Optimizations in Monte-Carlo Radiative Transfer

Thomas Robitaille (Max Planck Institute for Astronomy)

Propagate many photon packets by randomly sampling from probability distribution functions for directions, frequencies, path lengths, interactions.

Propagate many photon packets by randomly sampling from probability distribution functions for directions, frequencies, path lengths, interactions.

Problem solved?

Propagate many photon packets by randomly sampling from probability distribution functions for directions, frequencies, path lengths, interactions.

Problem solved?

Yes and No - Monte-Carlo RT will eventually give the right result, but for most real-life cases it will often be inefficient in its basic implementation.

When is Monte-Carlo inefficient?

When computing **temperatures** from absorptions

When making **images/SEDs** the easy way

When the dust is **very dense**

When the dust is **not dense enough**

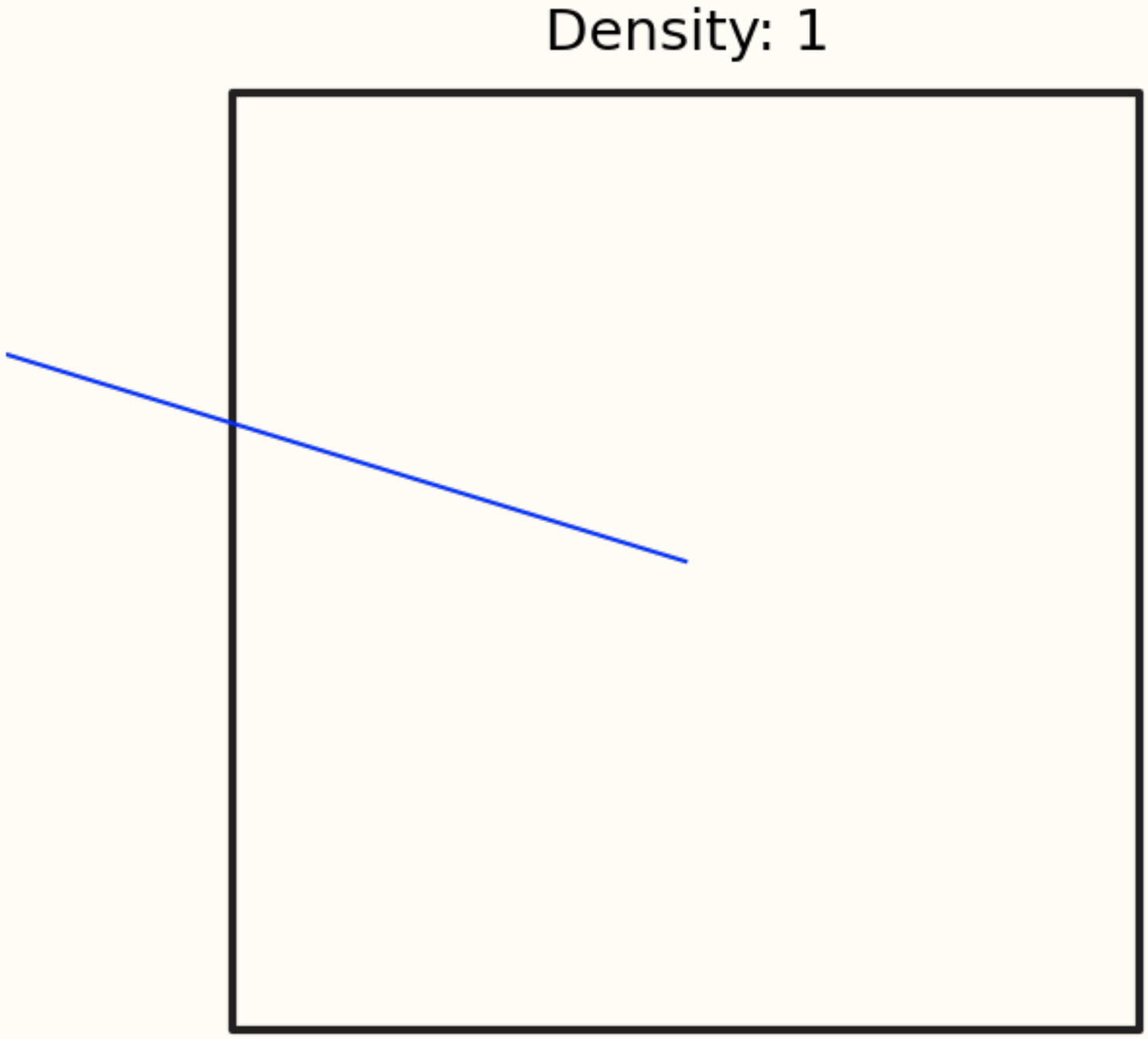
When computing **temperatures** from absorptions

When making **images/SEDs** the easy way

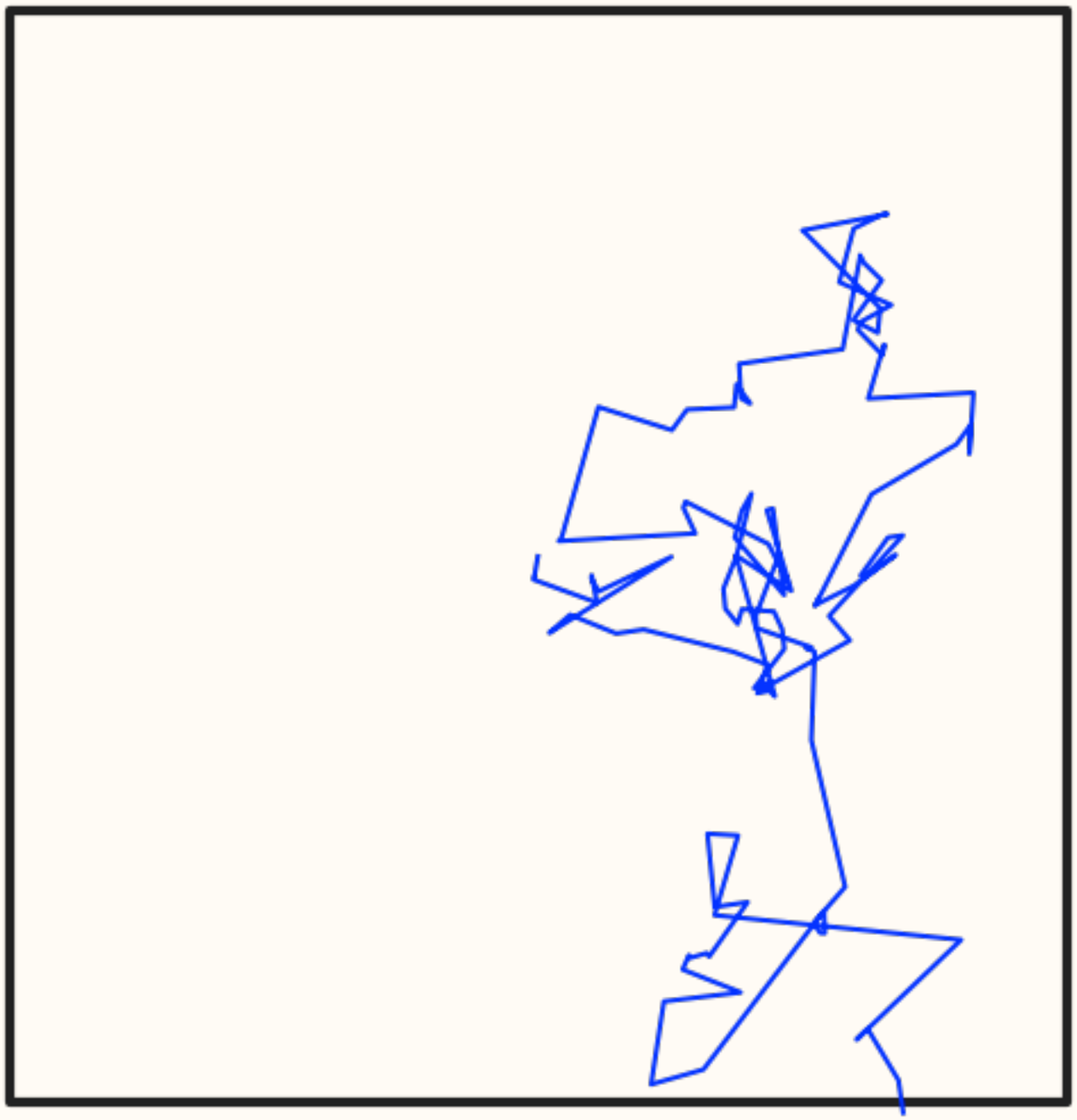
When the dust is **very dense**

When the dust is **not dense enough**

When the dust is **very dense**

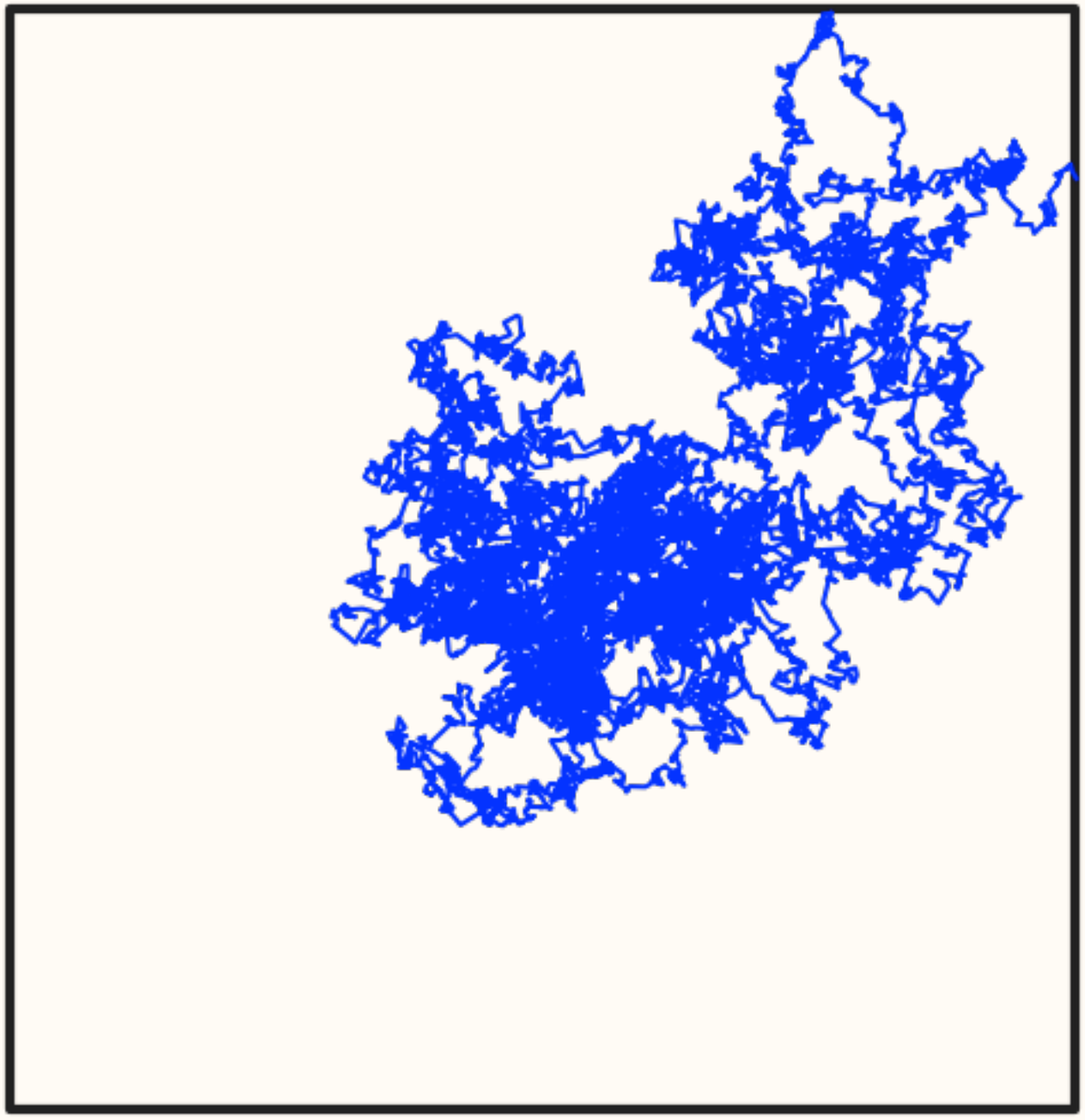

Density: 1
Interactions: 2

Density: 10



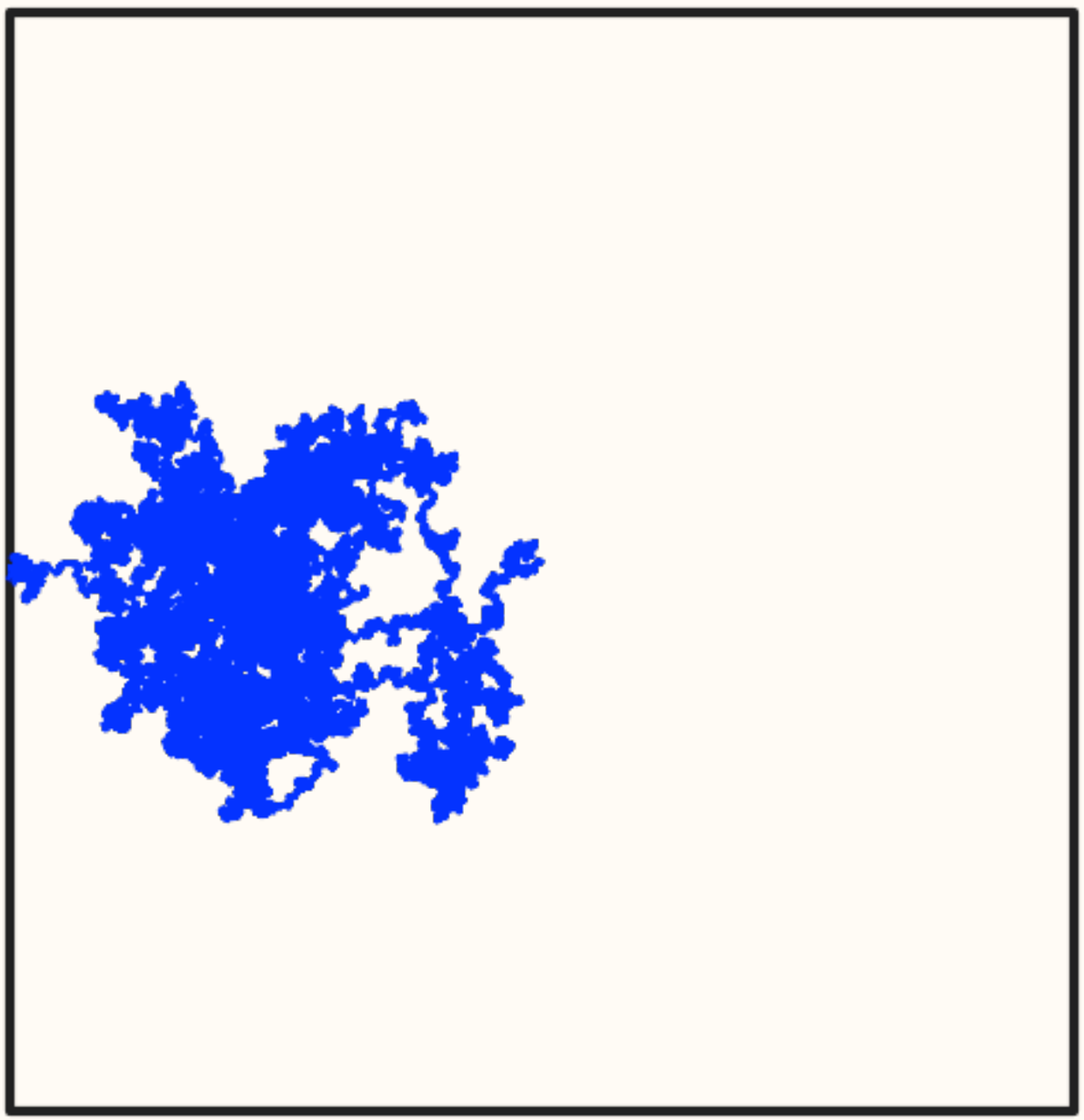
Interactions: 122

Density: 100



Interactions: 17918

Density: 1000



Interactions: 719222

Number of interactions goes
up super-linearly with density...

Analytical Radiative Transfer to the rescue!

$$\nabla \cdot (D \nabla E) = \frac{\partial E}{\partial t}$$

This is the **Diffusion Equation**

(not specific to Radiative Transfer)

See Fick's second law: http://en.wikipedia.org/wiki/Fick's_laws_of_diffusion

Analytical Radiative Transfer to the rescue!

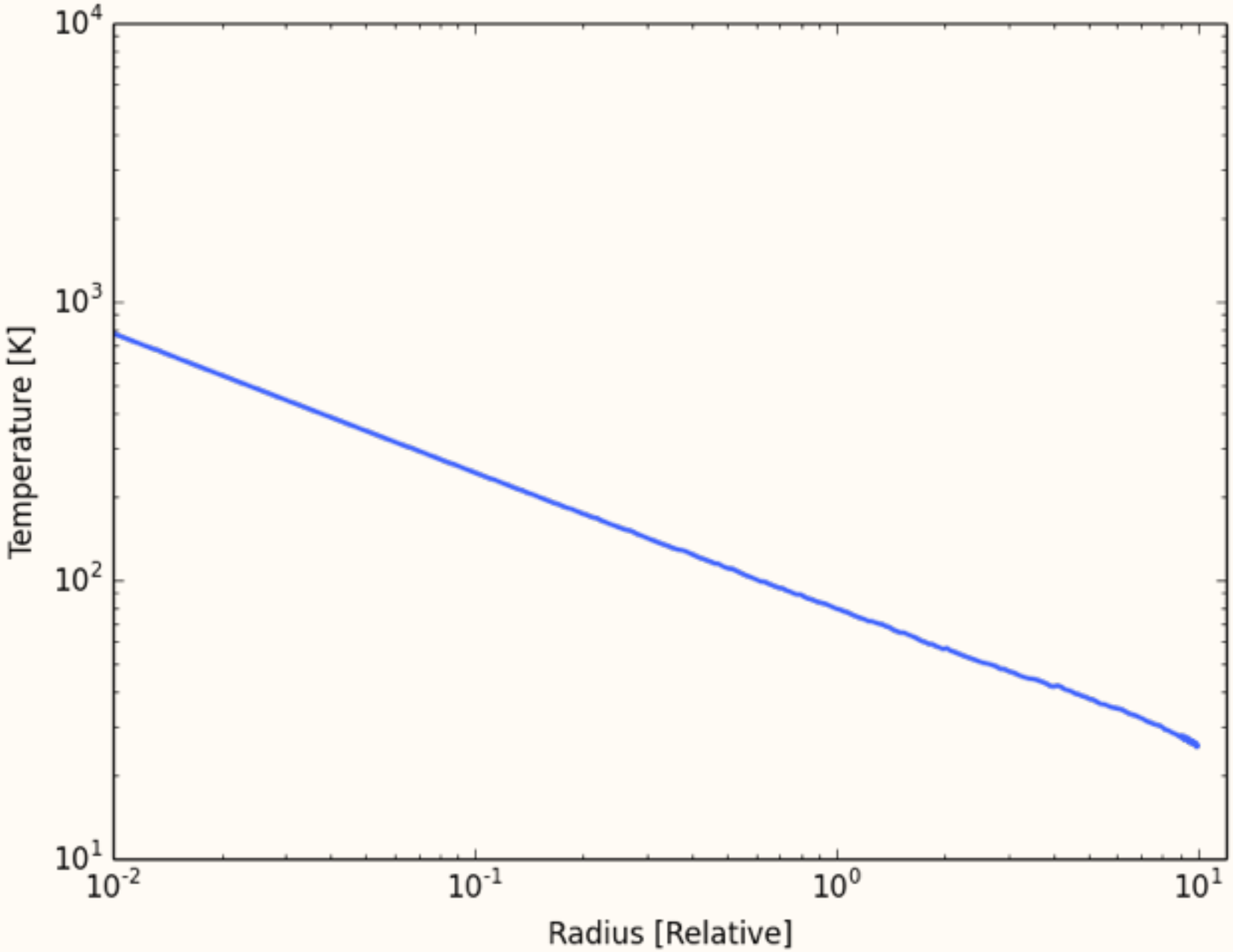
$$\nabla \cdot \left(\frac{1}{3\rho\chi_R} \nabla E \right) = \frac{1}{c} \frac{\partial E}{\partial t}$$

This is the **Diffusion Equation**

(not specific to Radiative Transfer)

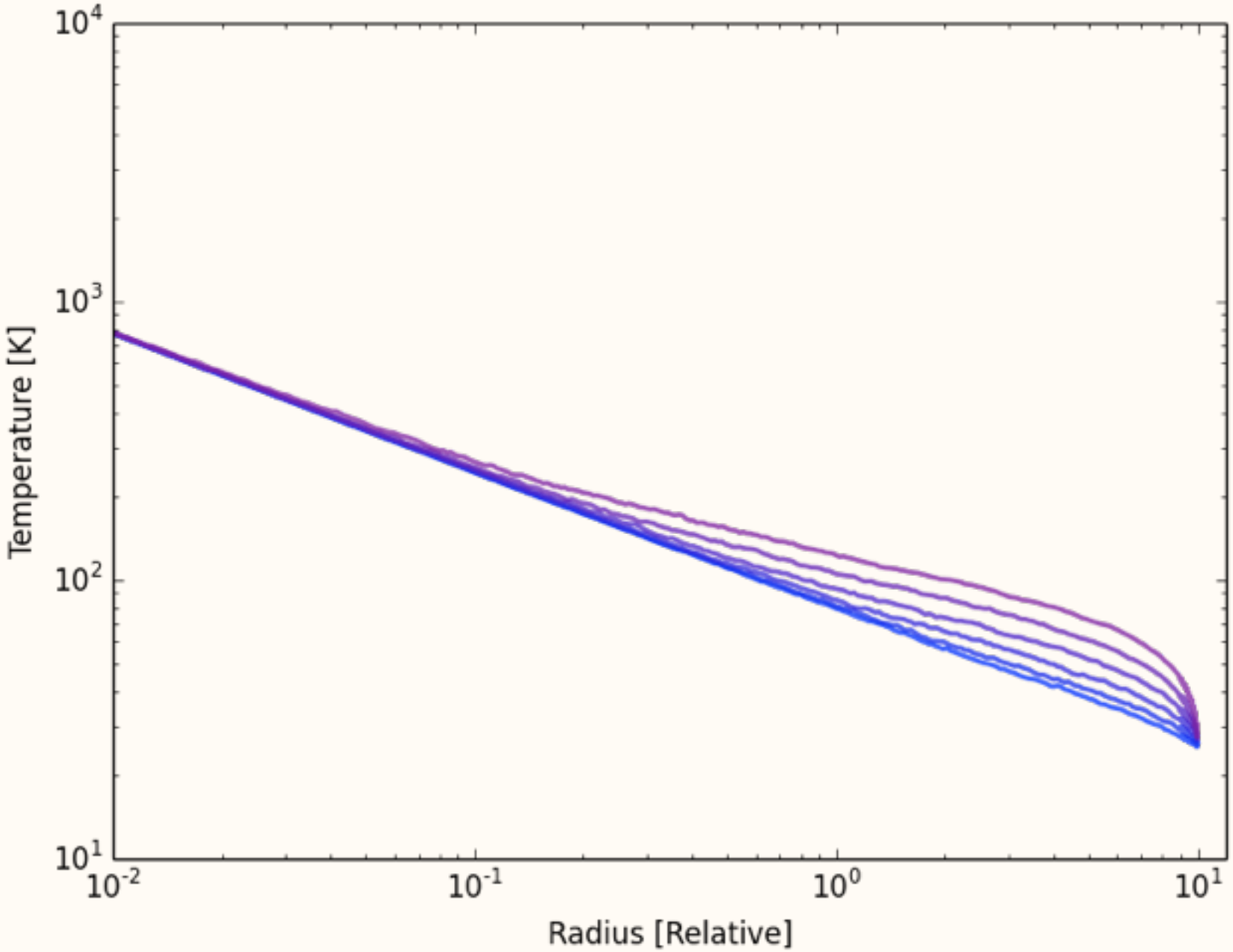
In the limit of very optically thick dust, solving this differential equation is much more efficient than Monte-Carlo photon propagation

Demonstration



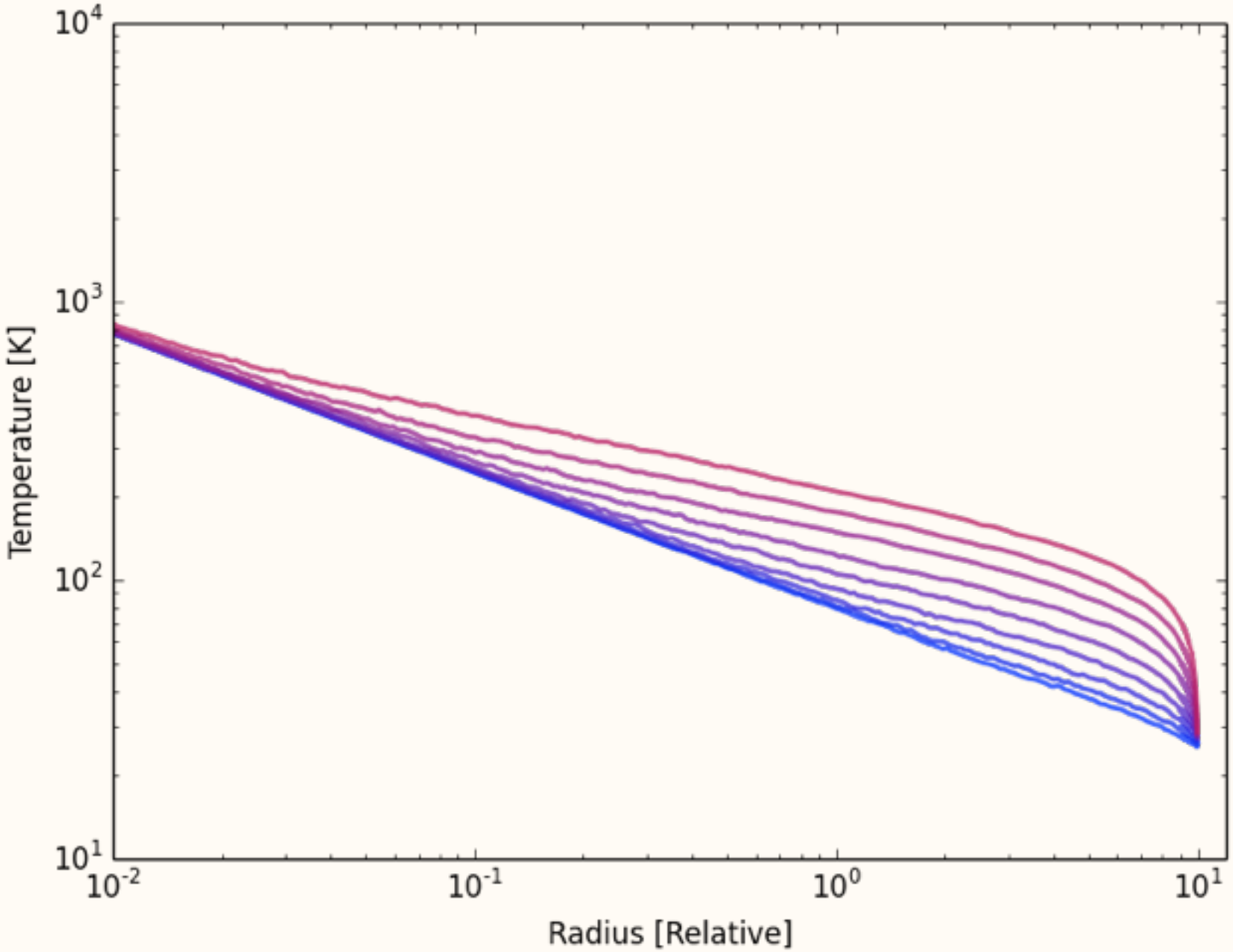
Constant density sphere, central point source

Demonstration



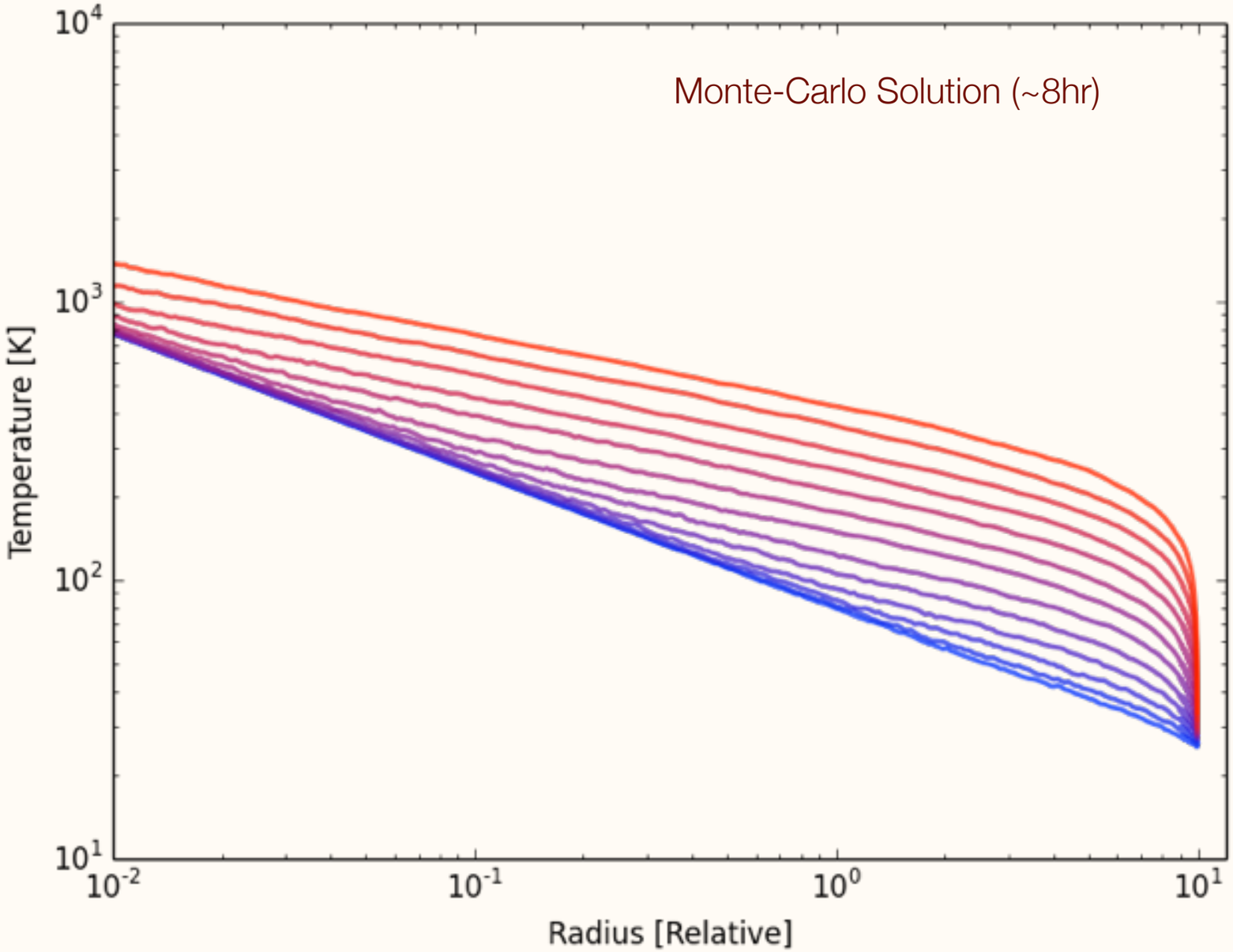
Constant density sphere, central point source

Demonstration



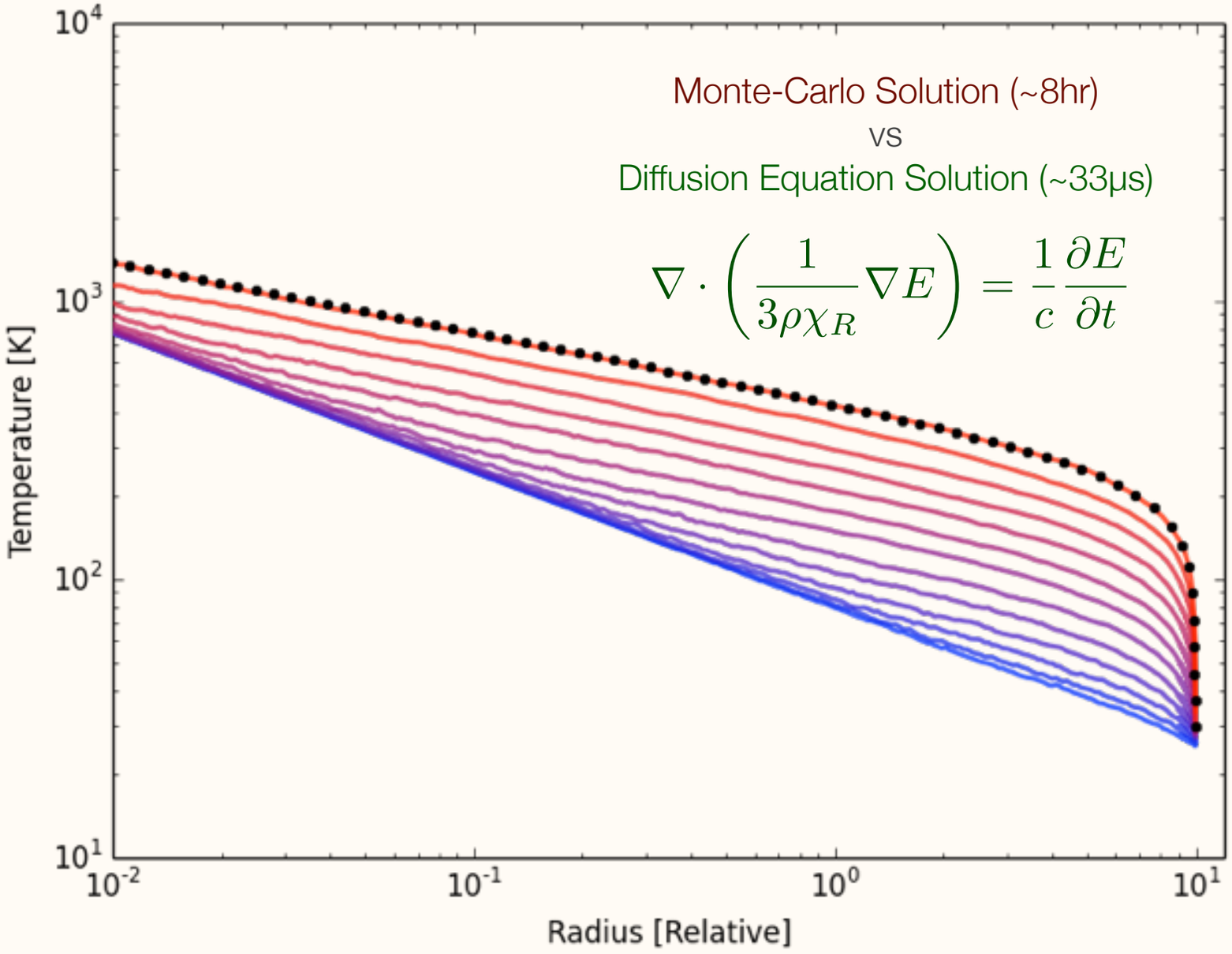
Constant density sphere, central point source

