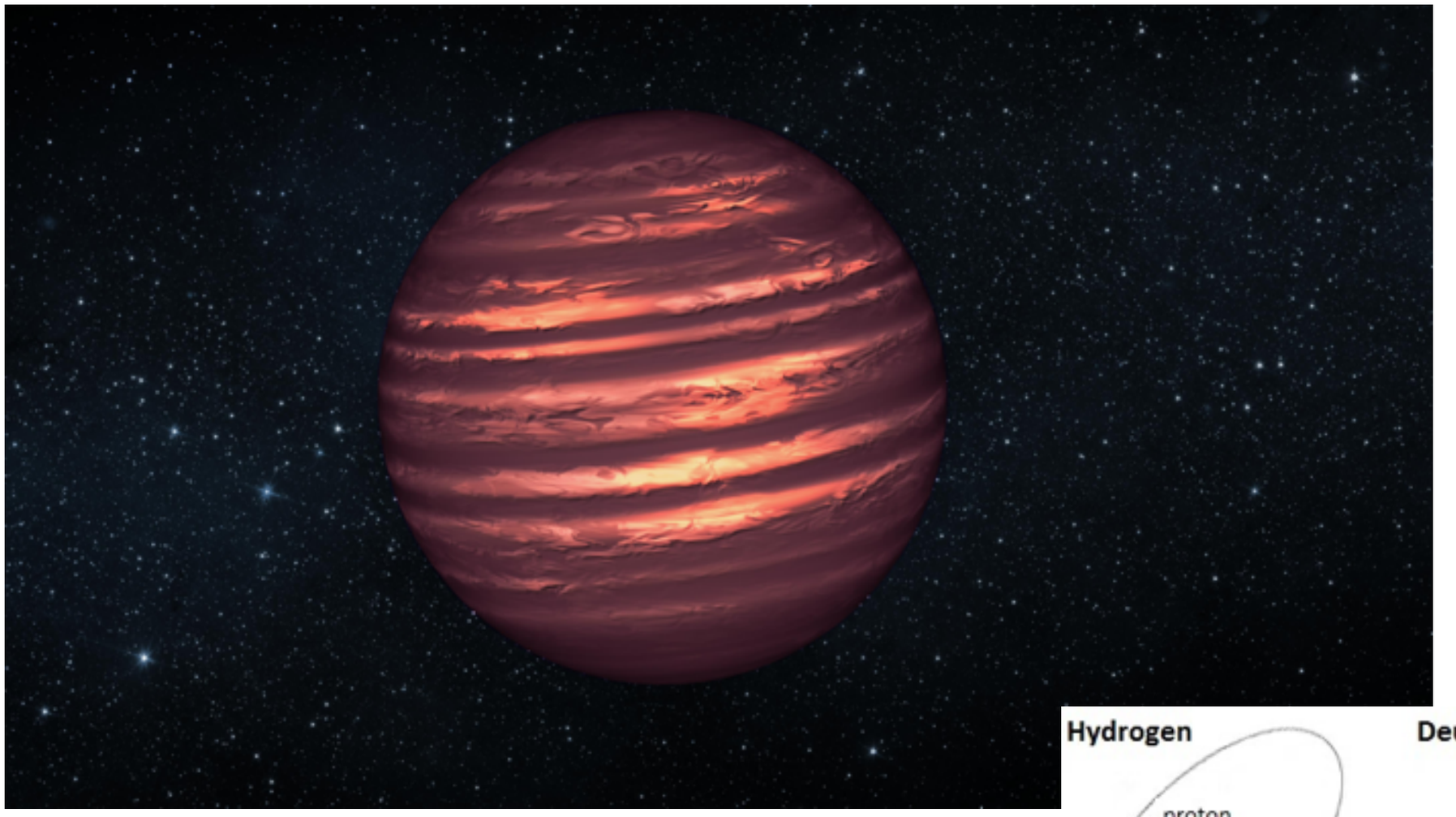
LAB 2
Taking the Temperature of a Cold Brown Dwarf

