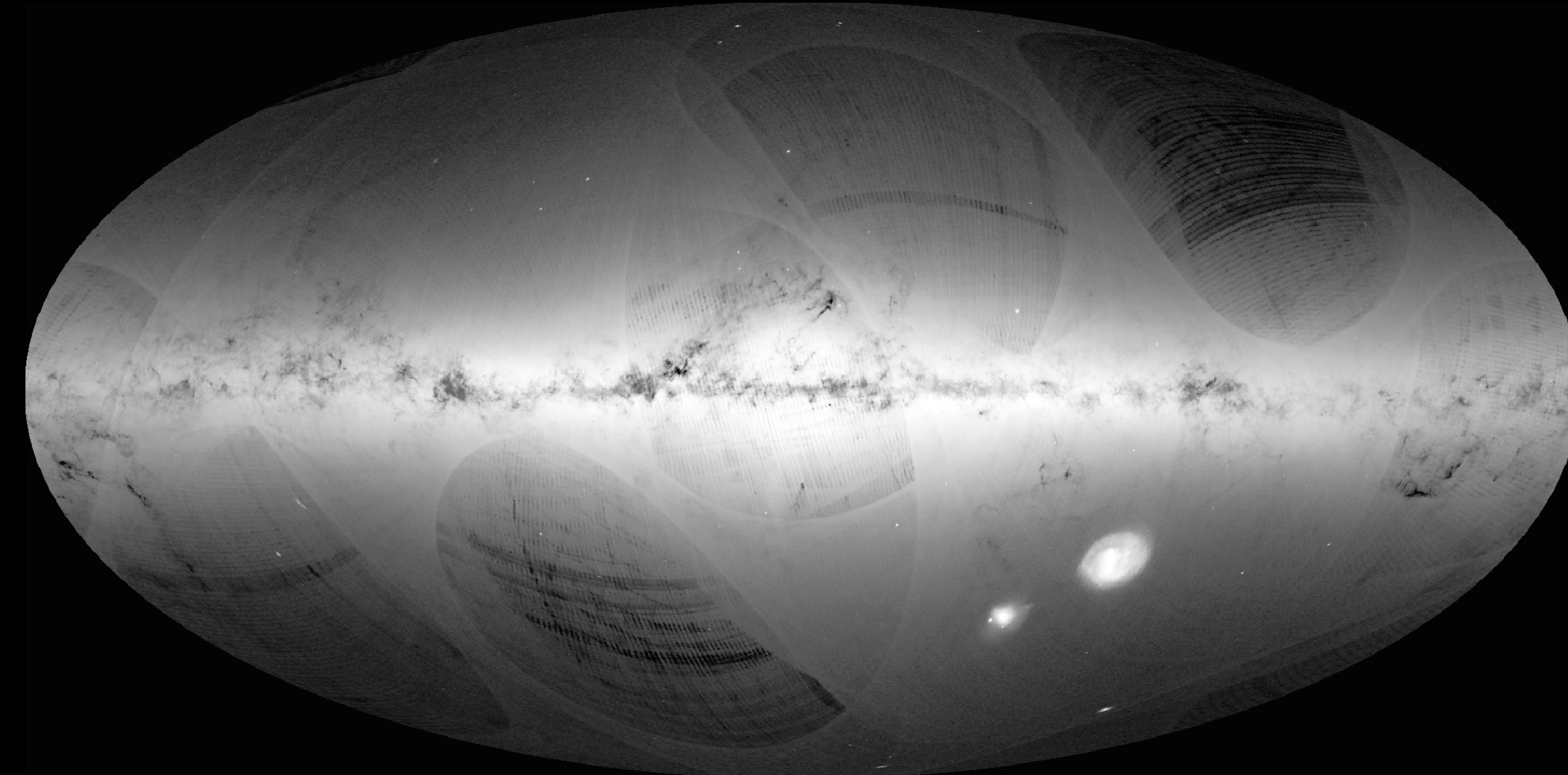


A Billion stars in the Jupyter notebook



@MaartenBreddels  

jupytercon - 2017 - NY



university of
groningen

faculty of mathematics
and natural sciences

kapteyn astronomical
institute

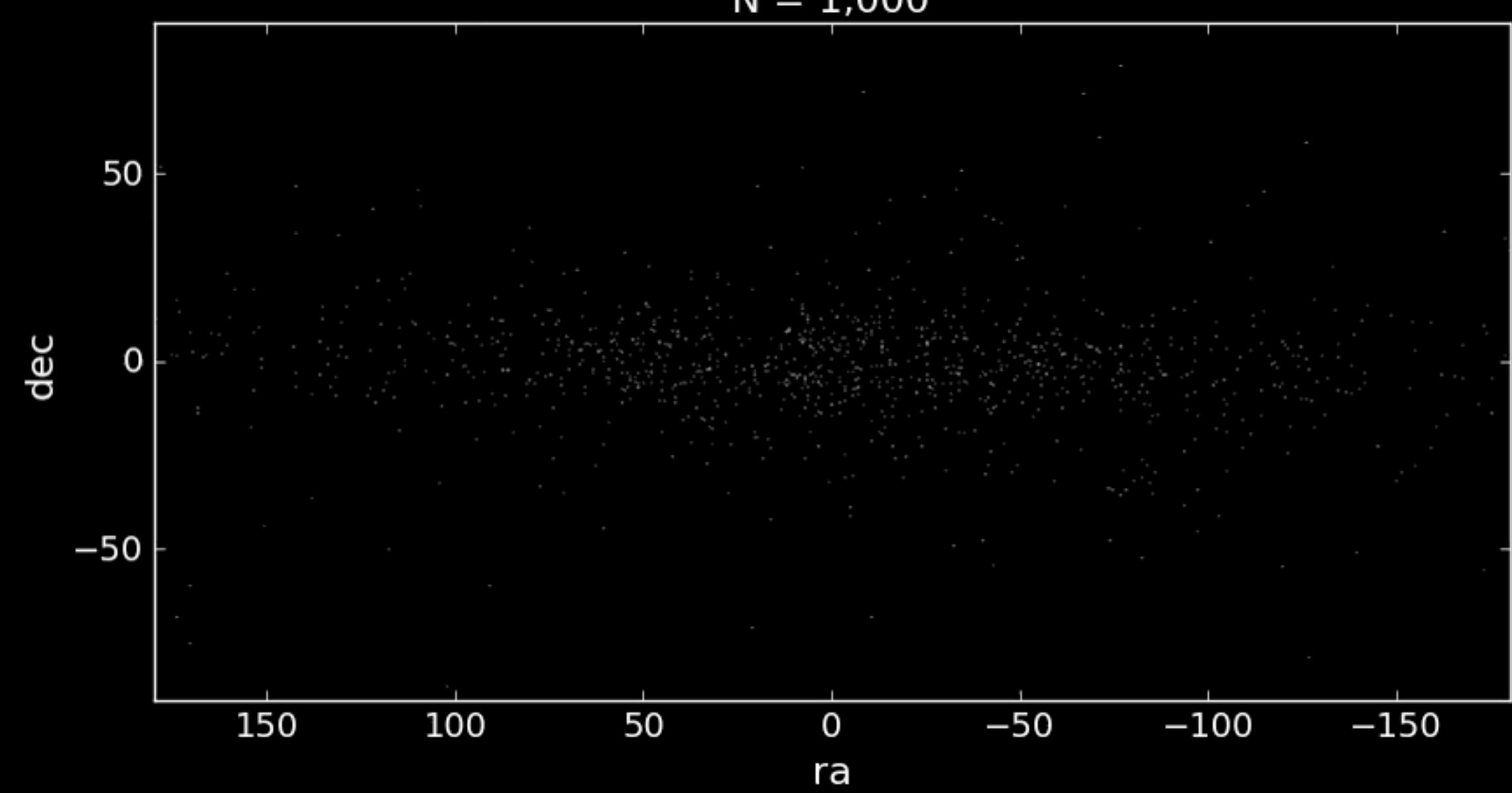
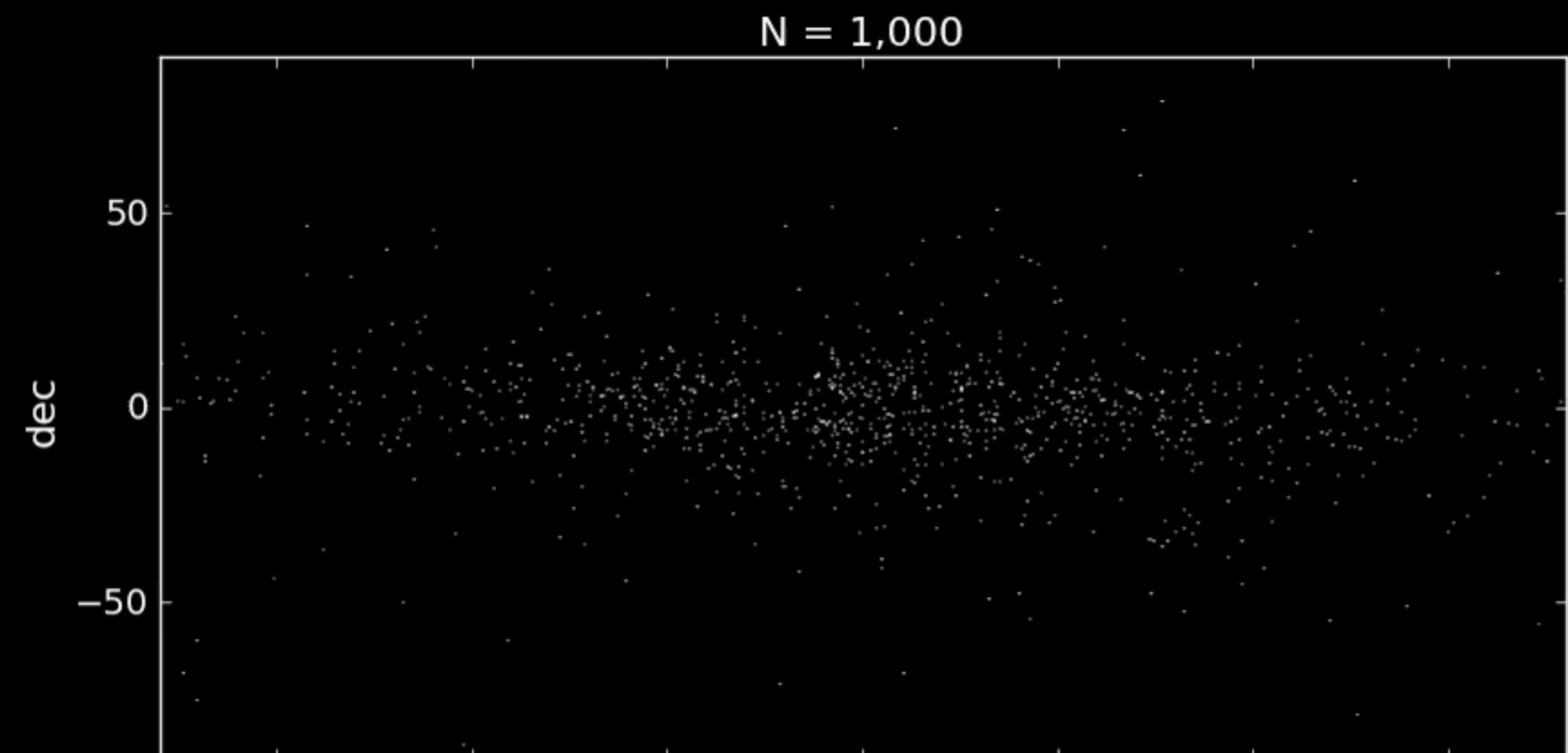
Agenda

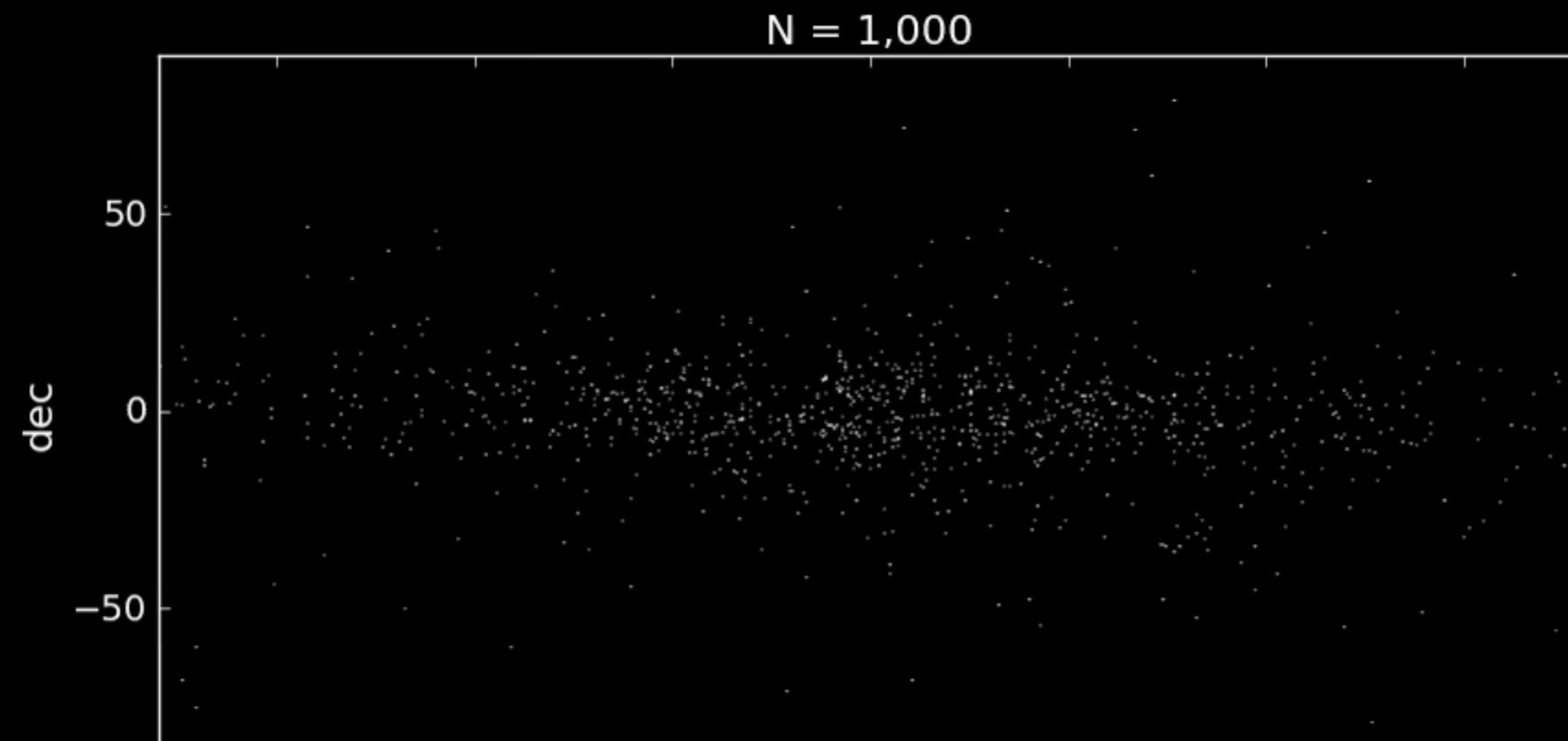
- Show how to deal with a billion objects/rows/stars?
- How you can do 3d visualization in the notebook?