Demonstration



Constant density sphere, central point source

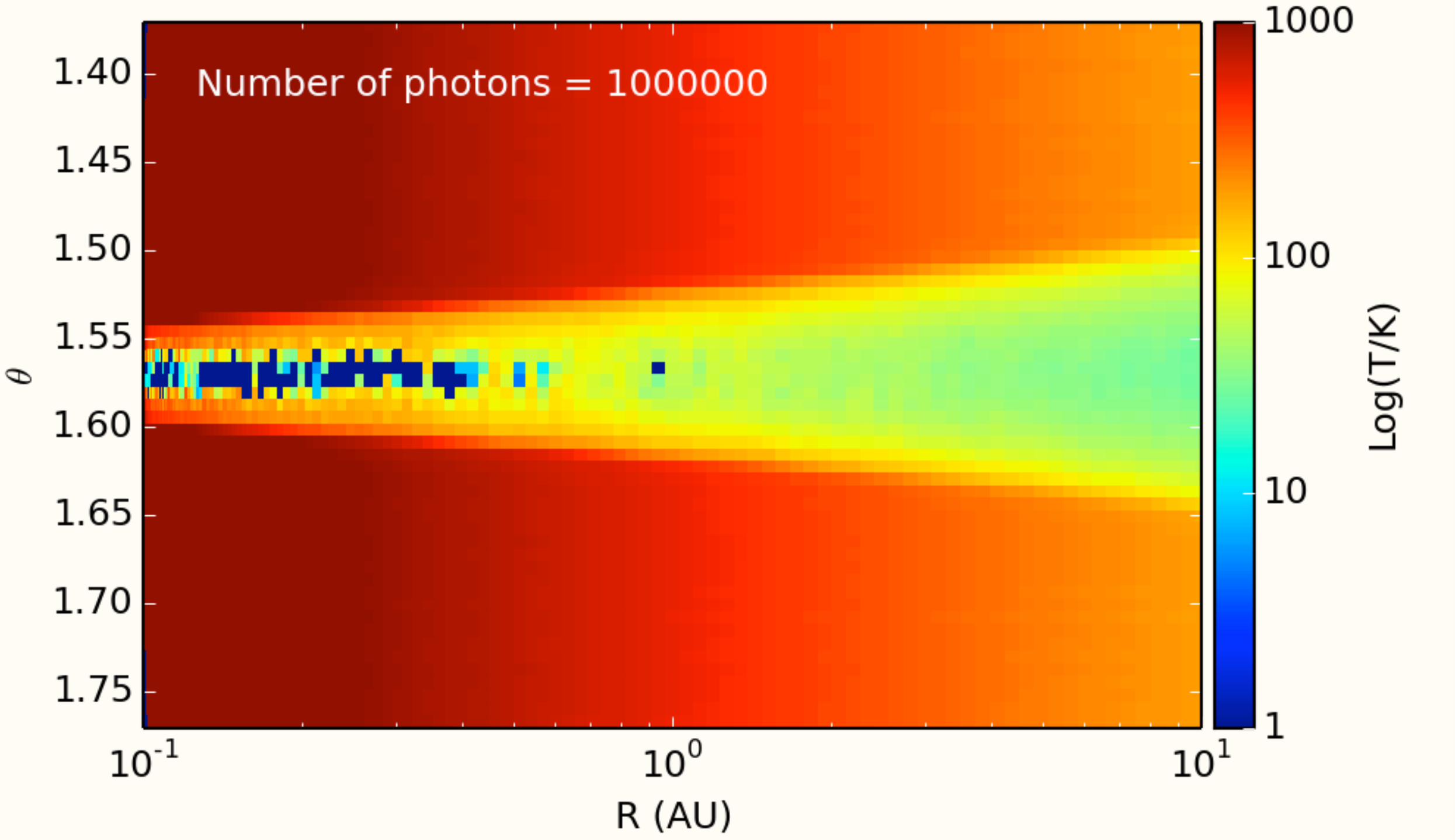
Demonstration



Constant density sphere, central point source

How do we make use of this in Monte-Carlo?

Dealing with regions with low photon count

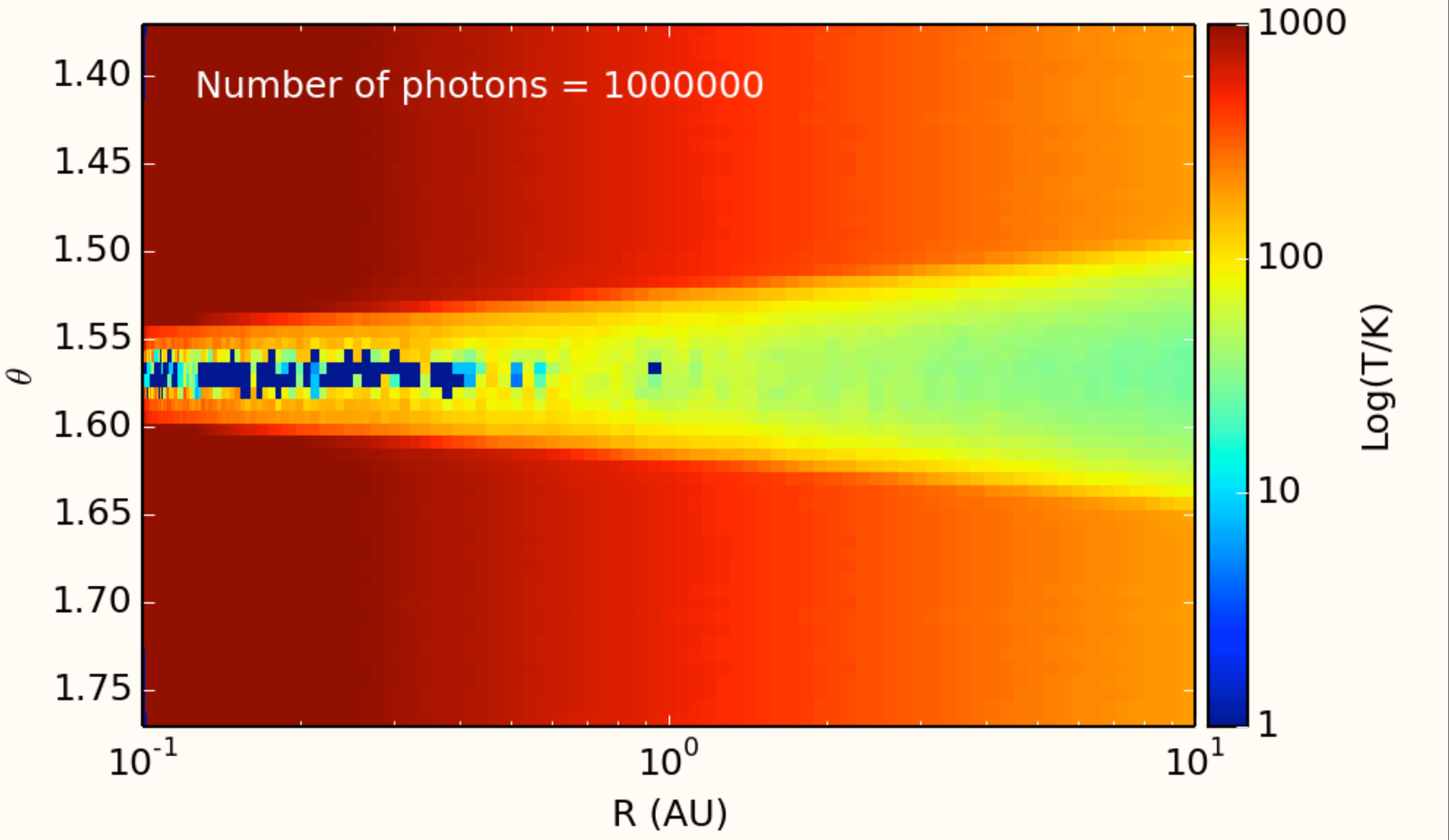


Dealing with regions with low photon count

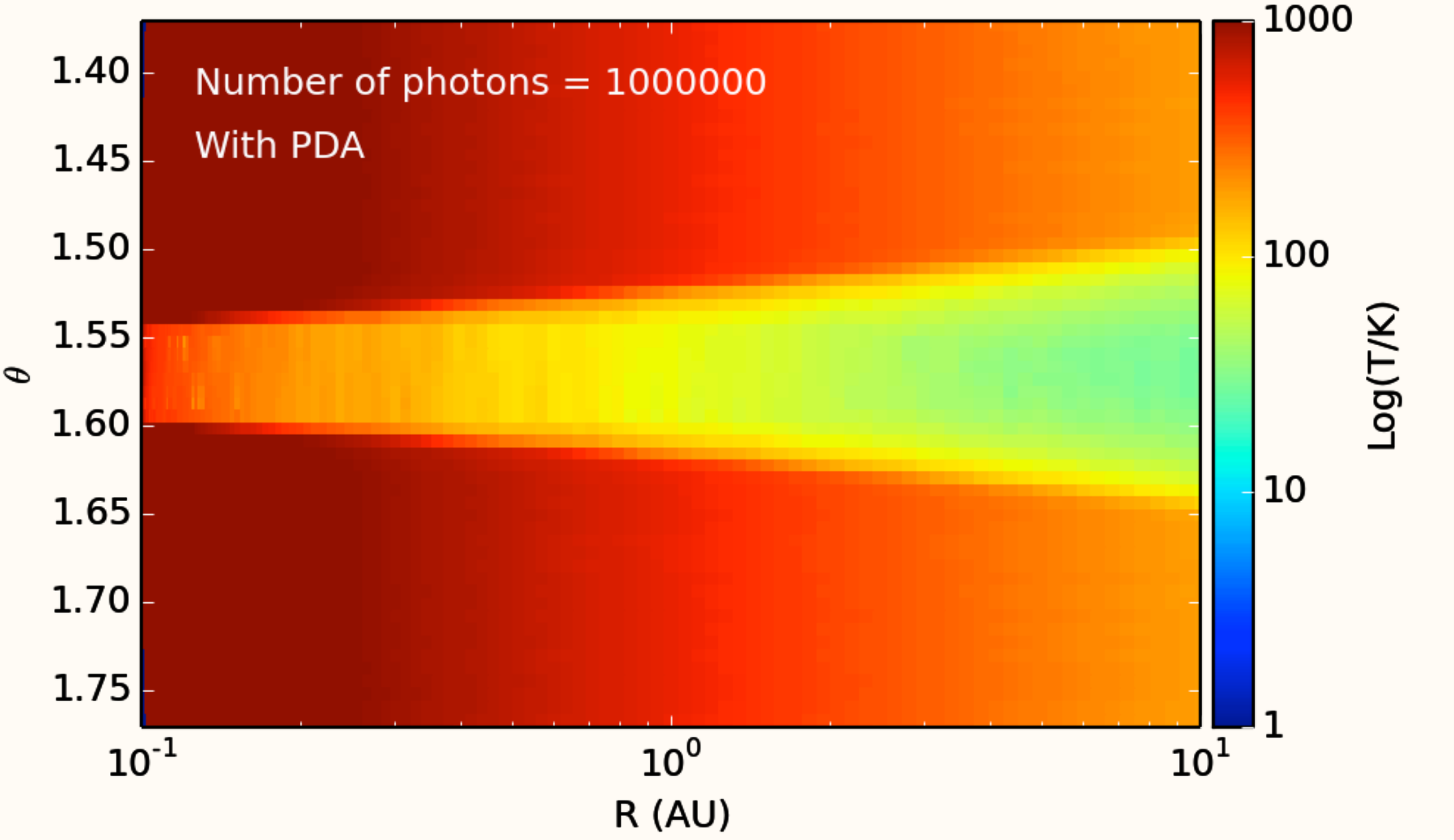
Algorithm: in high density regions where photons cannot penetrate, we locally solve the diffusion equation using neighboring cells as boundary conditions. This is the **partial diffusion approximation.**

See Min et al. (2009, A&A 497, 155)

Dealing with regions with low photon count



Dealing with regions with low photon count



$$\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2 D\,\frac{\partial T^4}{\partial r}\right)+\frac{1}{\sin\theta}\,\frac{\partial}{r\partial\theta}\left(\sin\theta\,D\,\frac{\partial T^4}{r\partial\theta}\right)+\frac{\partial}{r\sin\theta\partial\phi}\left(D\,\frac{\partial T^4}{r\sin\theta\partial\phi}\right)=0$$

(for Spherical Polar Coordinates)

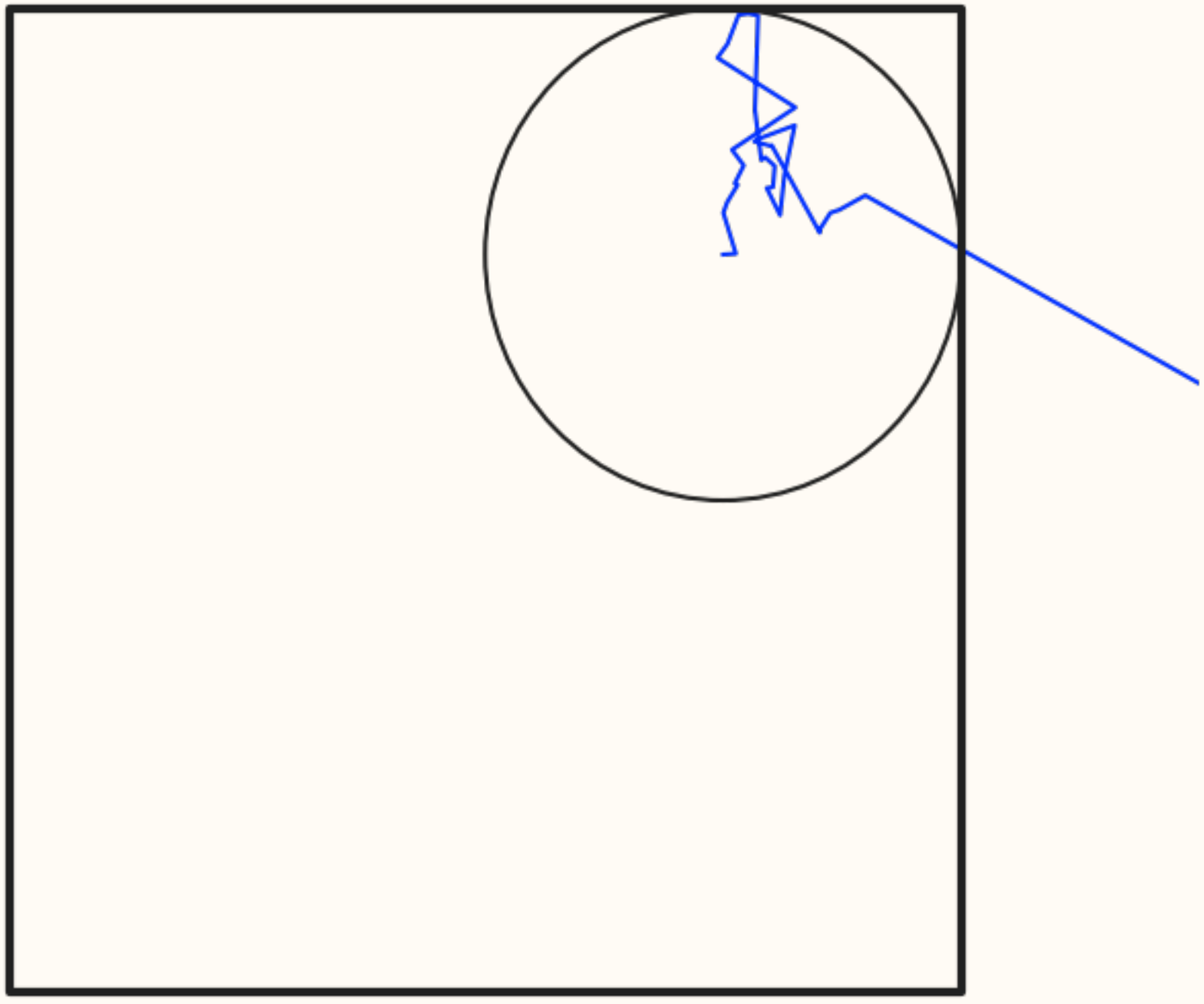
$$\frac{1}{r_i^2\,\Delta r_i}\left(r_{i+1/2}^2\,\frac{T_{i+1,j,k}^4-T_{i,j,k}^4}{\Delta\tau_{i+1,j,k}^r+\Delta\tau_{i,j,k}^r}+r_{i-1/2}^2\,\frac{T_{i-1,j,k}^4-T_{i,j,k}^4}{\Delta\tau_{i,j,k}^r+\Delta\tau_{i-1,j,k}^r}\right)$$

$$\frac{1}{\sin\theta_j}\frac{1}{r\Delta\theta_j}\left(\sin\theta_{j+1/2}\,\frac{T_{i,j+1,k}^4-T_{i,j,k}^4}{\Delta\tau_{i,j+1,k}^\theta+\Delta\tau_{i,j,k}^\theta}+\sin\theta_{j-1/2}\,\frac{T_{i,j-1,k}^4-T_{i,j,k}^4}{\Delta\tau_{i,j,k}^\theta+\Delta\tau_{i,j-1,k}^\theta}\right)$$

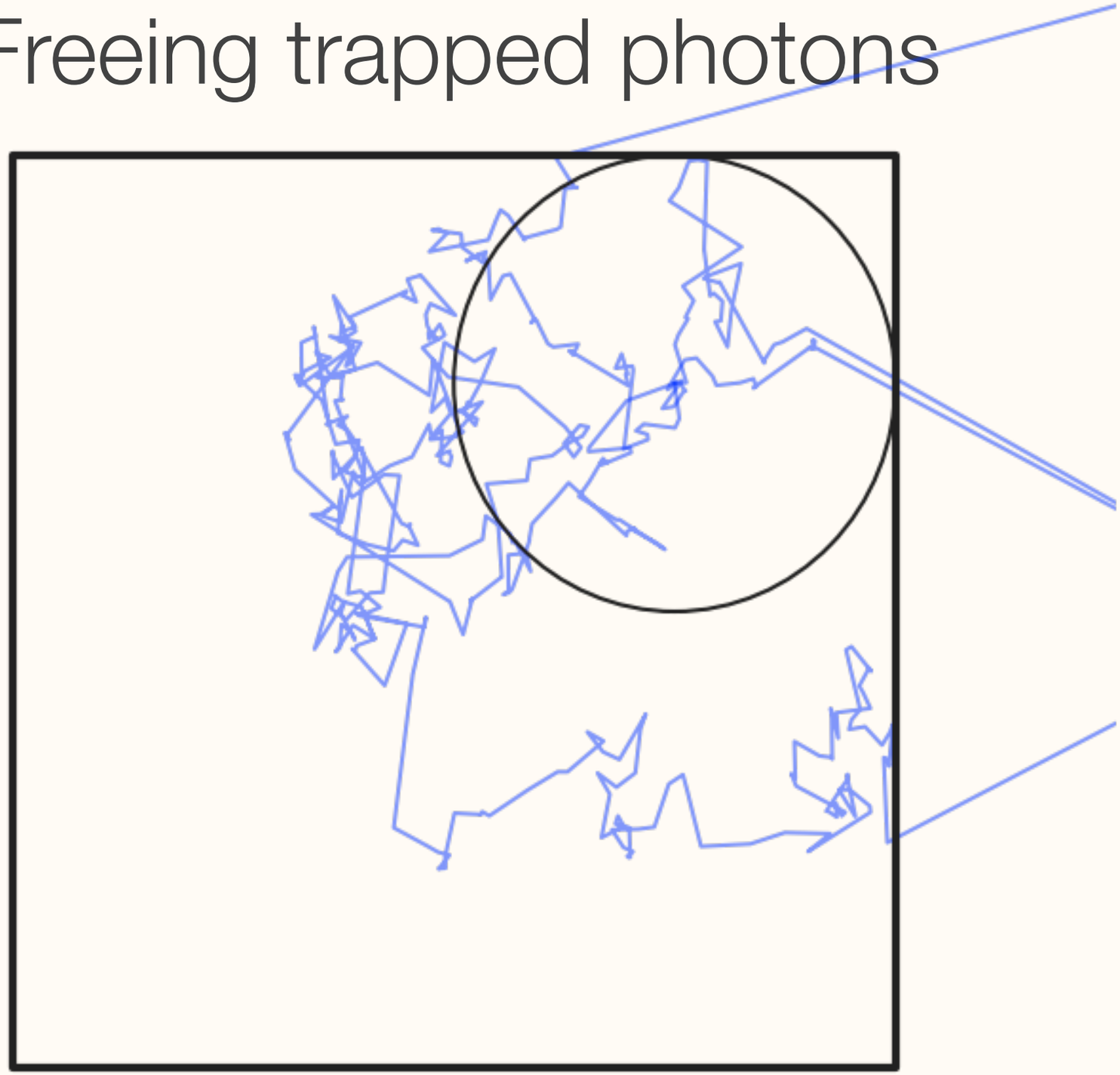
$$\frac{1}{r\sin\theta\Delta\phi_k}\left(\frac{T_{i,j,k+1}^4-T_{i,j,k}^4}{\Delta\tau_{i,j,k+1}^\phi+\Delta\tau_{i,j,k}^\phi}+\frac{T_{i,j,k-1}^4-T_{i,j,k}^4}{\Delta\tau_{i,j,k}^\phi+\Delta\tau_{i,j,k-1}^\phi}\right)=0$$

(for Spherical Polar Coordinates)