C3986: Astrostatistics

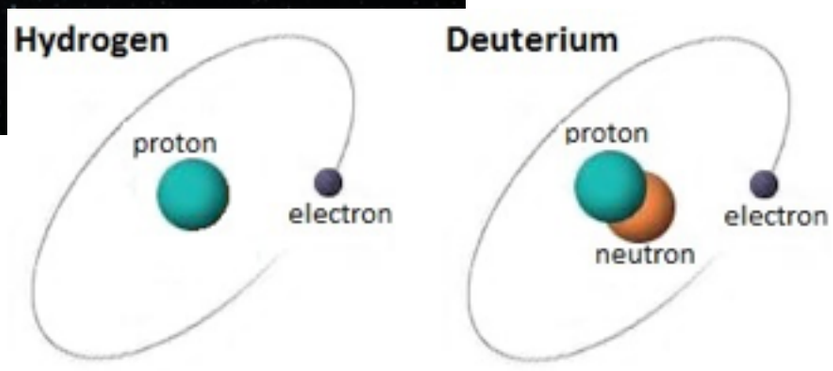
Quick background... Brown Dwarfs



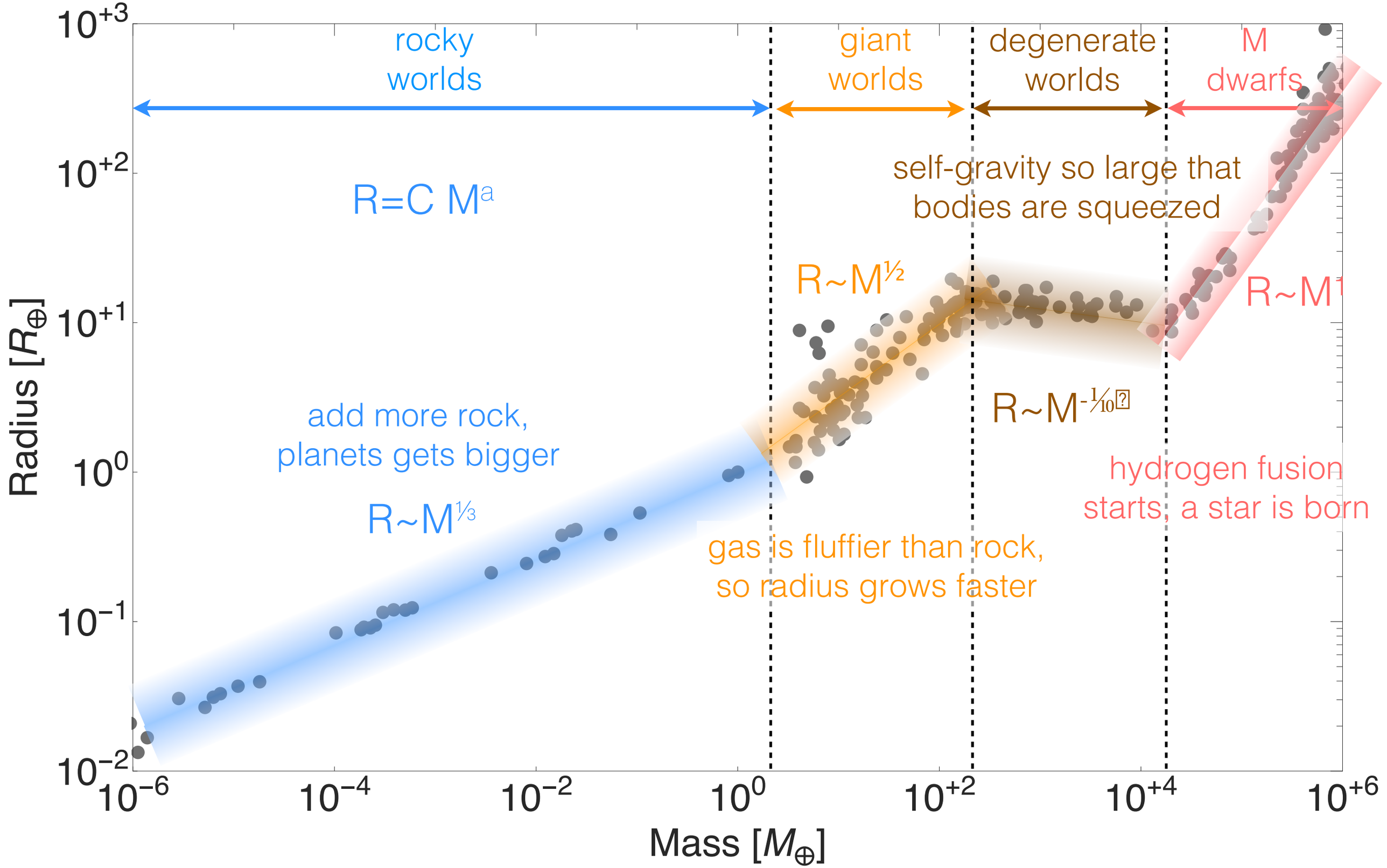
brown dwarfs



self-gravity so large that bodies are squeezed



- deuterium (isotope of hydrogen) is the easiest thing to fuse
- once $M \gtrsim 13 M_{Jup}$, pressures high enough for deuterium fusion
- but deuterium is rare (10^{-5}) and conditions inefficient and so not body won't start shining until it starts fusing hydrogen-1


rocky worlds
giant worlds
degenerate worlds
M dwarfs
R=C M^a
self-gravity so large that bodies are squeezed
R~M^½
R~M^1
add more rock, planets gets bigger
R~M^⅓
R~M^-1/10?
hydrogen fusion starts, a star is born
gas is fluffier than rock, so radius grows faster
Radius [R⊕]
Mass [M⊕]
10^+3
10^+2
10^+1
10^0
10^-1
10^-2
10^-6
10^-4
10^-2
10^0
10^+2
10^+4
10^+6

NASA Infrared 0.4m telescope, December 2009

In April 2014, Kevin Luhman announces WISE 0855-0714

Temperature 225 - 260 K
(below room temp!)