Motivation: Gaia

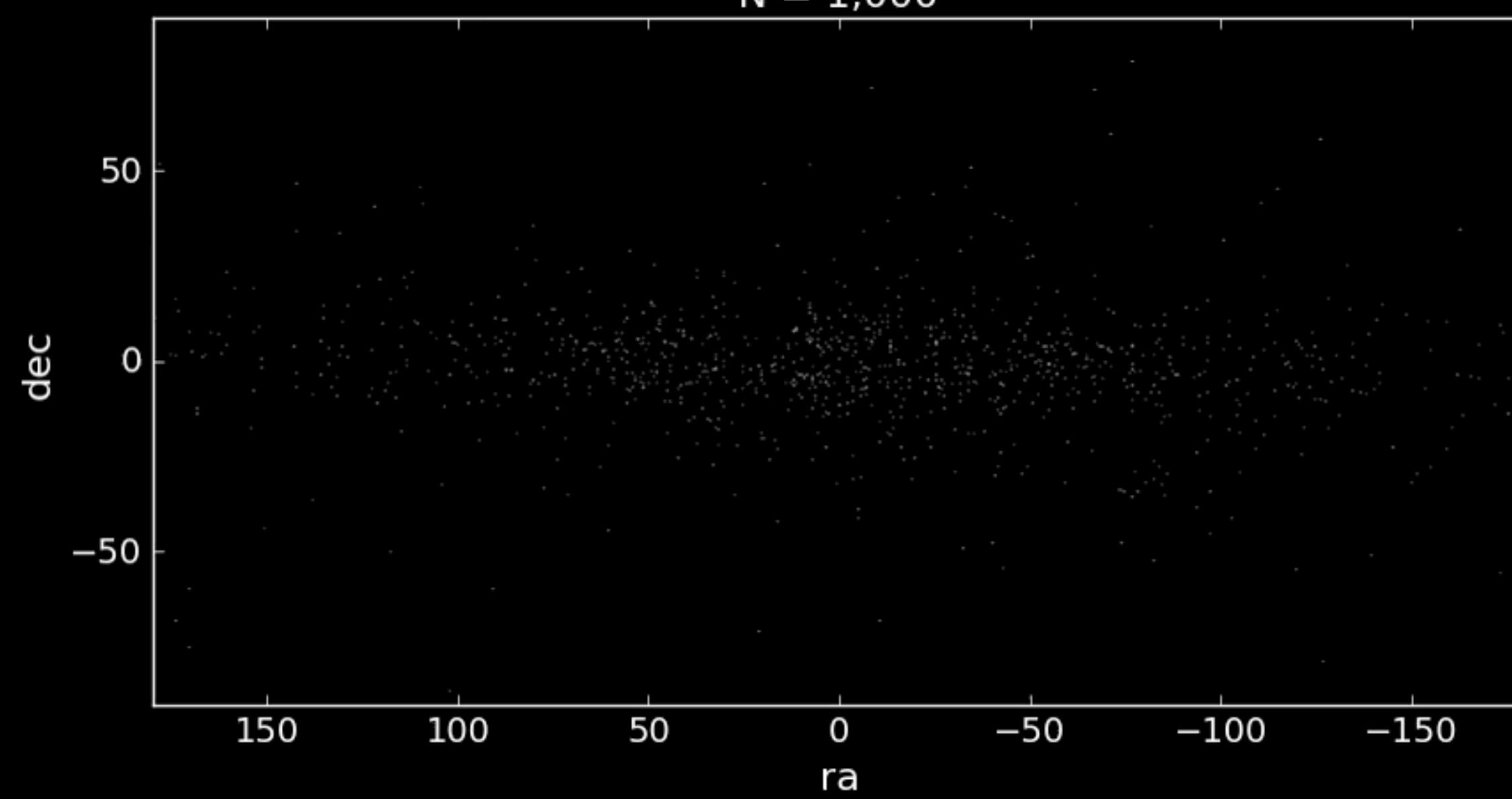


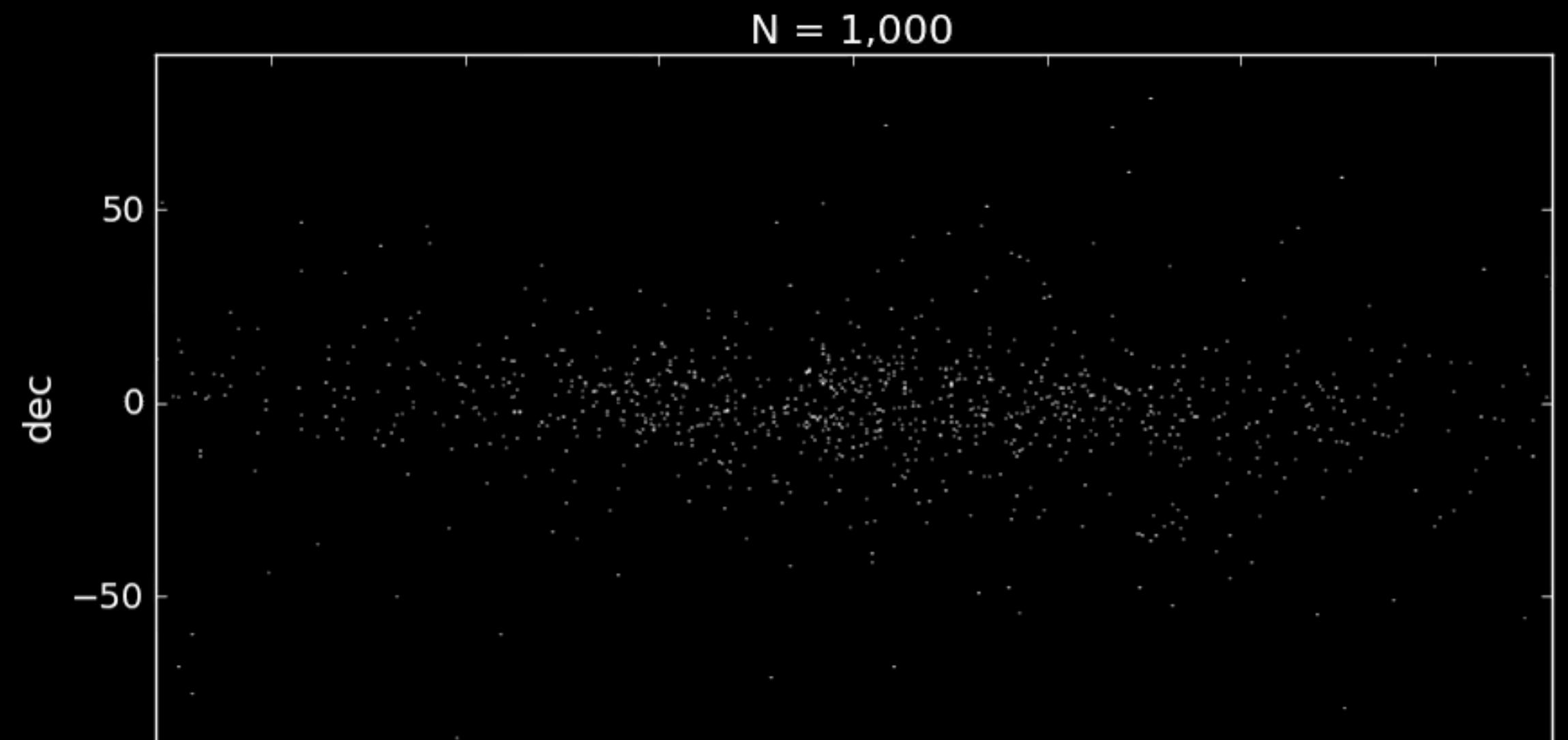
>1,000,000,000 stars/objects



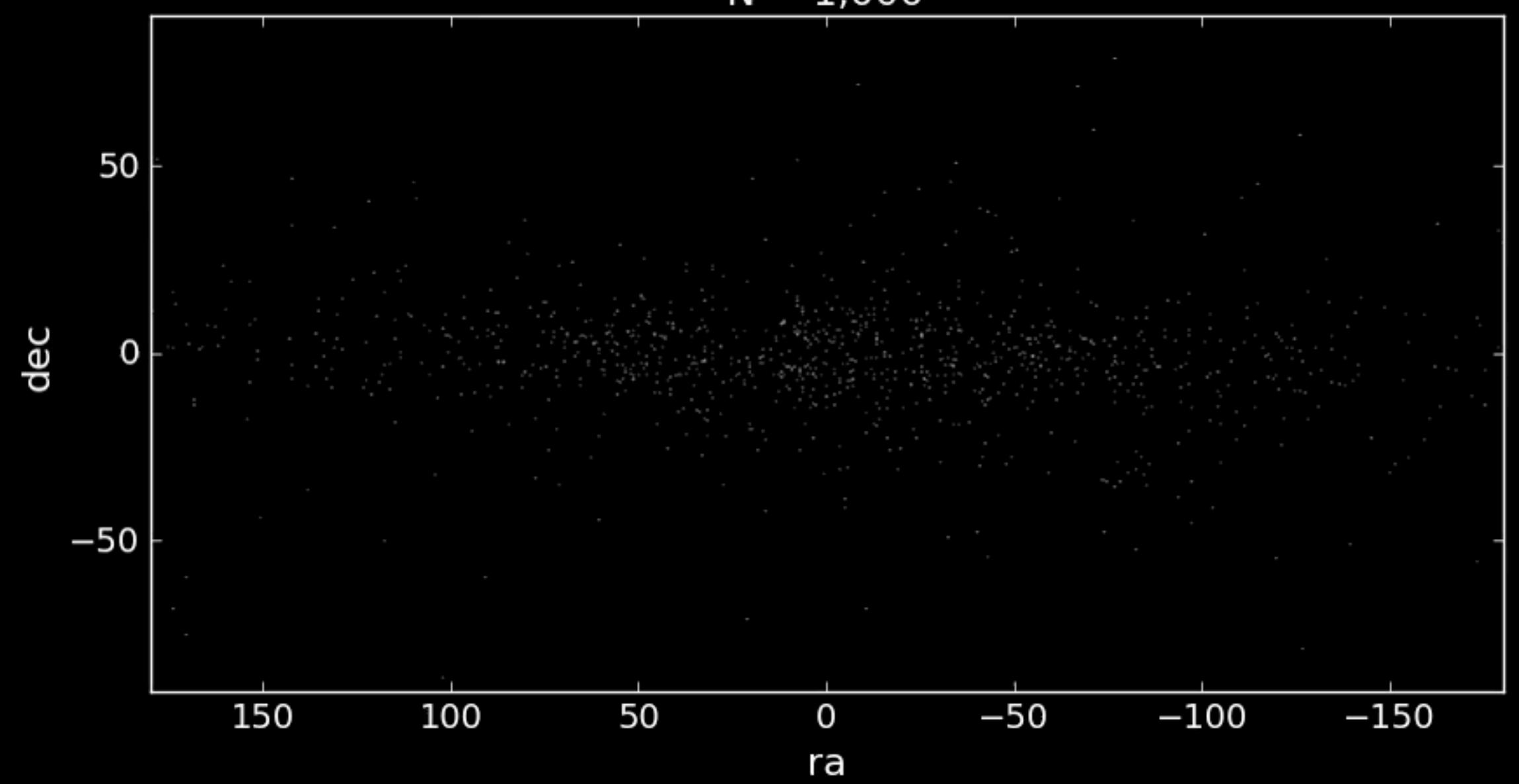


scatter



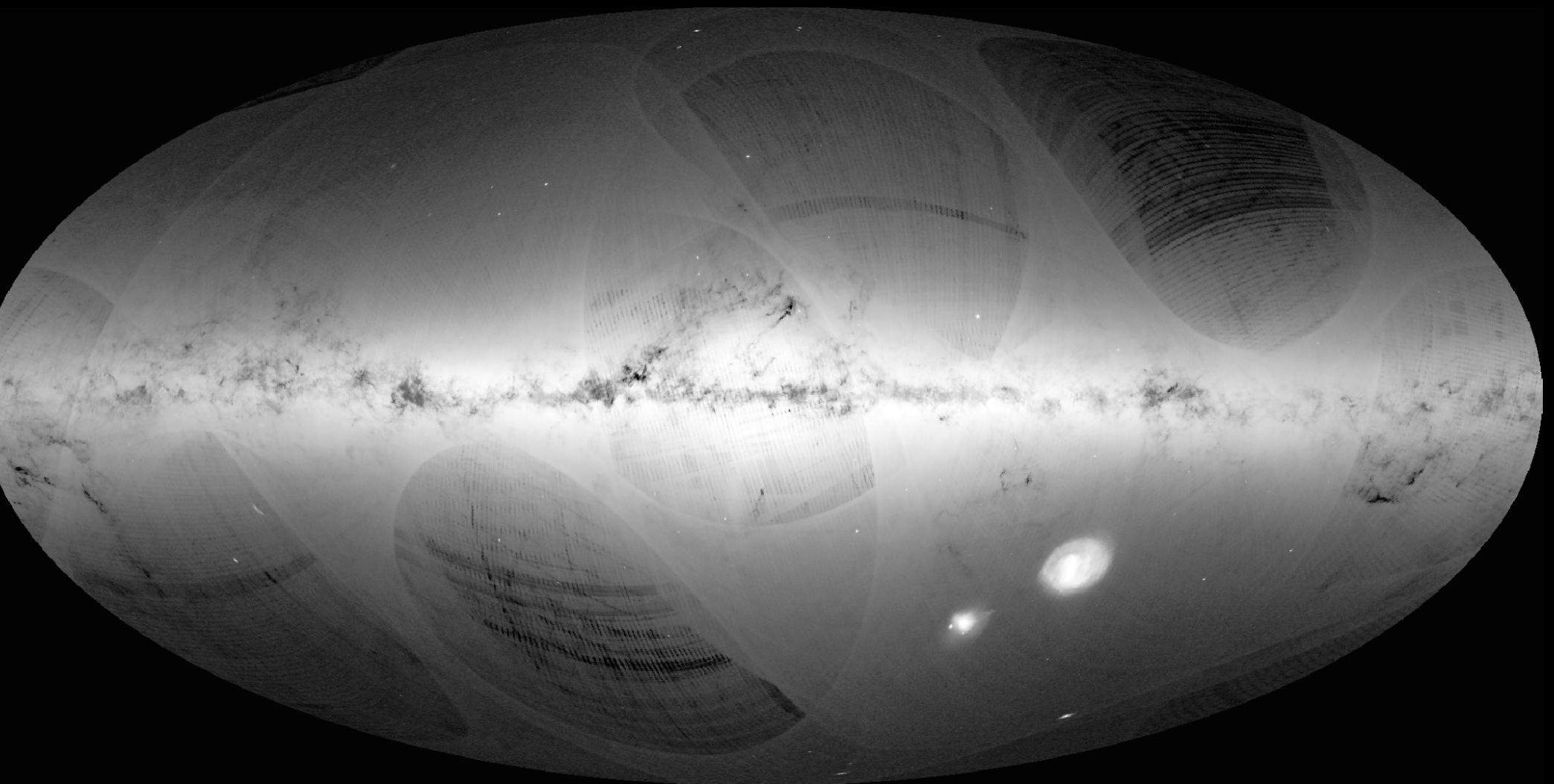


scatter

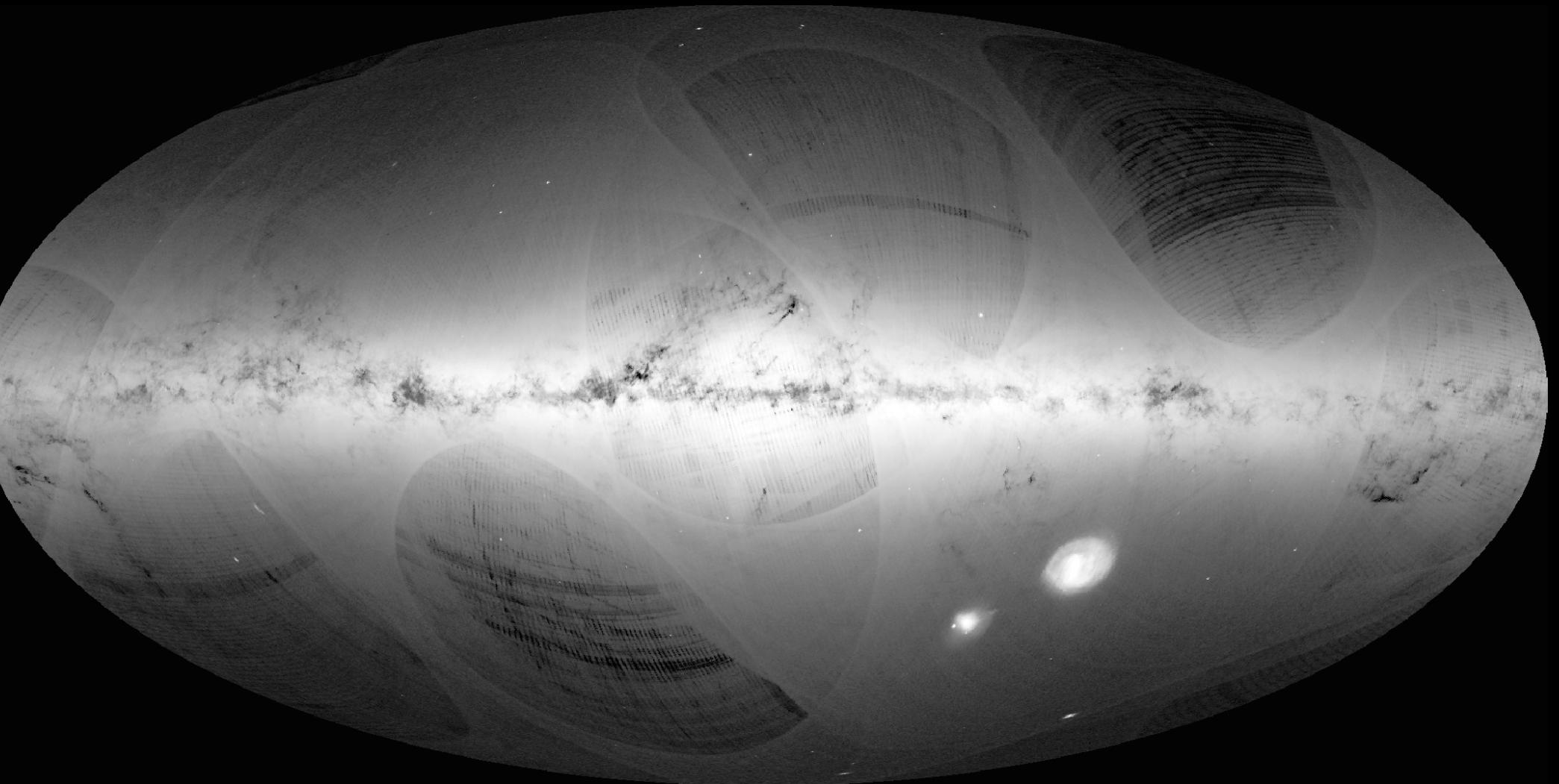


density

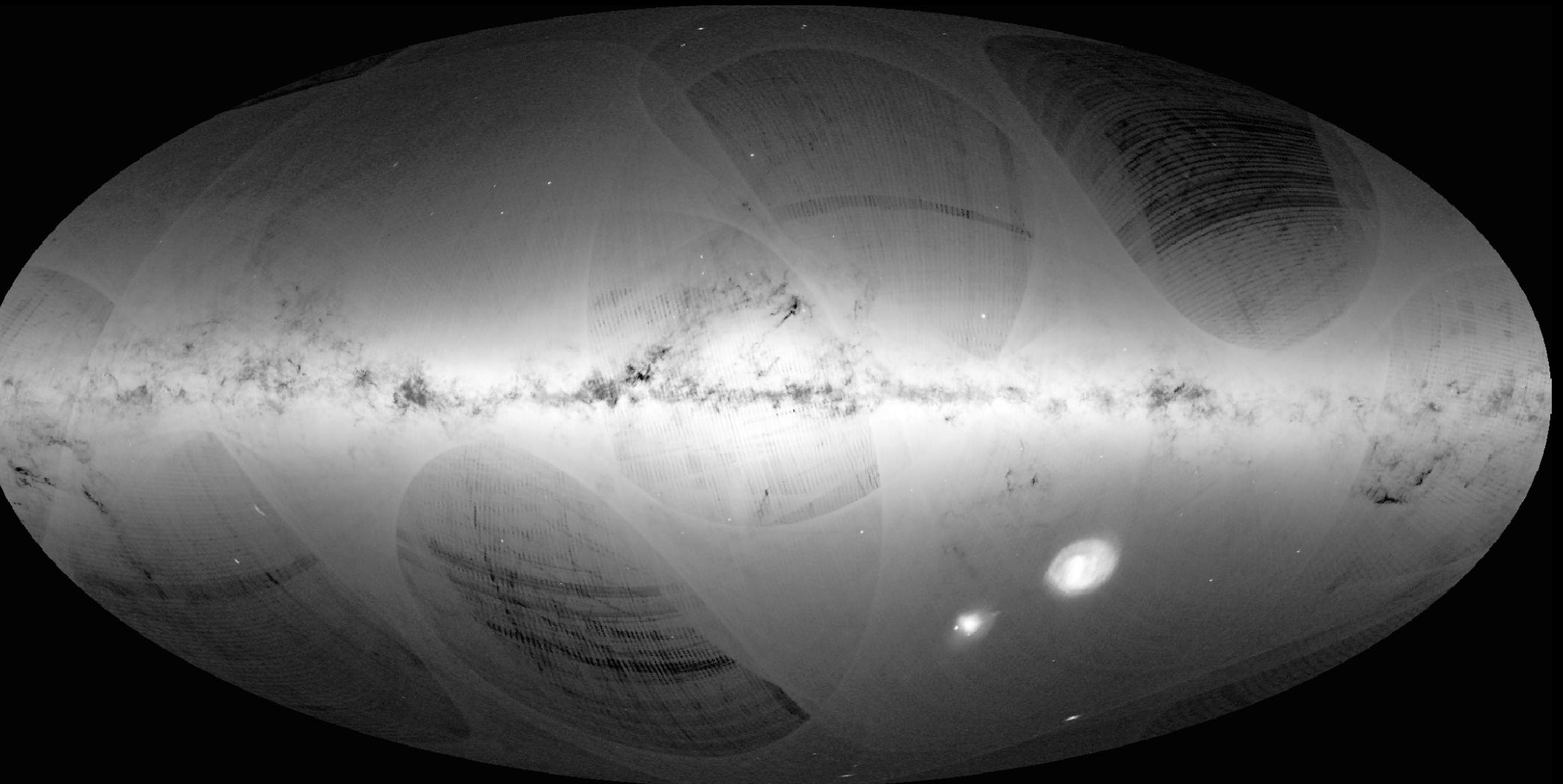