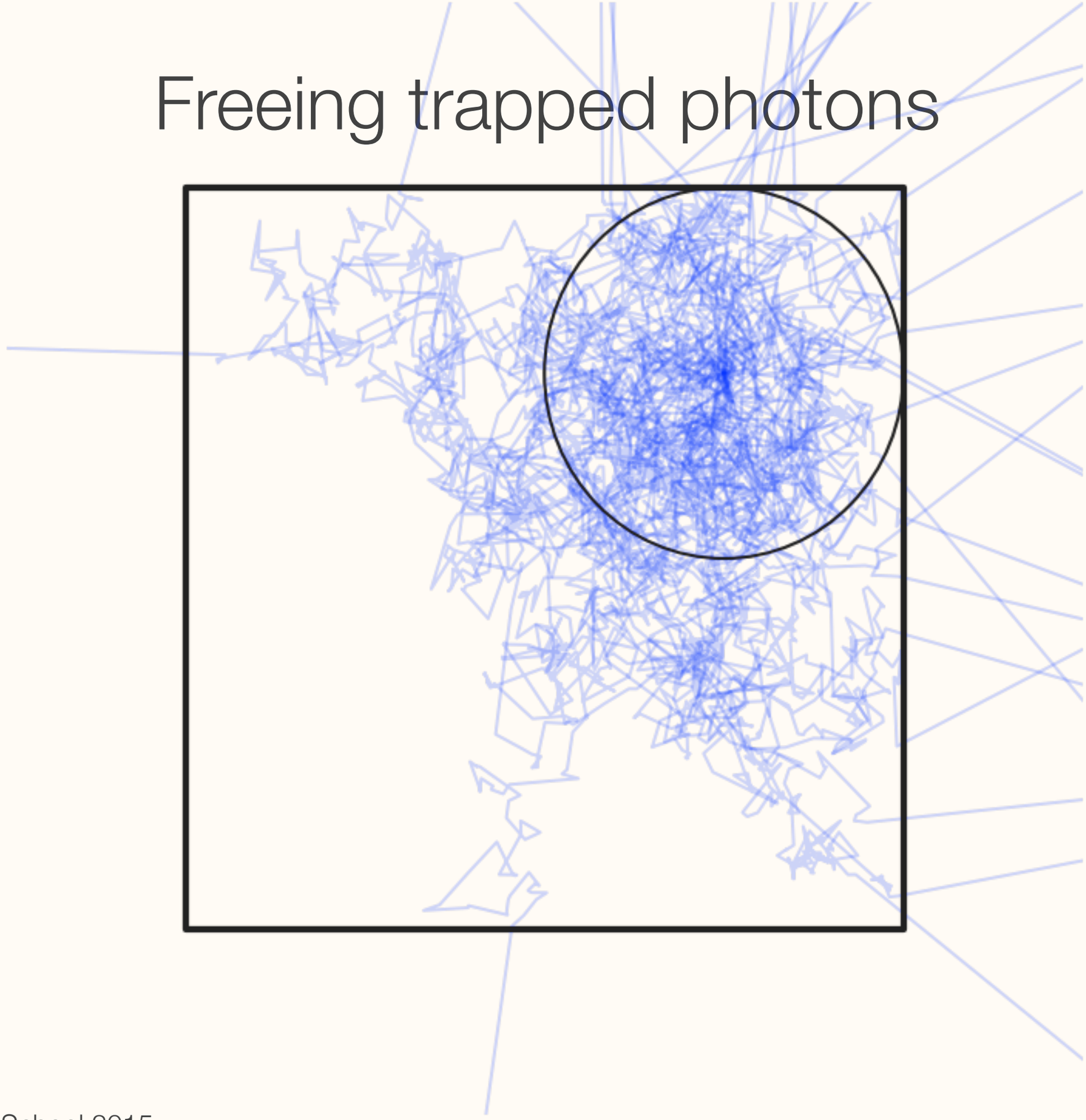
Freeing trapped photons



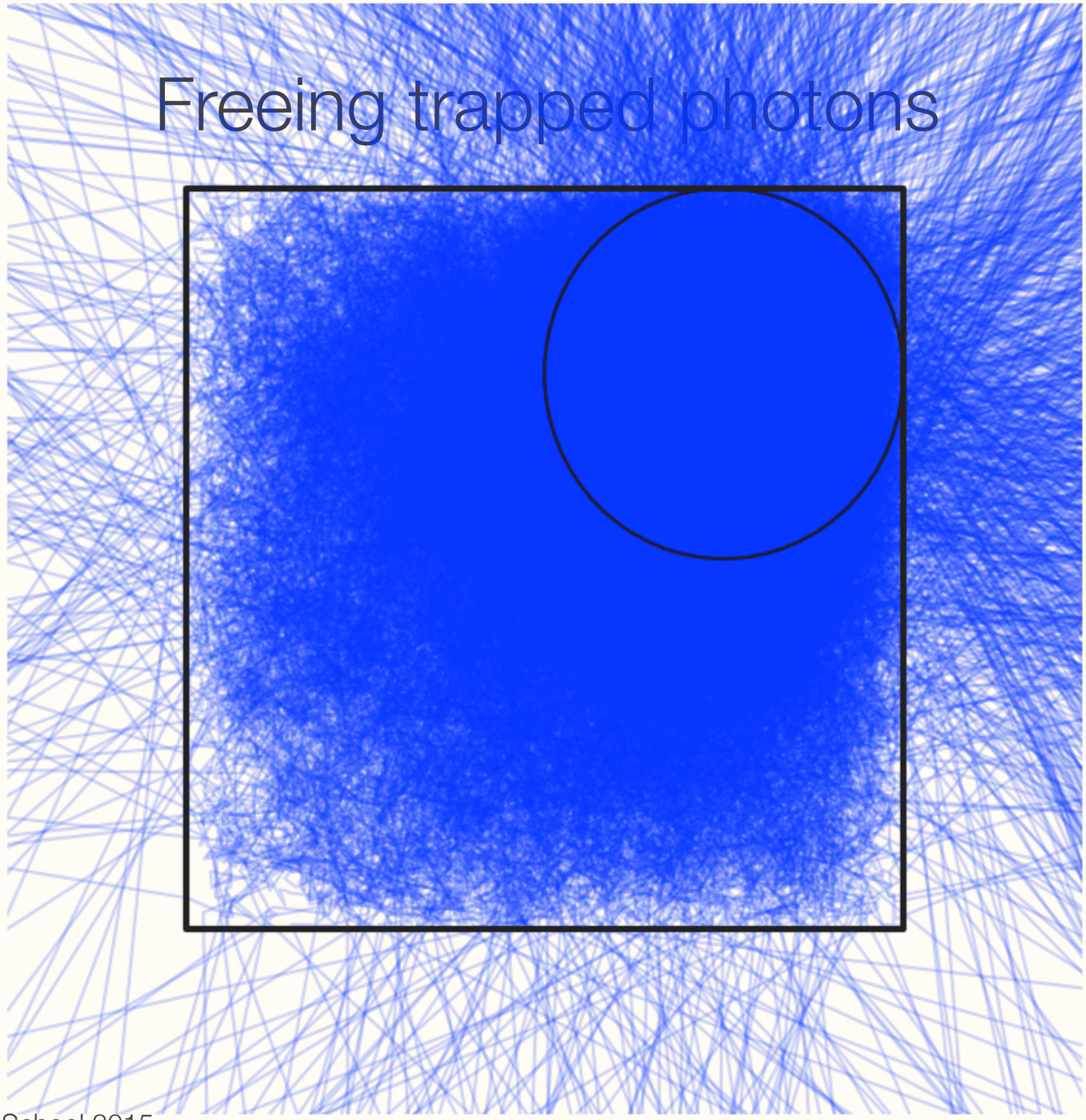
Freeing trapped photons



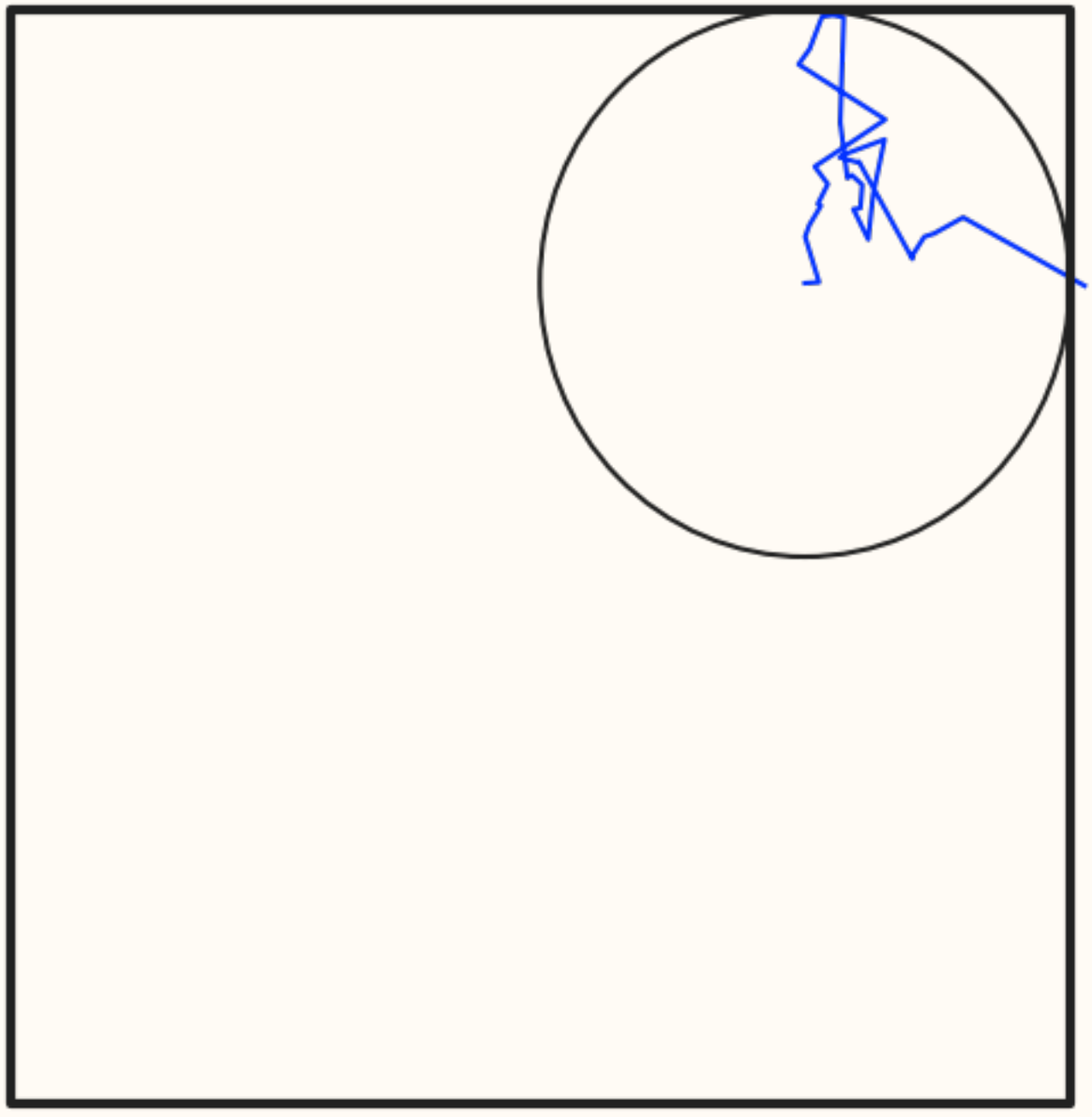
Freeing trapped photons



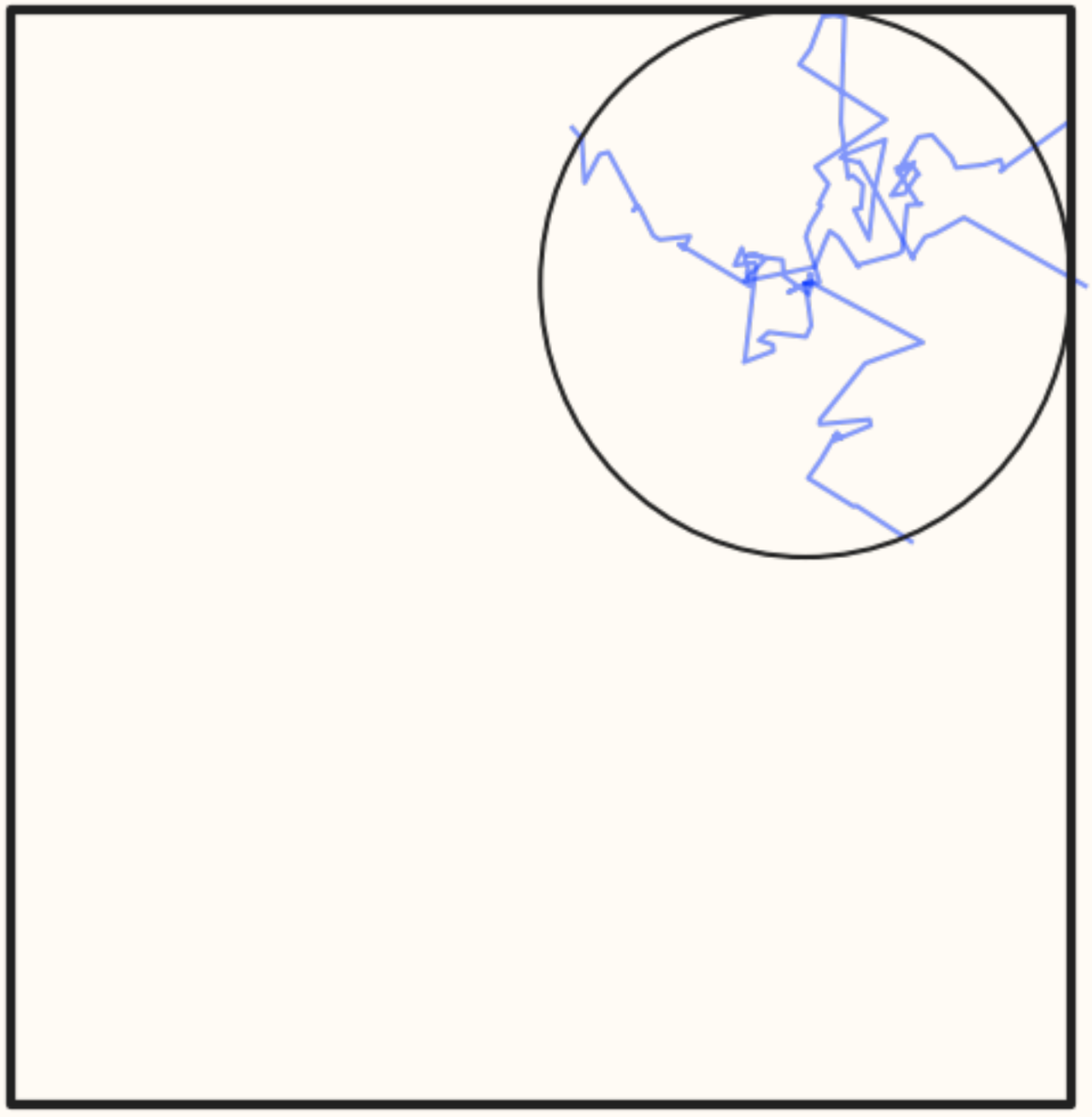
Freeing trapped photons



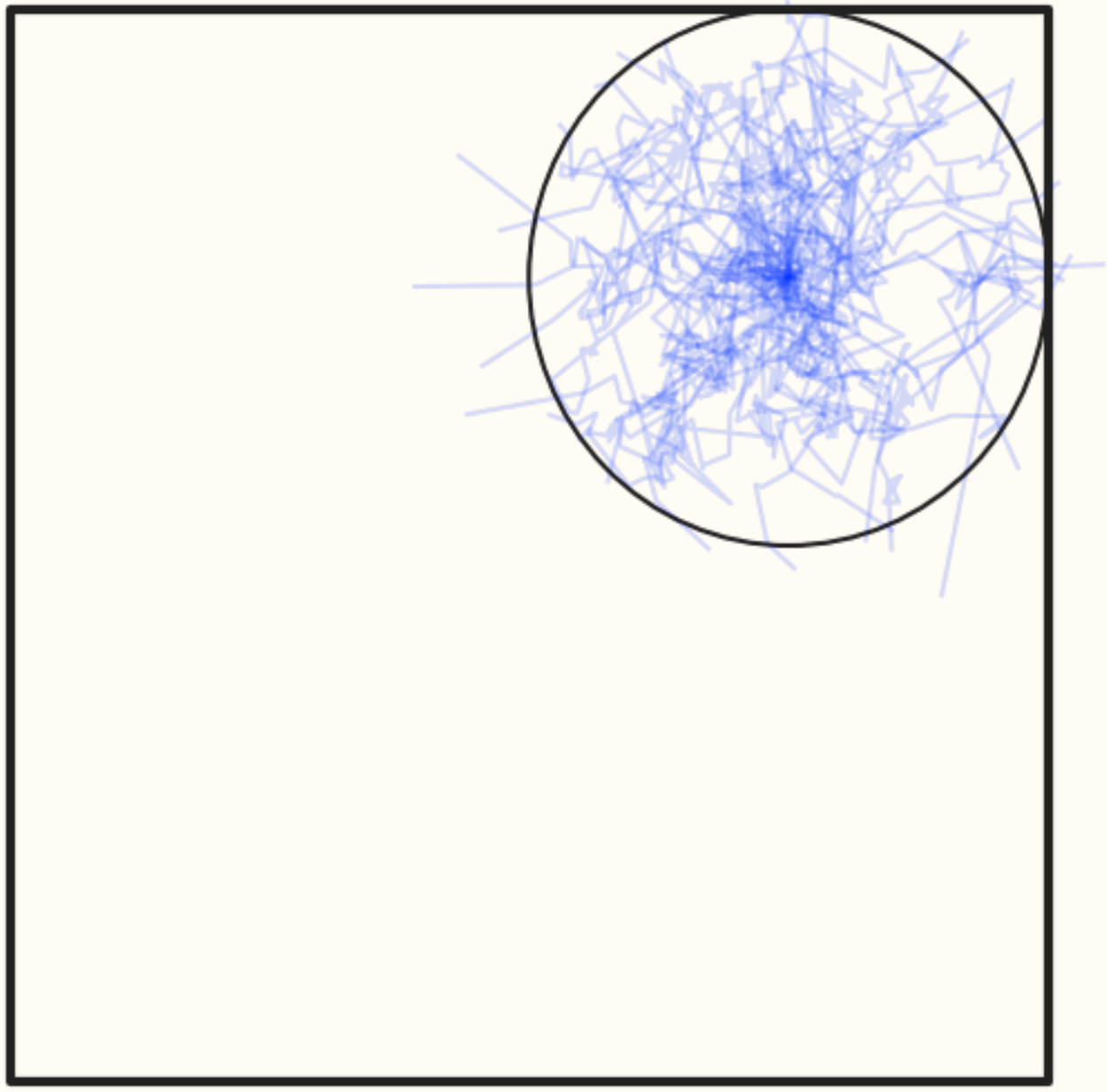
Freeing trapped photons



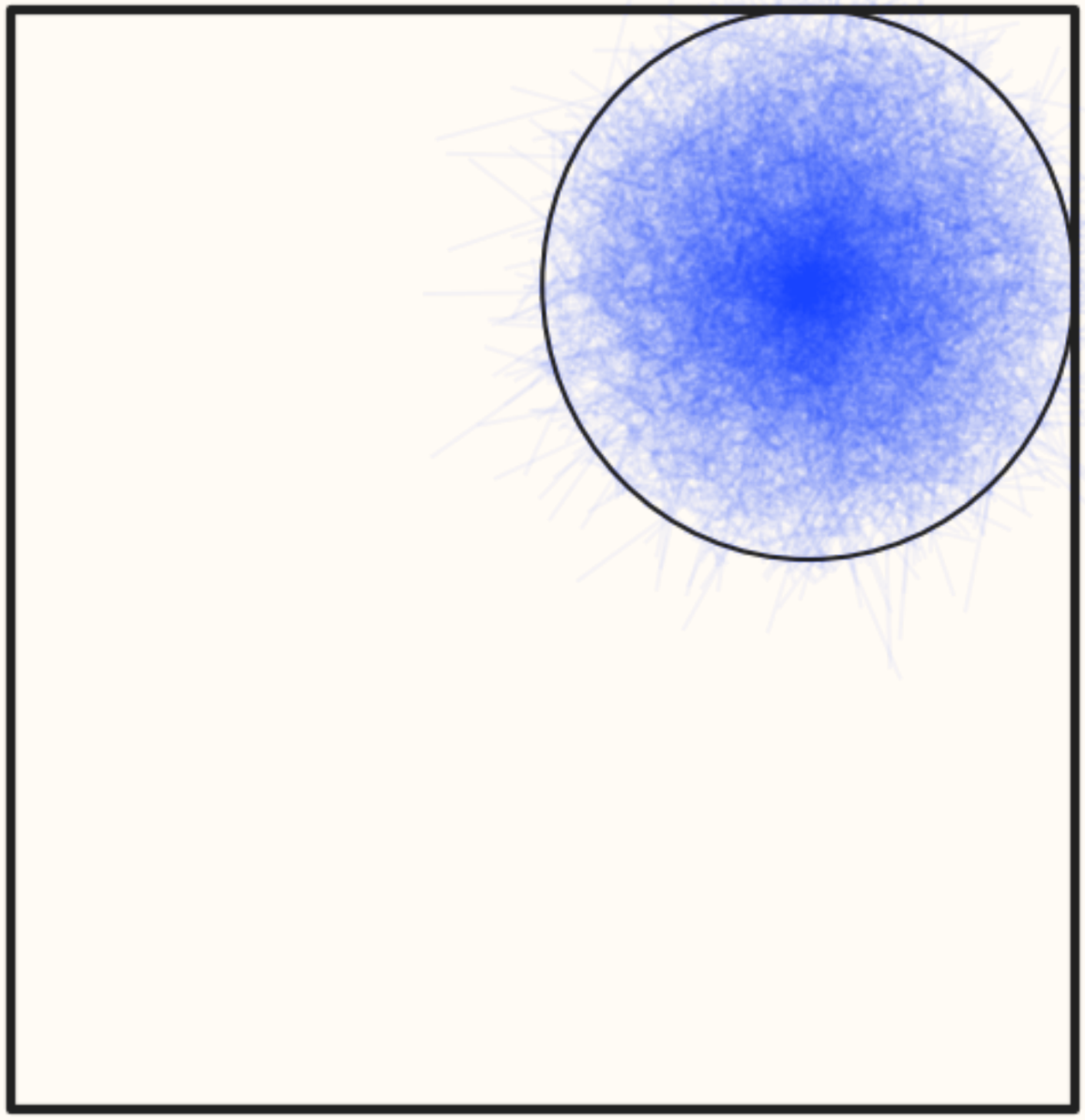
Freeing trapped photons



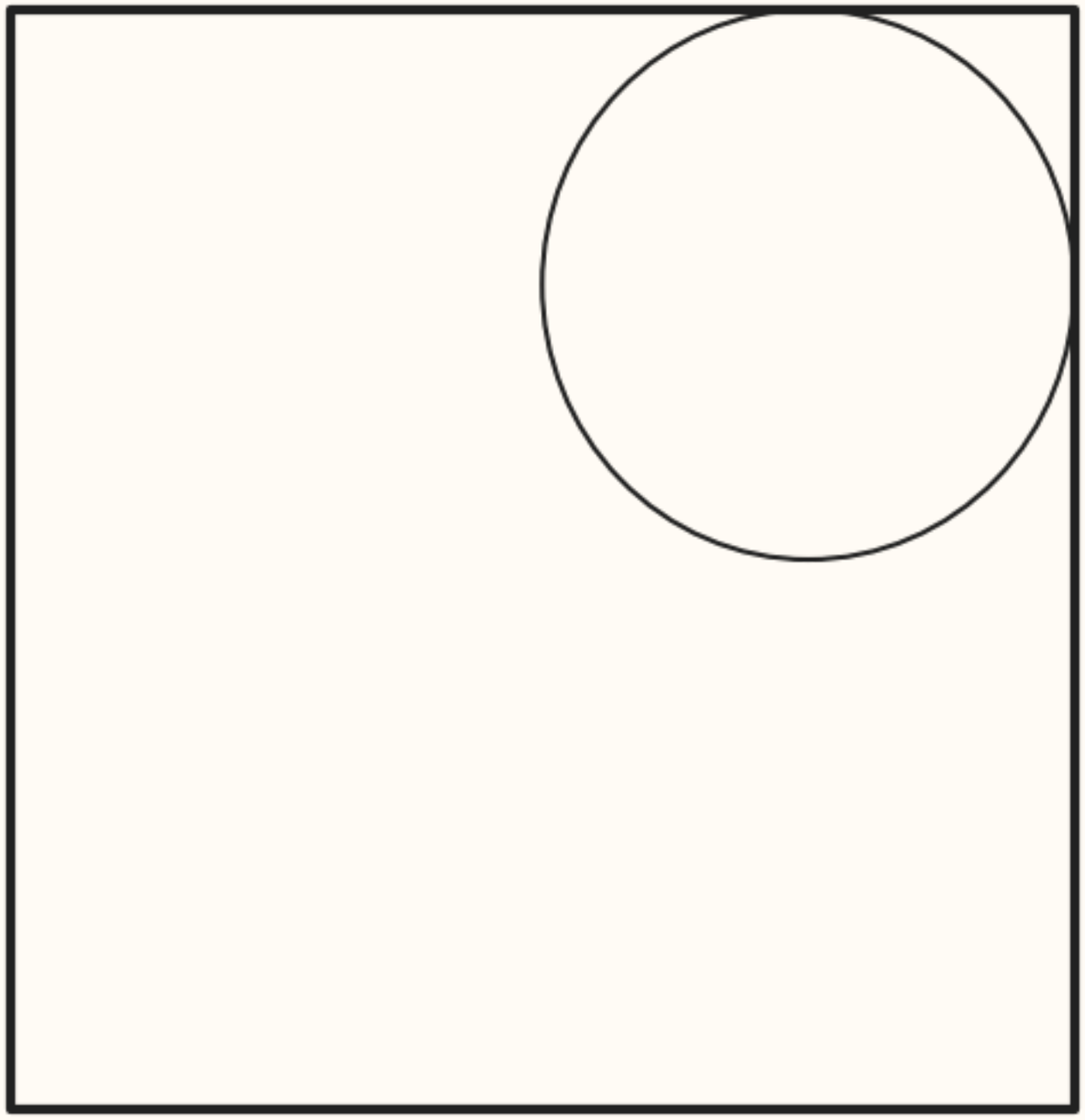
Freeing trapped photons



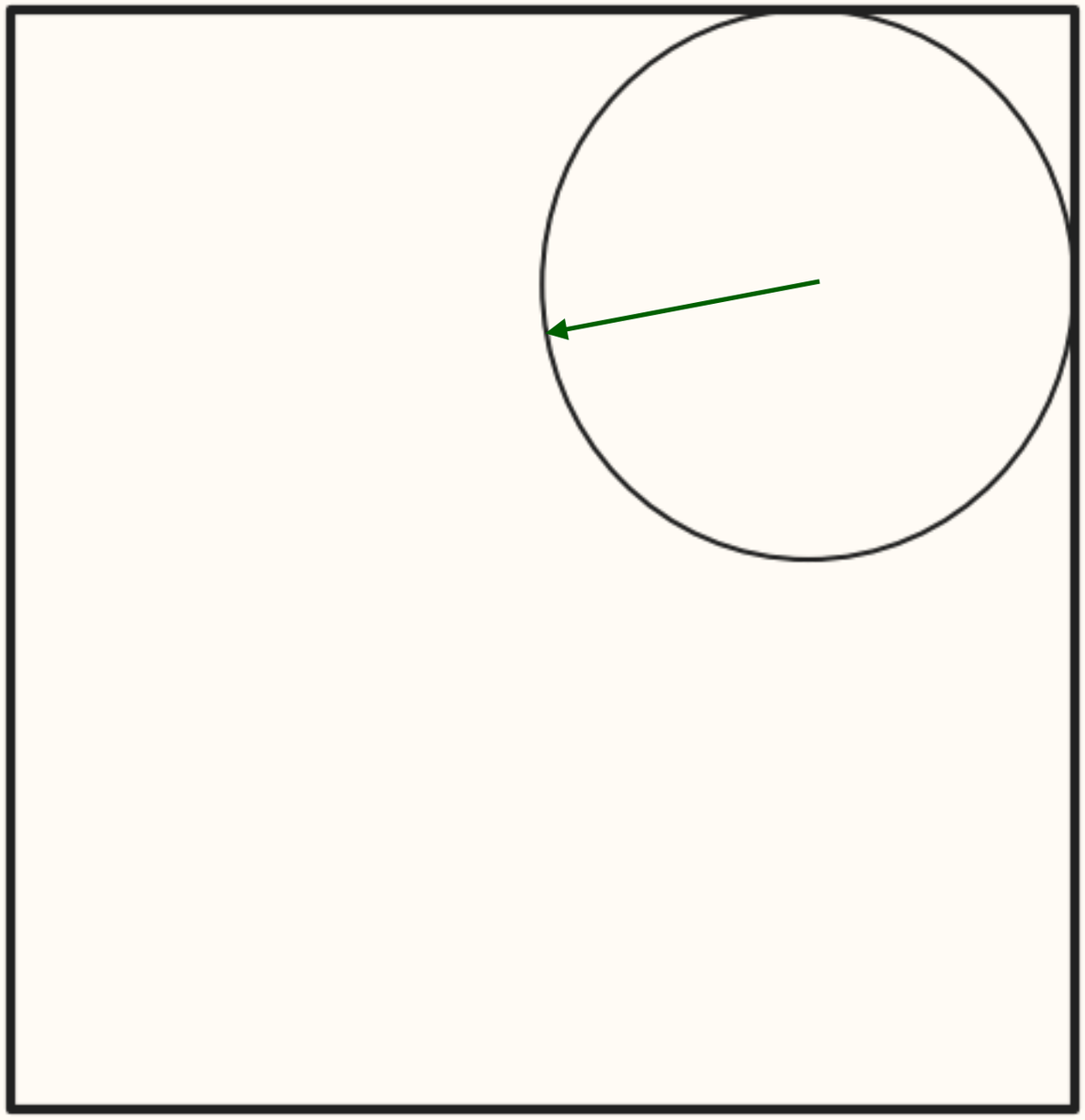
Freeing trapped photons



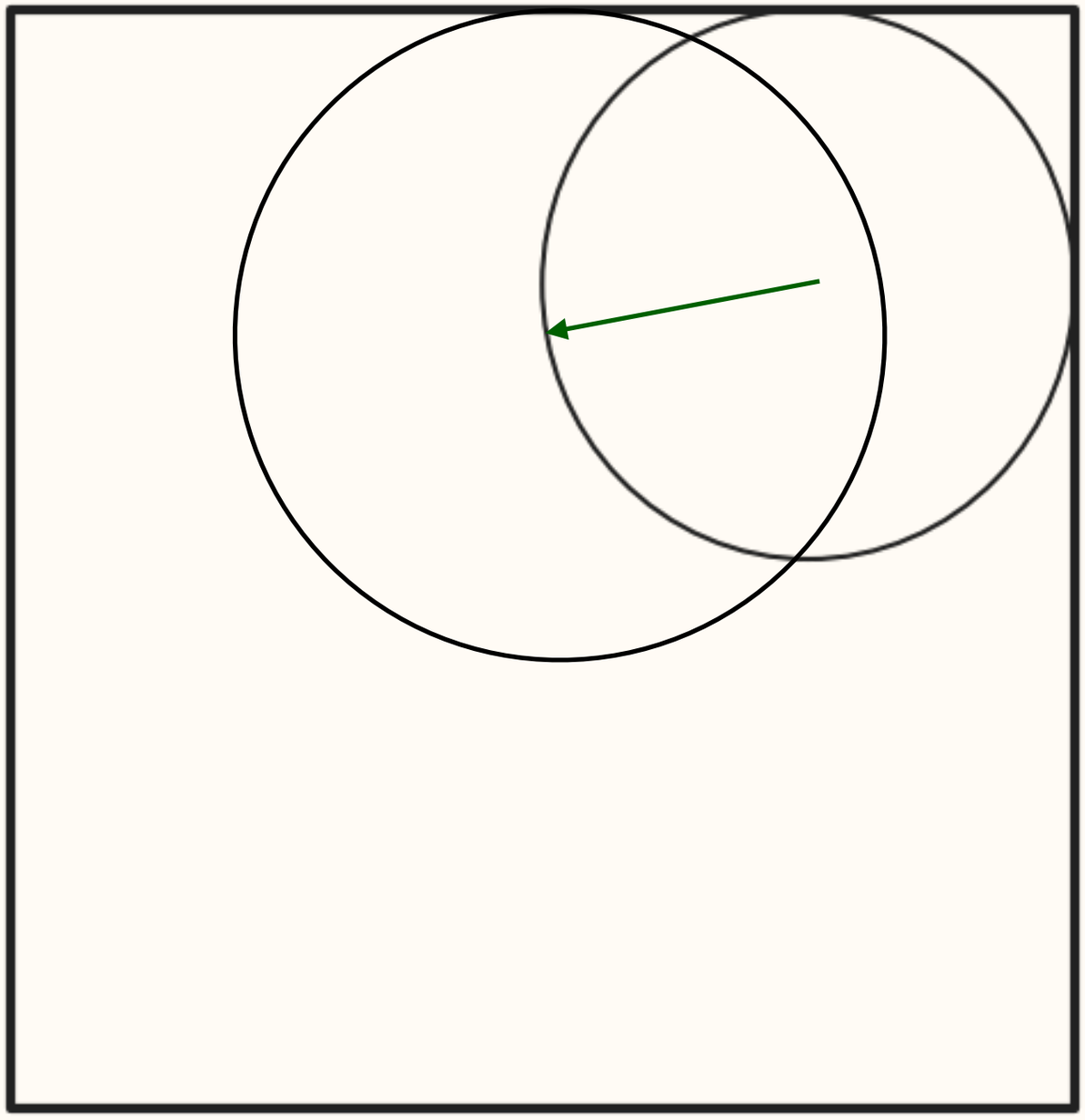
Freeing trapped photons



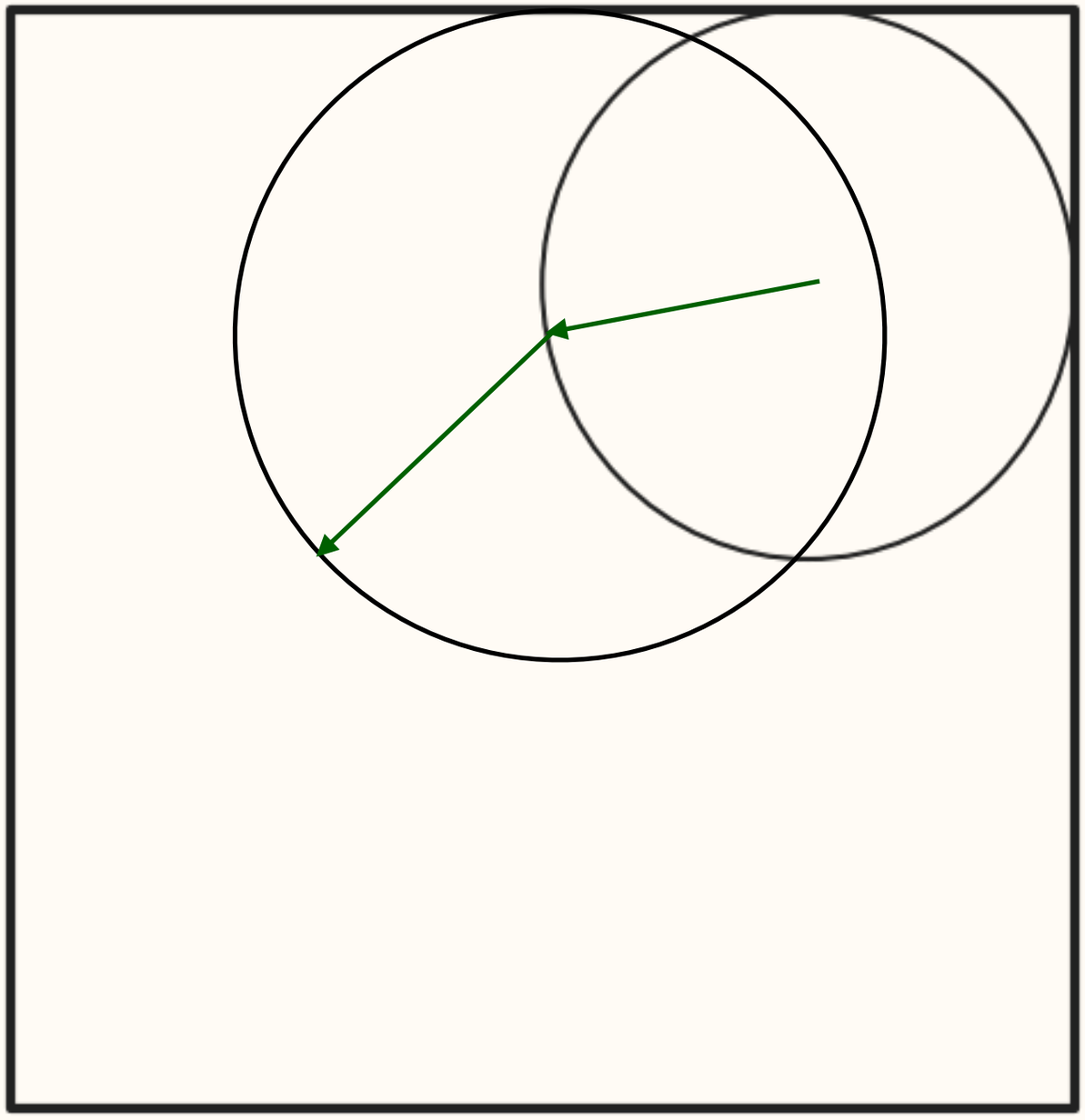
Freeing trapped photons



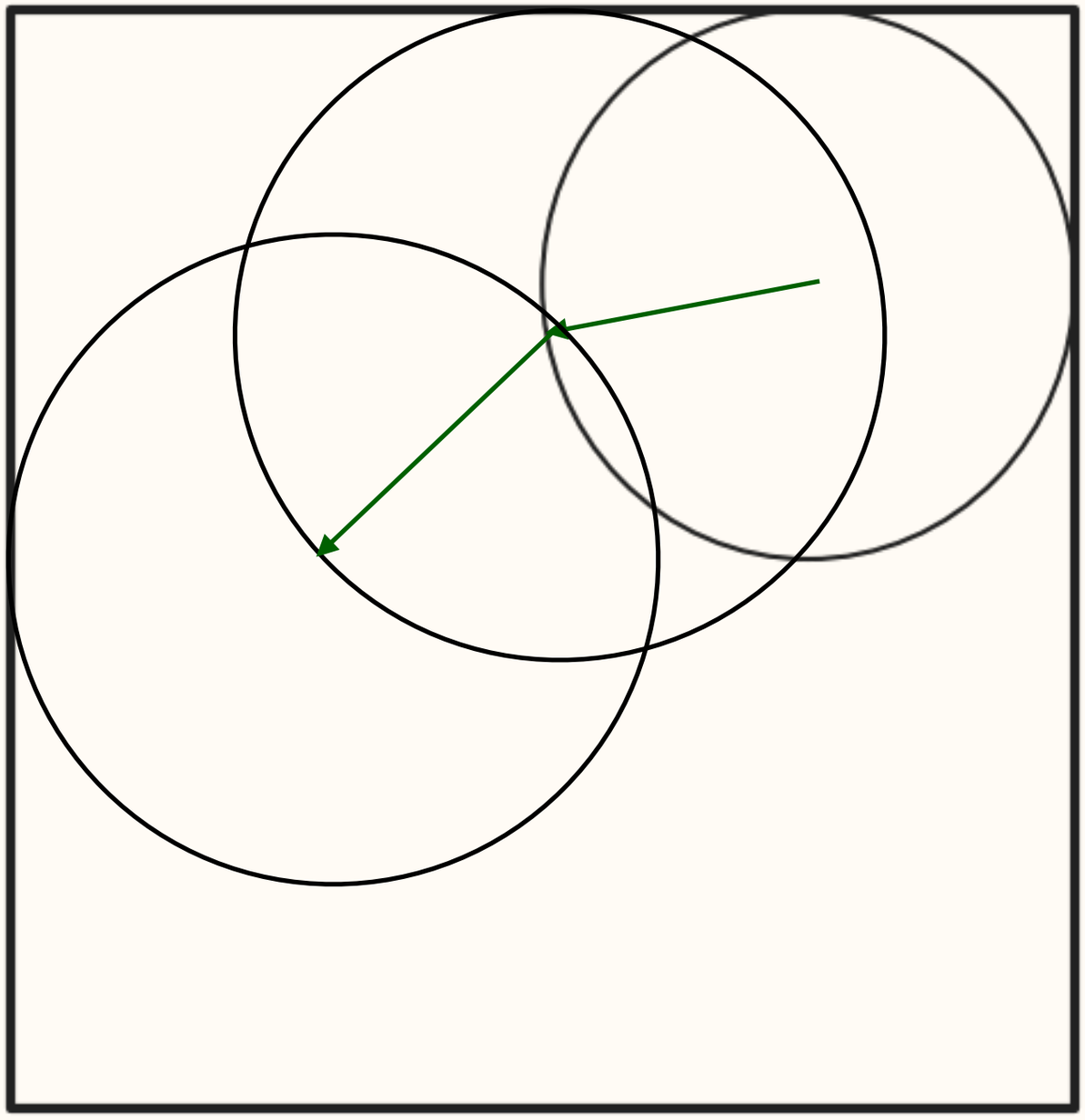
Freeing trapped photons



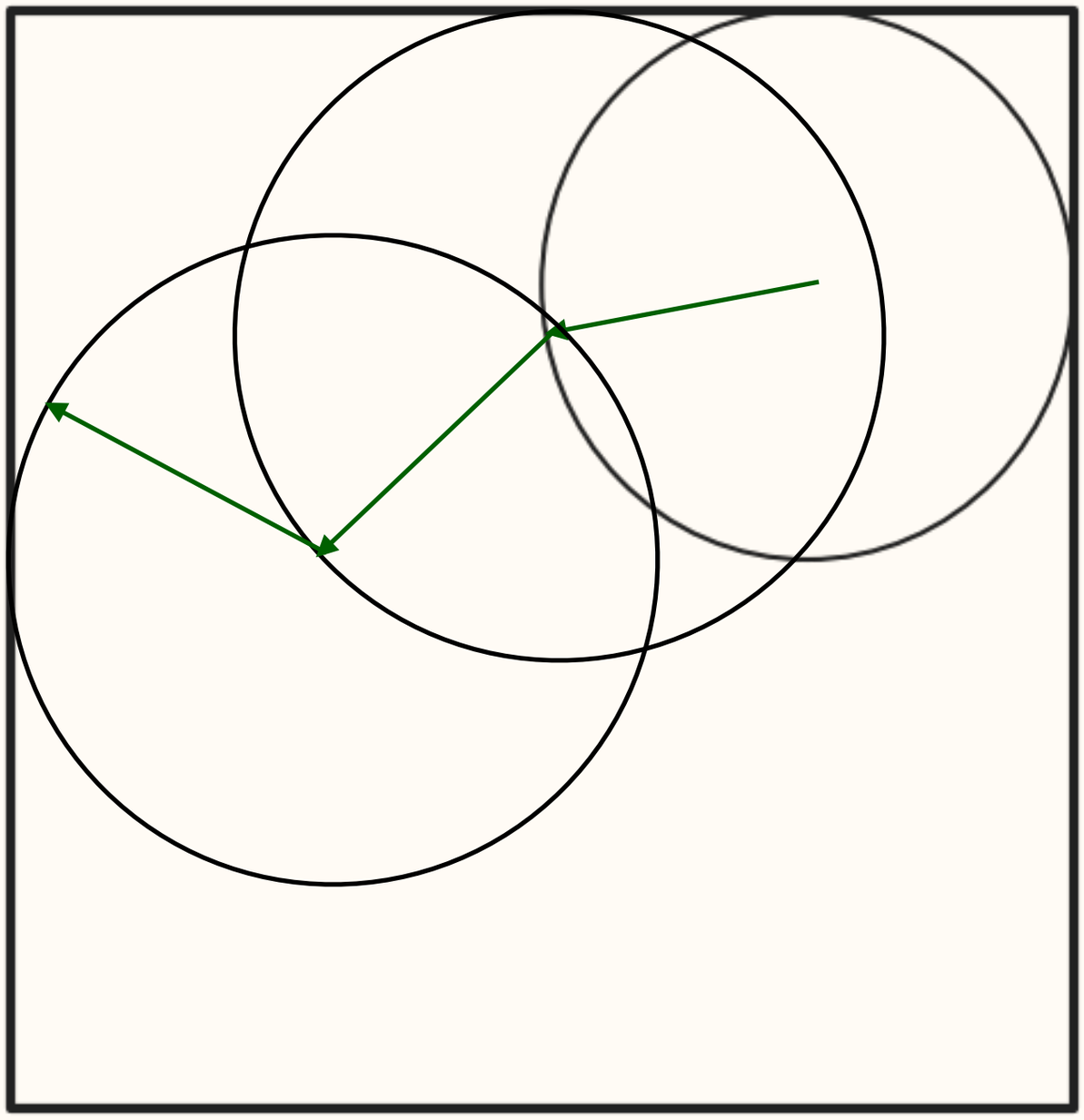
Freeing trapped photons



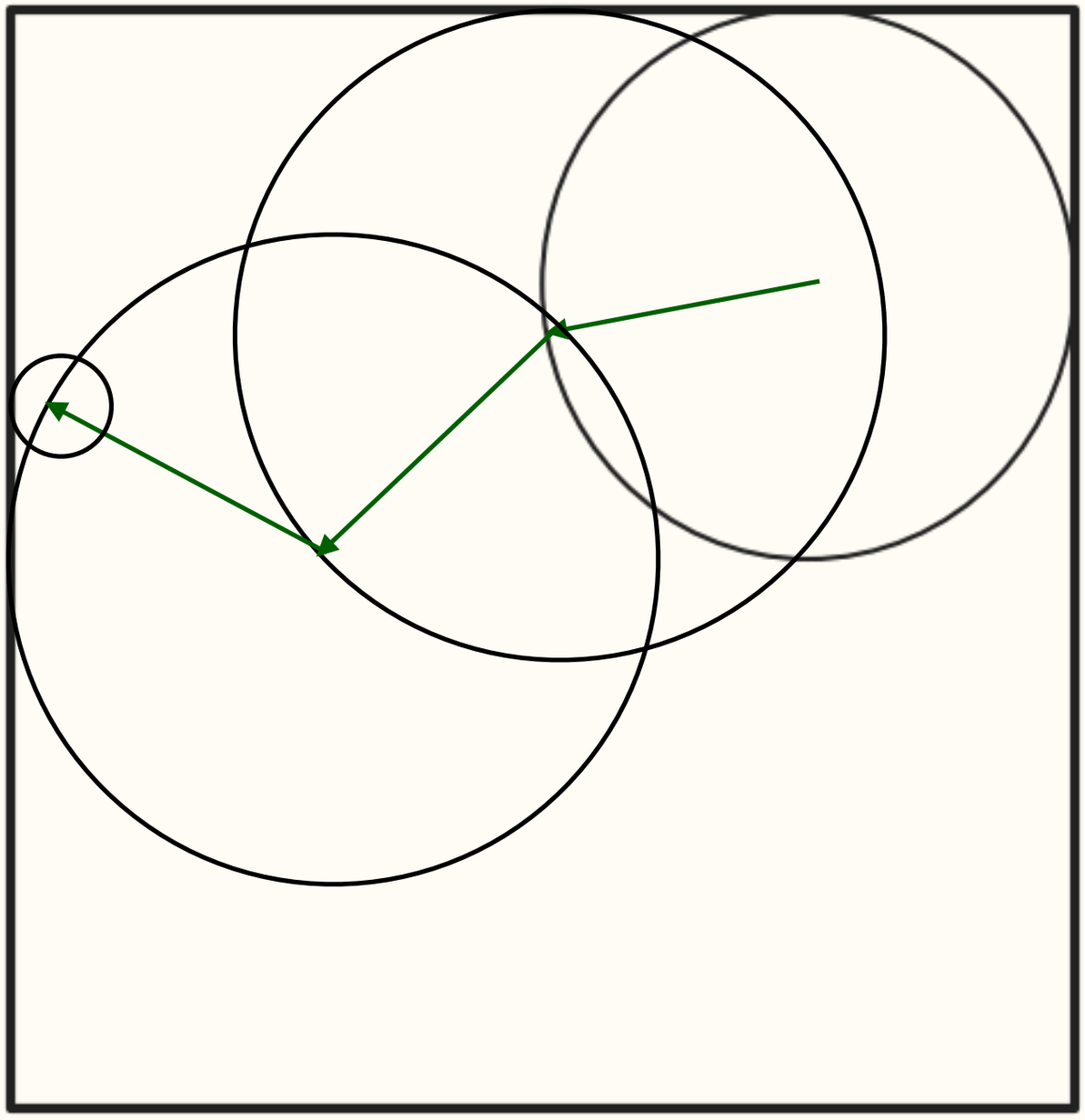
Freeing trapped photons



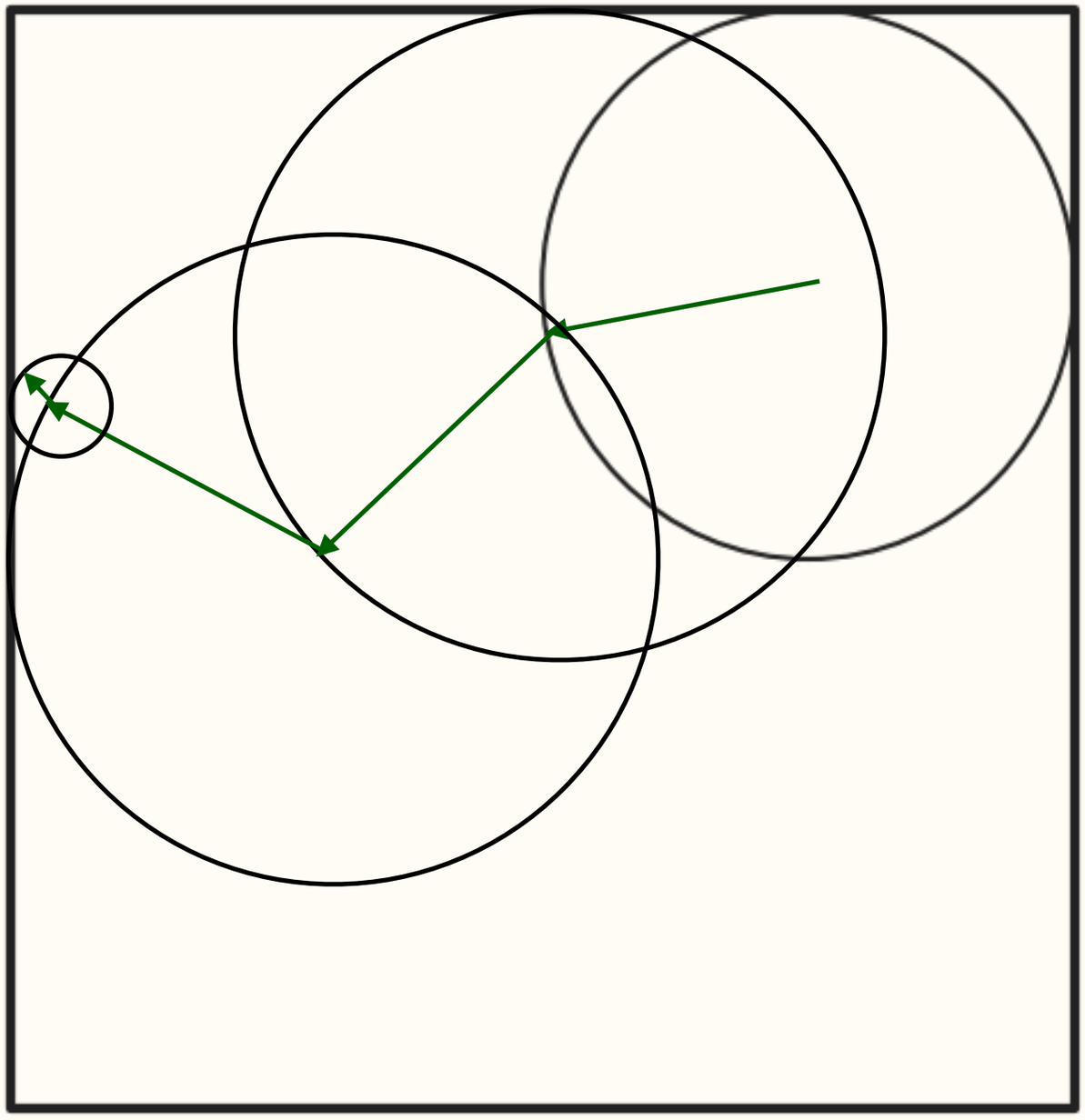
Freeing trapped photons



Freeing trapped photons



Freeing trapped photons



Freeing trapped photons

Algorithm: When a photon is in a high density cell, we can set up a sphere within the cell and use diffusion theory to move the photon to the surface of the sphere in a single step. Thus, millions of interactions can be replaced by a single step. This is the **modified random walk**.

See Min et al. (2009, A&A 497, 155)
Robitaille (2010, A&A 520, 70)

Step 1 - determine distance to closest cell edge $d_{\min}$