THE SUN'S CLOSEST NEIGHBORS

WISE 0855-0714
(distance 2014)

6 LIGHT YEARS

WISE 1049-5319
(distance 2013)

Barnard's Star
(distance 1916)

4 LIGHT YEARS

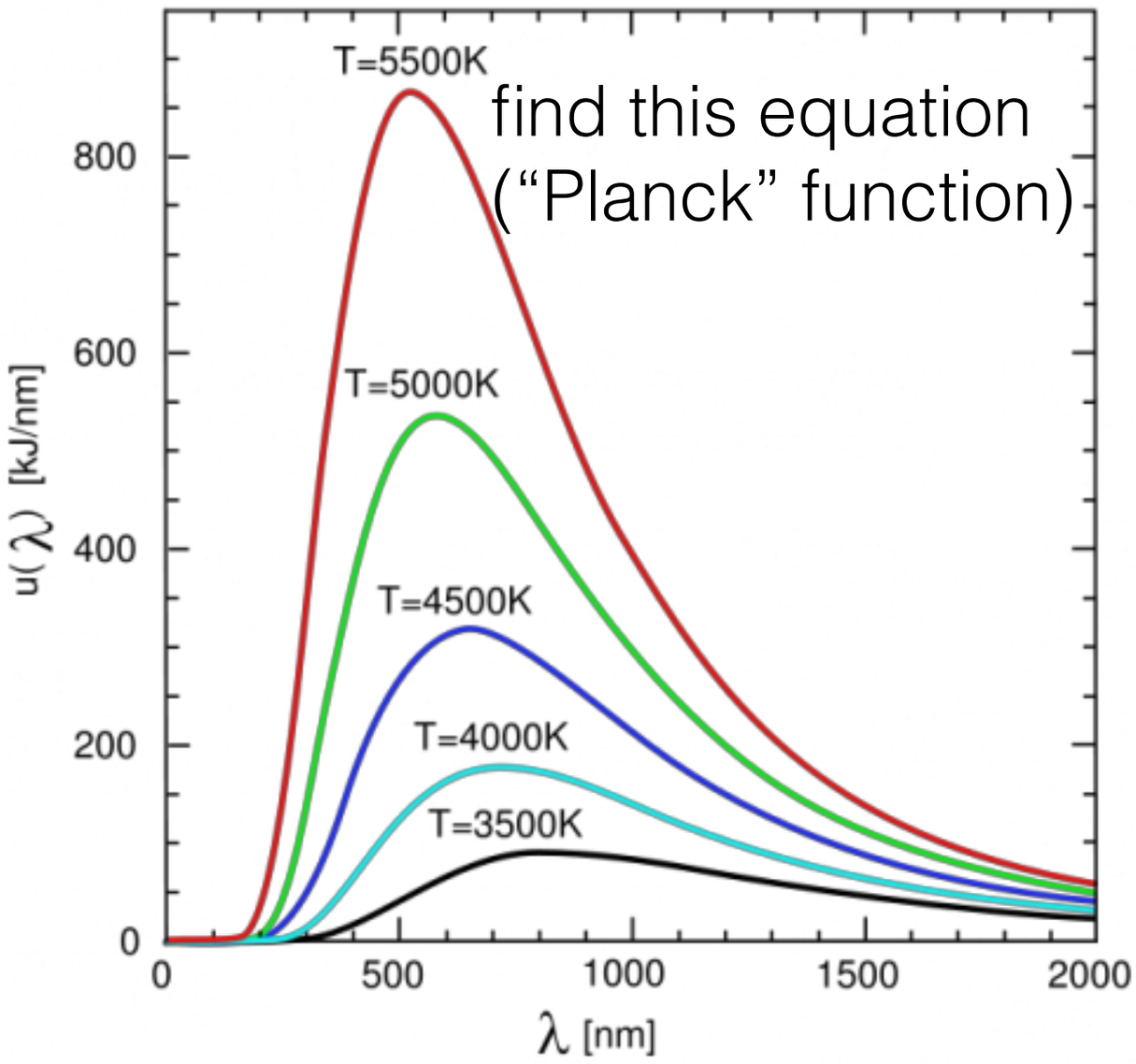
Alpha Centauri
(distance 1839)

Proxima
Centauri
(distance 1917)