- How fast can it be done?



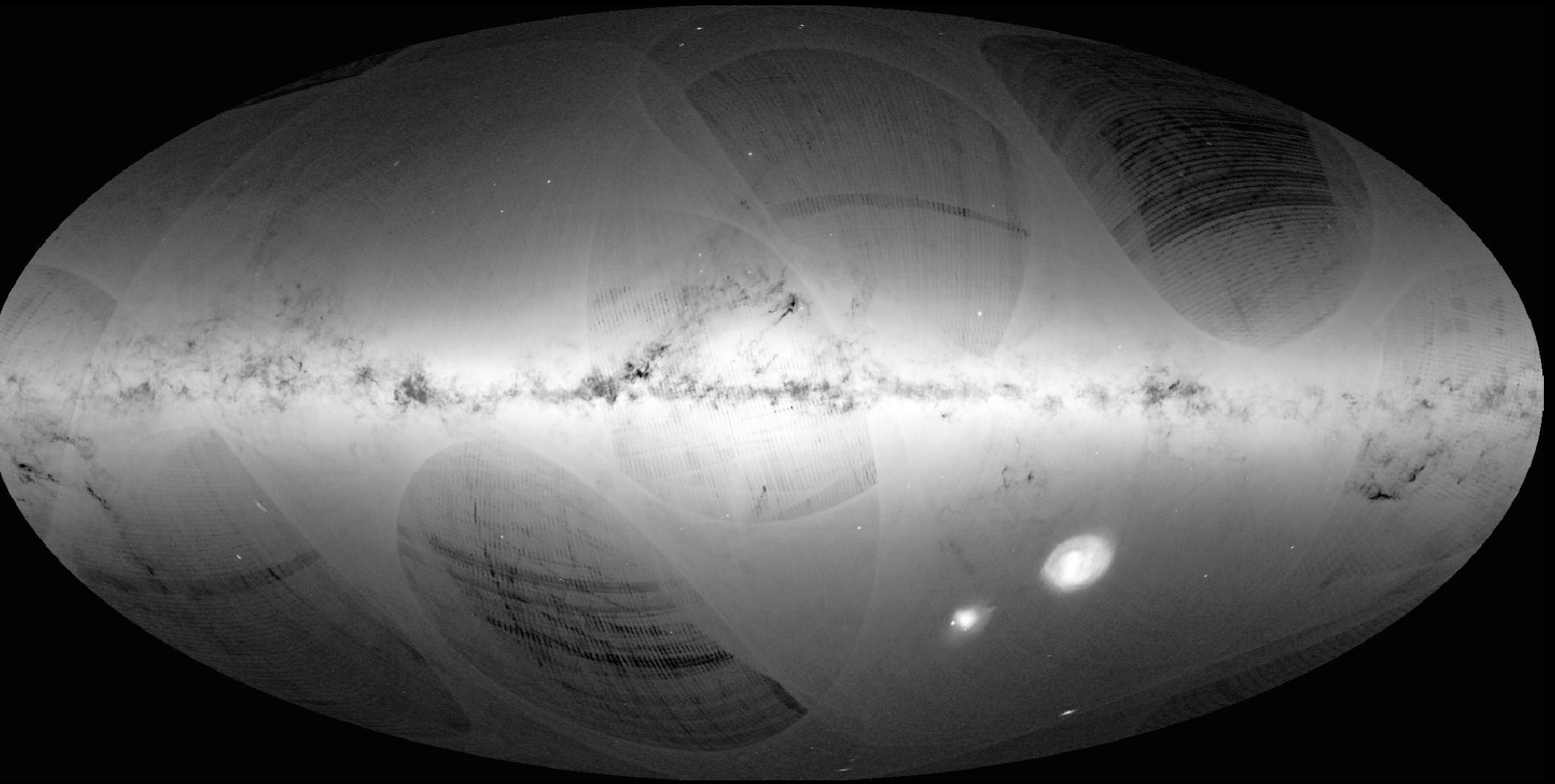
- How fast can it be done?
 - $10^9 * 2 * 8$ bytes = 15 GiB (double is 8 bytes)



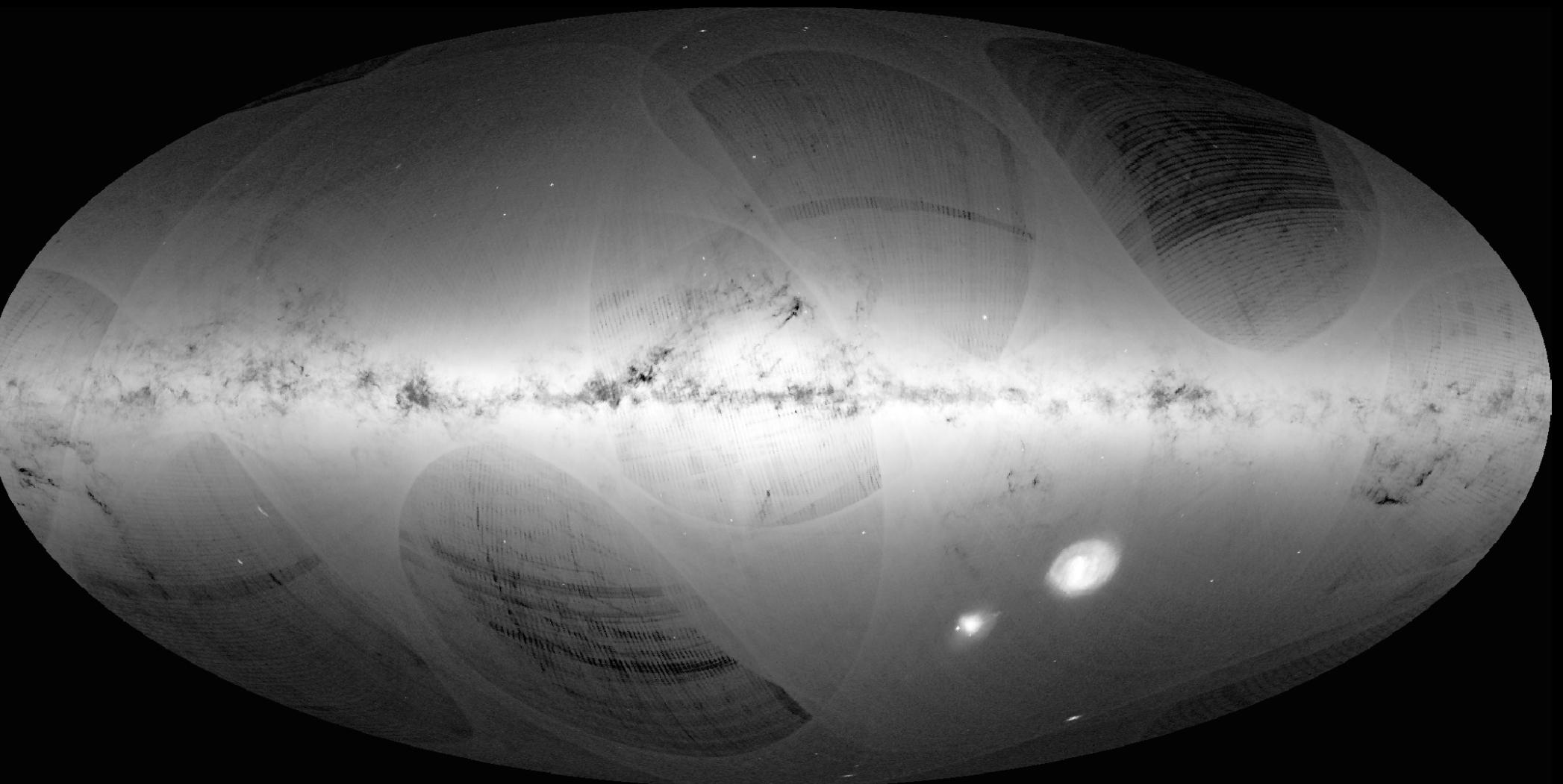
- How fast can it be done?
 - $10^9 * 2 * 8$ bytes = 15 GiB (double is 8 bytes)
 - Memory bandwidth: 10-20 GiB/s: ~1 second