Step 2 - determine whether to use the MRW:

$$d_{\min} > \frac{\gamma}{\rho \bar{\chi}_R}$$

i.e. should be more than a few optical depths away from cell edge.

Step 3 - determine distance travelled to escape the sphere:

$$\zeta = 2 \sum_{n=1}^{\infty} (-1)^{n+1} y^{n^2} \qquad ct = -\ln y \left(\frac{R_0}{\pi}\right)^2 \frac{1}{D}$$

Step 4 (if computing T) - determine energy absorbed by the dust:

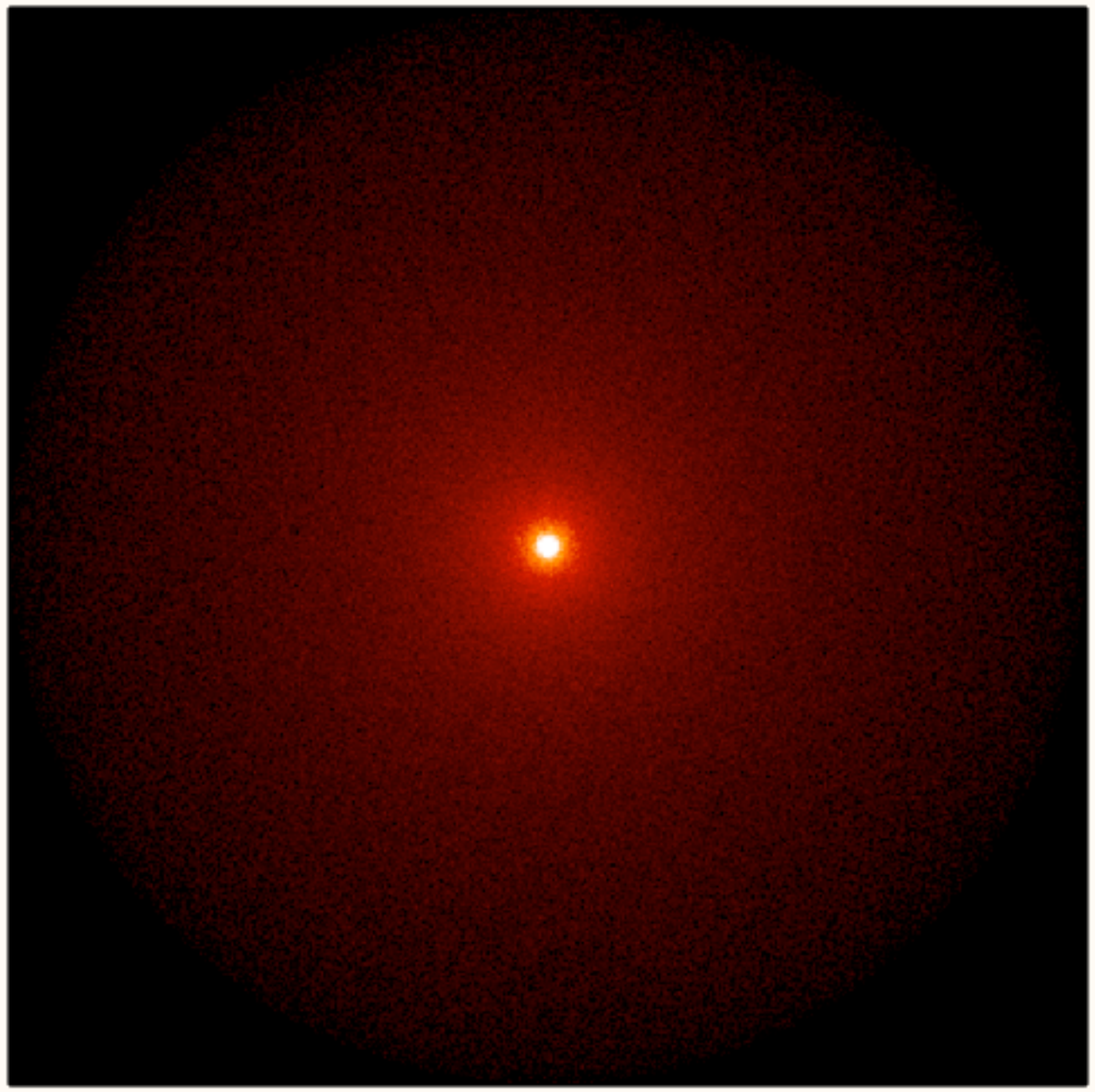
$$E = E_\gamma \, ct \, \rho \, \bar{\kappa_P}$$

Step 5 - emit photon from random position on sphere with a frequency sampled from a Planck function.

See Min et al. (2009, A&A 497, 155)
Robitaille (2010, A&A 520, 70)

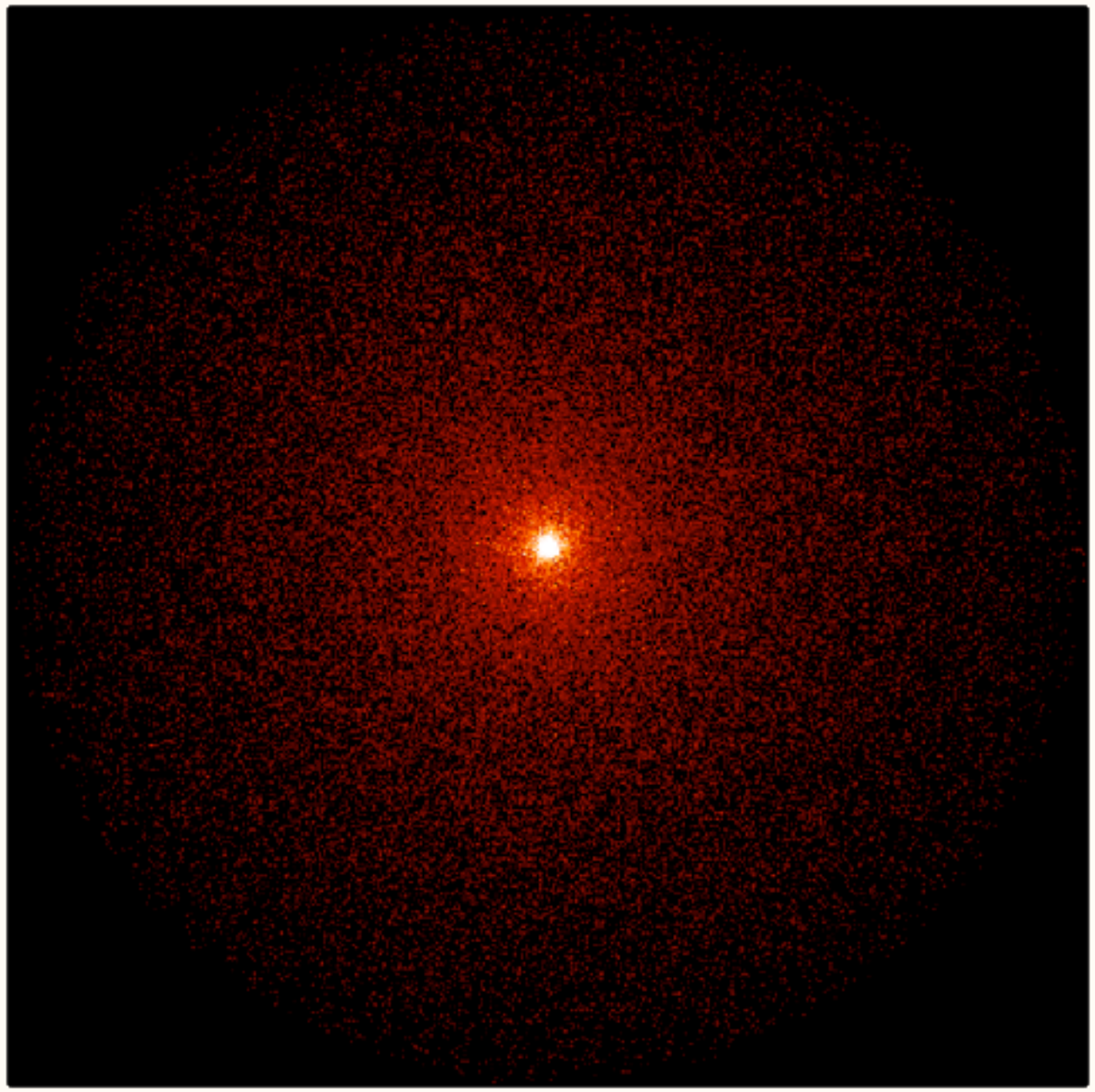
When the dust is **not dense enough**

Constant density sphere, central point source, 10^7 photons



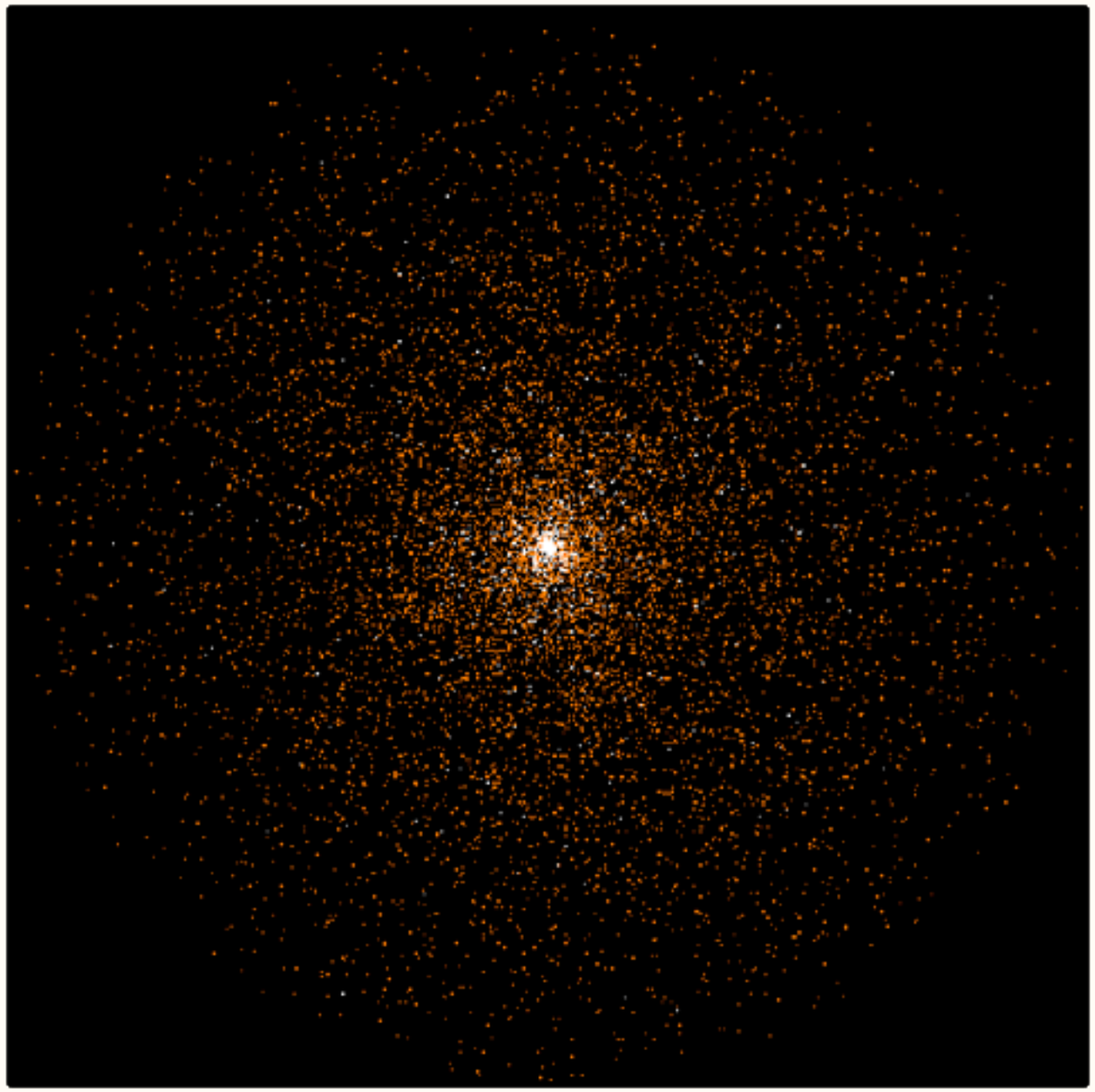
(lowering density by a factor of 10x each time)

Constant density sphere, central point source, 10^7 photons



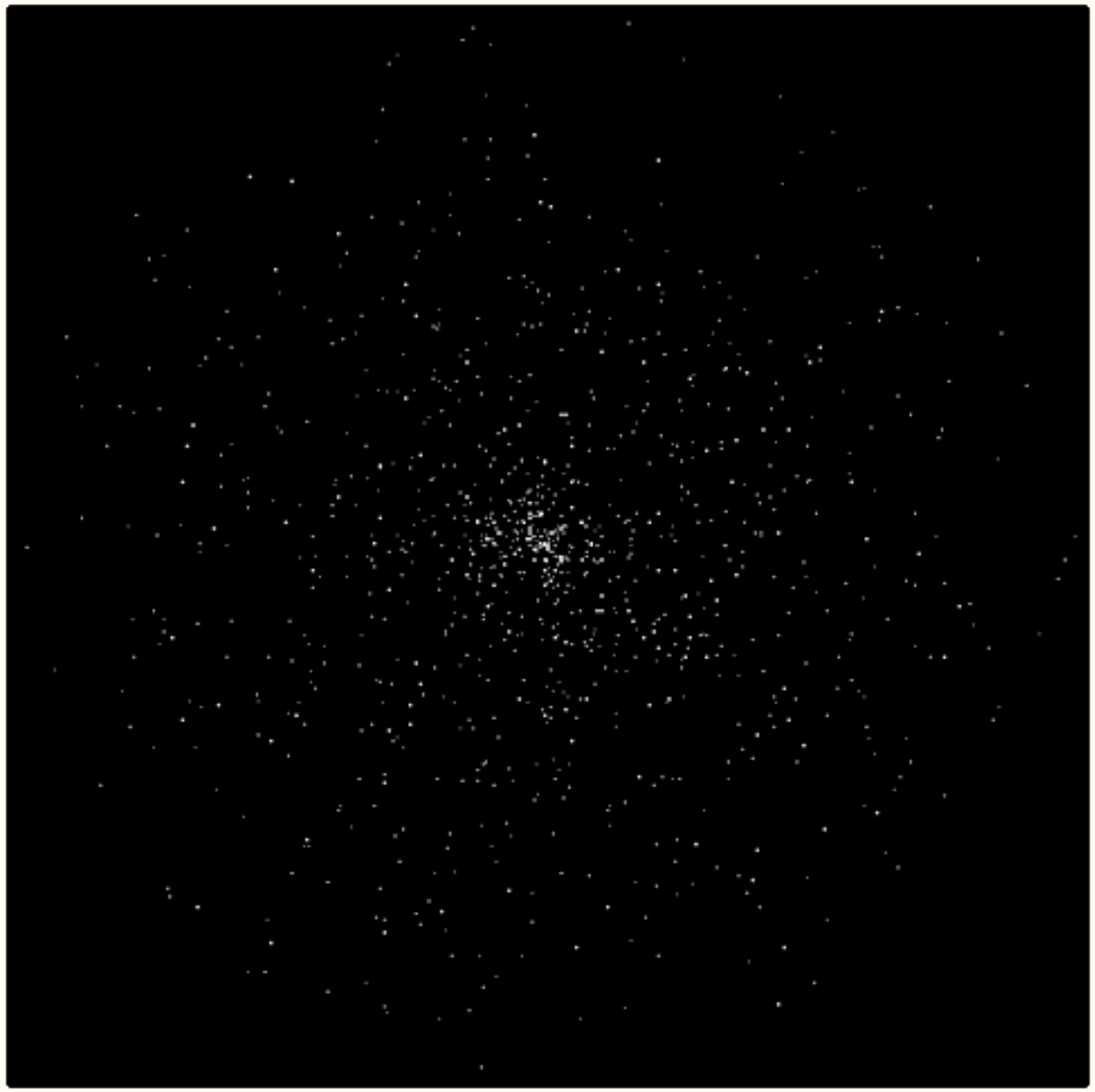
(lowering density by a factor of 10x each time)

Constant density sphere, central point source, 10^7 photons



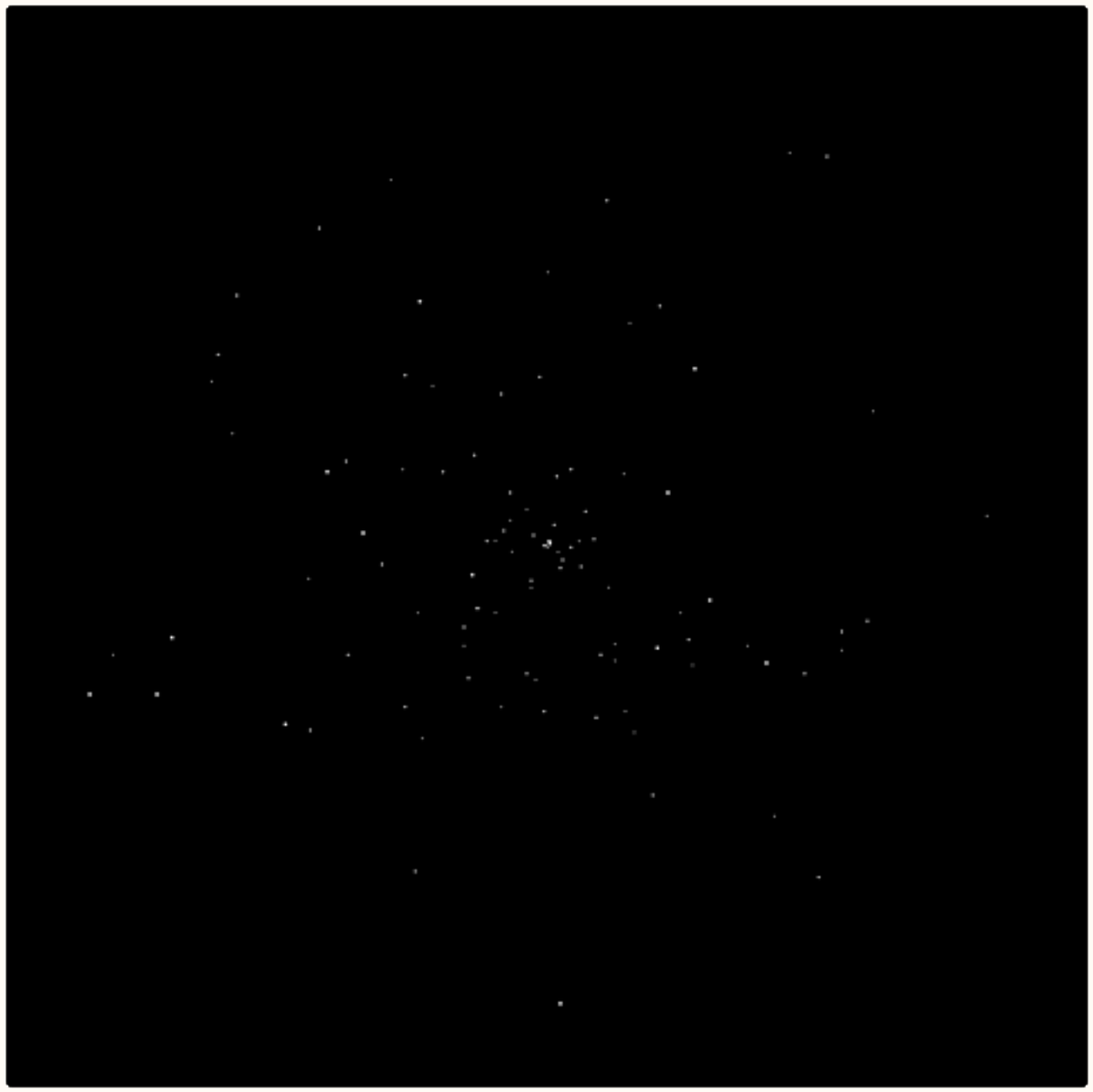
(lowering density by a factor of 10x each time)

Constant density sphere, central point source, 10^7 photons

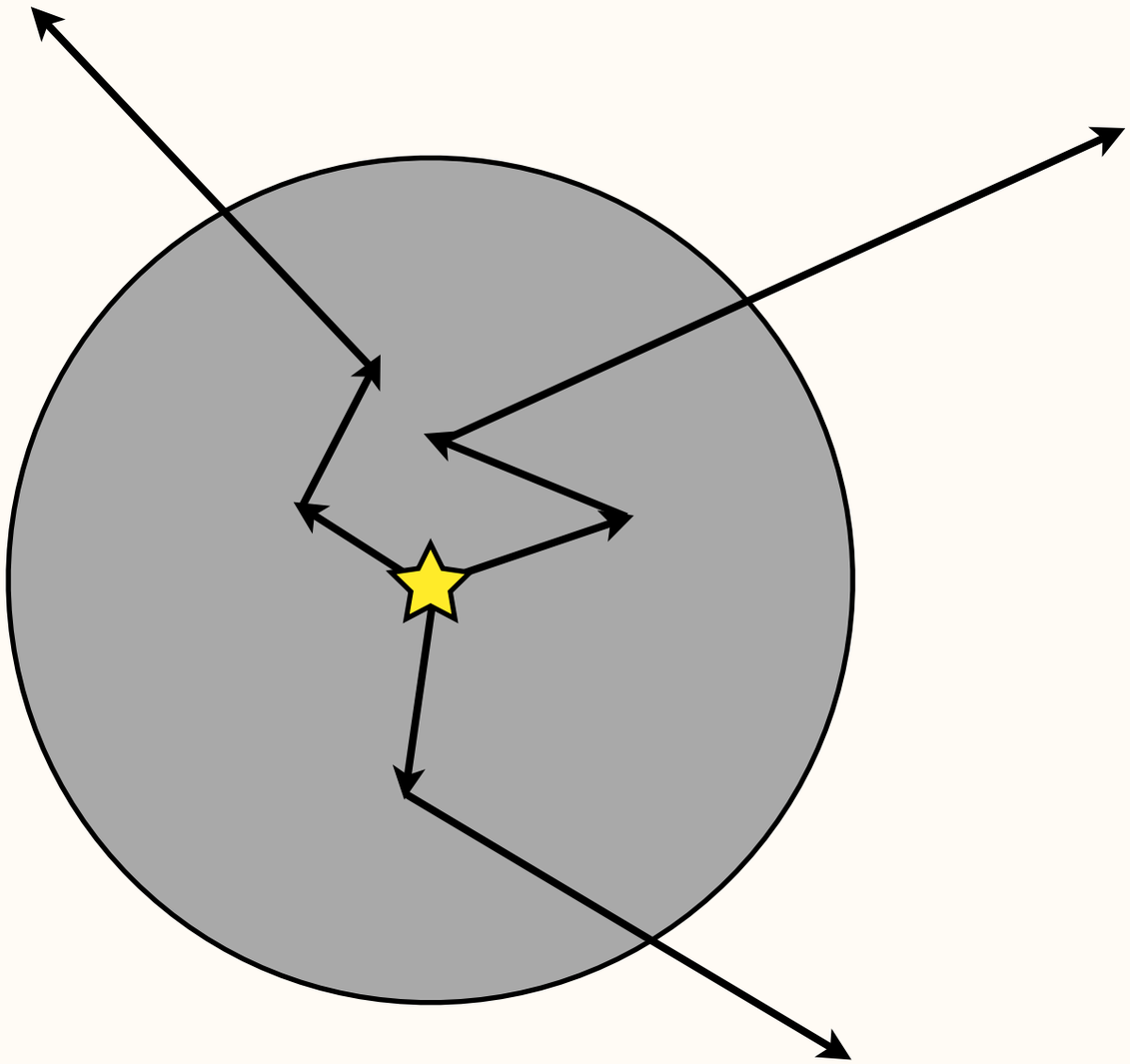


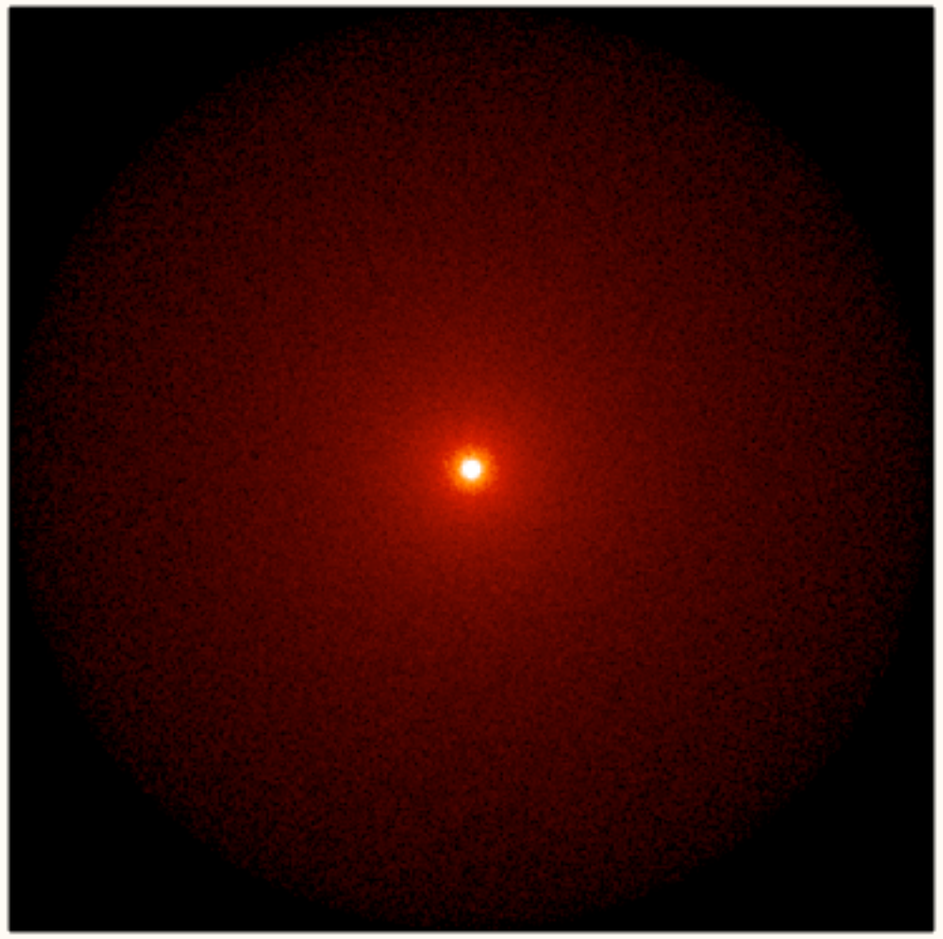
(lowering density by a factor of 10x each time)