2 LIGHT YEARS

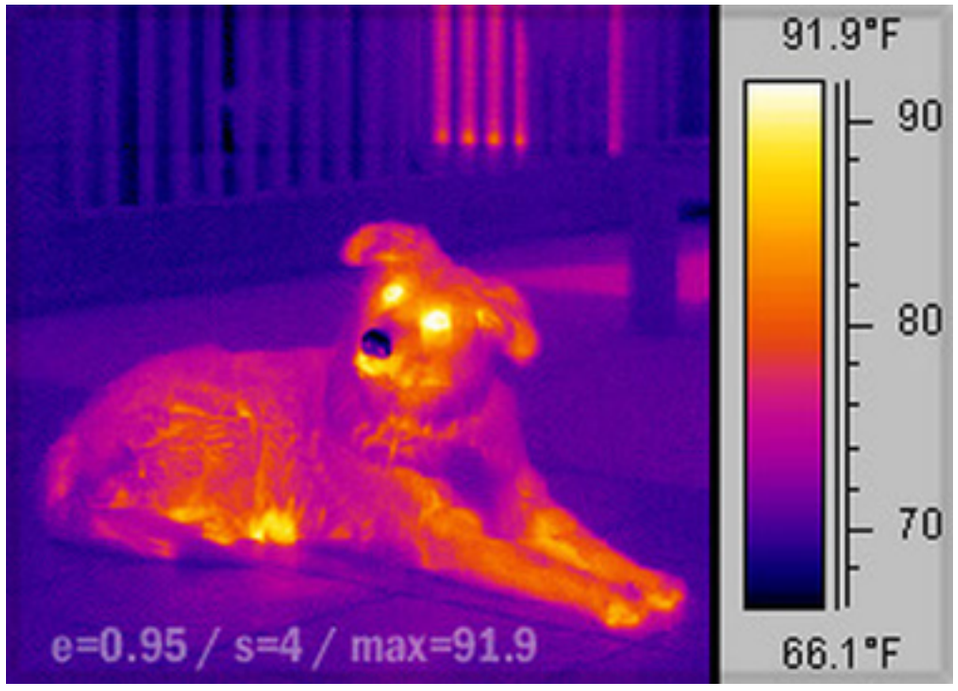
OORT CLOUD

Sun



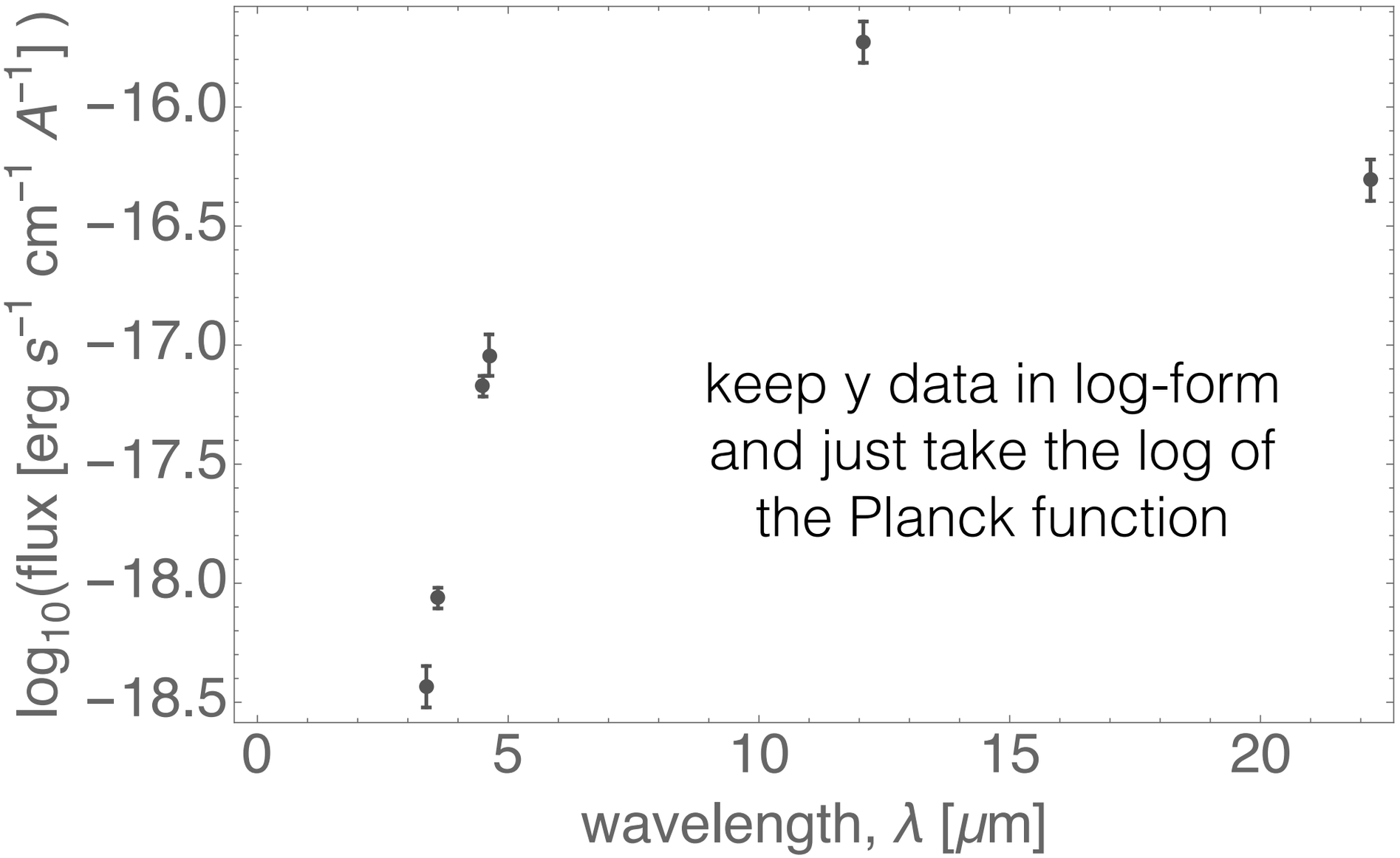
find this equation
("Planck" function)

all objects with T>0K emit radiation



ignoring clouds, molecular absorption/emission features, etc, expect blackbody radiation curve