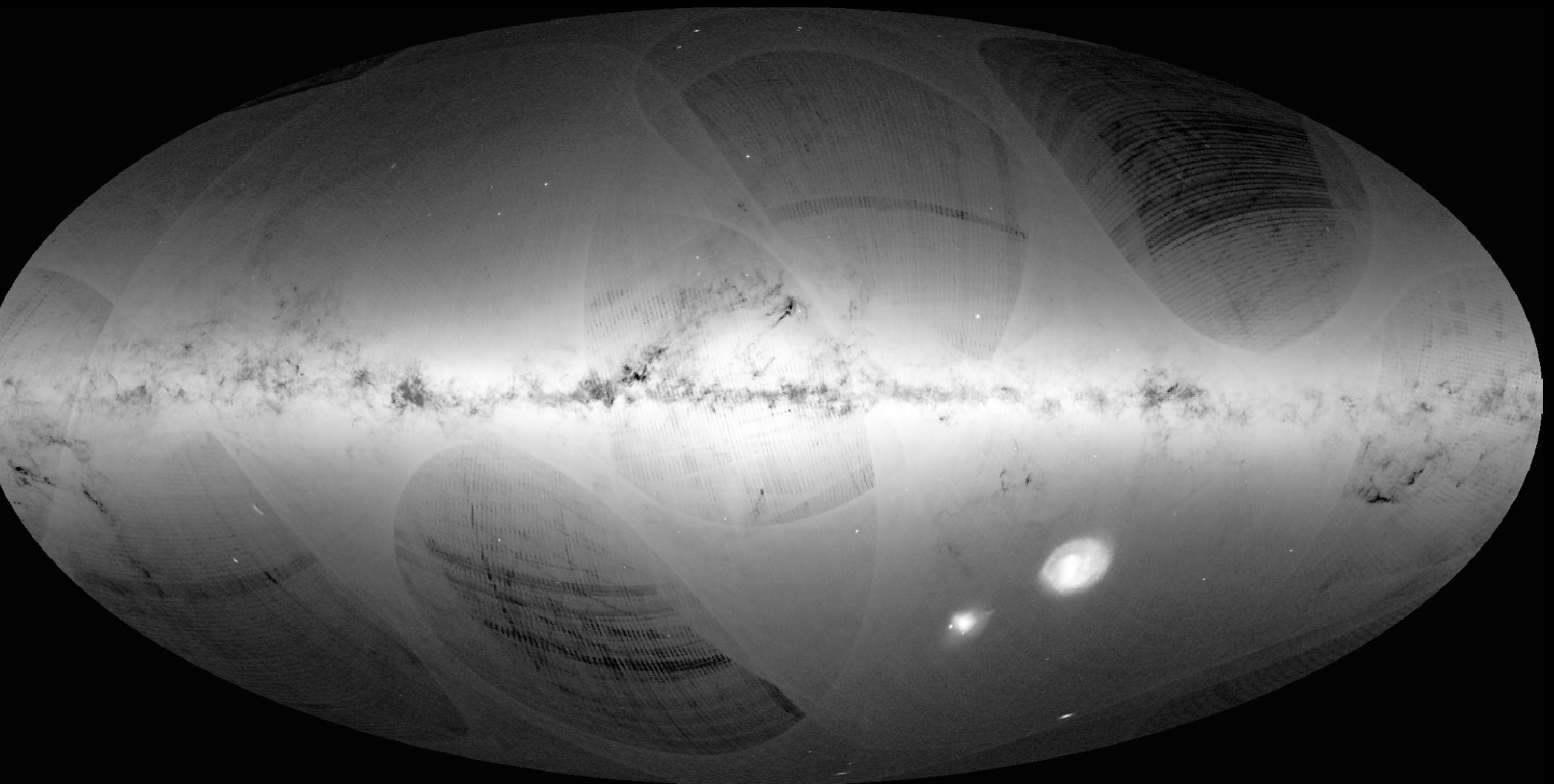
- How fast can it be done?
 - $10^9 * 2 * 8$ bytes = 15 GiB (double is 8 bytes)
 - Memory bandwidth: 10-20 GiB/s: ~1 second
 - CPU: 3 Ghz (but multicore, say 4-8): 12-24 cycles/second

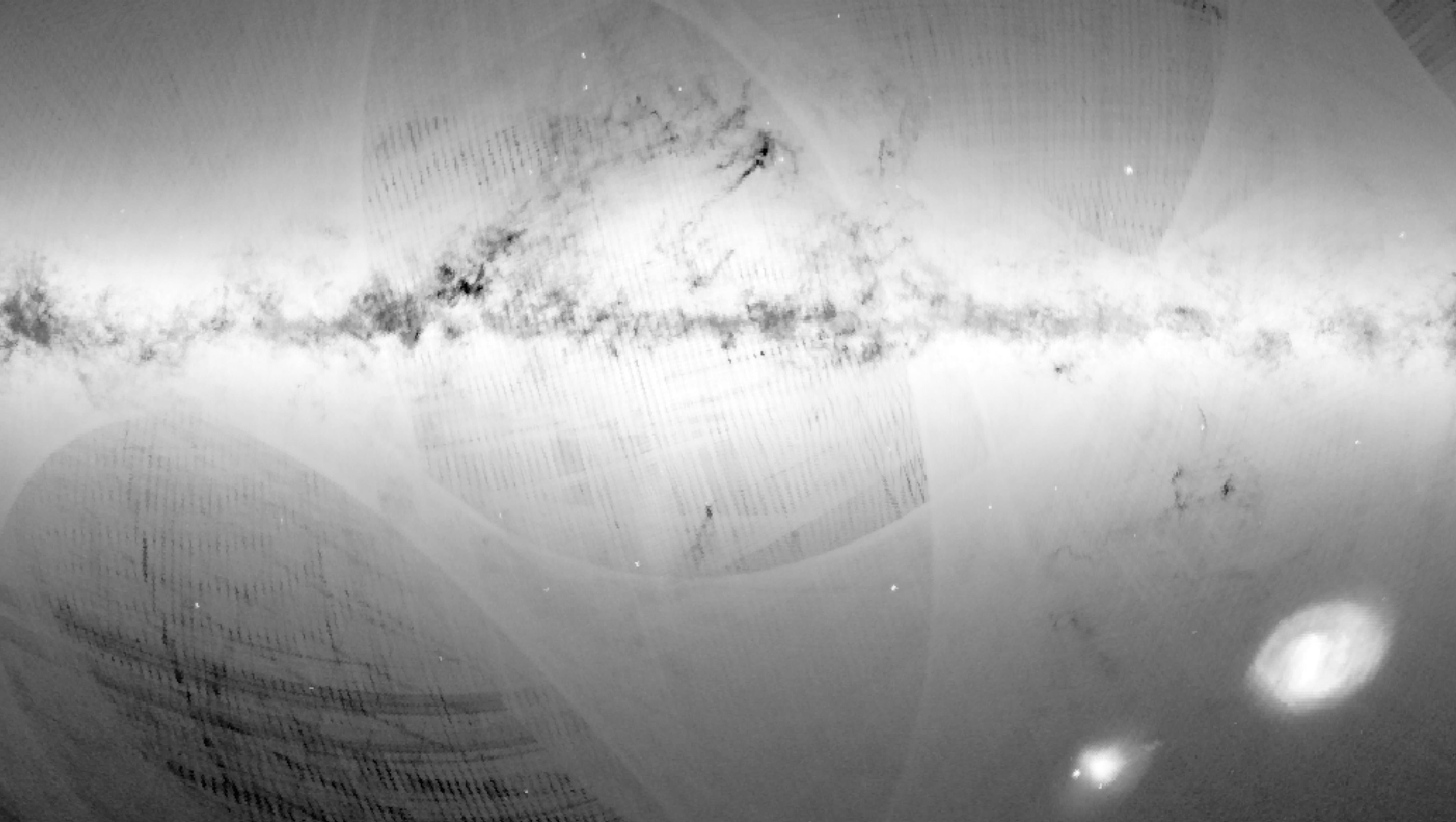


- How fast can it be done?
 - $10^9 * 2 * 8 \text{ bytes} = 15 \text{ GiB}$ (double is 8 bytes)
 - Memory bandwidth: 10-20 GiB/s: ~1 second
 - CPU: 3 Ghz (but multicore, say 4-8): 12-24 cycles/second
 - Few cycles per row/object, simple algorithm



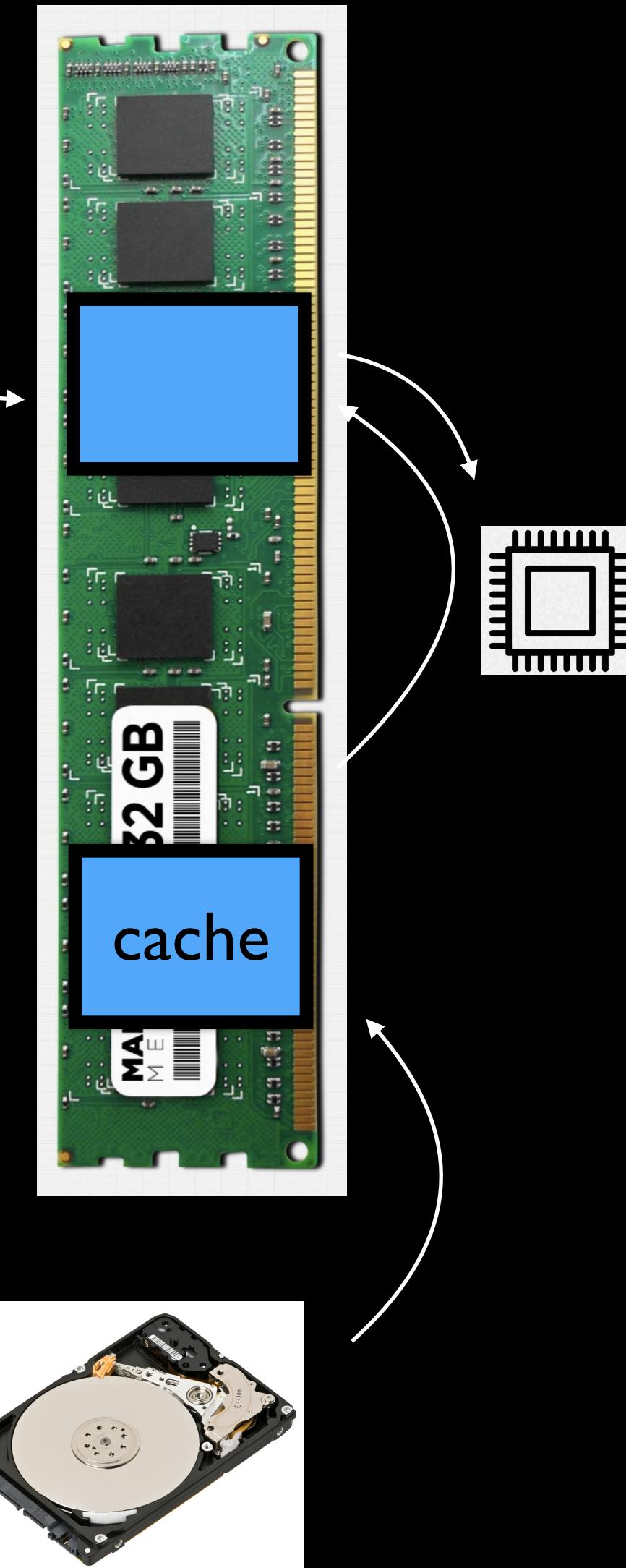
- How fast can it be done?
 - $10^9 * 2 * 8 \text{ bytes} = 15 \text{ GiB}$ (double is 8 bytes)
 - Memory bandwidth: 10-20 GiB/s: ~1 second
 - CPU: 3 Ghz (but multicore, say 4-8): 12-24 cycles/second
 - Few cycles per row/object, simple algorithm
 - Histograms/Density grids





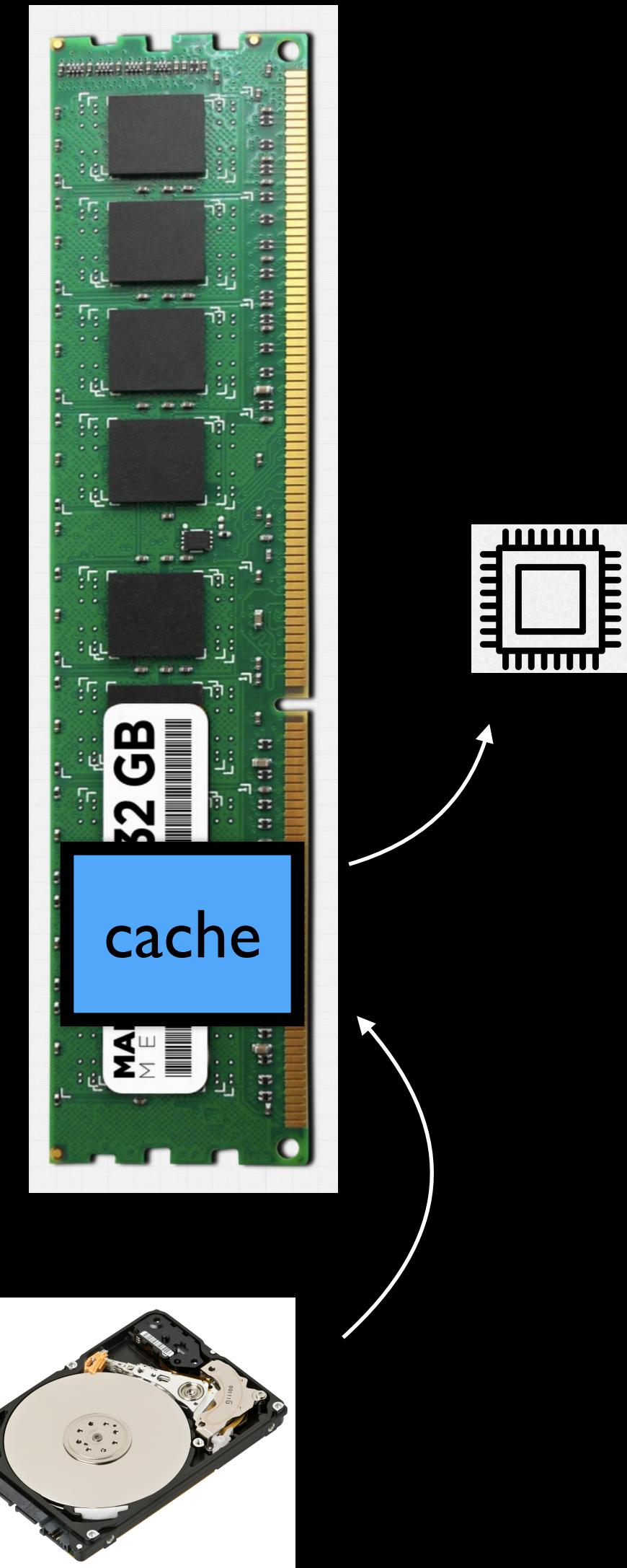
How to store and read the data

- Storage: native, column based (hdf5)
- Normal (POSIX read) method:
 - Allocate memory
 - read from disk to memory
 - Actually: from disk, to OS cache, to memory
 - Wastes memory/cache
 - 15 GB data , requires 30 GB if you want to use the file system cache



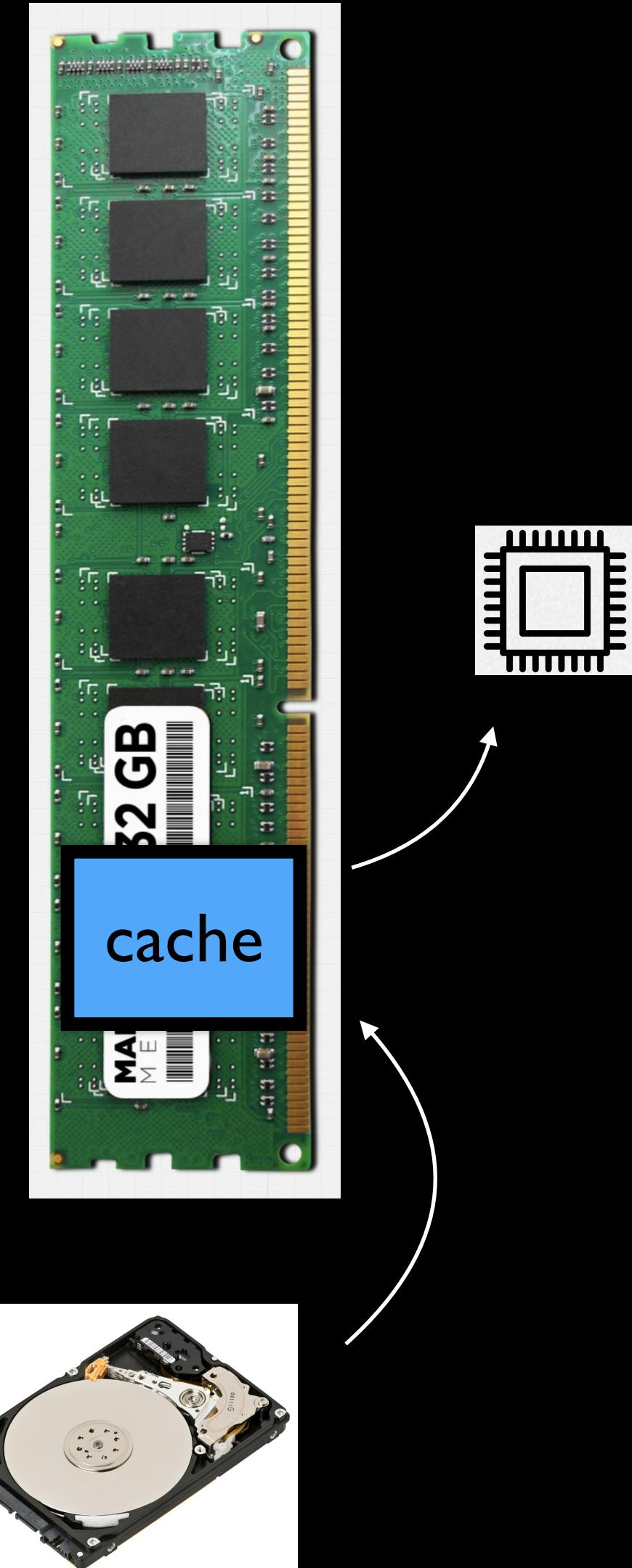
How to store and read the data

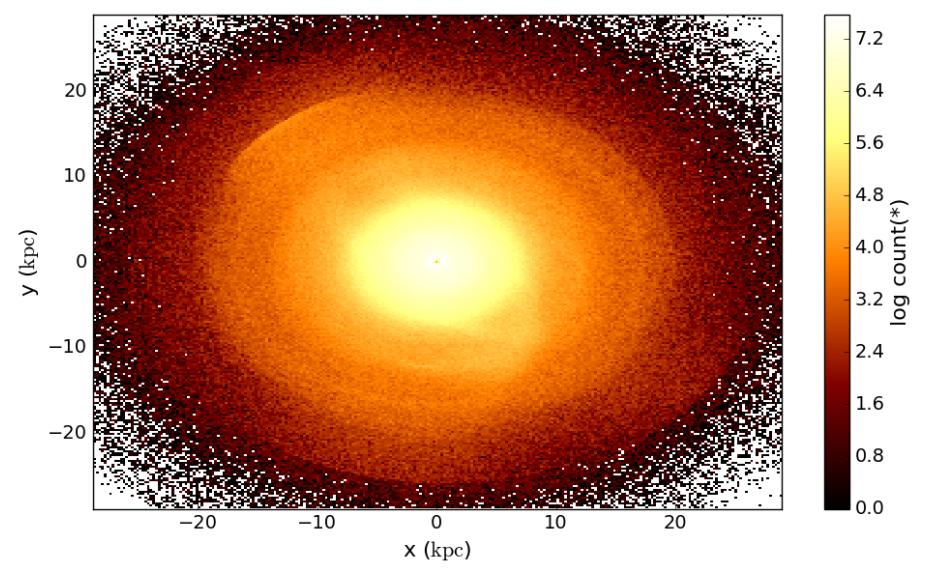
- Storage: native, column based (hdf5)
- Normal (POSIX read) method:
 - Allocate memory
 - read from disk to memory
 - Actually: from disk, to OS cache, to memory
 - Wastes memory/cache
 - 15 GB data , requires 30 GB if you want to use the file system cache