Constant density sphere, central point source, 10^7 photons

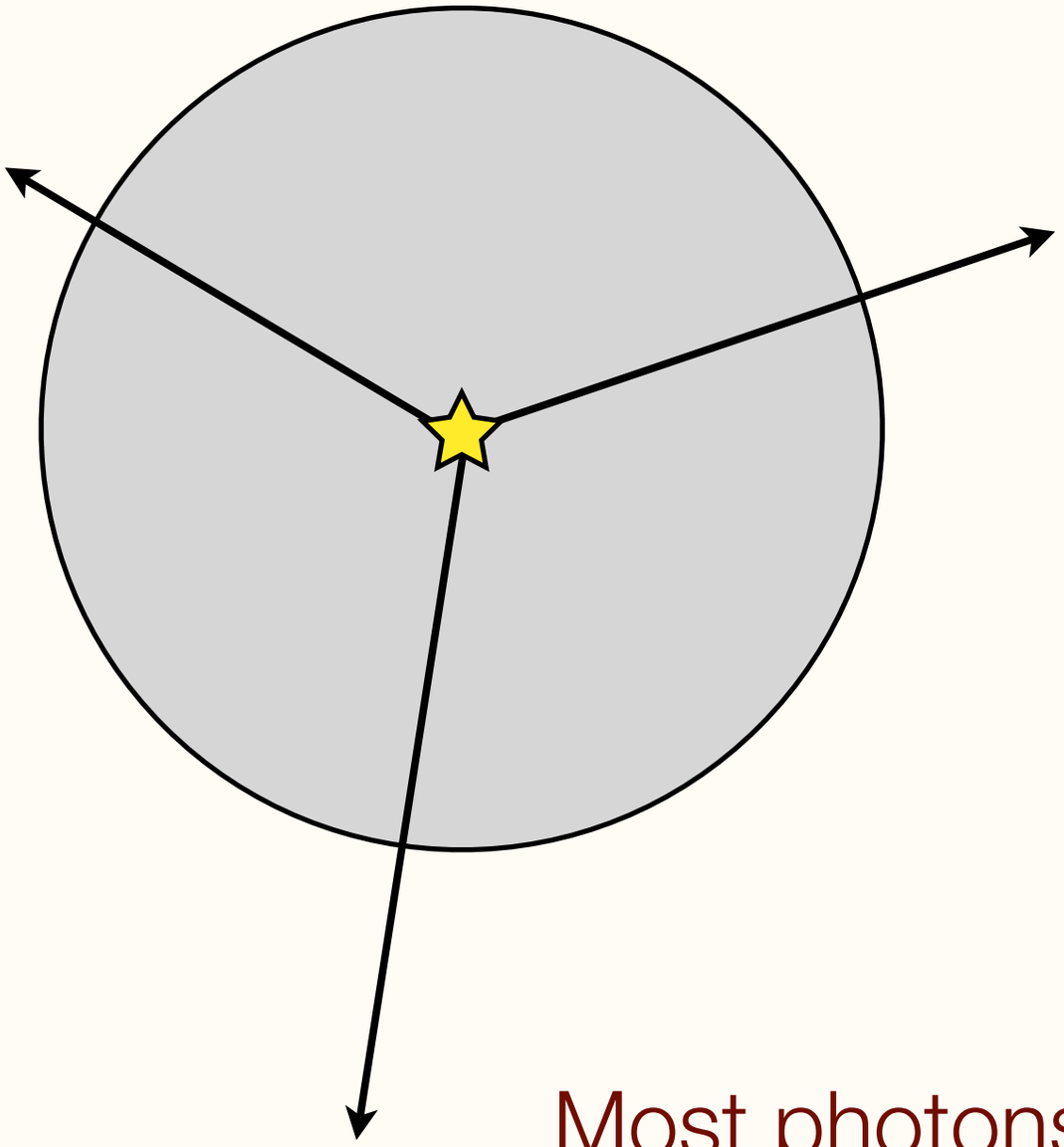


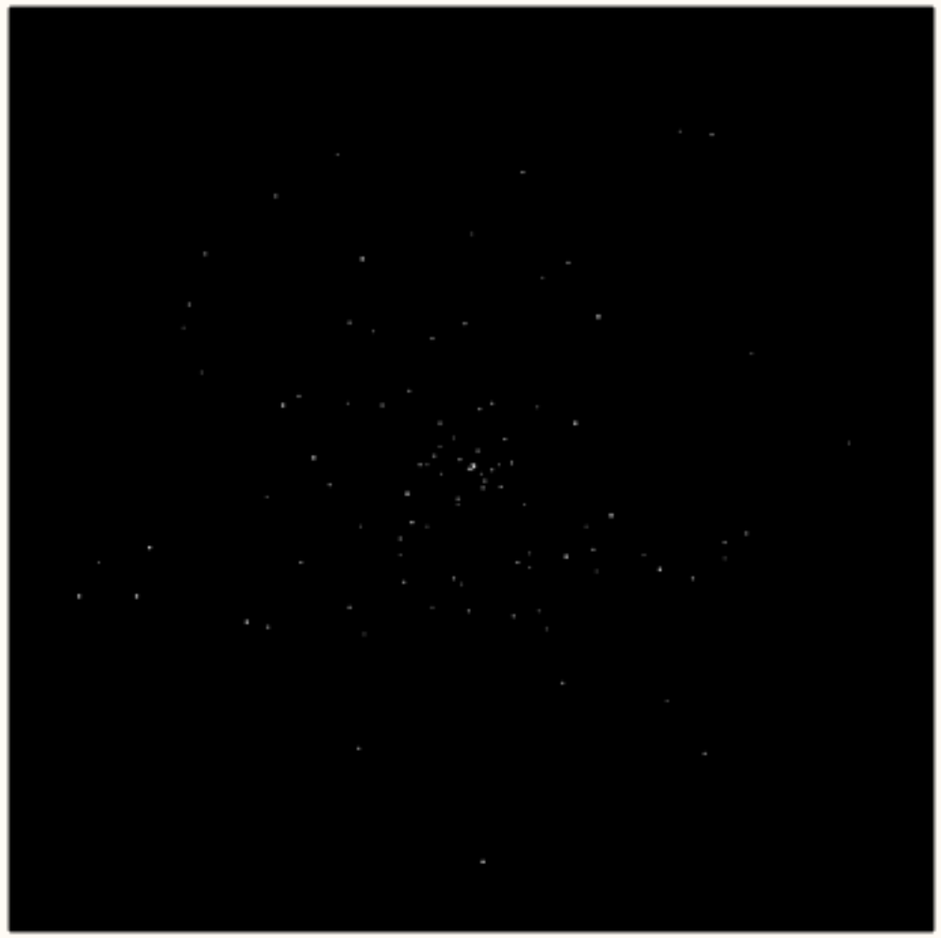
(lowering density by a factor of 10x each time)





Most photons scatter

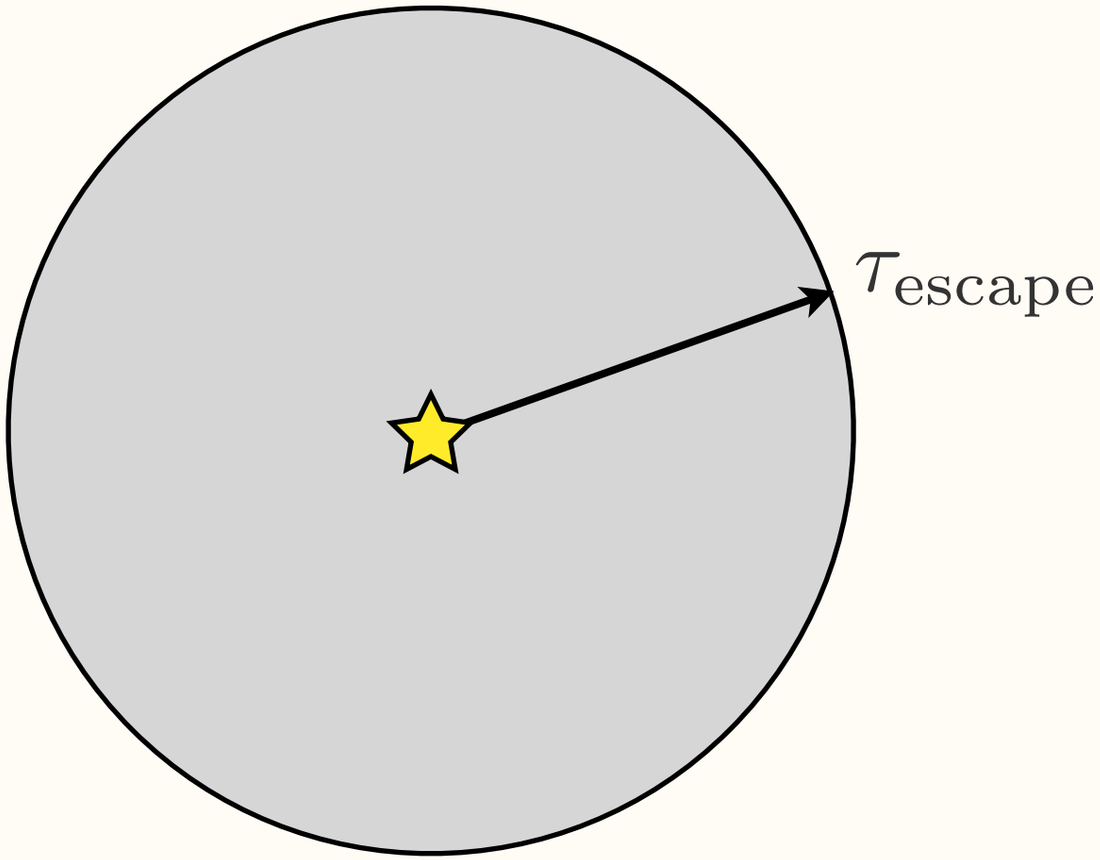




Most photons do not scatter

Algorithm: The first time a photon is emitted, we force it to scatter, and we weigh the energy of the photon according to the probability of scattering before escaping the grid. We call this **forced first scattering**

See Appendix in Wood & Reynolds (1999, ApJ 525, 799)



1. Compute τ_{escape}
2. Sample τ between 0 and τ_{escape}
3. Weight photon energy by:

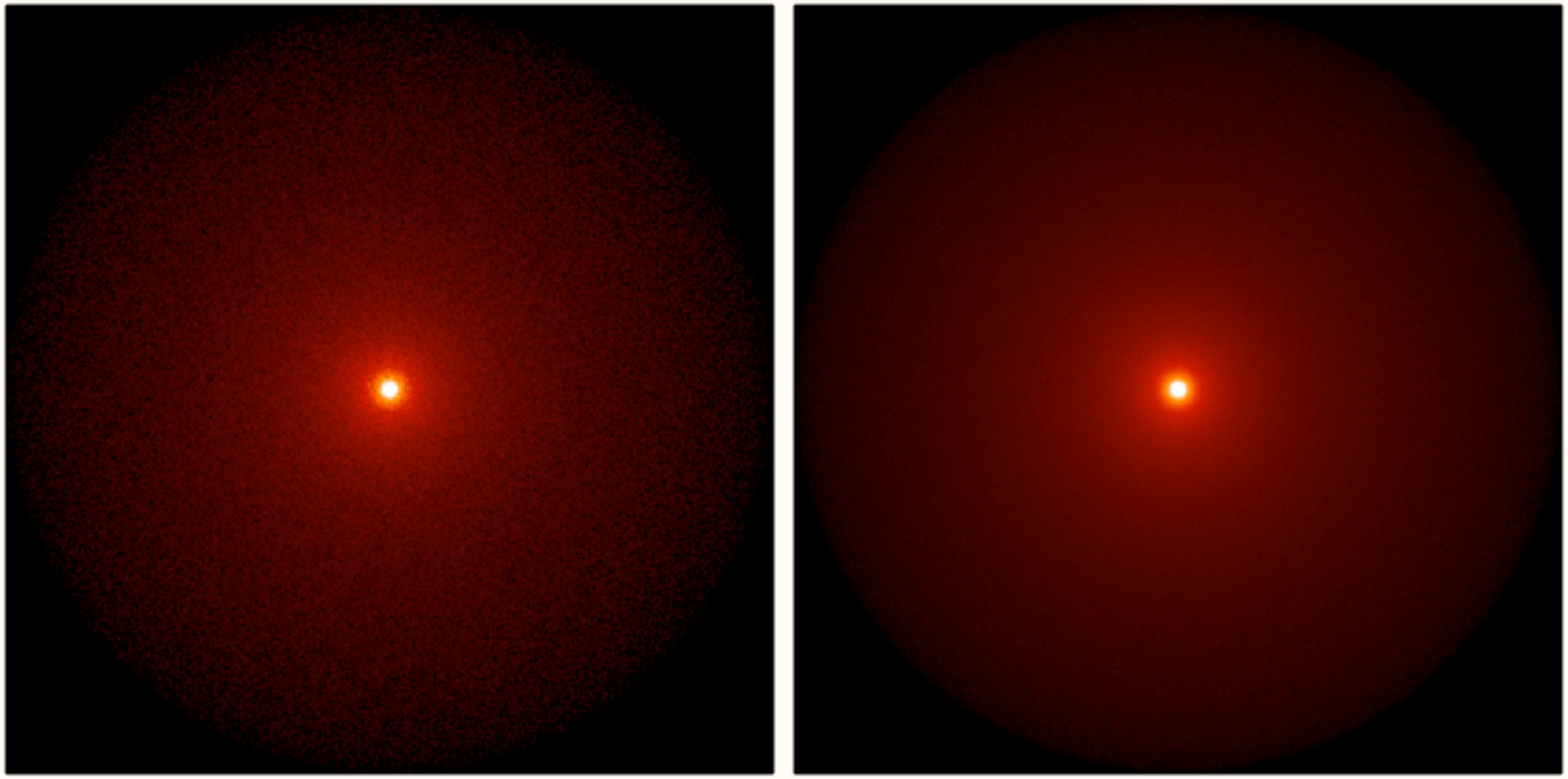
$$W = 1 - e^{-\tau_{\mathrm{escape}}}$$

4. Travel τ then scatter

All photons scatter!

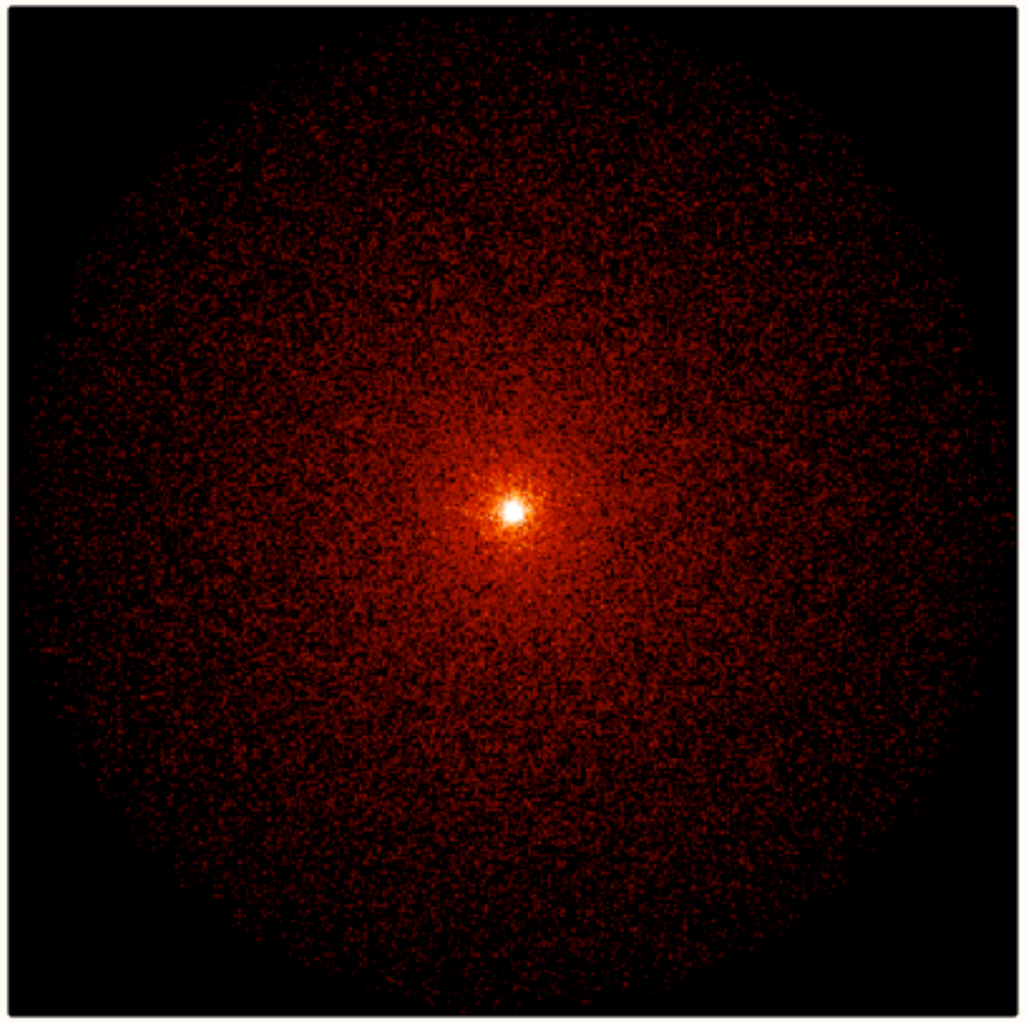
Regular MC

Force First Scattering

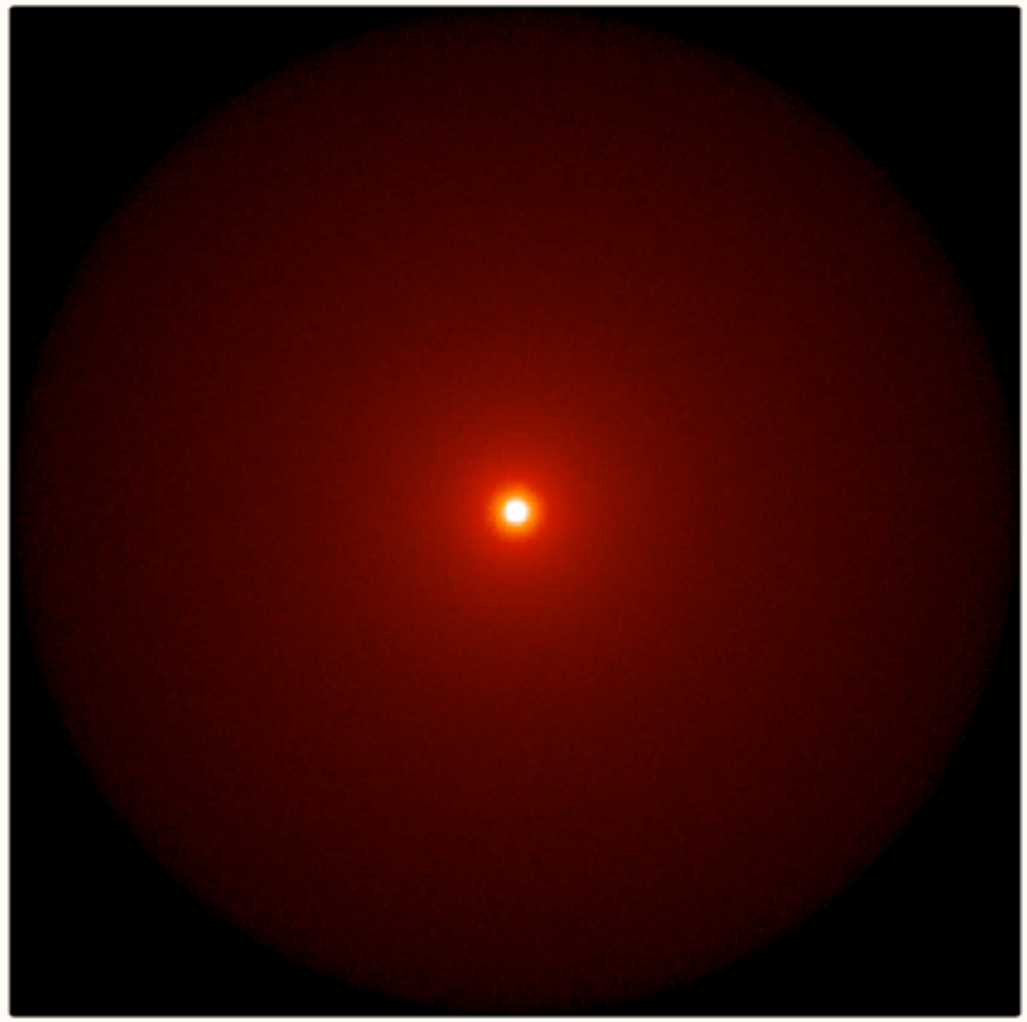


Constant density sphere, central point source, 10^7 photons
(lowering density by a factor of 10x each time)

Regular MC

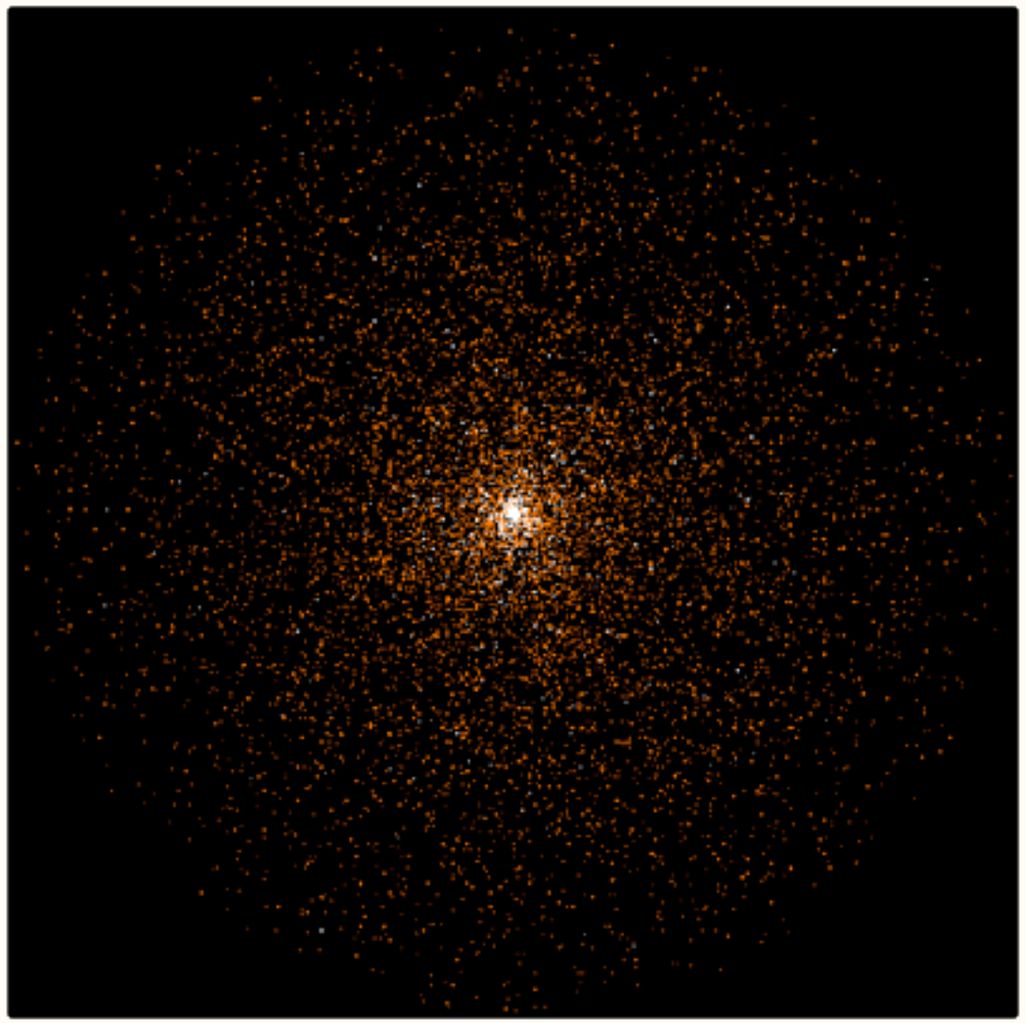


Force First Scattering

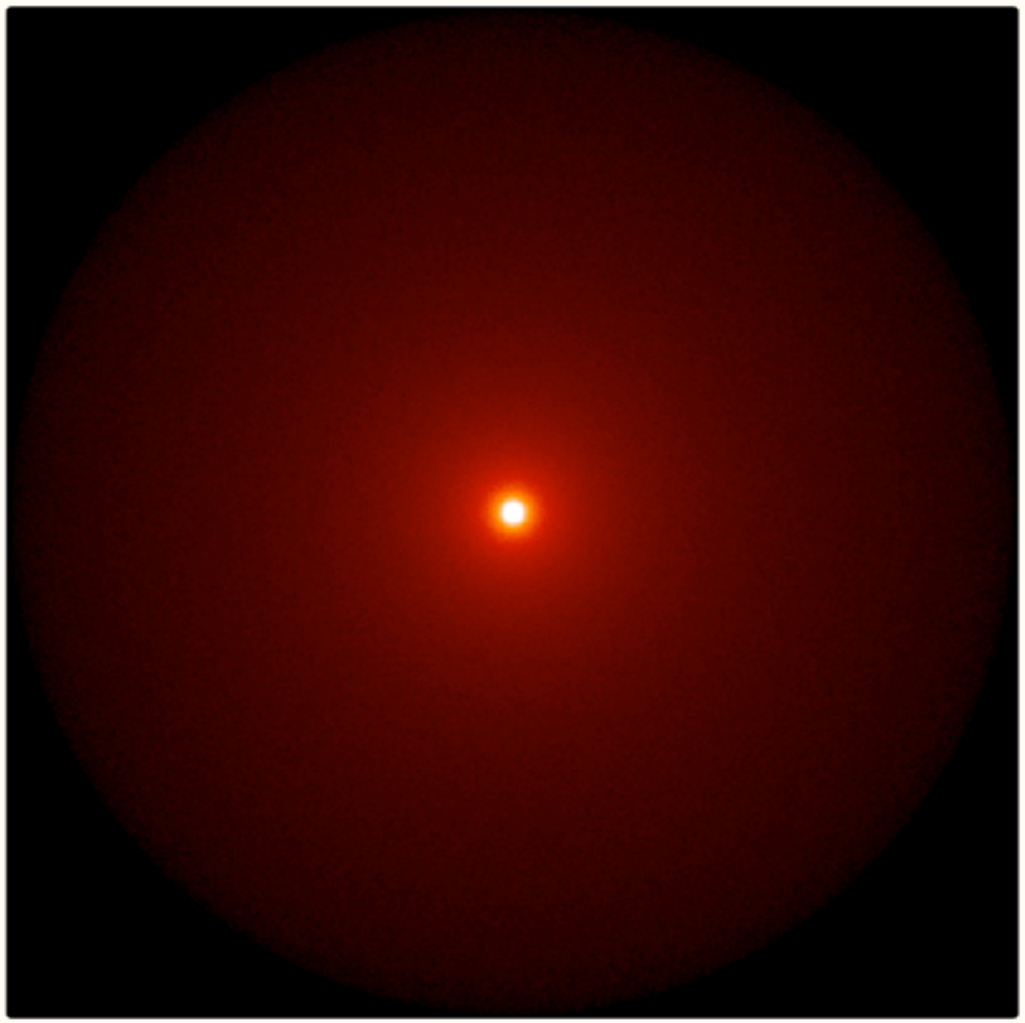


Constant density sphere, central point source, 10^7 photons
(lowering density by a factor of 10x each time)

Regular MC



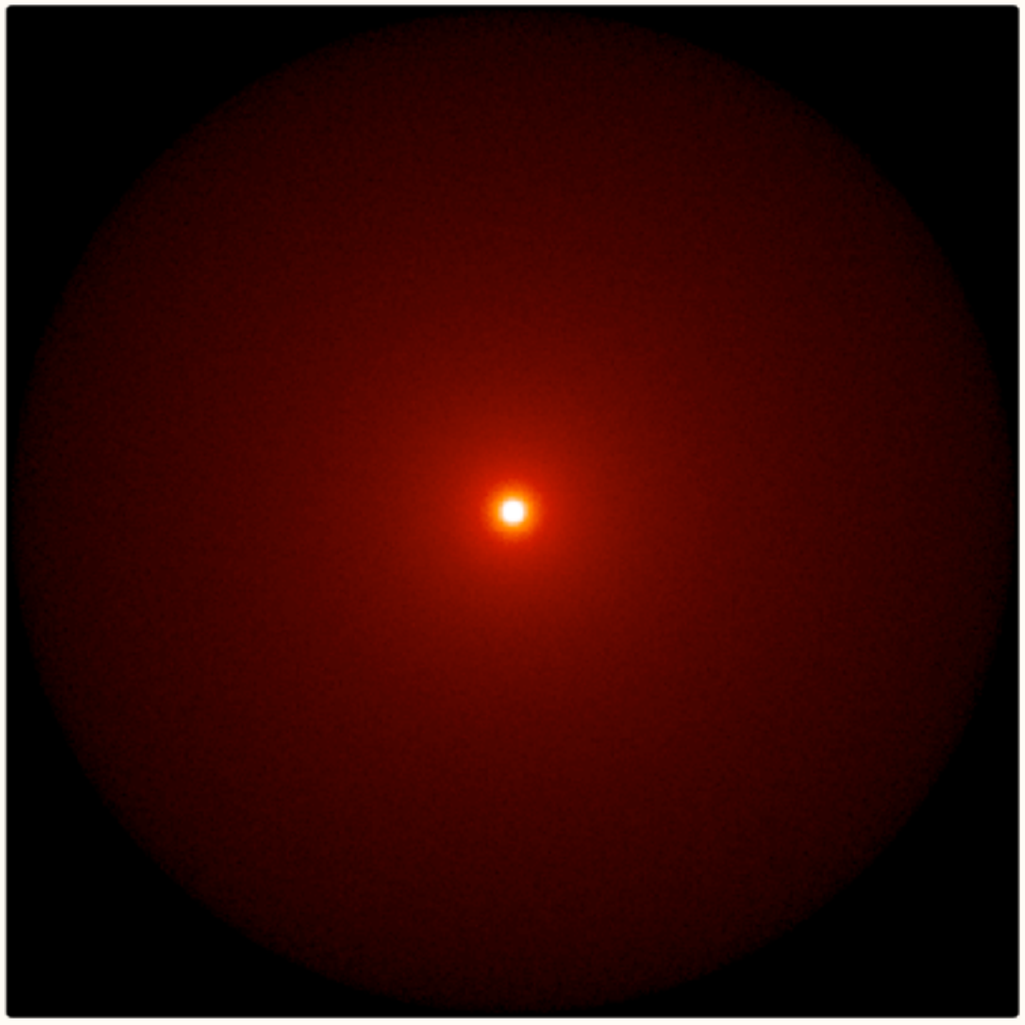
Force First Scattering



Constant density sphere, central point source, 10^7 photons
(lowering density by a factor of 10x each time)