here's your data of WISE 0855-0714

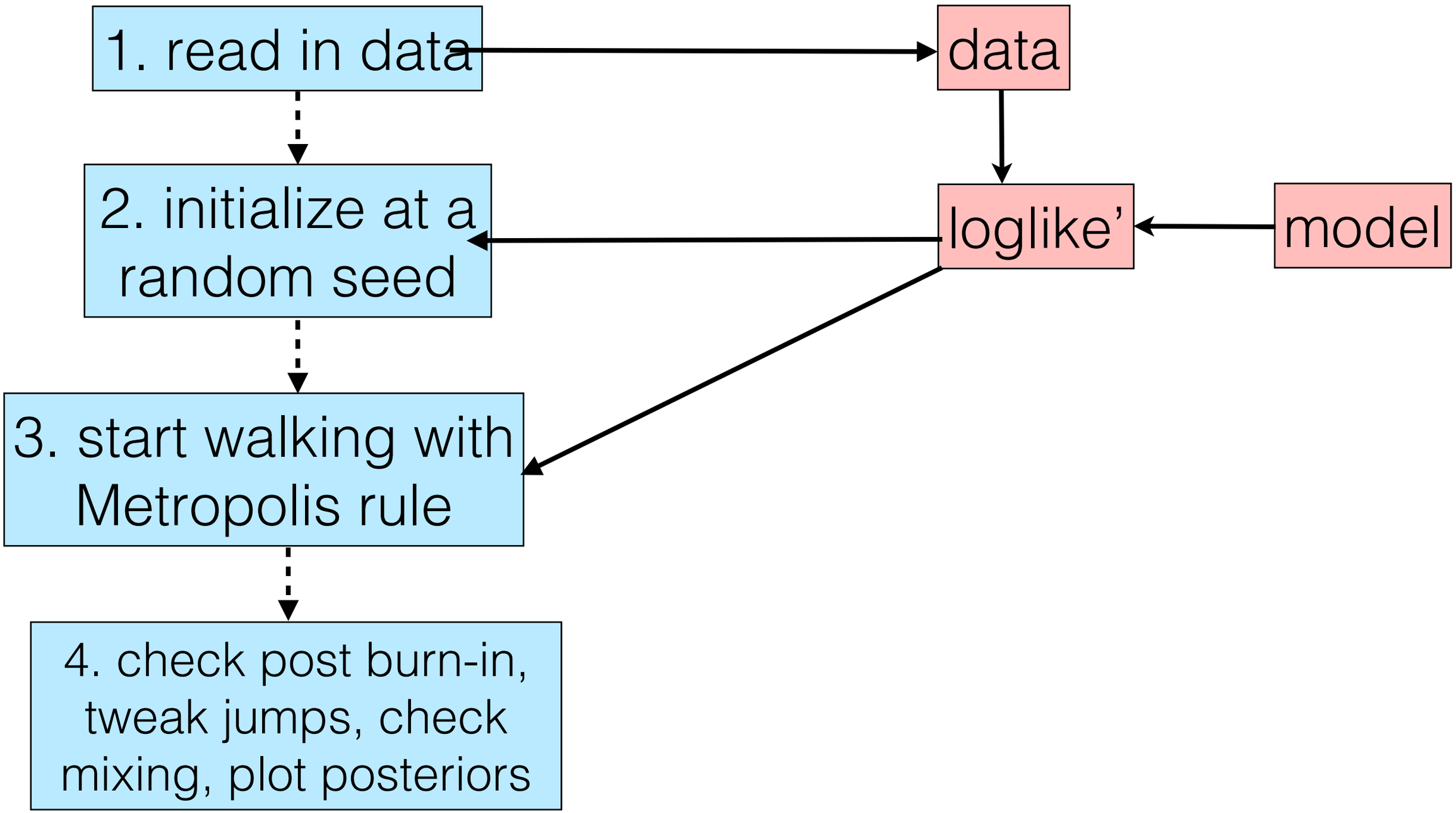


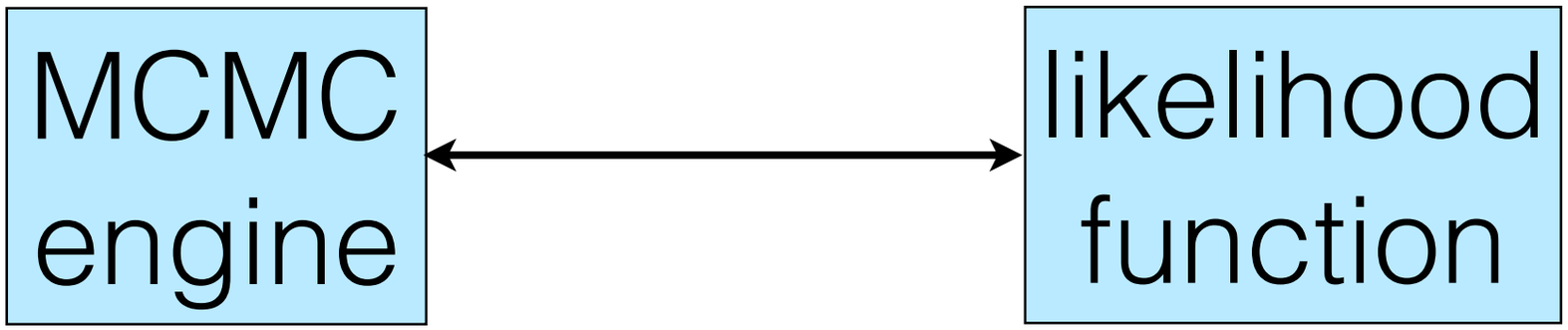
use an MCMC to get posteriors for Temp & multiplicative factor in front of Planck function

define likelihood

define priors

define model




MCMC
engine
likelihood
function

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
from astropy.io import ascii
import scipy.constants as con