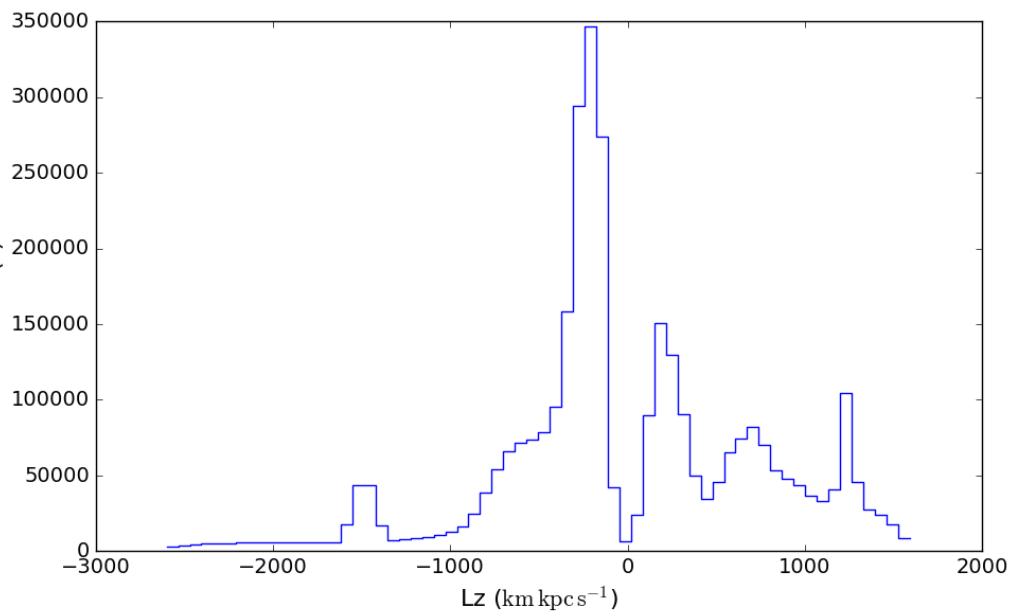
How to store and read the data

- Storage: native, column based (hdf5)
- Normal (POSIX read) method:
 - Allocate memory
 - read from disk to memory
 - Actually: from disk, to OS cache, to memory
 - Wastes memory/cache
 - 15 GB data , requires 30 GB if you want to use the file system cache
- Memory mapping:
 - get direct access to OS memory cache, no copy, no setup (apart from the kernel doing setting up the pages)
 - avoid memory copies, more cache available
- In previous example:
 - copying 15 GB will take about ~1.0 second, at 10-20 GB/s
 - Can be 2-3x slower (cpu cache helps a bit)