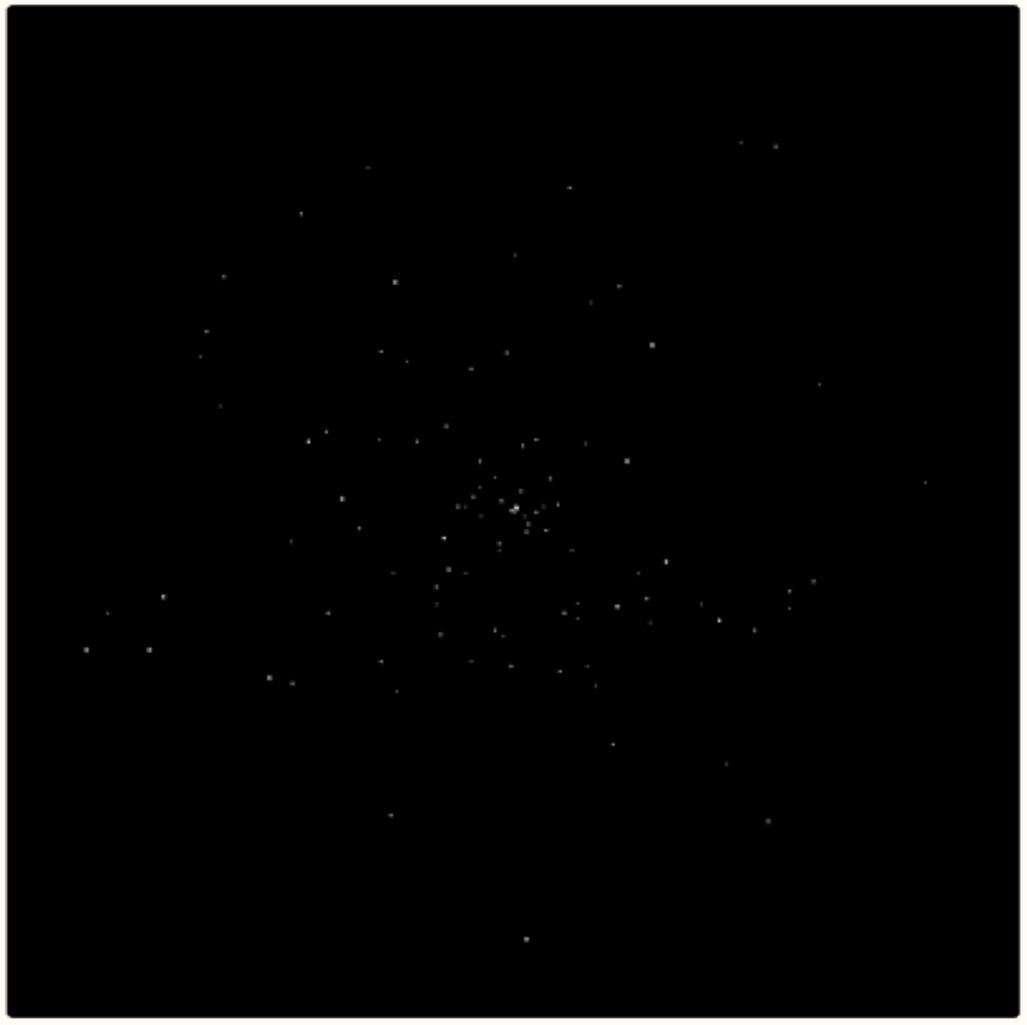
Regular MC

Force First Scattering

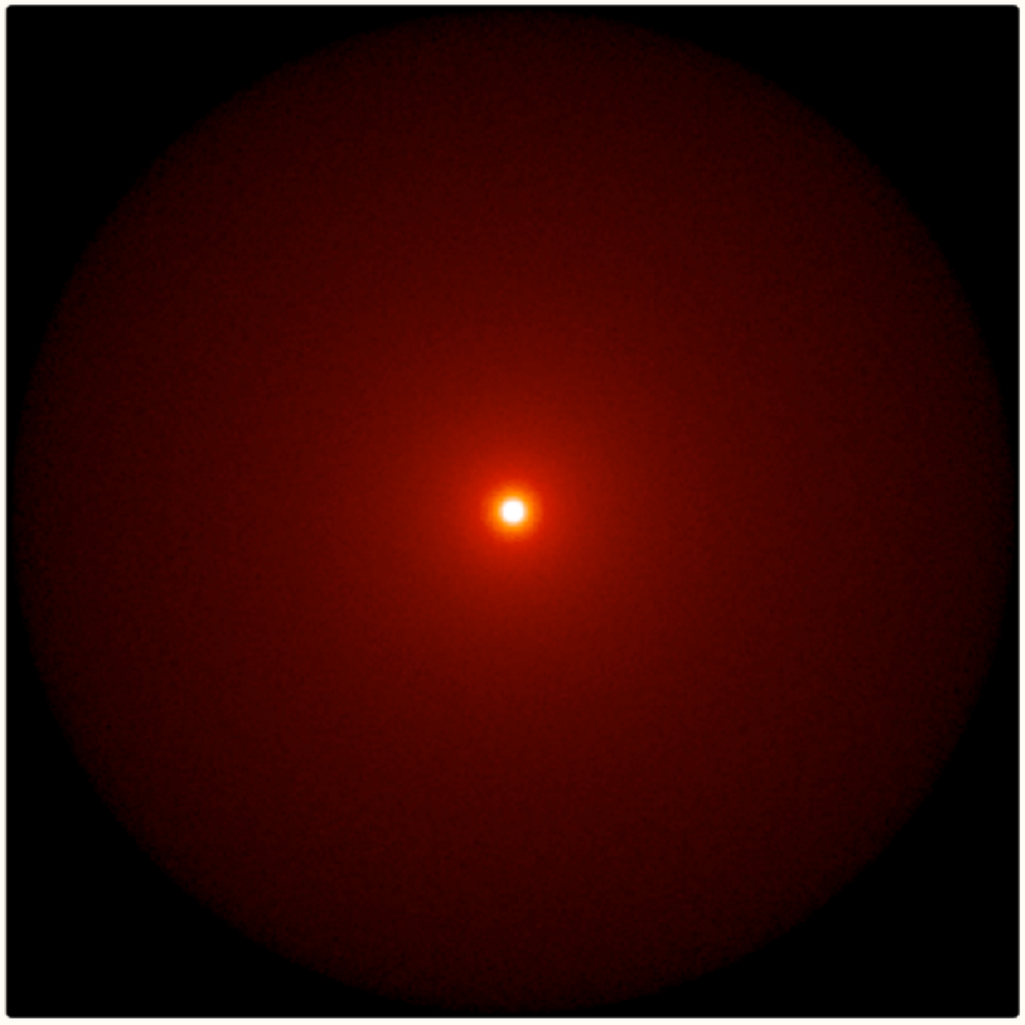


Constant density sphere, central point source, 10^7 photons
(lowering density by a factor of 10x each time)

Regular MC



Force First Scattering



Constant density sphere, central point source, 10^7 photons
(lowering density by a factor of 10x each time)

When the dust is **very dense**

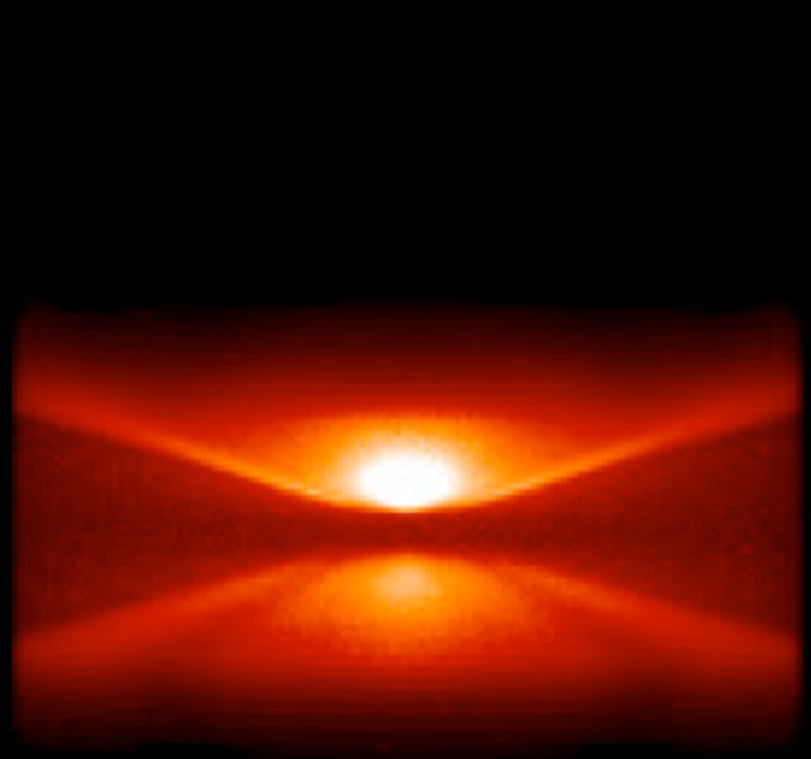
Use diffusion approximations (e.g. modified random walk and partial diffusion approximation)

When the dust is **not dense enough**

Use forced first scattering

HYPERION: 3d dust continuum radiative transfer

Thomas Robitaille (Max Planck Institute for Astronomy)



Homepage: http://www.hyperion-rt.org

Documentation: http://docs.hyperion-rt.org

Code: https://github.com/hyperion-rt/hyperion

Current version: 0.9.6 (February 27th 2015)

License: BSD (open source)

First released: July 2012

Main Characteristics:

Dust continuum radiative transfer (for now)

Lucy (1999) dust temperature calculation

SEDs, images, and polarization maps

Arbitrary 3-d geometry

Multiple sources and dust populations, and arbitrary dust properties

Cartesian, Polar, and Adaptive grids

High performance parallelized (MPI) Fortran 2003 core

Human-readable Python code to set up, run, and post-process models

Extensive test suite to ensure stability

Built-in optimizations:

Modified Random Walk

Peeling-off for SEDs and images

Raytracing for source and thermal dust emission

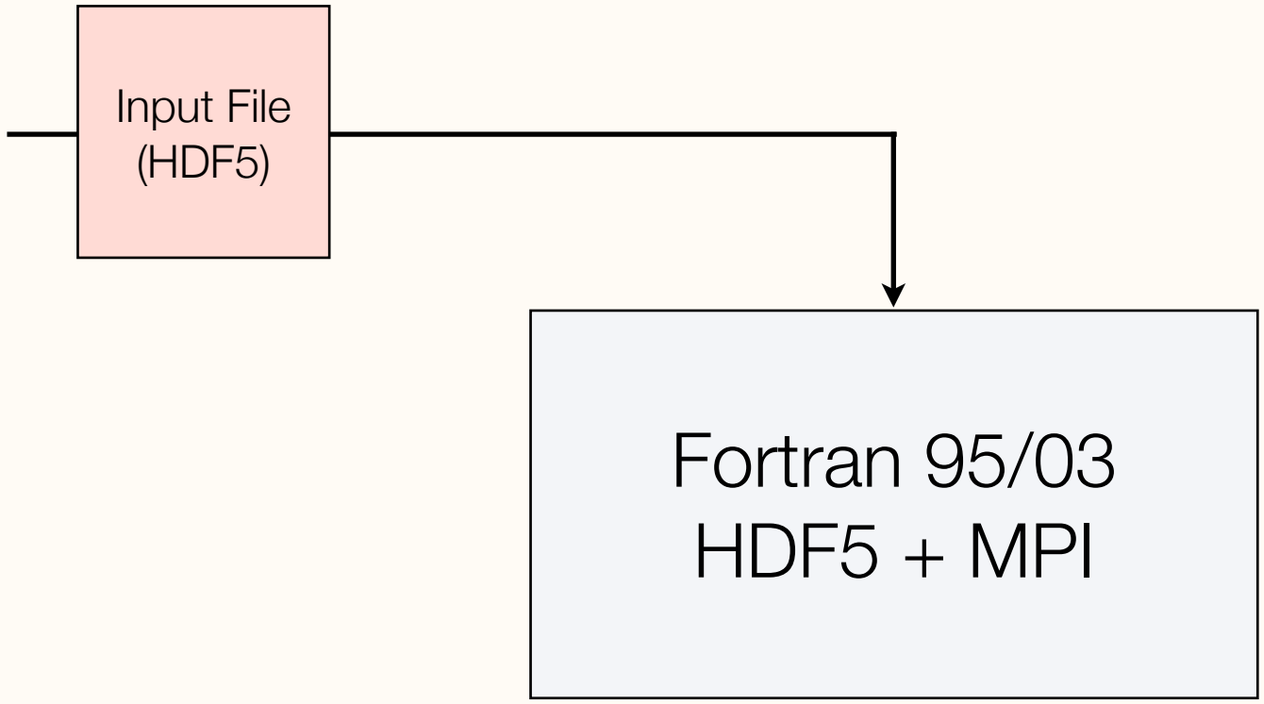
Monochromatic radiative transfer

Partial Diffusion Approximation

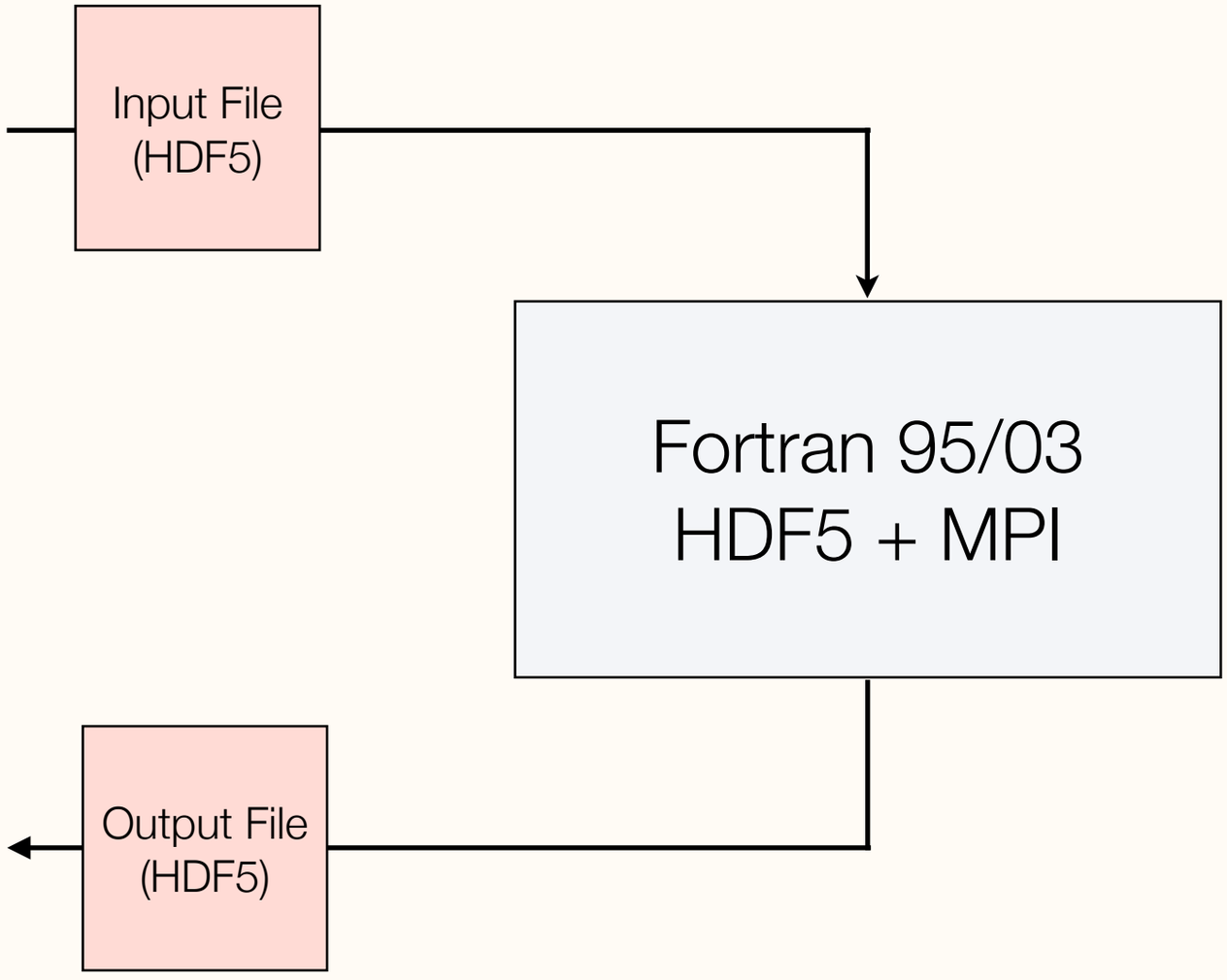
Forced first scattering (enabled by default)

Fortran 95/03
HDF5 + MPI

(Geometry, Densities, Sources,
Dust properties, Parameters)



(Geometry, Densities, Sources,
Dust properties, Parameters)



(Input, specific energy/temperature,
SEDs, images, statistics)

(Geometry, Densities, Sources,
Dust properties, Parameters)

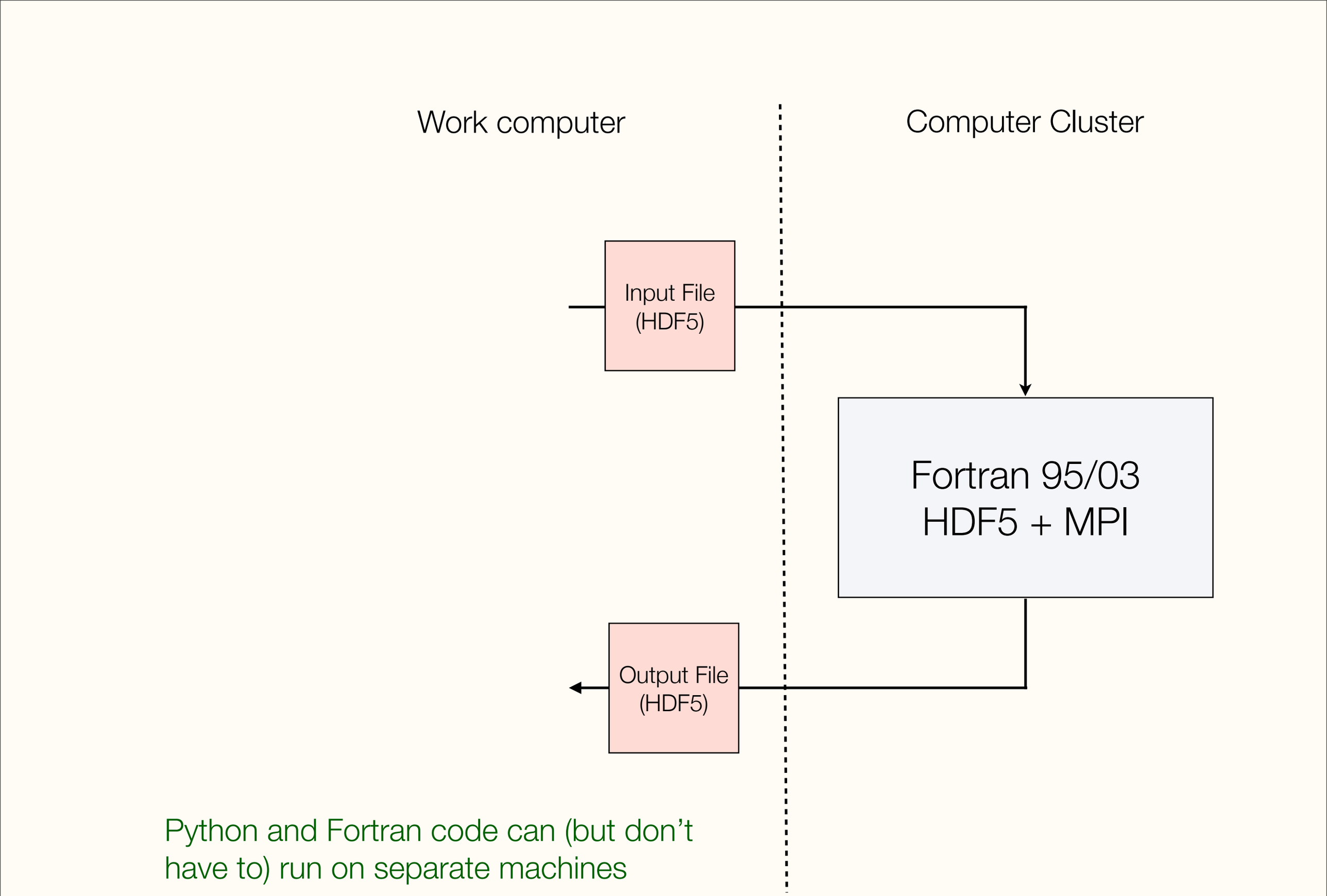
Input File
(HDF5)

python™

Fortran 95/03
HDF5 + MPI

Output File
(HDF5)

(Input, specific energy/temperature,
SEDs, images, statistics)



Python and Fortran code can (but don't have to) run on separate machines

Work computer

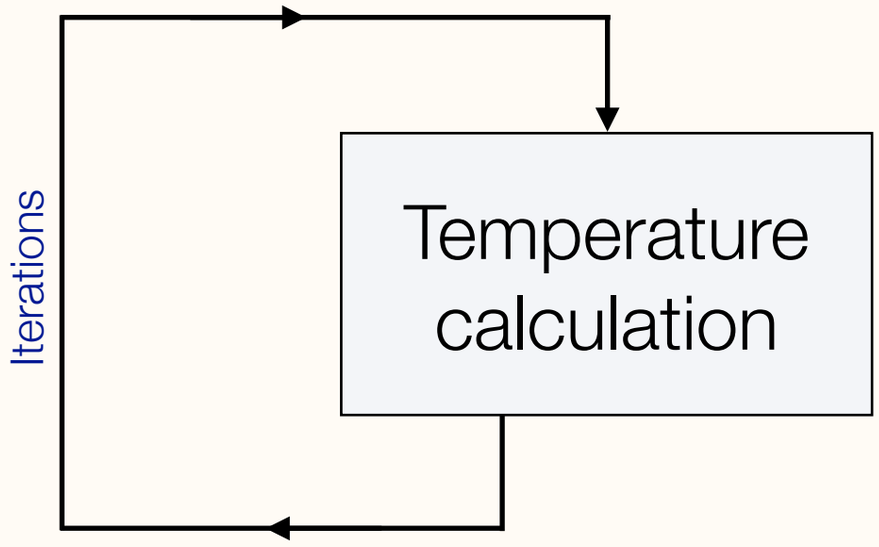
Computer Cluster

Input File
(HDF5)

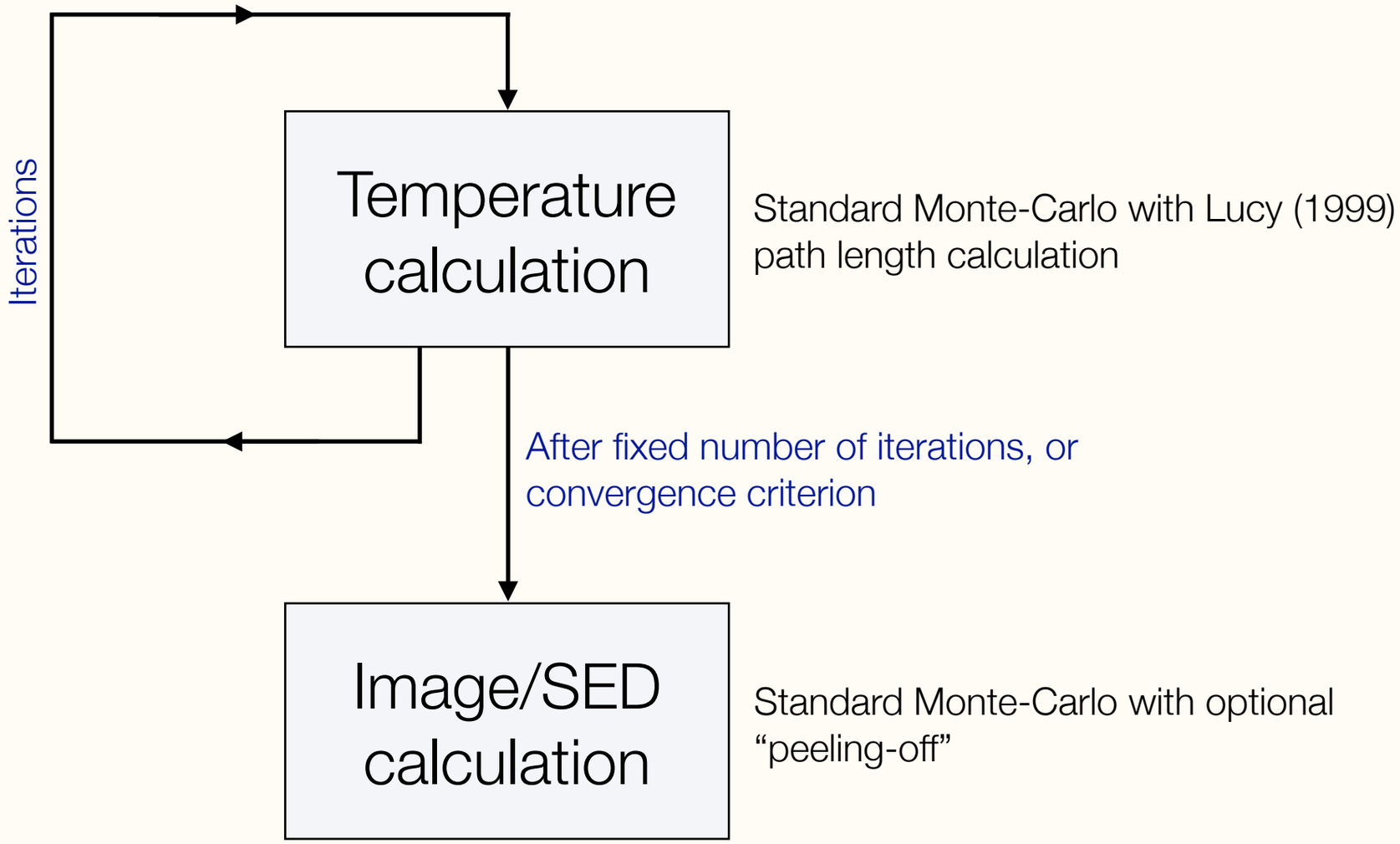
Fortran 95/03
HDF5 + MPI

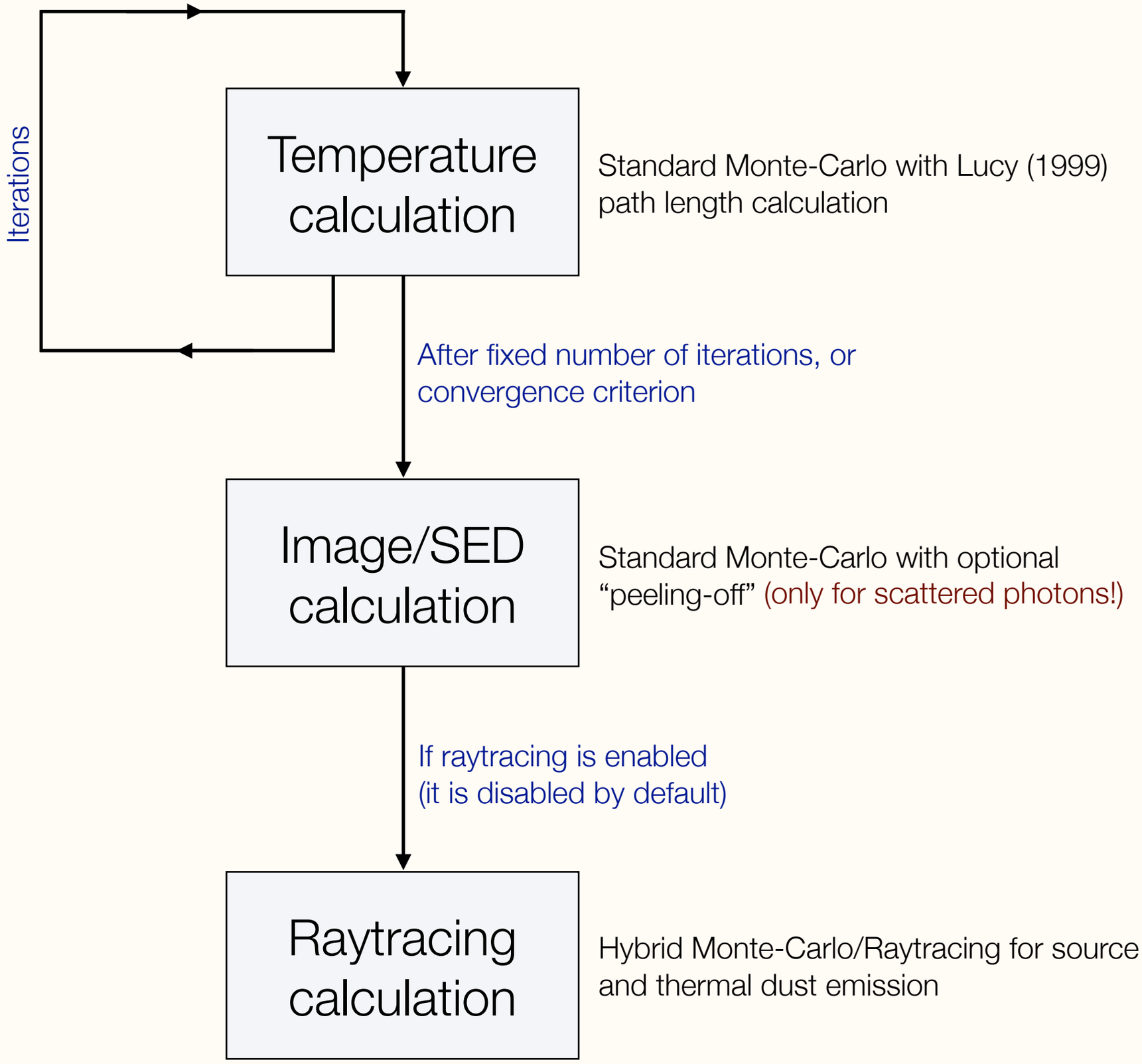
Output File
(HDF5)

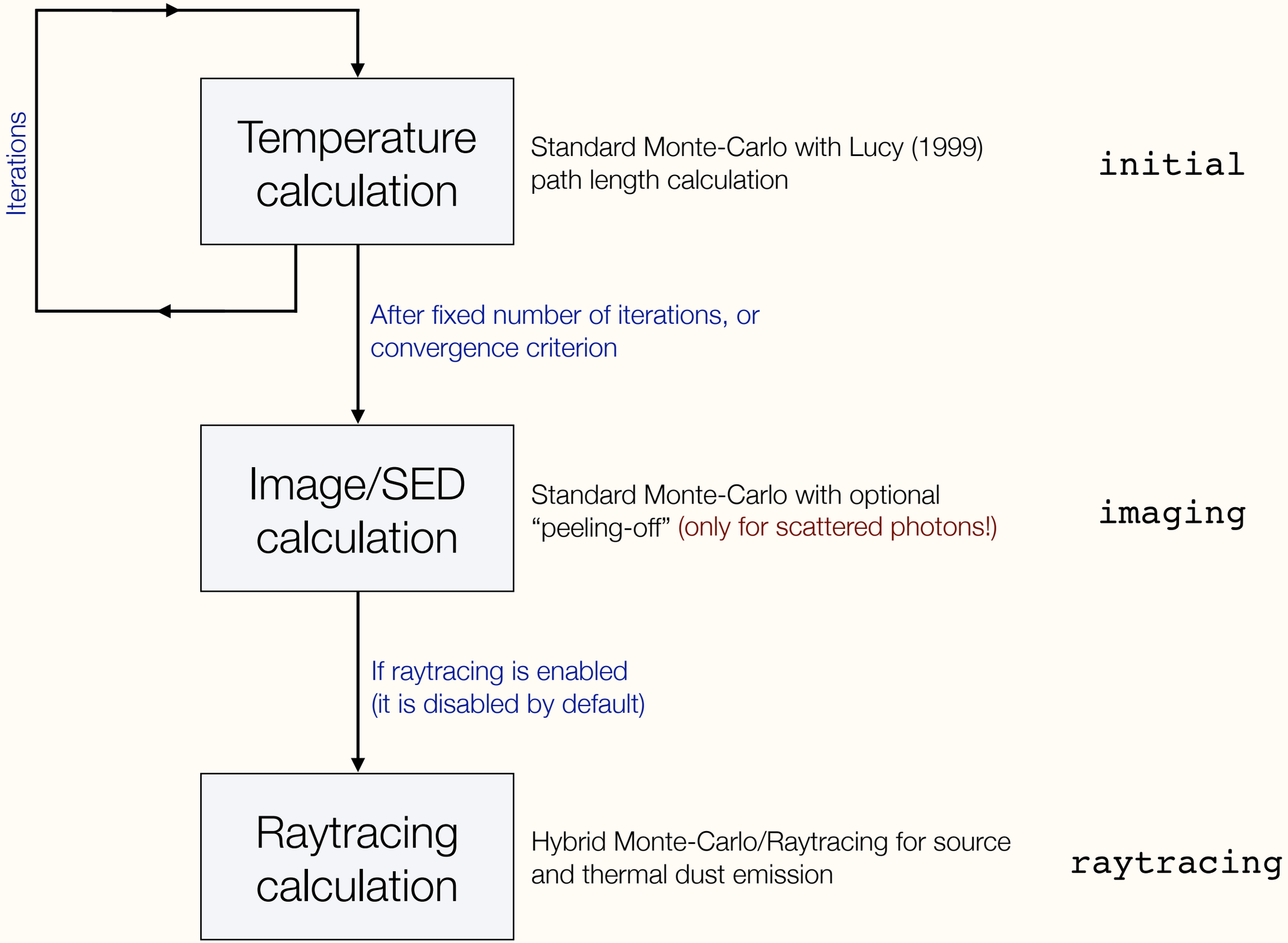
Python and Fortran code can (but don't have to) run on separate machines



Standard Monte-Carlo with Lucy (1999) path length calculation


Iterations
Temperature calculation
Standard Monte-Carlo with Lucy (1999) path length calculation
After fixed number of iterations, or convergence criterion
Image/SED calculation
Standard Monte-Carlo with optional "peeling-off"


Iterations
Temperature calculation
Standard Monte-Carlo with Lucy (1999) path length calculation
After fixed number of iterations, or convergence criterion
Image/SED calculation
Standard Monte-Carlo with optional "peeling-off" (only for scattered photons!)
If raytracing is enabled (it is disabled by default)
Raytracing calculation
Hybrid Monte-Carlo/Raytracing for source and thermal dust emission


Iterations
Temperature calculation
Standard Monte-Carlo with Lucy (1999) path length calculation
initial
After fixed number of iterations, or convergence criterion
Image/SED calculation
Standard Monte-Carlo with optional "peeling-off" (only for scattered photons!)
imaging
If raytracing is enabled (it is disabled by default)
Raytracing calculation
Hybrid Monte-Carlo/Raytracing for source and thermal dust emission
raytracing

Example Workflow 1

setup_basic.py

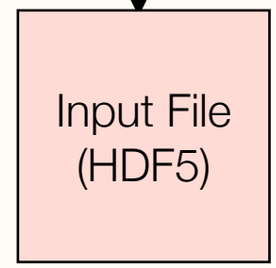
```
from hyperion.model import Model

m = Model()

# ... code here to set up model ...

m.write('basic_model.rtin')
```

Example Workflow 1

setup_basic.py

```
from hyperion.model import Model

m = Model()

# ... code here to set up model ...

m.write('basic_model.rtin')
```

python setup_basic.py



Example Workflow 1

setup_basic.py

```
from hyperion.model import Model

m = Model()

# ... code here to set up model ...

m.write('basic_model.rtin')
```

python setup_basic.py

Input File
(HDF5)

hyperion basic_model.rtin basic_model.rtout

Output File
(HDF5)

Example Workflow 1

setup_basic.py

```
from hyperion.model import Model

m = Model()

# ... code here to set up model ...

m.write('basic_model.rtin')
```

python setup_basic.py

Input File
(HDF5)

hyperion basic_model.rtin basic_model.rtout

Output File
(HDF5)

extract_basic.py

```
from hyperion.model import ModelOutput

m = ModelOutput('basic_model.rtout')

# ... extract required output here ...
```

python extract_basic.py

SEDs, Images, Plots,
FITS files, etc...