# Get the data using the astropy ascii
data = ascii.read("../SED.dat",data_start=4)
lam=data[0][:]     # Wavelength column
logf=data[1][:]    # log10(flux)
errlogf=data[2][:] # Error on log10(flux)

# Shape parameters for priors
```

1. read in data

define priors

In [3]:
```
# Set a definition for the model
def model(microns,Teff,logfactor):
```

model

```
th[i]) ) - 1.0 )
  return logflux
```

In [4]:
```
# Set a definition for the loglikelihood, assuming normally distributed data
def log_like(lam,logf,errlogf,theta):
```

loglike’

```
    return loglike
```

In [5]:
```
# Set a definition for the logpriors
def log_prior(theta,thetashape):

    [redacted]

    # Prior for theta[0]: Teff~logU[Teffmin,Teffmax]
    [redacted]

    # Prior for theta[1]: logfac~U[logfacmin,logfacmax]
    [redacted]

    #logprior = np.sum(logpriors)
    return np.sum(logpriors)
```

loglike'

model

In [6]:
```
# Initialize the MCMC from a random point drawn from the prior
[redacted]

# Calculate the associated modified loglike
[redacted]
```

2. initialize at a random seed

In [5]:
```
# Set a definition for the logpriors
def log_prior(theta,thetashape):

    [redacted]

    # Prior for theta[0]: Teff~logU[Teffmin,Teffmax]
    [redacted]

    # Prior for theta[1]: logfac~U[logfacmin,logfacmax]
    [redacted]

    #logprior = np.sum(logpriors)
    return np.sum(logpriors)
```

loglike’

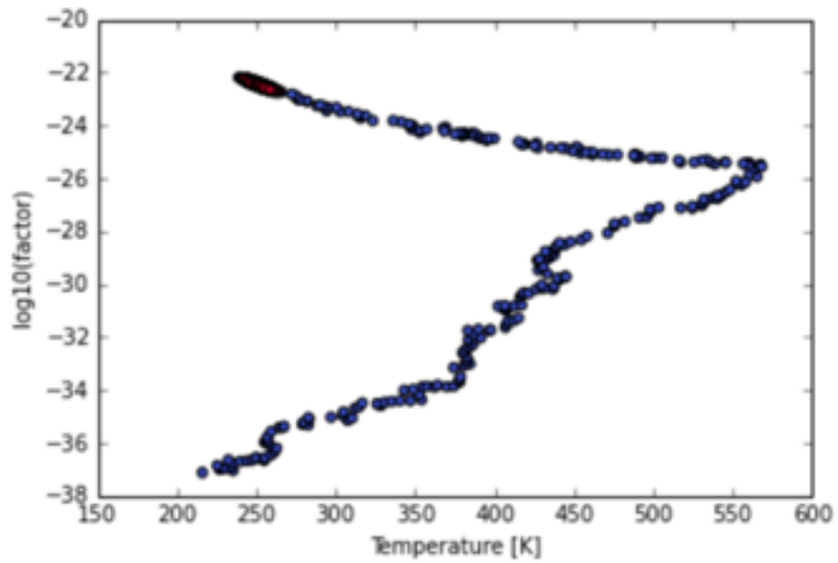
In [6]:
```
# Initialize the MCMC from a random point drawn from the prior
[redacted]

# Calculate the associated modified loglike
[redacted]
```

2. initialize at a random seed

In [7]:
```
# Define the proposal jump size
[redacted]

# Starting walking
j=0
jmax=10000
while True:

    # Generate a proposal (or jump)
    [redacted]

    # Compute Metropolis Rule
    [redacted]

    if probjump>np.random.uniform(0,1):
        # Accept the jump
        j = j + 1
        loglikechain = np.append(loglikechain,logliketrial)
        thetachain = np.vstack((thetachain,thetatrial))

    if j==jmax:
        break
```

3. start walking with Metropolis rule

In [52]:
```
%matplotlib inline