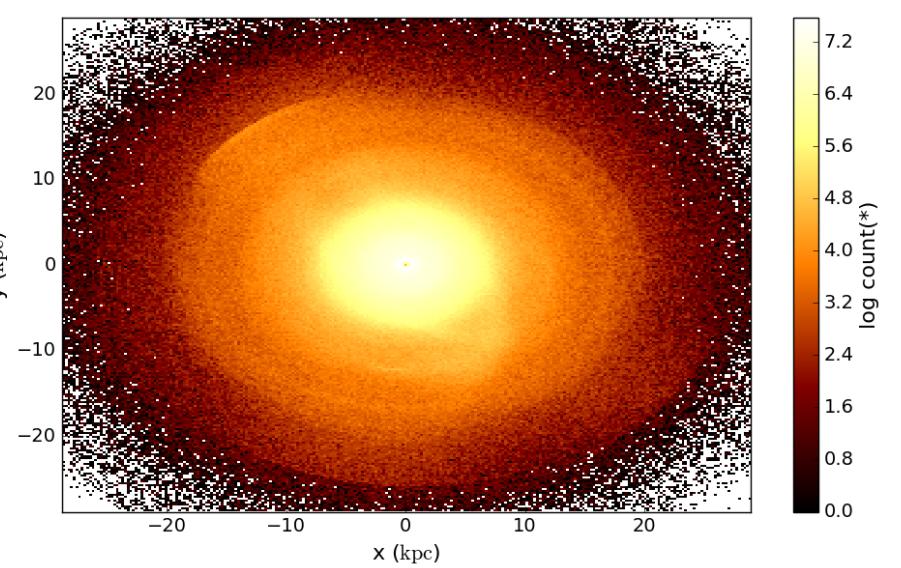




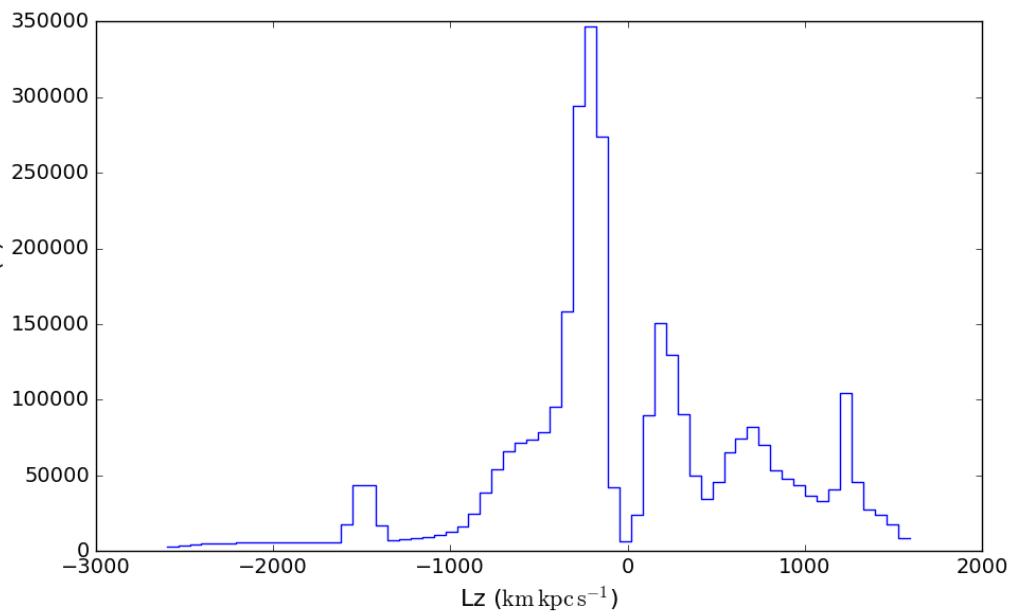
1d



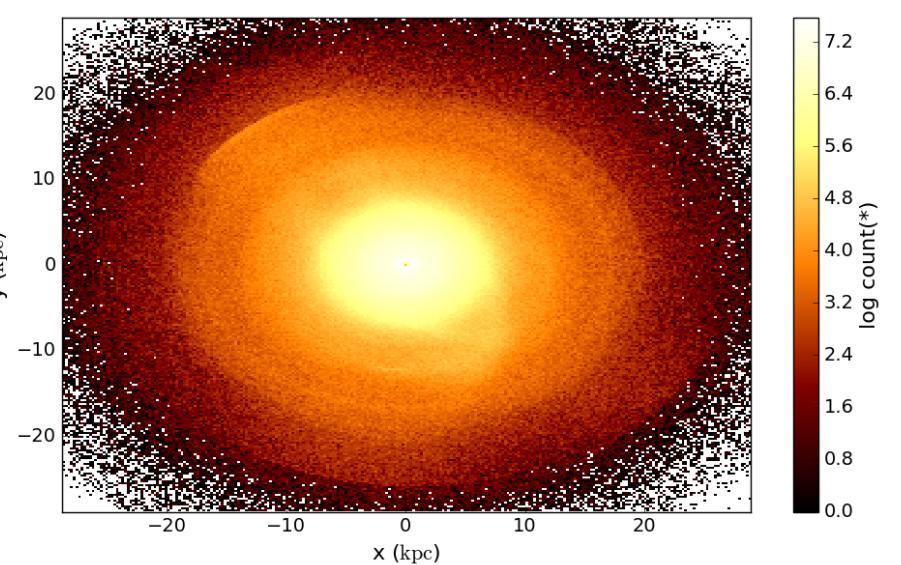
2d



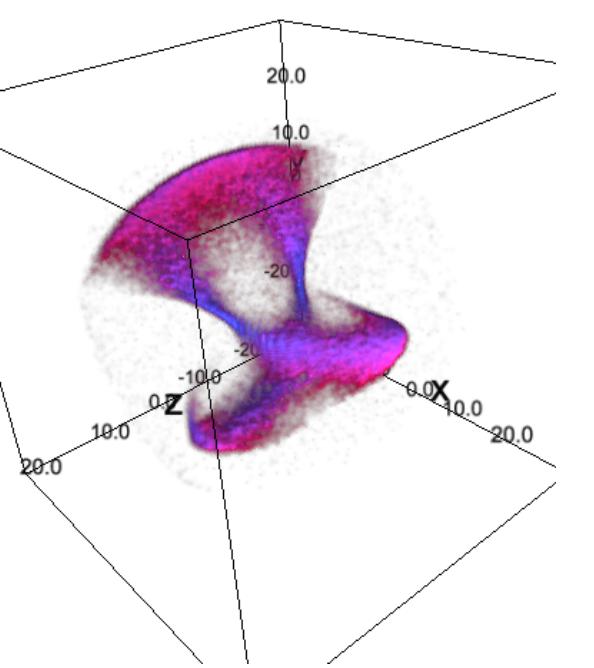
1d



2d



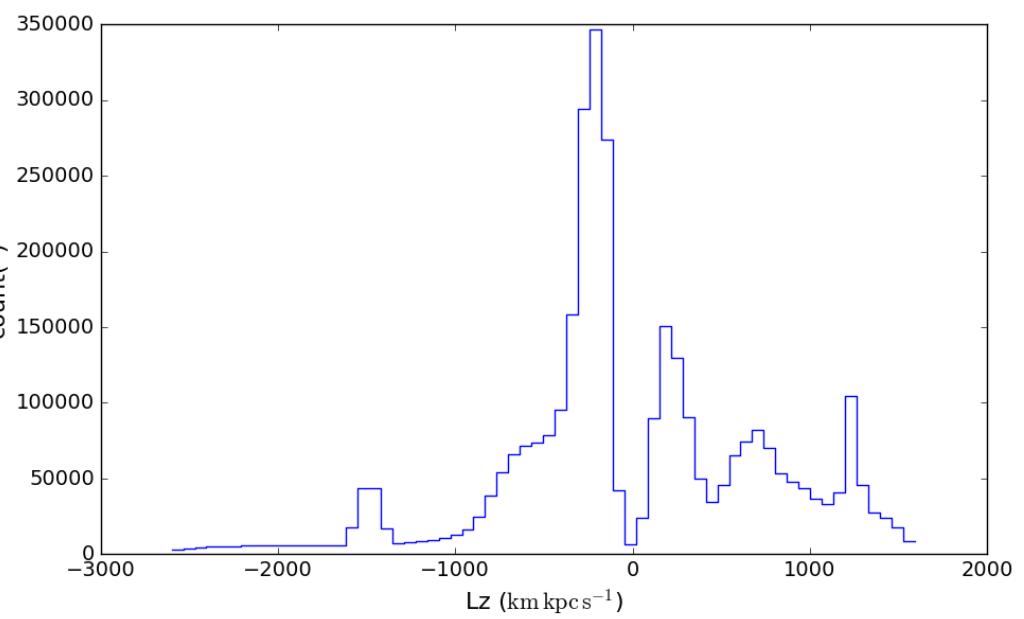
3d



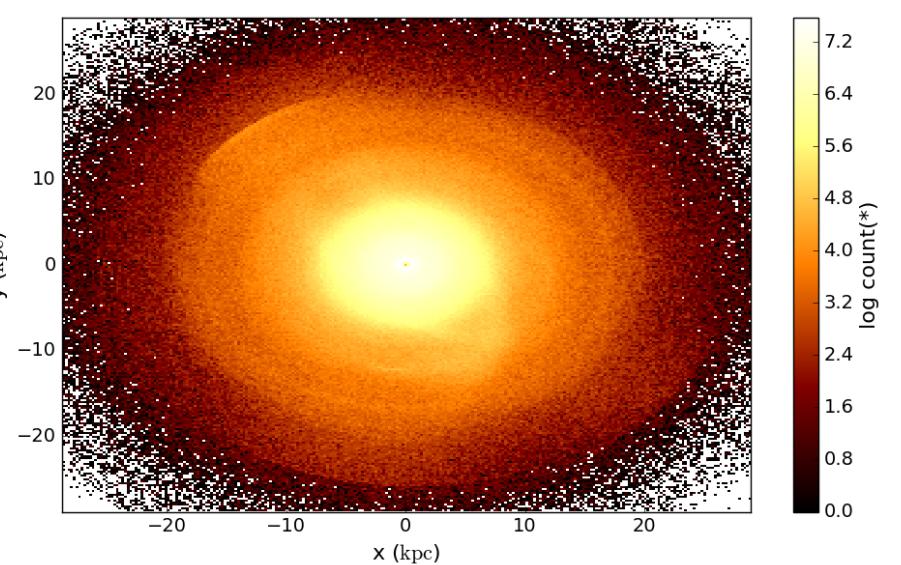
0d

330,000 rows

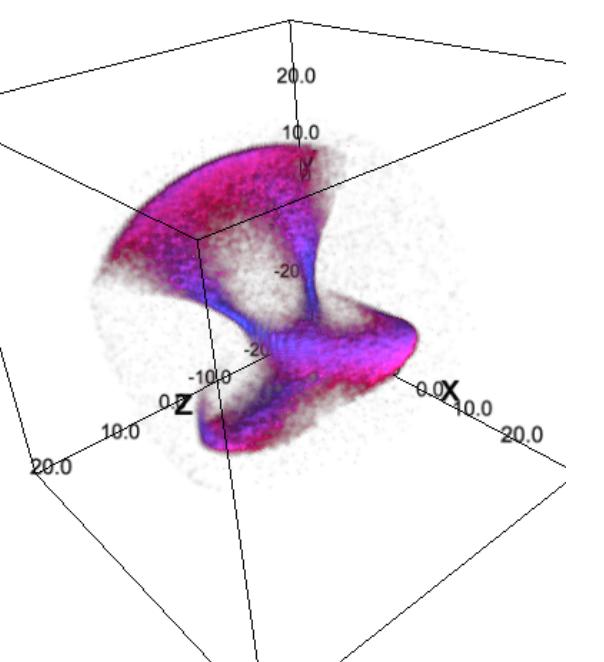
1d



2d



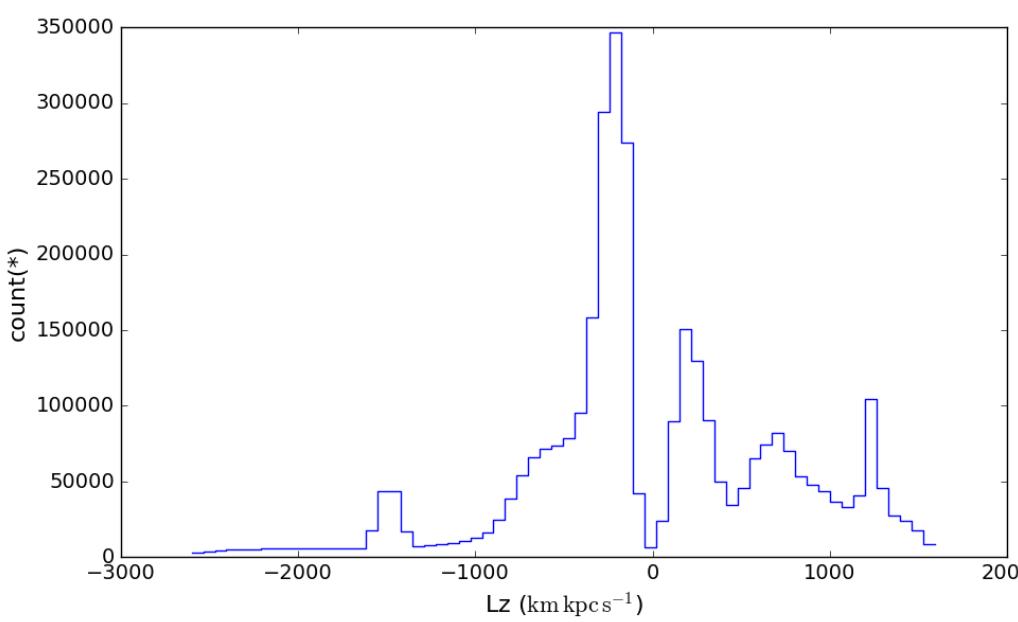
3d



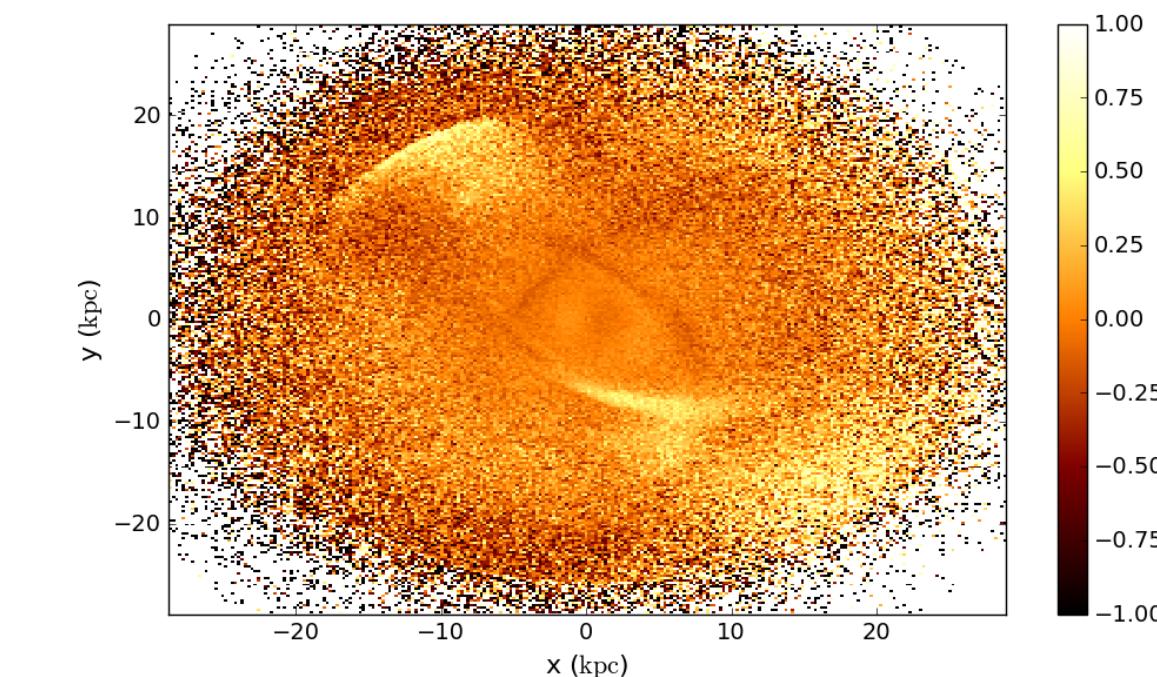
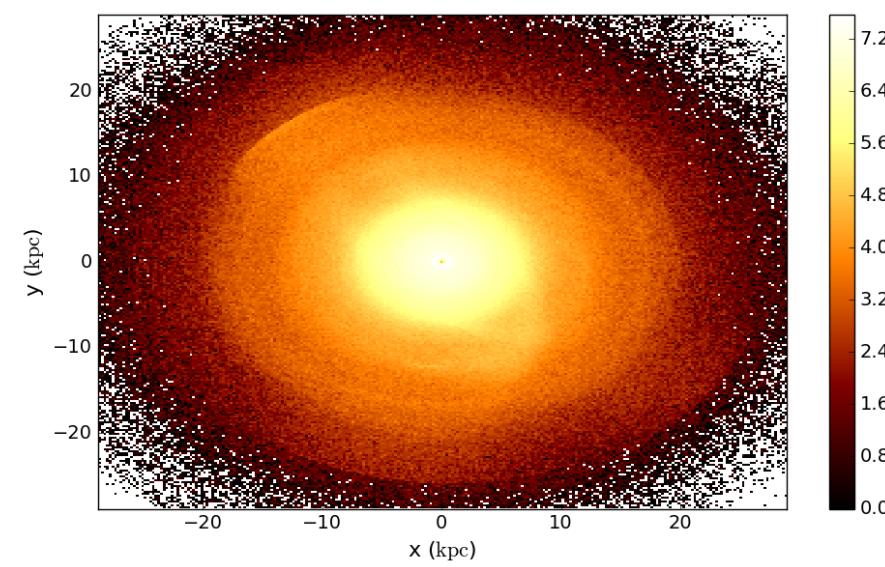
0d

330,000 rows

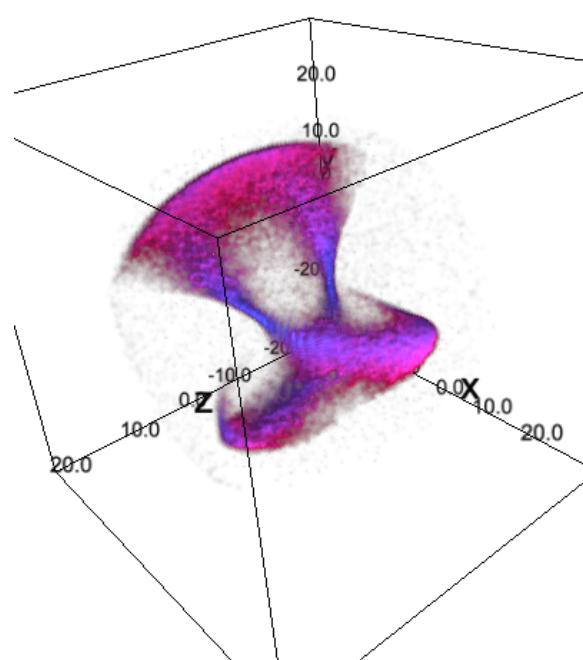
1d



2d



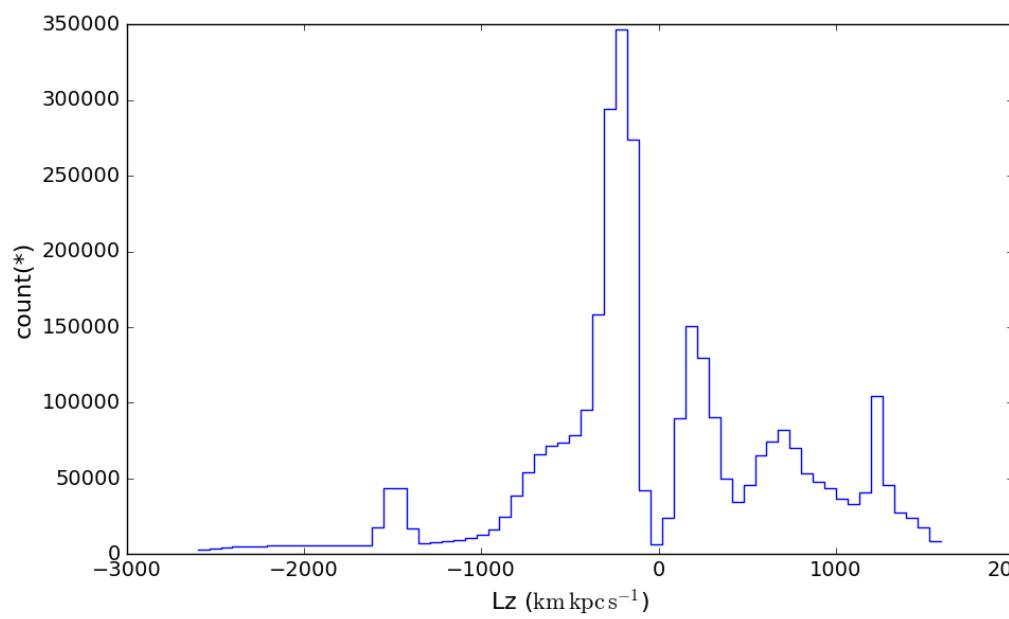
3d



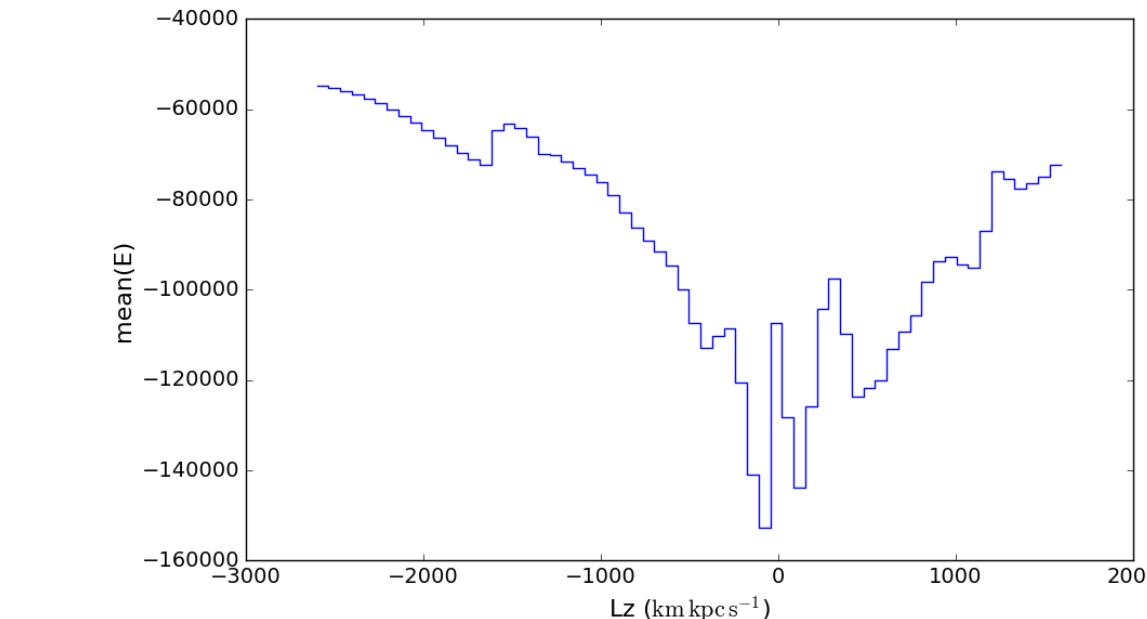
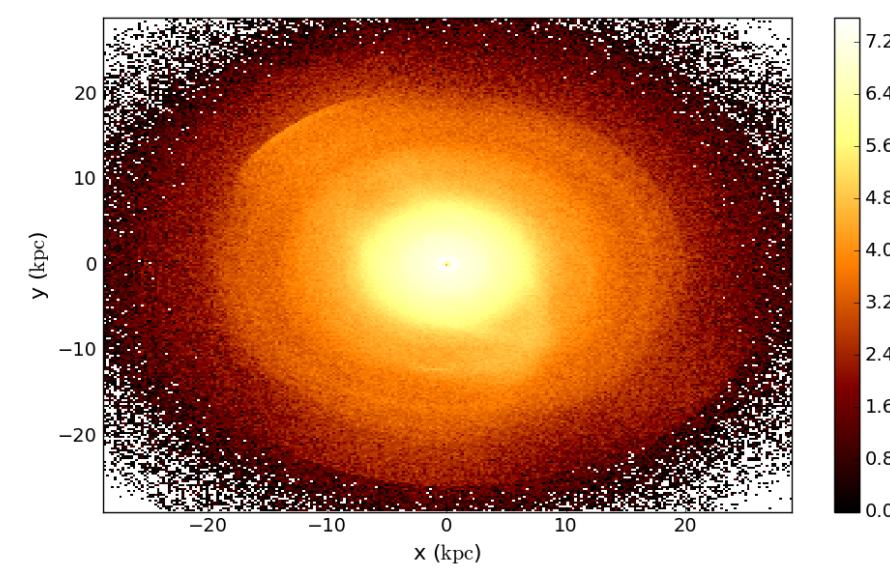
0d