Example Workflow 1

Required if running model on separate machine, best if any of the steps are slow.

setup_basic.py

```
from hyperion.model import Model

m = Model()

# ... code here to set up model ...

m.write('basic_model.rtin')
```

python setup_basic.py

Input File (HDF5)

hyperion basic_model.rtin basic_model.rtout

Output File (HDF5)

extract_basic.py

```
from hyperion.model import ModelOutput

m = ModelOutput('basic_model.rtout')

# ... extract required output here ...
```

python extract_basic.py

SEDs, Images, Plots, FITS files, etc...

Example Workflow 2

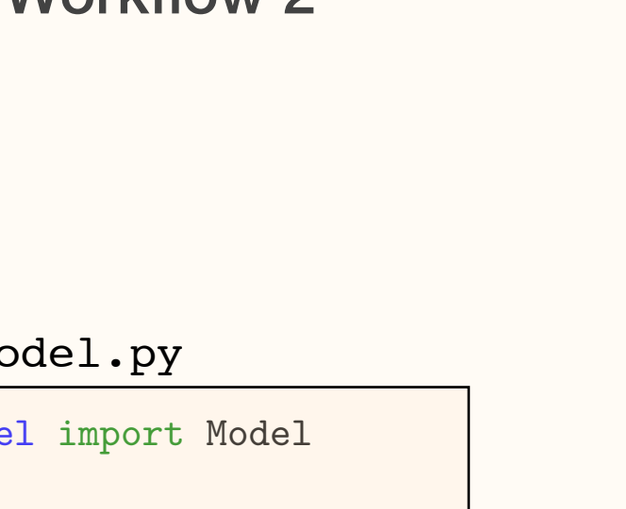
basic_model.py

```
from hyperion.model import Model

m = Model()

# ... code here to set up model ...

m.write('basic_model.rtin')

mo = m.run('basic_model.rtout')

# ... extract required output here ...
```

Example Workflow 2

basic_model.py

```
from hyperion.model import Model

m = Model()

# ... code here to set up model ...

m.write('basic_model.rtin')

mo = m.run('basic_model.rtout')

# ... extract required output here ...
```



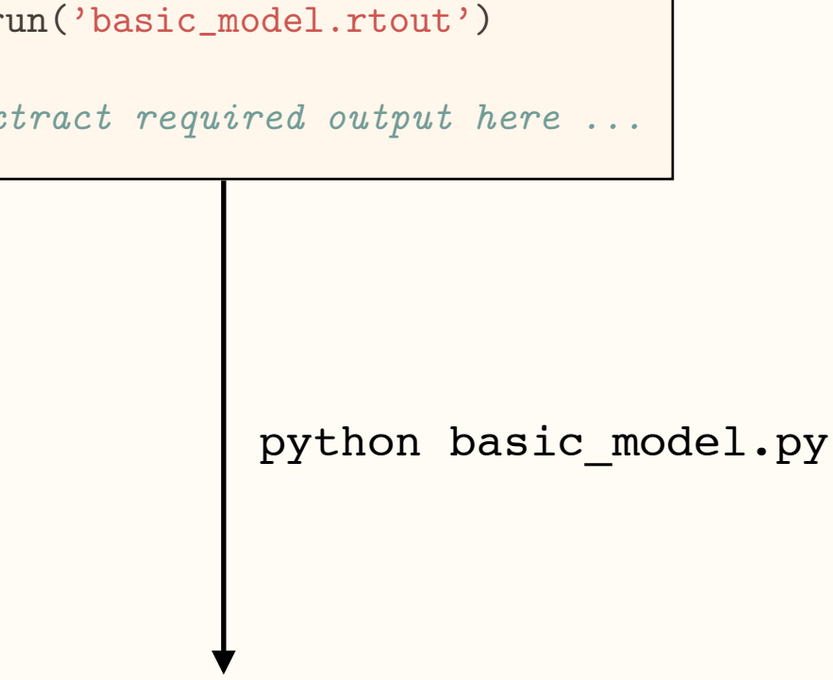
SEDs, Images, Plots,
FITS files, etc...

Example Workflow 2

Convenient once code is well tested, and running model on same machine. Easy to run batch jobs from a single script.

basic_model.py

```
from hyperion.model import Model

m = Model()

# ... code here to set up model ...

m.write('basic_model.rtin')

mo = m.run('basic_model.rtout')

# ... extract required output here ...
```



```python
from hyperion.model import Model

m = Model()

# ... code here to set up model ...

m.write('basic_model.rtin')
```

Example of an **arbitrary** 3-d model

```python
from hyperion.model import Model


# Initialize model
m = Model()
```

```python
# Write out input file
m.write('simple_cube.rtin')
```

```python
import numpy as np

from hyperion.model import Model
from hyperion.util.constants import pc

# Initialize model
m = Model()

# Set up 64x64x64 cartesian grid
w = np.linspace(-pc, pc, 64)
m.set_cartesian_grid(w, w, w)

# Write out input file
m.write('simple_cube.rtin')
```

```python
import numpy as np

from hyperion.model import Model
from hyperion.util.constants import pc

# Initialize model
m = Model()

# Set up 64x64x64 cartesian grid
w = np.linspace(-pc, pc, 64)
m.set_cartesian_grid(w, w, w)

# Add density grid
density = np.ones(m.grid.shape) * 1e-21
density[26:38, 26:38, 26:38] = 1.e-18
m.add_density_grid(density, 'kmh_lite.hdf5')

# Write out input file
m.write('simple_cube.rtin')
```

```python
import numpy as np

from hyperion.model import Model
from hyperion.util.constants import pc, lsun

# Initialize model
m = Model()

# Set up 64x64x64 cartesian grid
w = np.linspace(-pc, pc, 64)
m.set_cartesian_grid(w, w, w)

# Add density grid
density = np.ones(m.grid.shape) * 1e-21
density[26:38, 26:38, 26:38] = 1.e-18
m.add_density_grid(density, 'kmh_lite.hdf5')

# Add a point source in the center
s = m.add_point_source()
s.position = (0.4 * pc, 0., 0.)
s.luminosity = 1000 * lsun
s.temperature = 6000.

# Write out input file
m.write('simple_cube.rtin')
```

```python
import numpy as np

from hyperion.model import Model
from hyperion.util.constants import pc, lsun

# Initialize model
m = Model()

# Set up 64x64x64 cartesian grid
w = np.linspace(-pc, pc, 64)
m.set_cartesian_grid(w, w, w)

# Add density grid
density = np.ones(m.grid.shape) * 1e-21
density[26:38, 26:38, 26:38] = 1.e-18
m.add_density_grid(density, 'kmh_lite.hdf5')

# Add a point source in the center
s = m.add_point_source()
s.position = (0.4 * pc, 0., 0.)
s.luminosity = 1000 * lsun
s.temperature = 6000.

# Add multi-wavelength image for a single viewing angle
image = m.add_peeled_images(sed=False, image=True)
image.set_wavelength_range(20, 1., 1000.)
image.set_viewing_angles([60.], [80.])
image.set_image_size(400, 400)
image.set_image_limits(-1.5 * pc, 1.5 * pc, -1.5 * pc, 1.5 * pc)

# Write out input file
m.write('simple_cube.rtin')
```

```python
import numpy as np

from hyperion.model import Model
from hyperion.util.constants import pc, lsun

# Initialize model
m = Model()

# Set up 64x64x64 cartesian grid
w = np.linspace(-pc, pc, 64)
m.set_cartesian_grid(w, w, w)

# Add density grid
density = np.ones(m.grid.shape) * 1e-21
density[26:38, 26:38, 26:38] = 1.e-18
m.add_density_grid(density, 'kmh_lite.hdf5')

# Add a point source in the center
s = m.add_point_source()
s.position = (0.4 * pc, 0., 0.)
s.luminosity = 1000 * lsun
s.temperature = 6000.

# Add multi-wavelength image for a single viewing angle
image = m.add_peeled_images(sed=False, image=True)
image.set_wavelength_range(20, 1., 1000.)
image.set_viewing_angles([60.], [80.])
image.set_image_size(400, 400)
image.set_image_limits(-1.5 * pc, 1.5 * pc, -1.5 * pc, 1.5 * pc)

# Set runtime parameters
m.set_n_initial_iterations(5)
m.set_raytracing(True)
m.set_n_photons(initial=4e6, imaging=4e7,
                raytracing_sources=1, raytracing_dust=1e7)

# Write out input file
m.write('simple_cube.rtin')
```

```python
import numpy as np

from hyperion.model import Model
from hyperion.util.constants import pc, lsun

# Initialize model
m = Model()

# Set up 64x64x64 cartesian grid
w = np.linspace(-pc, pc, 64)
m.set_cartesian_grid(w, w, w)

# Add density grid
density = np.ones(m.grid.shape) * 1e-21
density[26:38, 26:38, 26:38] = 1.e-18
m.add_density_grid(density, 'kmh_lite.hdf5')

# Add a point source in the center
s = m.add_point_source()
s.position = (0.4 * pc, 0., 0.)
s.luminosity = 1000 * lsun
s.temperature = 6000.

# Add multi-wavelength image for a single viewing angle
image = m.add_peeled_images(sed=False, image=True)
image.set_wavelength_range(20, 1., 1000.)
image.set_viewing_angles([60.], [80.])
image.set_image_size(400, 400)
image.set_image_limits(-1.5 * pc, 1.5 * pc, -1.5 * pc, 1.5 * pc)

# Set runtime parameters
m.set_n_initial_iterations(5)
m.set_raytracing(True)
m.set_n_photons(initial=4e6, imaging=4e7,
                raytracing_sources=1, raytracing_dust=1e7)

# Write out input file
m.write('simple_cube.rtin')
m.run('simple_cube.rtout', mpi=True)
```

```python
import numpy as np

from hyperion.model import Model
from hyperion.util.constants import pc, lsun
```

Initialization

```python
# Initialize model
m = Model()
```

Geometry

```python
# Set up 64x64x64 cartesian grid
w = np.linspace(-pc, pc, 64)
m.set_cartesian_grid(w, w, w)
```

Density and Dust

```python
# Add density grid
density = np.ones(m.grid.shape) * 1e-21
density[26:38, 26:38, 26:38] = 1.e-18
m.add_density_grid(density, 'kmh_lite.hdf5')
```

Sources

```python
# Add a point source in the center
s = m.add_point_source()
s.position = (0.4 * pc, 0., 0.)
s.luminosity = 1000 * lsun
s.temperature = 6000.
```

Images/SEDs

```python
# Add multi-wavelength image for a single viewing angle
image = m.add_peeled_images(sed=False, image=True)
image.set_wavelength_range(20, 1., 1000.)
image.set_viewing_angles([60.], [80.])
image.set_image_size(400, 400)
image.set_image_limits(-1.5 * pc, 1.5 * pc, -1.5 * pc, 1.5 * pc)
```

Parameters

```python
# Set runtime parameters
m.set_n_initial_iterations(5)
m.set_raytracing(True)
m.set_n_photons(initial=4e6, imaging=4e7,
                raytracing_sources=1, raytracing_dust=1e7)
```

Writing and Running

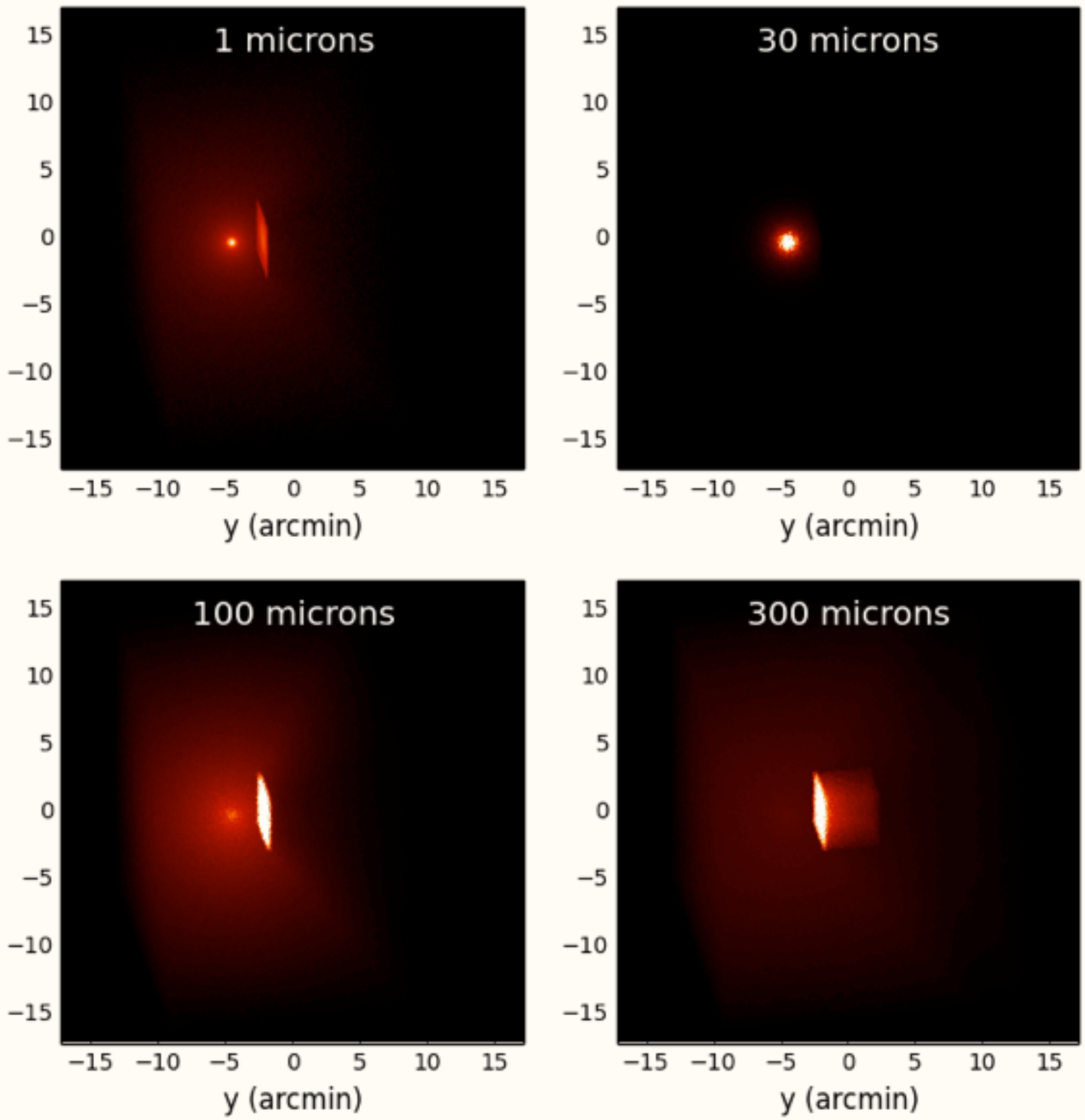
```python
# Write out input file
m.write('simple_cube.rtin')
m.run('simple_cube.rtout', mpi=True)
```

```python
from hyperion.model import ModelOutput

mo = ModelOutput('basic_model.rtout')

# Extract SED
sed = mo.get_sed(...)

# Extract image
image = mo.get_image(...)

# Extract physical quantities
grid = mo.get_quantities(...)
```

See full documentation at
http://docs.hyperion-rt.org/en/stable/postprocessing/postprocessing.html

```python
import numpy as np
import matplotlib.pyplot as plt

from hyperion.model import ModelOutput
from hyperion.util.constants import pc

# Open the model
m = ModelOutput('simple_cube.rtout')

# Extract the image for the first inclination, and scale to 300pc. We
# have to specify group=1 as there is no image in group 0.
image = m.get_image(inclination=0, distance=300 * pc, units='MJy/sr')

# ... plotting code ...
```

See full tutorial at http://docs.hyperion-rt.org/en/stable/tutorials/tutorial_images.html



See full tutorial at http://docs.hyperion-rt.org/en/stable/tutorials/tutorial_images.html

See full tutorial at http://docs.hyperion-rt.org/en/stable/tutorials/tutorial_images.html

Dust properties

HYPERION supports arbitrary optical properties for **randomly oriented** dust grains ('4-element scattering matrix').

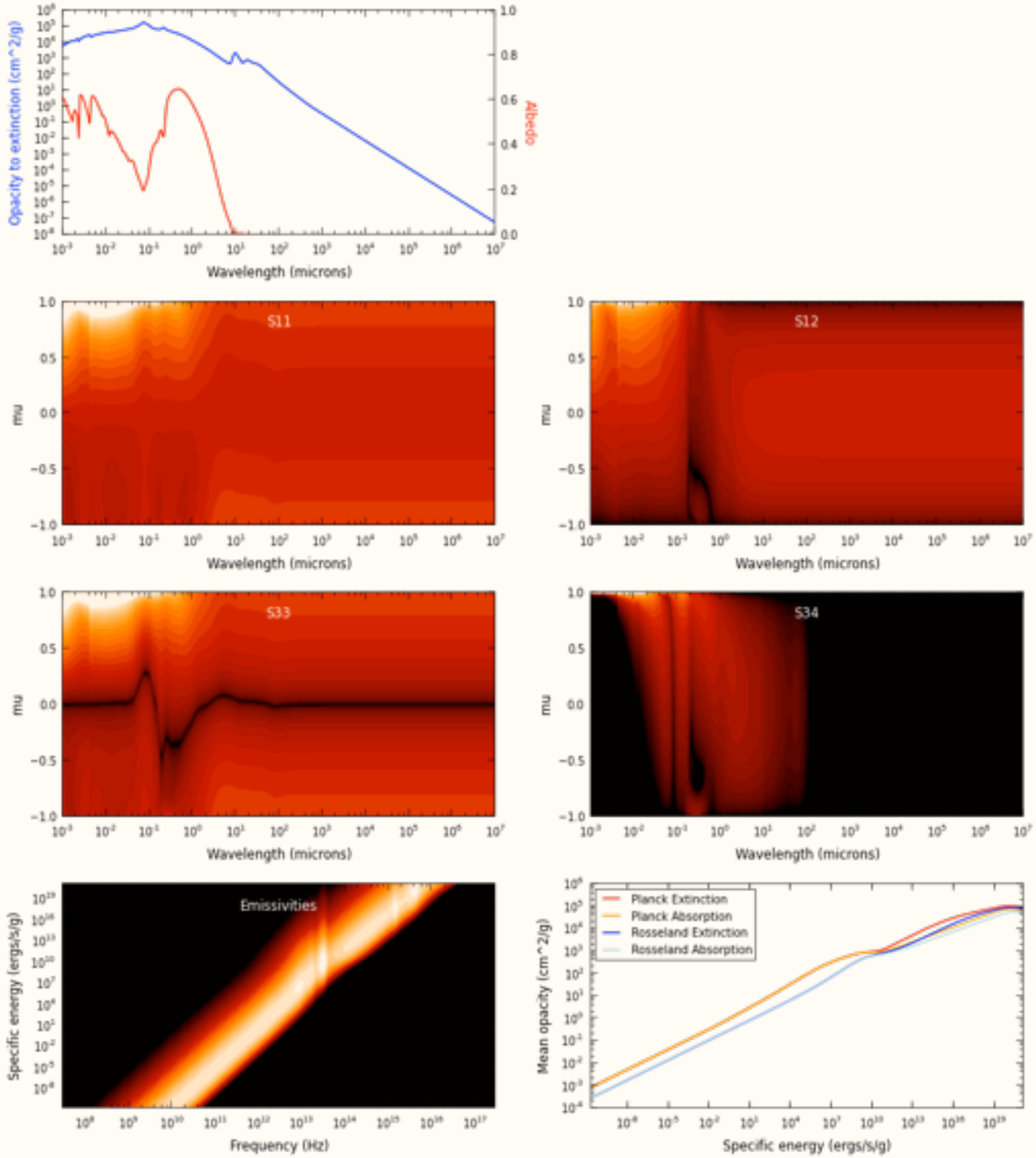
Dust properties should be stored as a single HDF5 file.

Python classes are available to make it easy to set up such files:

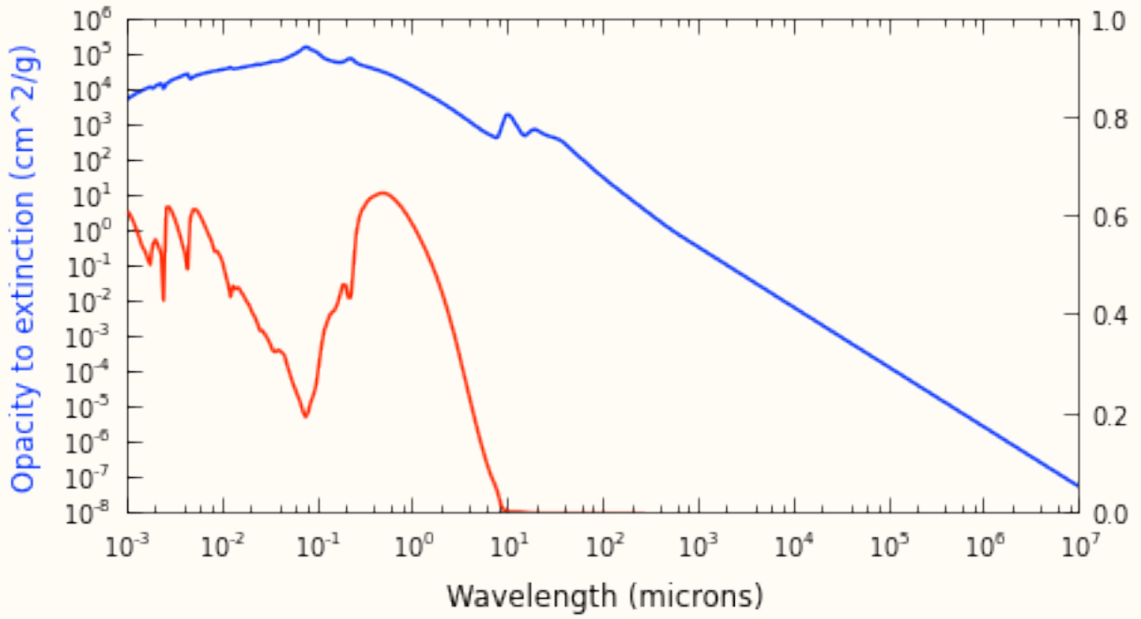
```
from hyperion.dust import IsotropicDust
d = IsotropicDust(nu, albedo, chi)
d.write('my_isotropic_dust.hdf5')
d.plot('my_isotropic_dust.png')
```

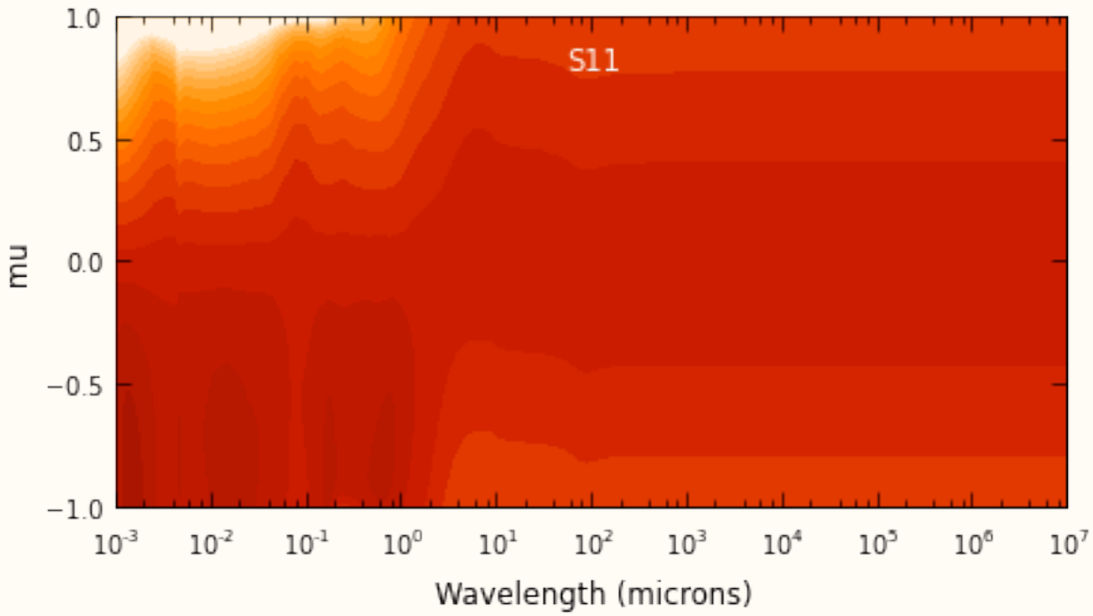
See documentation at http://docs.hyperion-rt.org/en/stable/setup/setup_dust.html

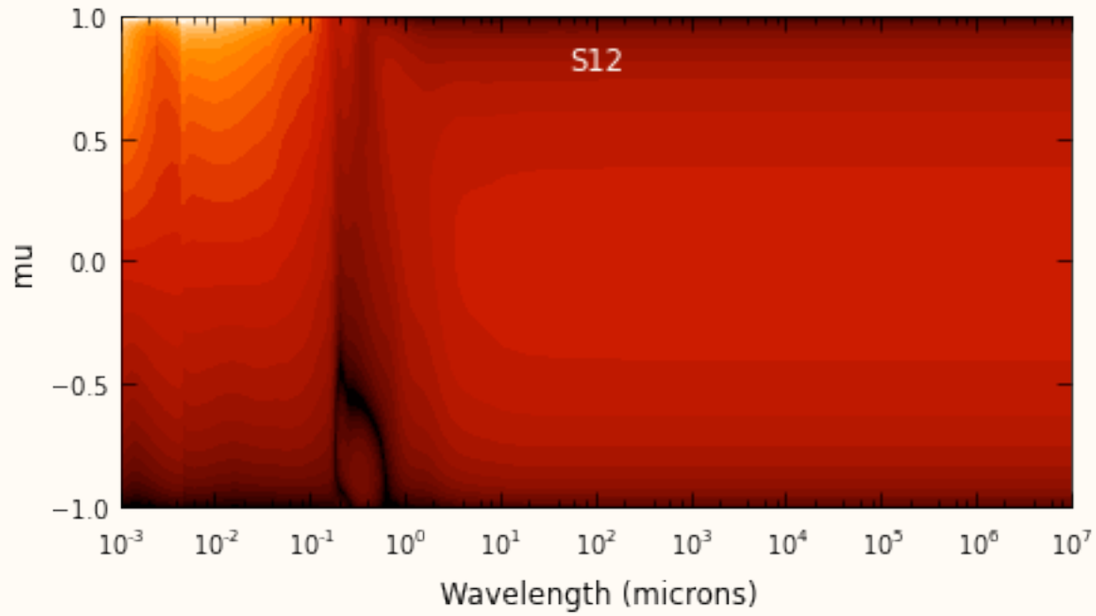
Example from dust model library at http://docs.hyperion-rt.org/en/stable/dust/dust.html

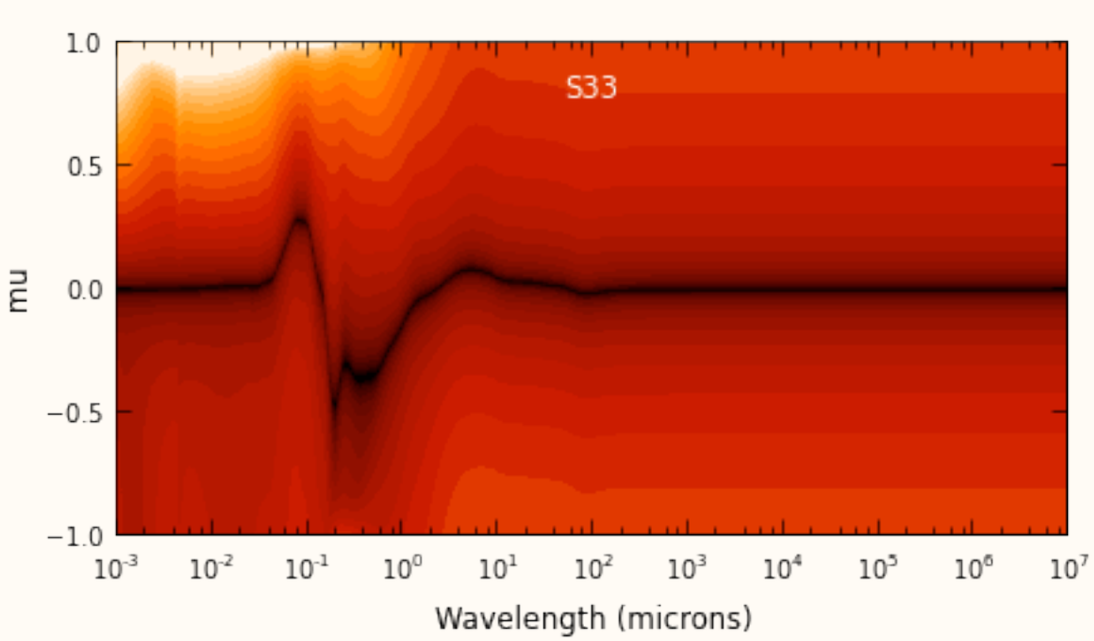


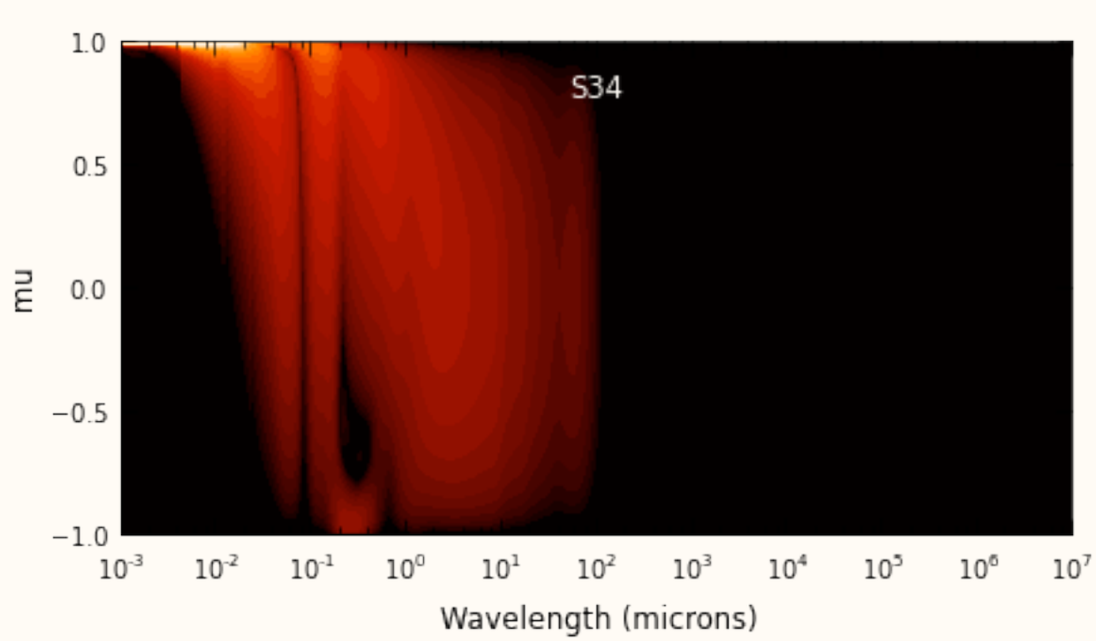
Draine et al. (2003) Milky-Way dust, Rv=3.1, b_c=6.0

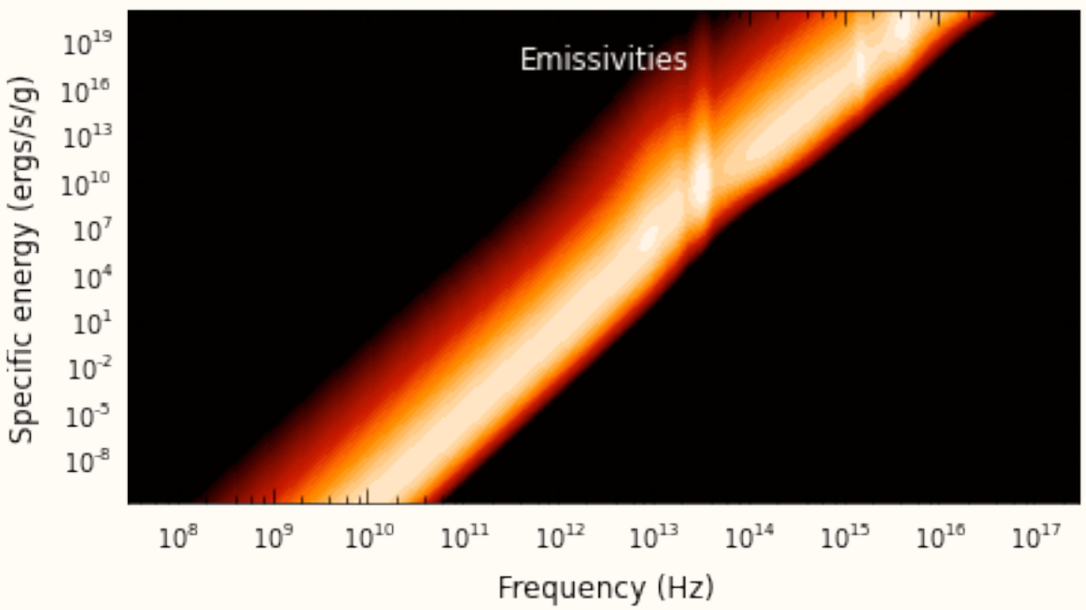

Opacity to extinction (cm^2/g)
Albedo
Wavelength (microns)
10^{6}
10^{5}
10^{4}
10^{3}
10^{2}
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
1.0
0.8
0.6
0.4
0.2
0.0
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
10^{5}
10^{6}
10^{7}

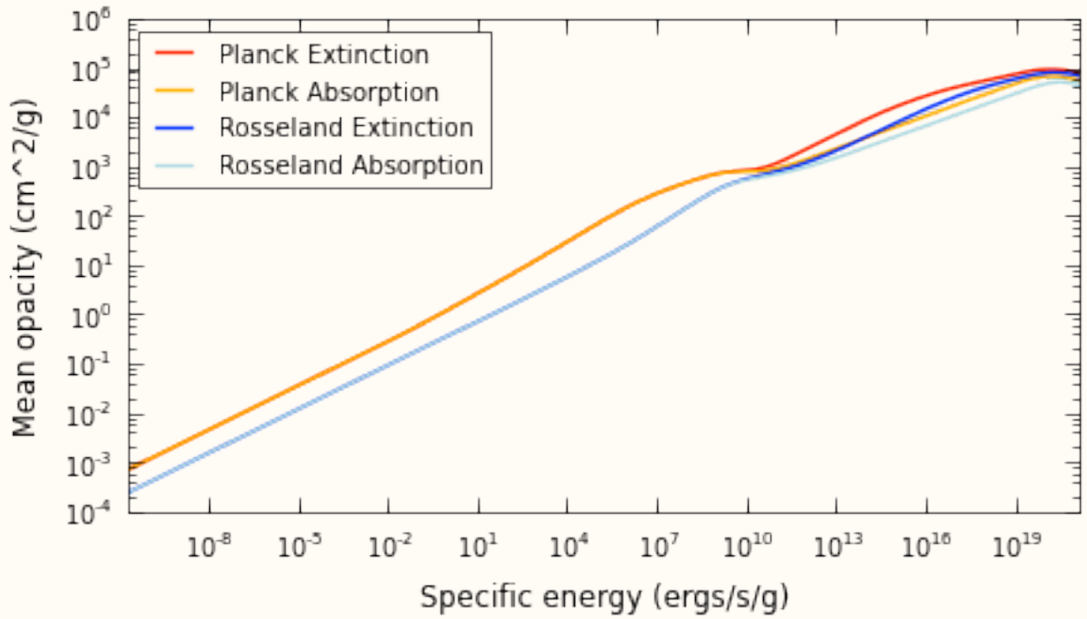

S11
mu
1.0
0.5
0.0
−0.5
−1.0
10^{-3} 10^{-2} 10^{-1} 10^{0} 10^{1} 10^{2} 10^{3} 10^{4} 10^{5} 10^{6} 10^{7}
Wavelength (microns)


S12
mu
1.0
0.5
0.0
−0.5
−1.0
10^{-3} 10^{-2} 10^{-1} 10^{0} 10^{1} 10^{2} 10^{3} 10^{4} 10^{5} 10^{6} 10^{7}
Wavelength (microns)


S33
mu
1.0
0.5
0.0
−0.5
−1.0
10^{-3} 10^{-2} 10^{-1} 10^{0} 10^{1} 10^{2} 10^{3} 10^{4} 10^{5} 10^{6} 10^{7}
Wavelength (microns)


S34
mu
1.0
0.5
0.0
−0.5
−1.0
10^{-3} 10^{-2} 10^{-1} 10^{0} 10^{1} 10^{2} 10^{3} 10^{4} 10^{5} 10^{6} 10^{7}
Wavelength (microns)


Emissivities
Specific energy (ergs/s/g)
Frequency (Hz)


Planck Extinction
Planck Absorption
Rosseland Extinction
Rosseland Absorption
Mean opacity (cm^2/g)
Specific energy (ergs/s/g)

Important Notes

HYPERION does not make any assumptions about whether or not gas is included. If the opacities are specified per unit dust mass, the densities should be given as dust densities, and if the opacities are specified per unit dust+gas mass then the densities should be given in dust+gas densities.