jlist=np.arange(len(thetachain))
plt.scatter(thetachain[:,0], thetachain[:,1], c=jlist, cmap='coolwarm')
plt.xlabel('Temperature [K]')
plt.ylabel('log10(factor)')
plt.show()
```

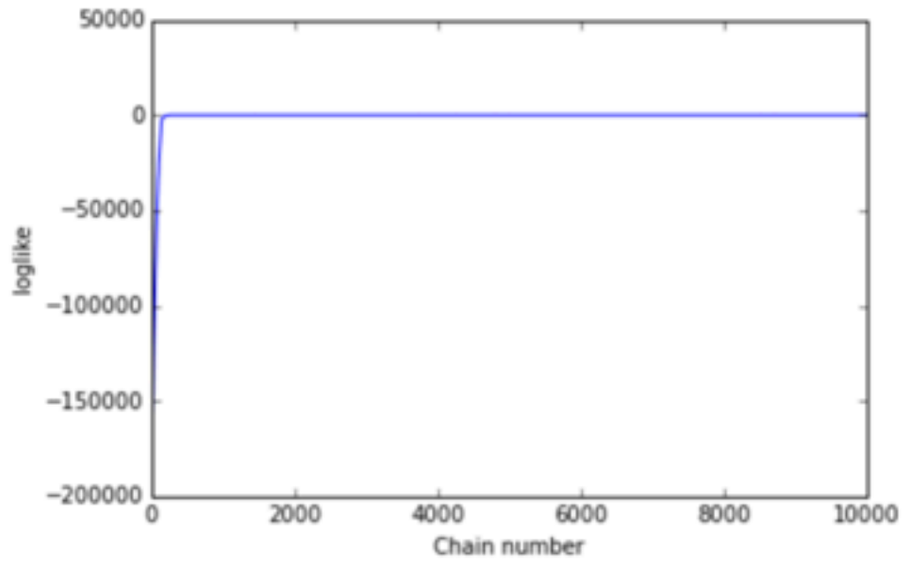


4. check post burn-in, tweak jumps, check mixing, plot posteriors

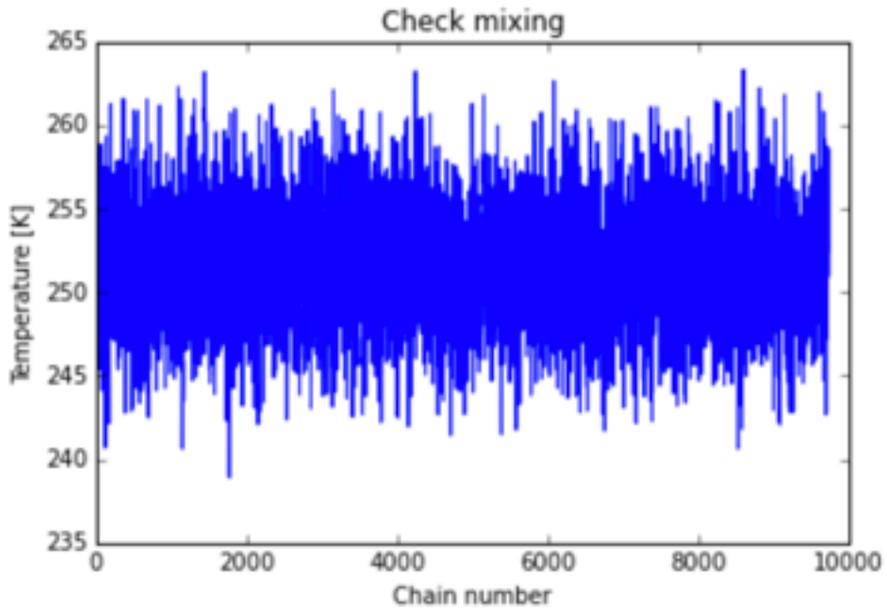
In [53]:
```
np.max(loglikechain)
```

Out[53]: 8.6442438885663009

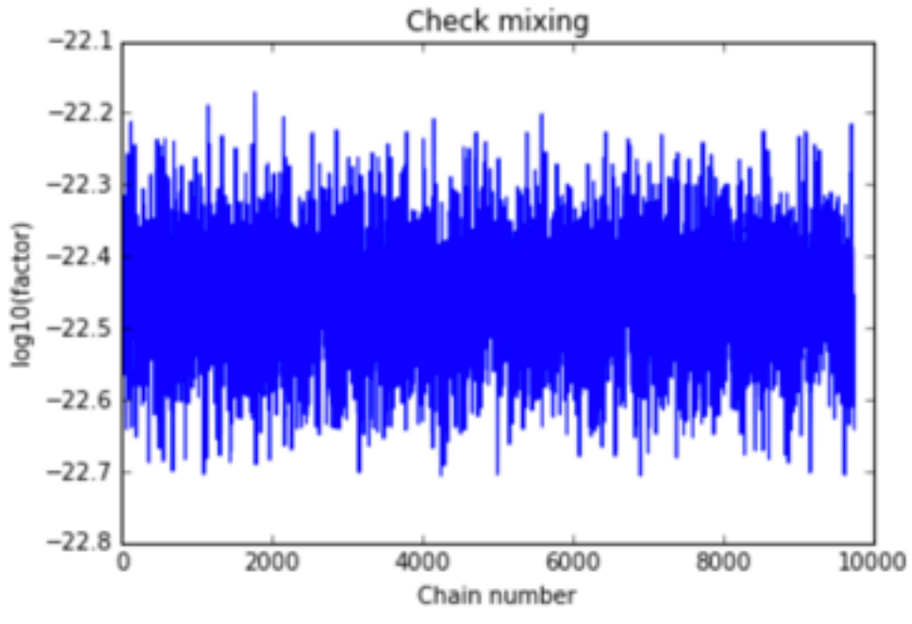
In [54]:
```
%matplotlib inline