330,000 rows

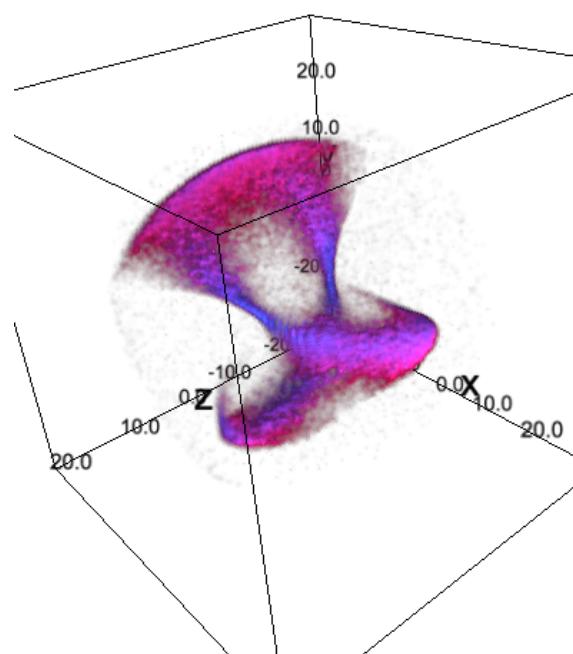
1d



2d



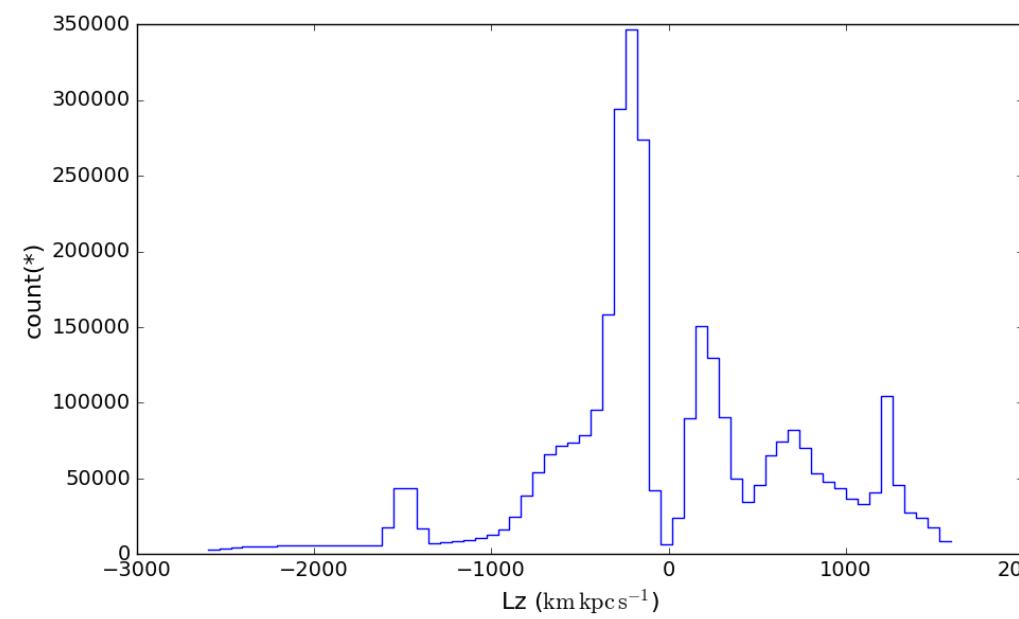
3d



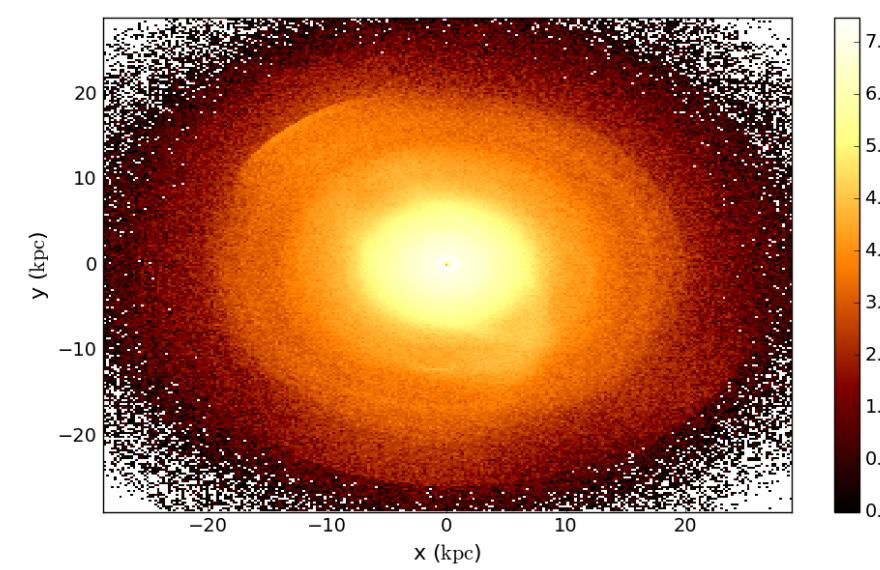
0d

330,000 rows

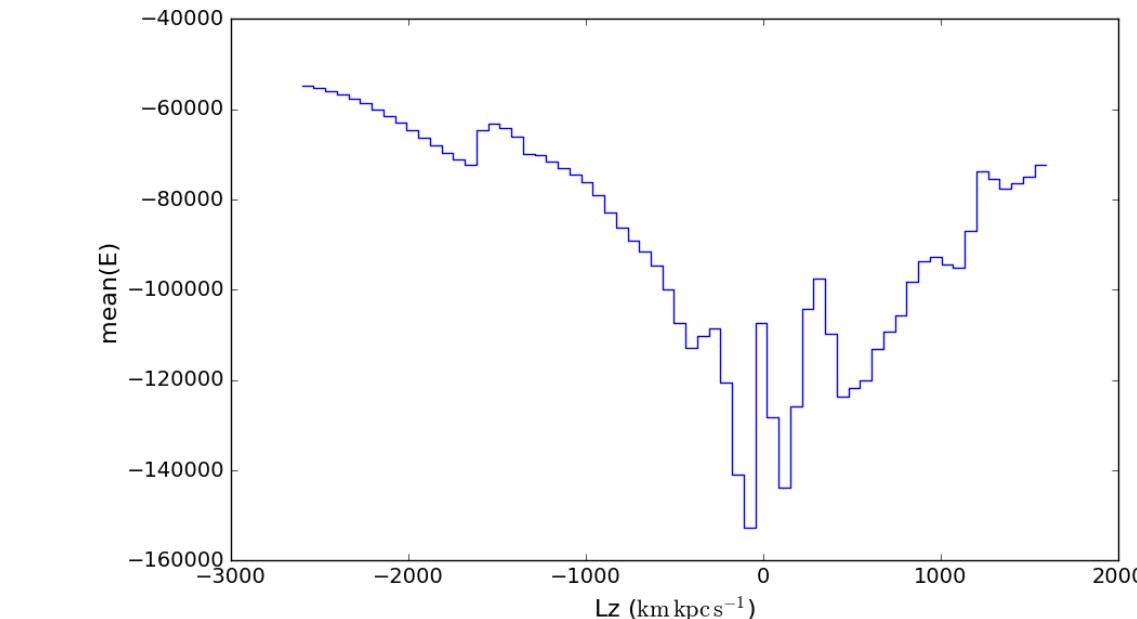
1d



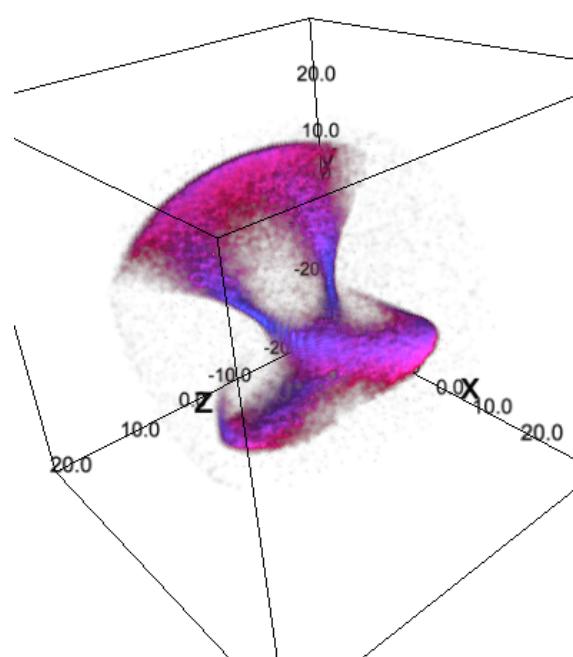
2d



mean: -0.083



3d

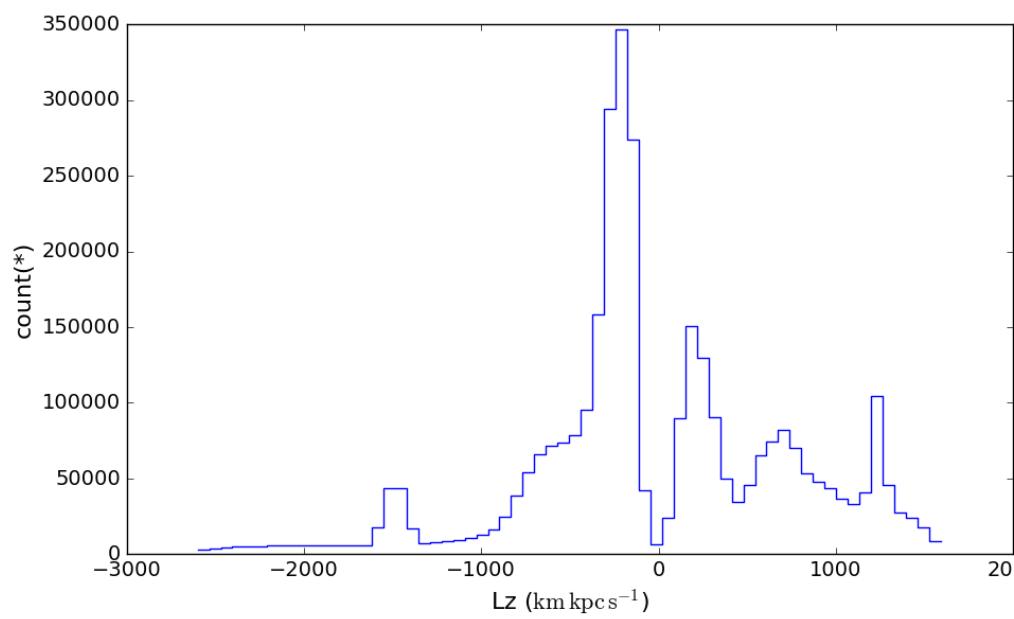


0d

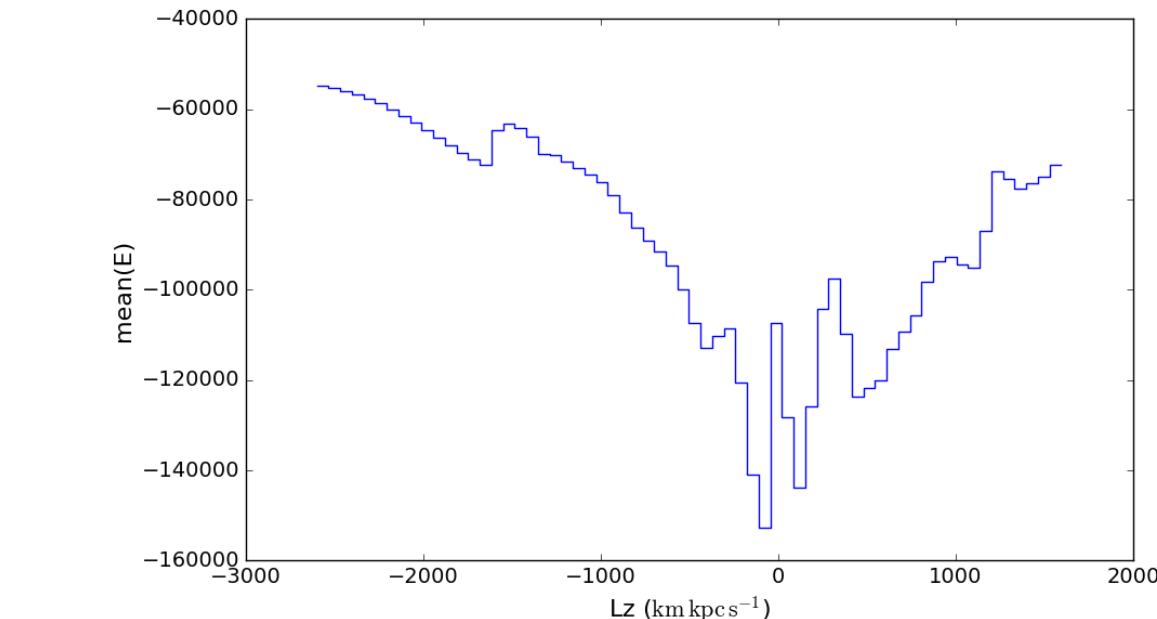
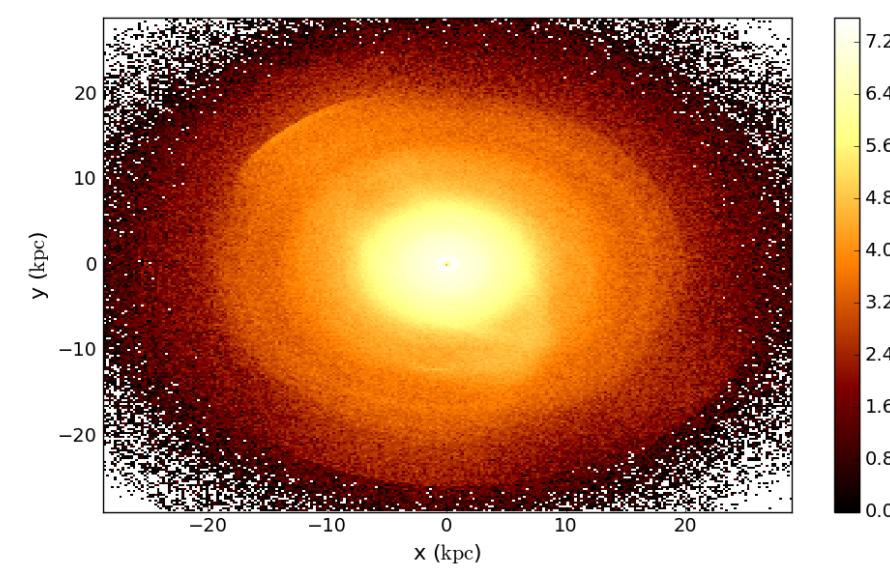
330,000 rows

mean: -0.083

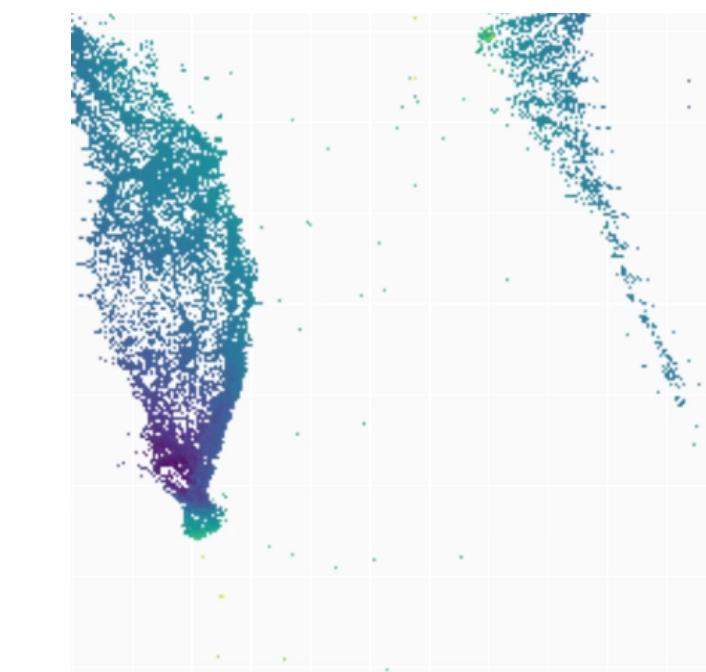
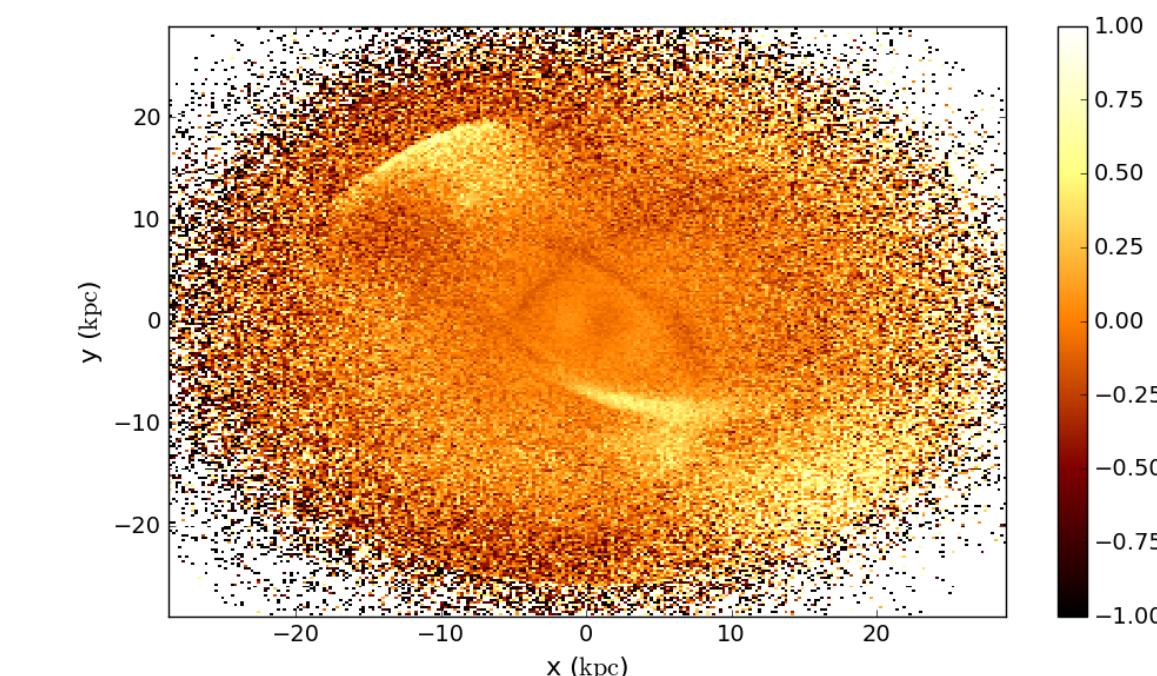
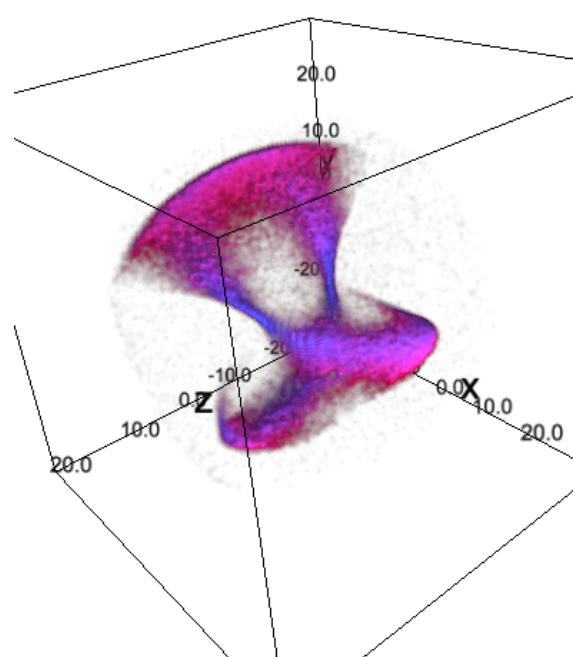
1d