HYPERION does not try and determine the optimal gridding or number of photons to use - this is up to the user. Note that it is possible for example to run Hyperion one iteration at a time and adjust these parameters between each iteration.

Interfacing with Hydro+ Simulations

HYPERION aims to support reading in density grids from various simulation codes. Python library already includes support for ORION AMR code. Ask about support for other formats!

Support exporting output cartesian, AMR, and Octree grids to yt:

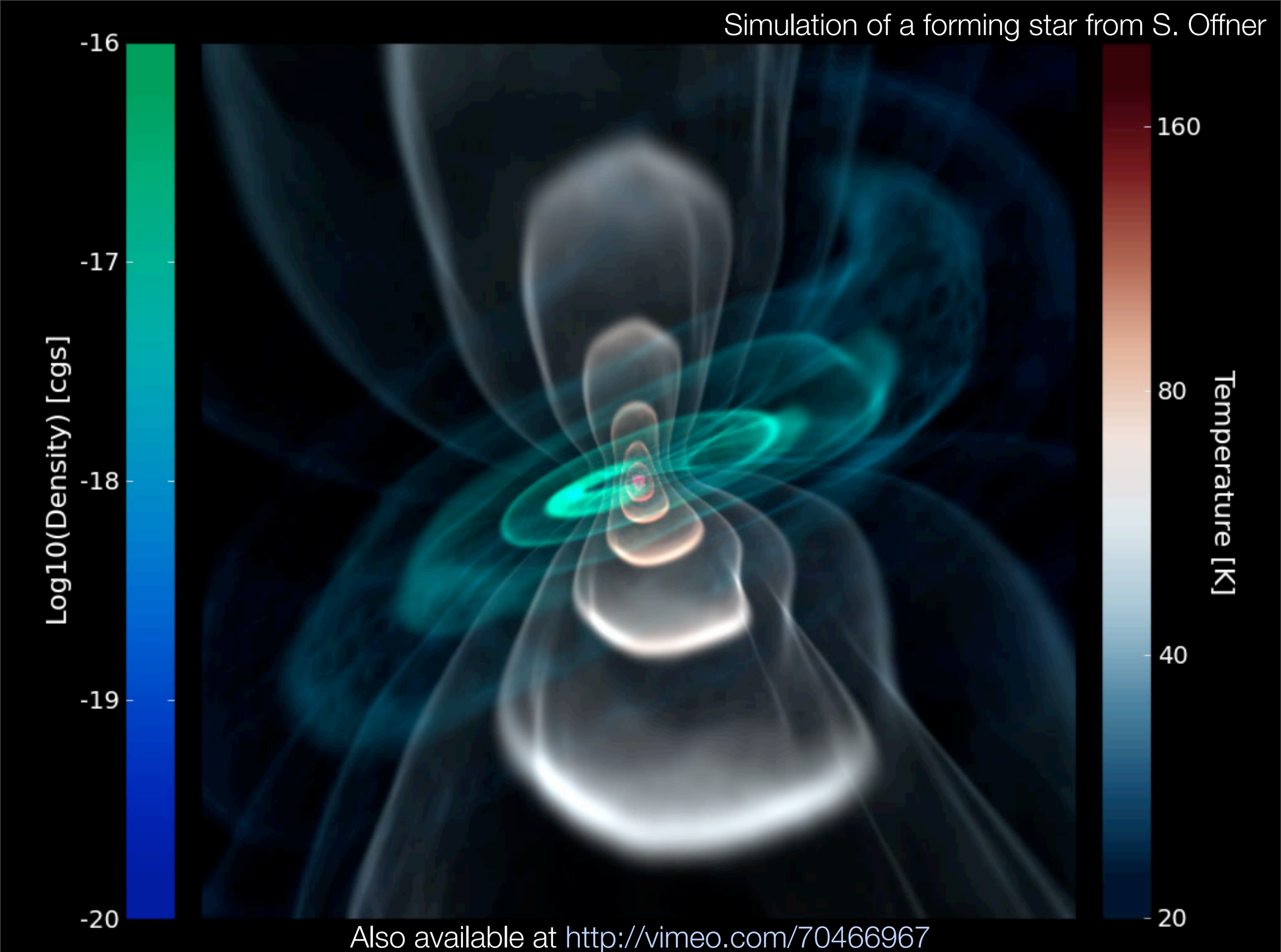
```
# Read in grid (with quantities) from output model file
from hyperion.model import ModelOutput
m = ModelOutput('model.rtout')
grid = m.get_quantities()

# Export to yt object
pf = grid.to_yt()

# Use yt to make a projection plot of density and temperature
from yt.mods import ProjectionPlot
prj = ProjectionPlot(pf, 'y', ['density', 'temperature'],
                     center=[0.0, 0.0, 0.0])
prj.set_cmap('temperature', 'gist_heat')
prj.set_cmap('density', 'gist_heat')
prj.set_log('density', True)
prj.save()
```

See tutorial at http://docs.hyperion-rt.org/en/stable/tutorials/tutorial_quantities_yt.html

Simulation of a forming star from S. Offner

Also available at http://vimeo.com/70466967

Simulation of a forming star from S. Offner



Also available at http://vimeo.com/70466967

Planned features

Scattering surfaces (e.g. planetary surfaces)

Temperature-dependent opacities

Raytracing for scattered light

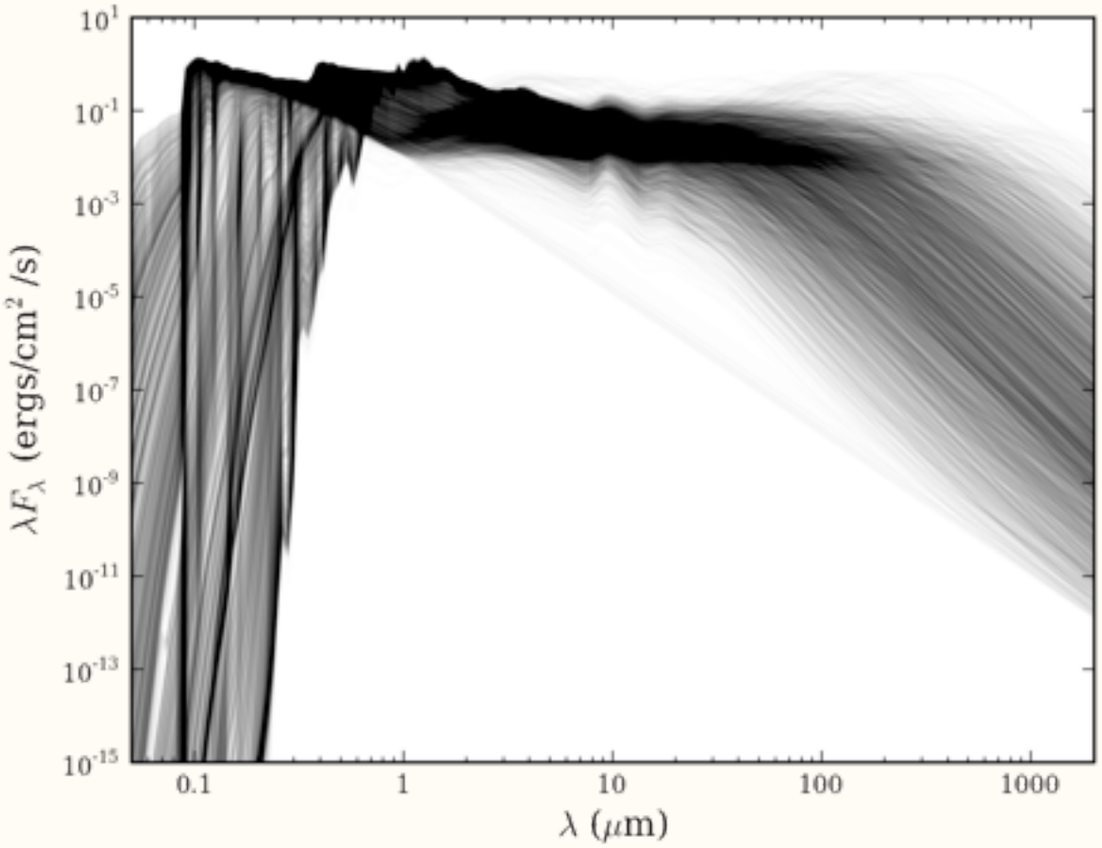
Basic photoionization

Molecular lines

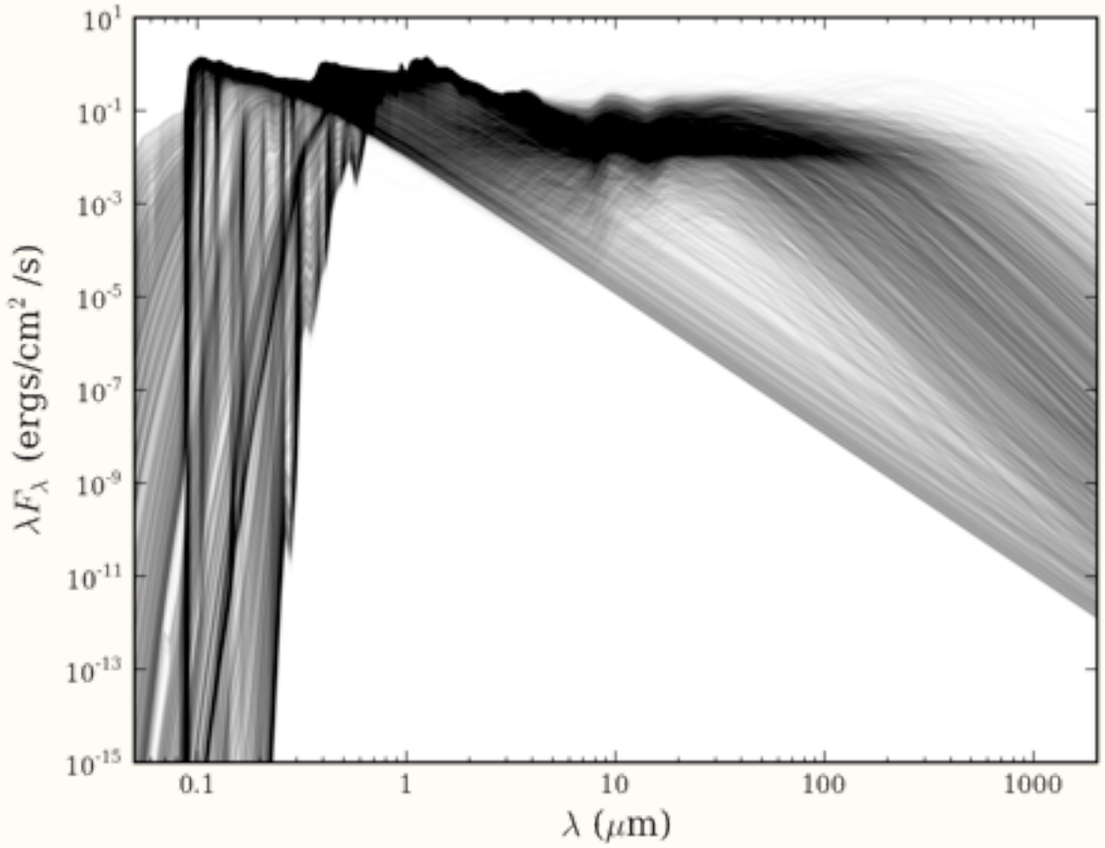
Non-spherical grains aligned along magnetic fields

Gallery of models

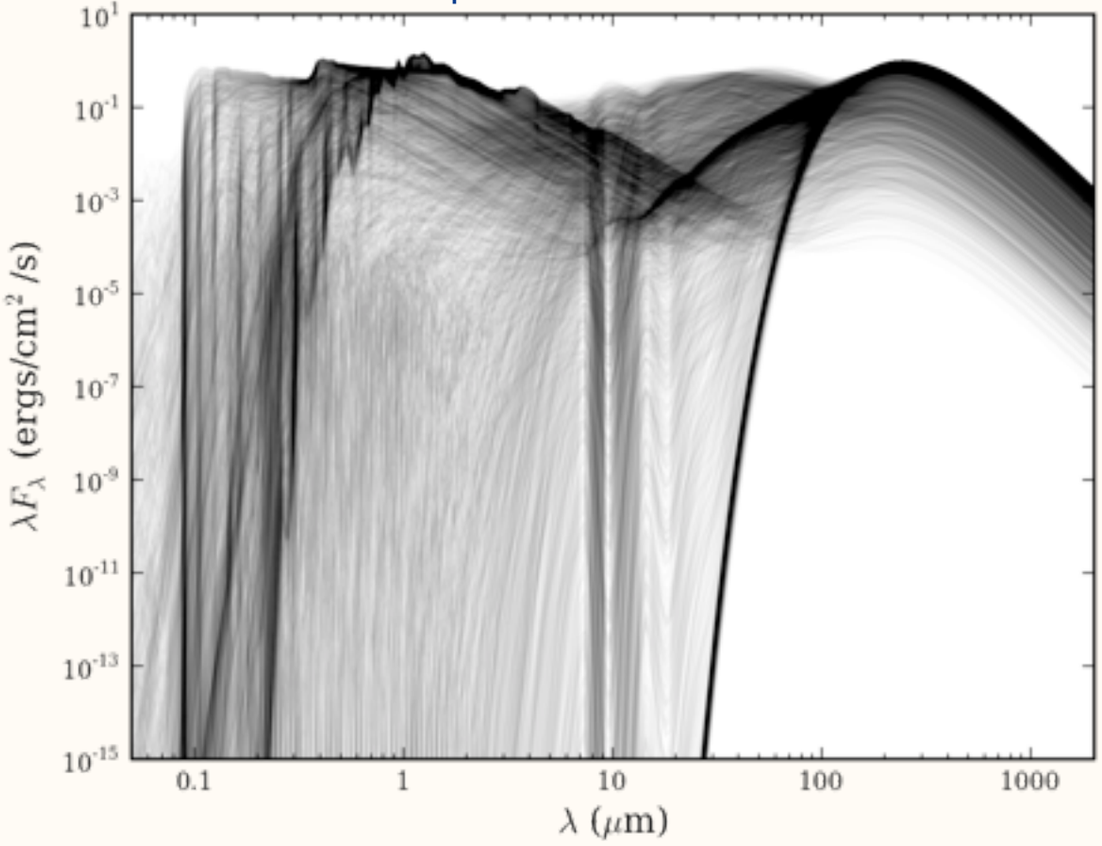
Flared Disk



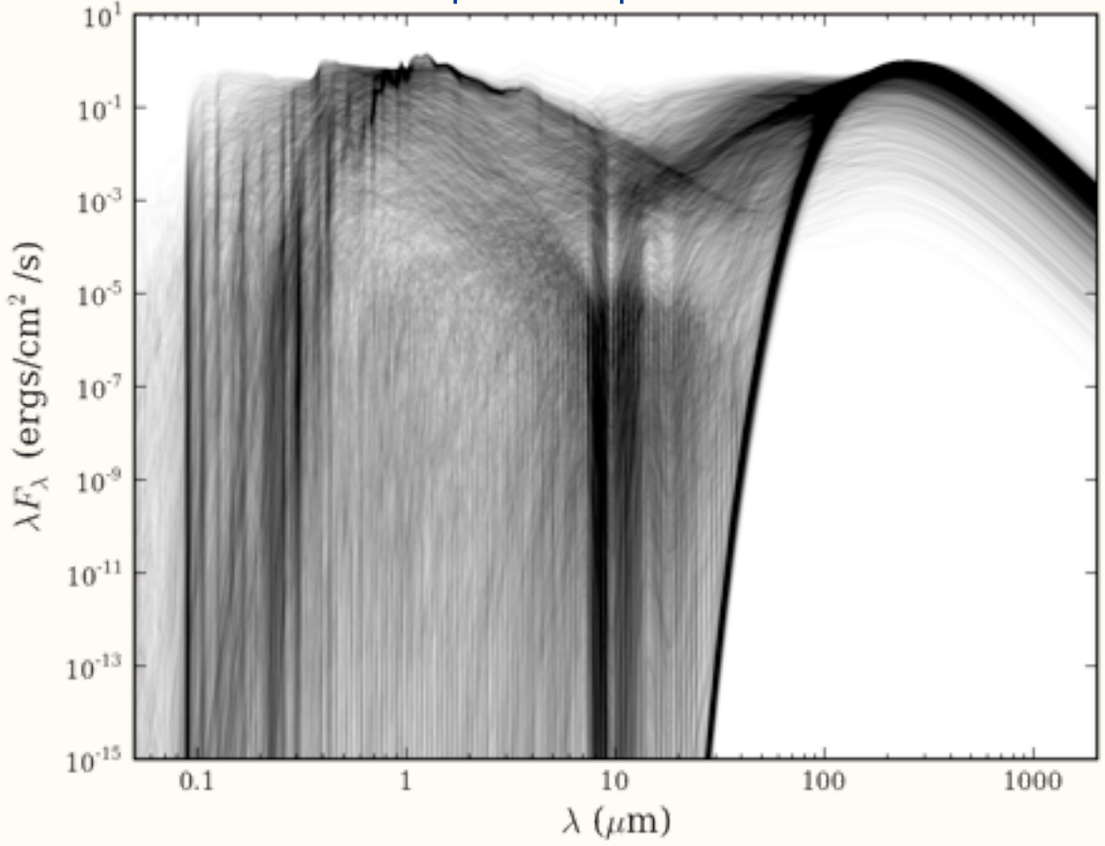
Flared disk with inner hole



Ulrich envelope

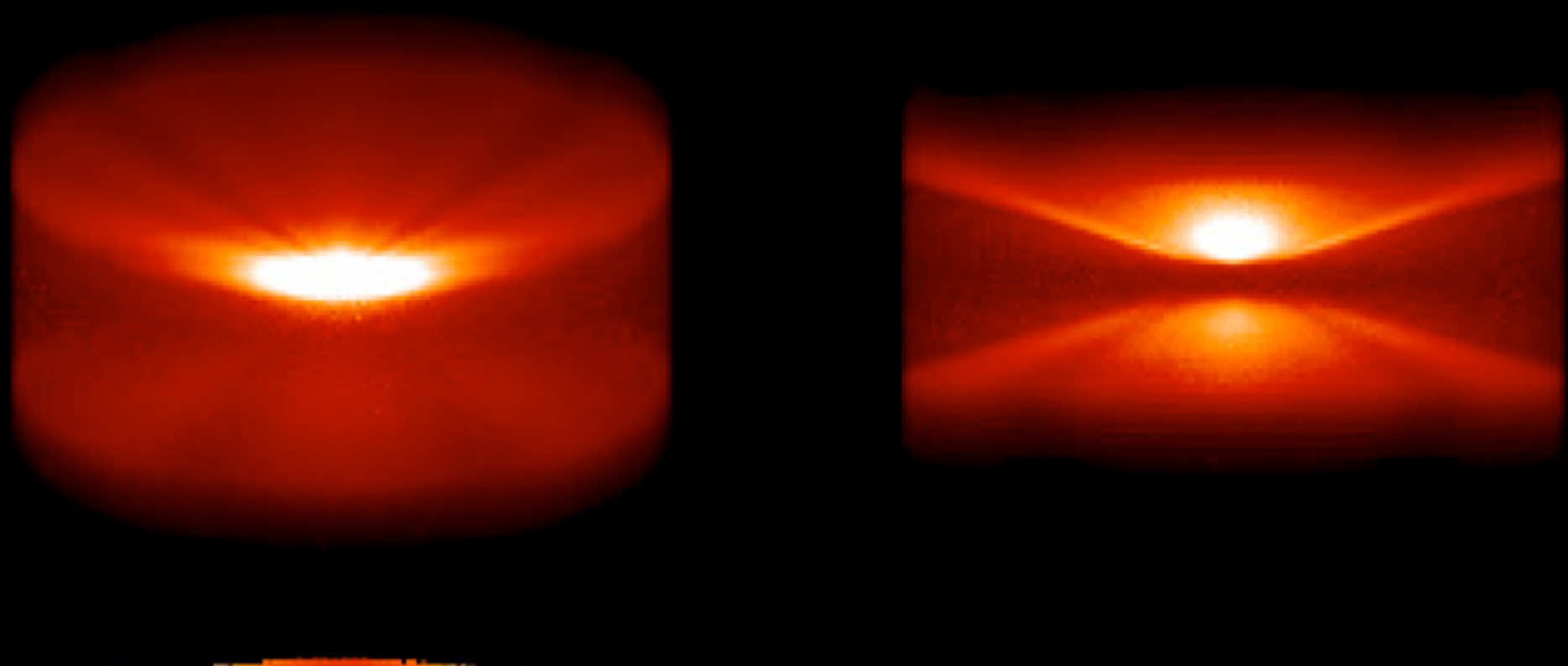


Ulrich envelope + bipolar cavities

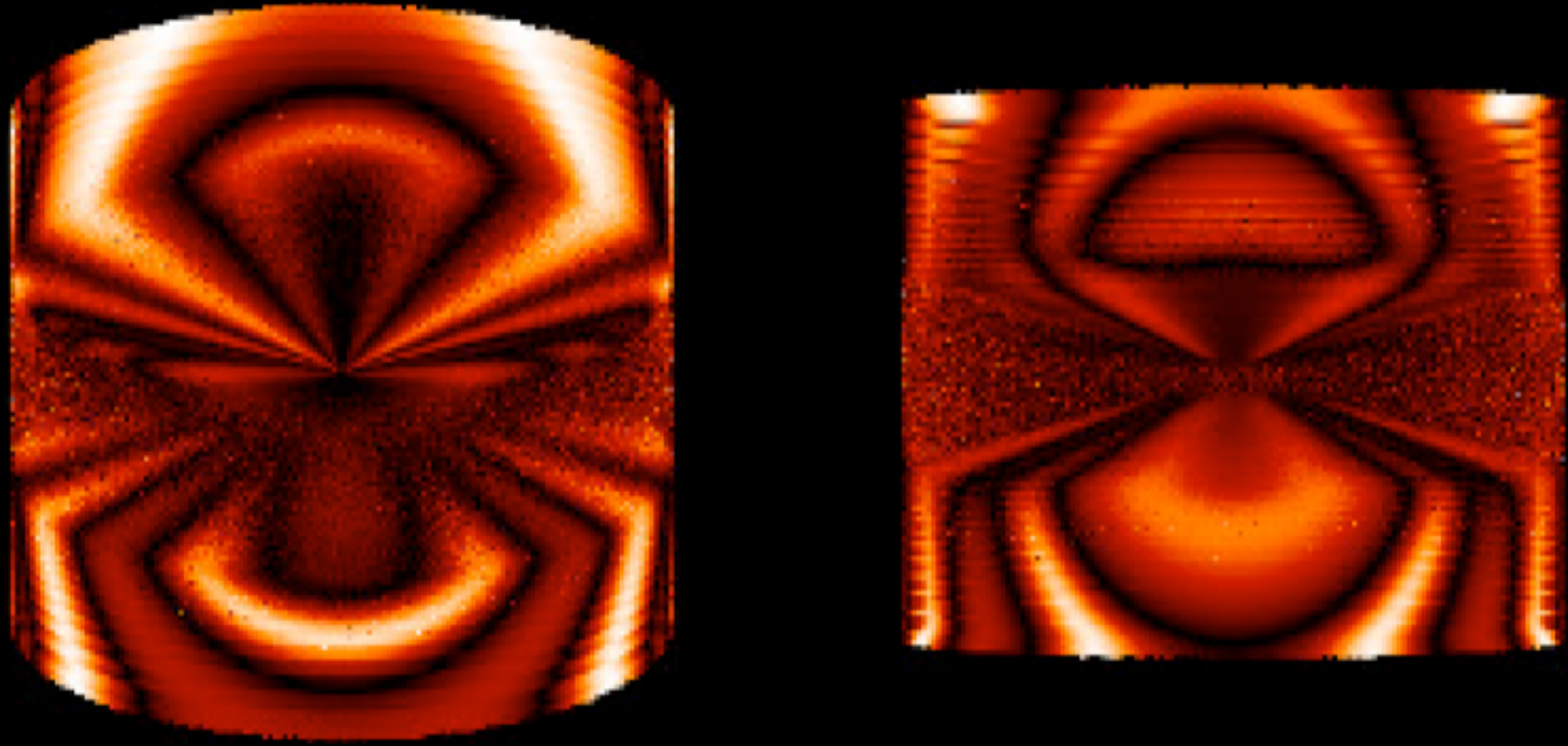


Pinte et al (2009) protoplanetary disk benchmark

Flux



Linear Polarization



i=69.5°

i=87.1°

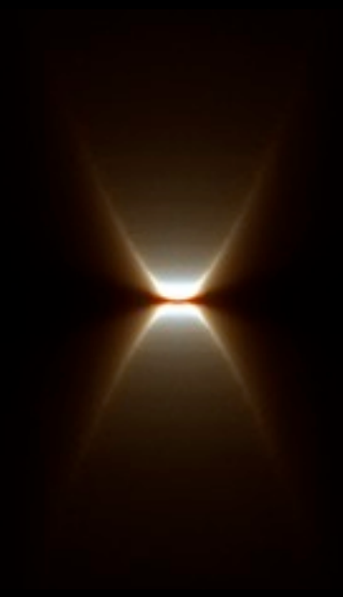
Analytical model of a Young Stellar Object

1 - 3 μm

Scattered light + extinction

20 - 60 μm

Warm dust



400 - 800 μm

Cool dust

$6x10^{14}$ m (4000 AU)

Simulation of a forming star from S. Offner

2 - 7 μm

Scattered light + extinction

300 - 1000 μm

Cool dust

$3x10^{15}$ m (0.1 pc)

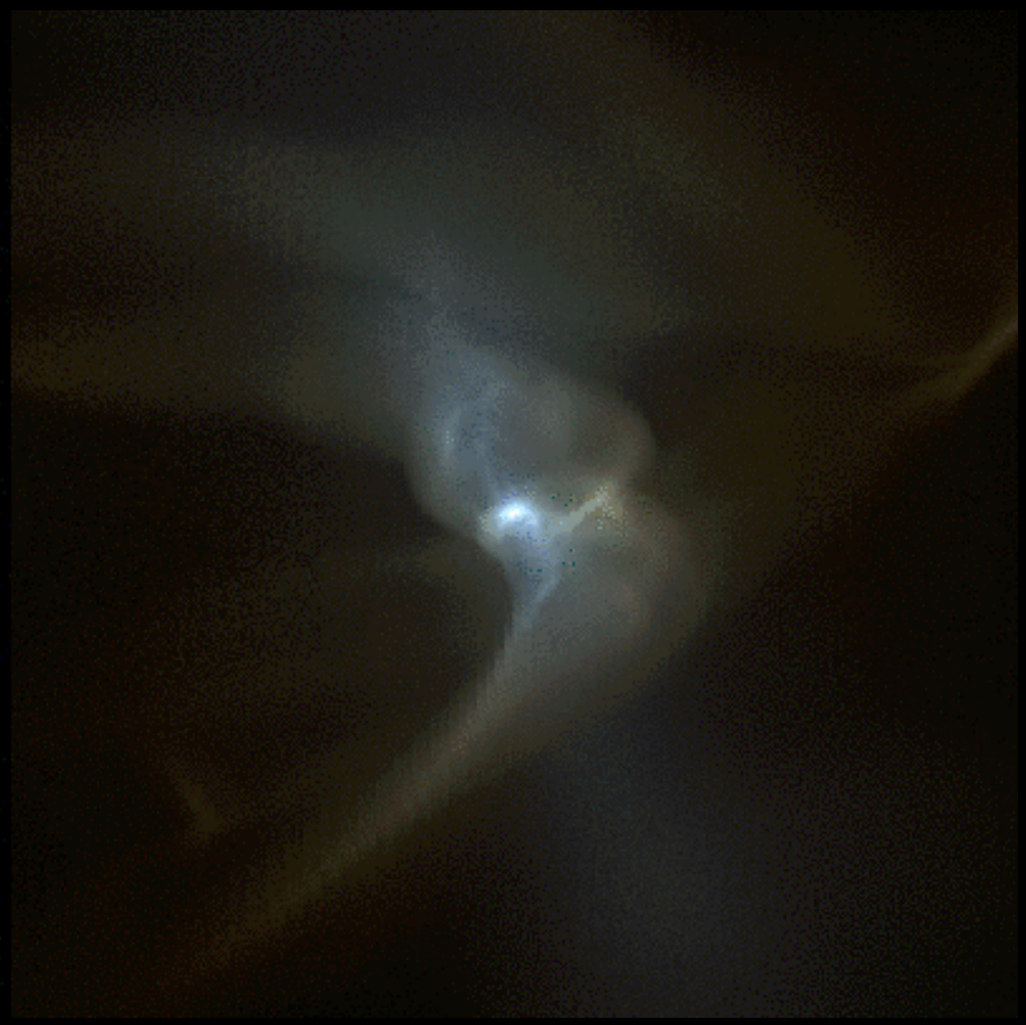
Simulation of a forming star from S. Offner

2 - 7 μm

Scattered light + extinction

300 - 1000 μm

Cool dust



$3x10^{15}$ m (0.1 pc)

Far-IR image based on a simulation with AREPO by R. Smith

Where to get help?

https://github.com/hyperion-rt/hyperion/issues

To report bugs, or request features, please use the GitHub Issue tracker (shown on right)

For general help, you can simply email

help@hyperion-rt.org

This week - ask me in person!