plt.plot(loglikechain)
plt.xlabel('Chain number')
plt.ylabel('loglike')
plt.show()
```



In [16]:
```
%matplotlib inline

plt.plot(thetachain[burnj:,0])
plt.title('Check mixing')
plt.xlabel('Chain number')
plt.ylabel('Temperature [K]')
plt.show()
```



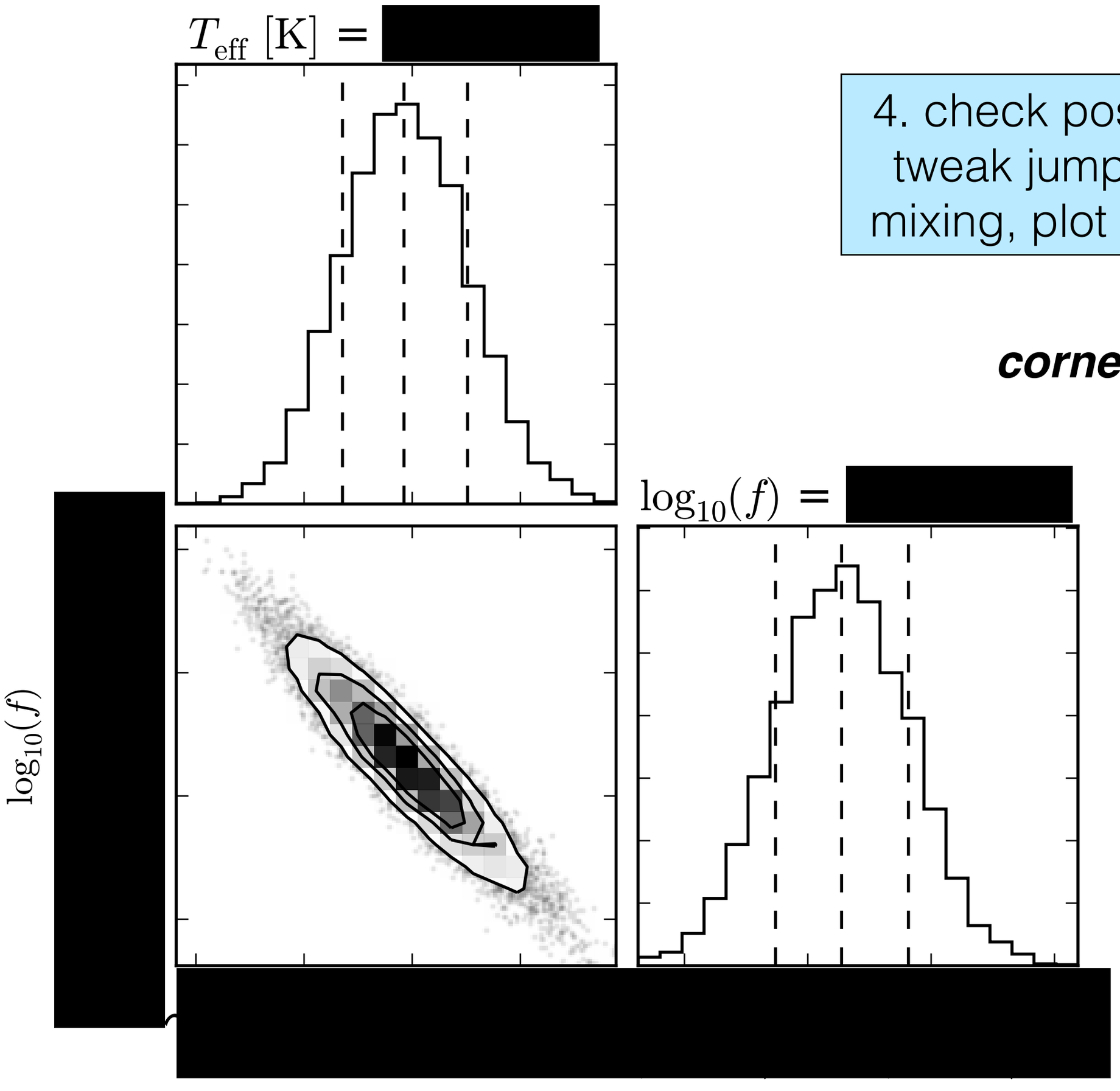
In [17]:
```
%matplotlib inline

plt.plot(thetachain[burnj:,1])
plt.title('Check mixing')
plt.xlabel('Chain number')
plt.ylabel('log10(factor)')
plt.show()
```



4. check post burn-in, tweak jumps, check mixing, plot posteriors

4. check post burn-in, tweak jumps, check mixing, plot posteriors

corner.py



YOUR TASK: Use this data set to calculate the temperature of WISE 0855

1] Write down your equations for likelihood, (priors) & model

2] Import data

3] Initialize from a random point and compute loglike'

4] Walk from there, using Metropolis rule and Gaussian proposals

5] Try just a few steps, tweak jump sizes till things run smoothly, may have to play around with priors if too wide

6] Once a good MCMC has ran, filter out the burn-in points

7] Make some nice plots proving i) burnt-in ii) good mixing and then your posteriors as a histogram

8] Write up your slides!