2d



3d



vaex

vaex

- Python library (conda/pip installable)

vaex

- Python library (conda/pip installable)
- Think pandas-like but for large datasets

vaex

- Python library (conda/pip installable)
- Think pandas-like but for large datasets
- Focuses mostly on statistics on N-d grids (count/mean/max/std/...)

vaex

- Python library (conda/pip installable)
- Think pandas-like but for large datasets
- Focuses mostly on statistics on N-d grids (count/mean/max/std/...)
- >1 billion rows / sec on a `decent` desktop (quad core 3Gz)
 - >50x faster than `scipy.stats.binned_statistic_2d`

vaex

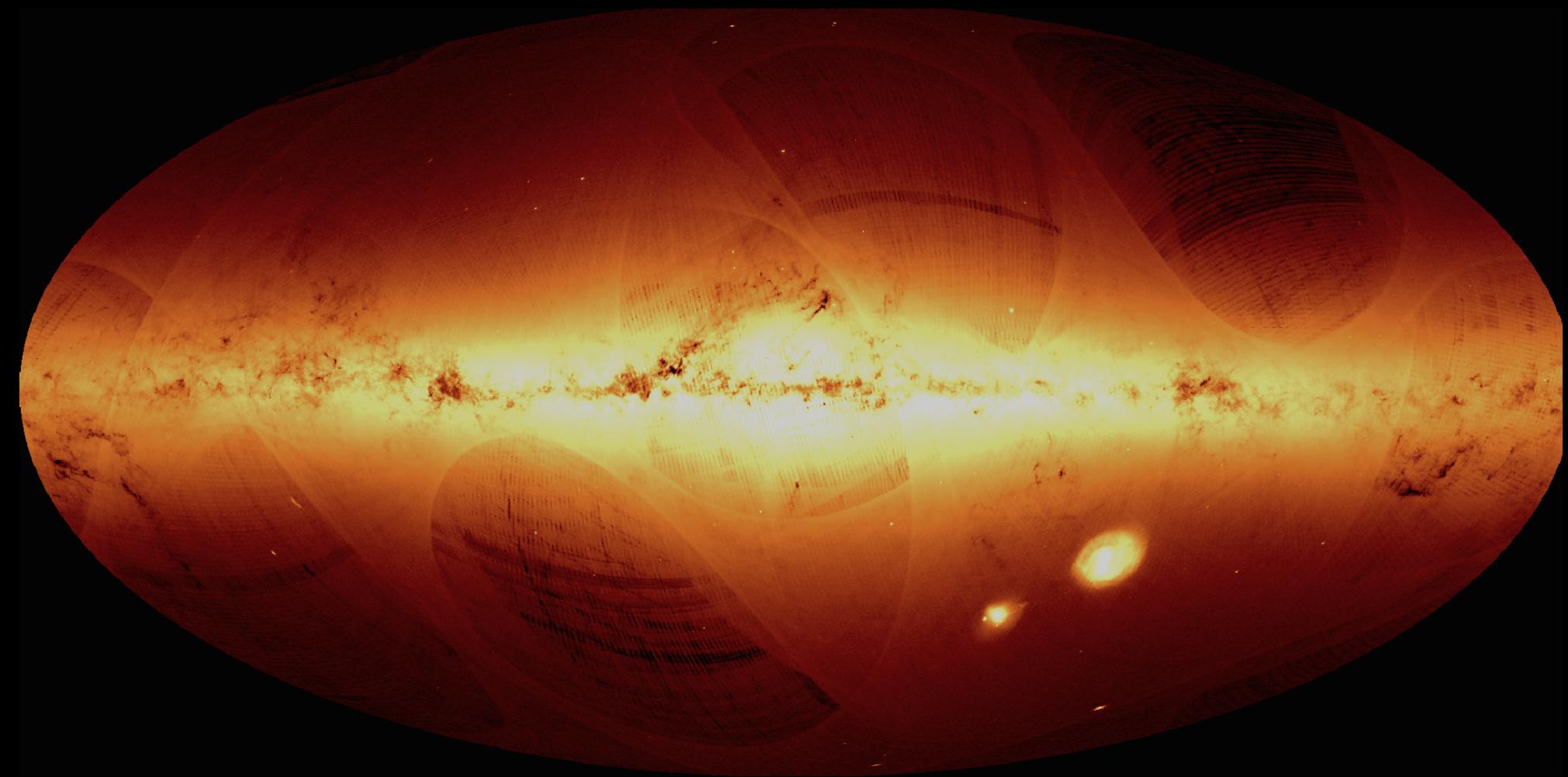
- Python library (conda/pip installable)
- Think pandas-like but for large datasets
- Focusses mostly on statistics on N-d grids (count/mean/max/std/...)
- >1 billion rows / sec on a `decent` desktop (quad core 3Gz)
 - >50x faster than `scipy.stats.binned_statistic_2d`
- 0.1-0.2 billion rows on this 2yr old Macbook Air (CPU bound)

vaex

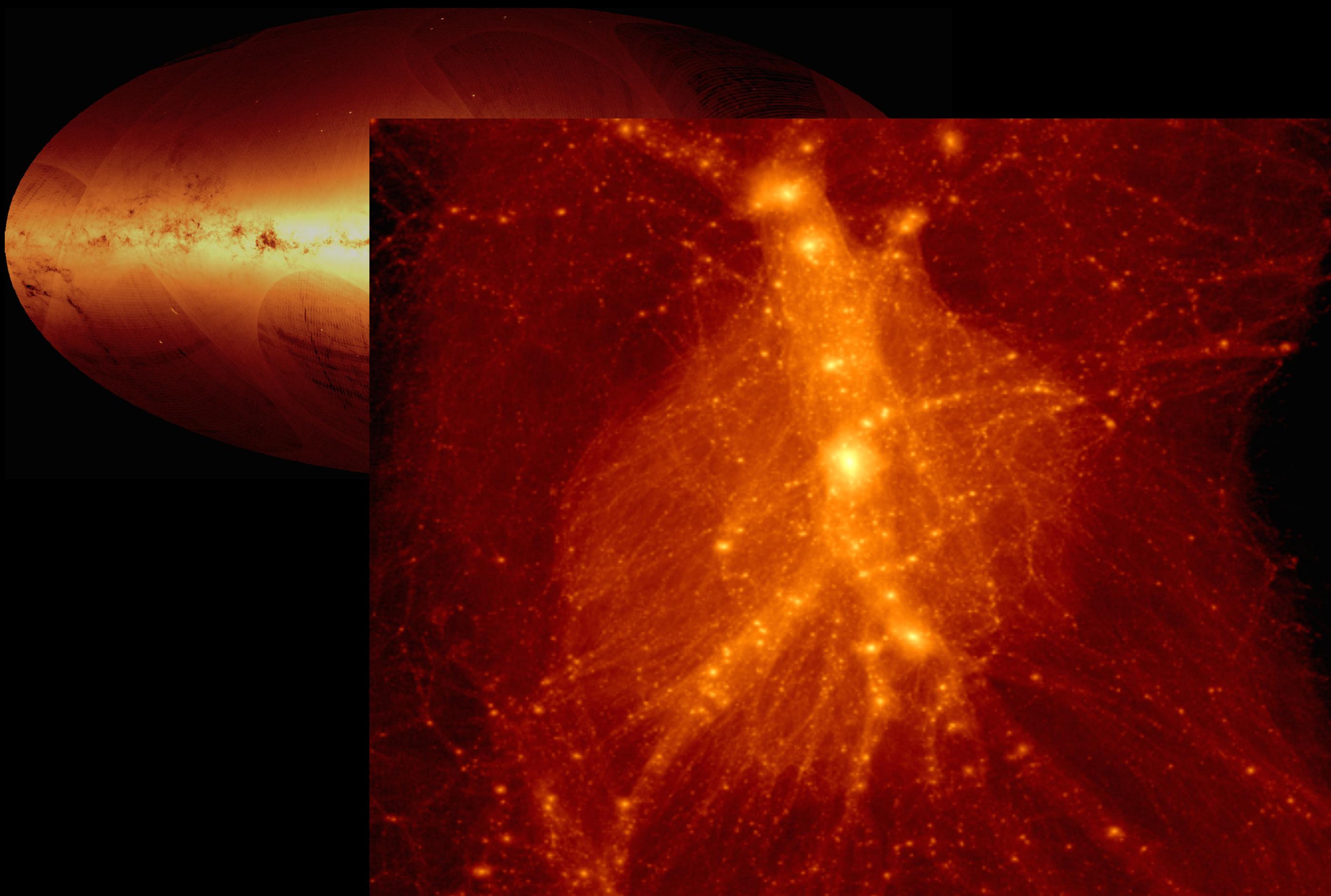
- Python library (conda/pip installable)
- Think pandas-like but for large datasets
- Focusses mostly on statistics on N-d grids (count/mean/max/std/...)
- >1 billion rows / sec on a `decent` desktop (quad core 3Gz)
 - >50x faster than `scipy.stats.binned_statistic_2d`
- 0.1-0.2 billion rows on this 2yr old Macbook Air (CPU bound)
- Does visualisation / matplotlib / bqplot / ipyvolume / ipyleaflet

What kind of data?

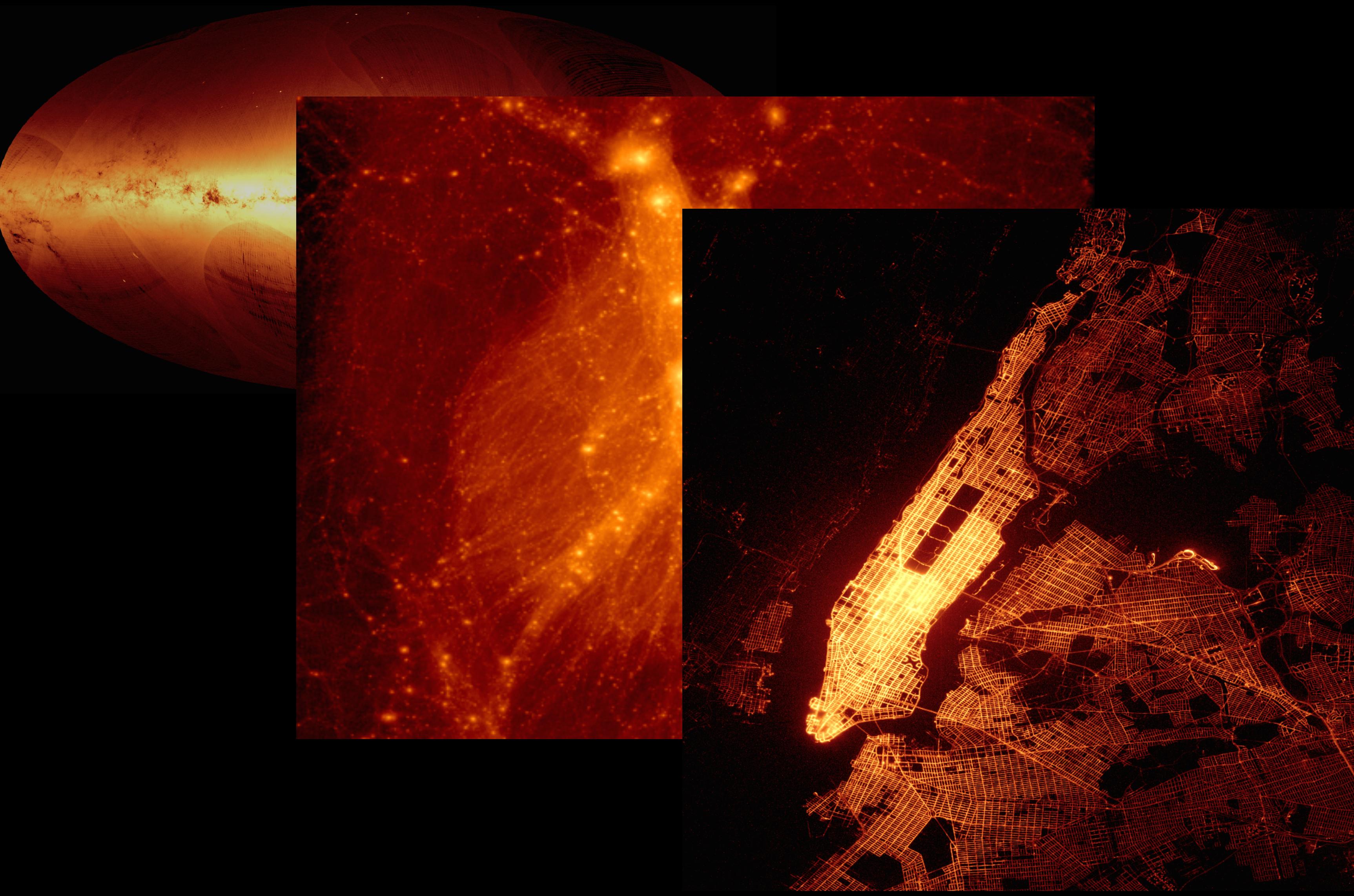
What kind of data?



What kind of data?



What kind of data?



“Never do a live demo”

-Many people

Answers

- How to deal with a billion objects/rows/stars?
 - statistics in N-d grids / vaex
- How you can do 3d visualization in the notebook?
 - ipyvolume

- maartenbreddels@gmail.com
- [twitter @maartenbreddels](https://twitter.com/maartenbreddels)
- vaex
 - <http://vaex.astro.rug.nl>
 - <https://github.com/maartenbreddels/vaex>
 - pip install —pre vaex / conda install -c conda-forge vaex
- ipyvolume
 - <https://ipywidgets.readthedocs.io>
 - <https://github.com/maartenbreddels/ipyvolume>
 - pip install ipyvolume / conda install -c conda-forge ipyvolume
 - if pip: jupyter nbextension enable --py --user ipyvolume (but really, use anaconda)
- Material:
 - <https://github.com/maartenbreddels/jupytercon-2017>