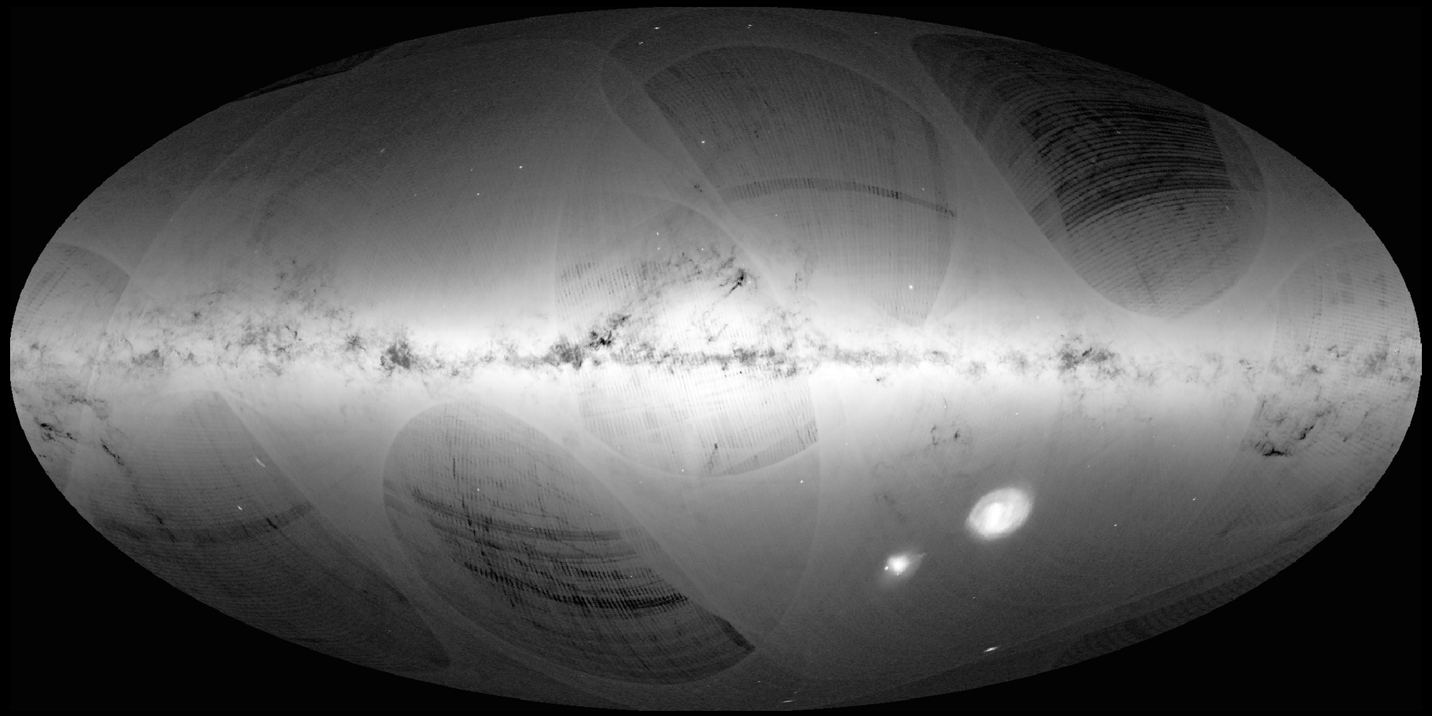
A Billion stars in the Jupyter notebook



@MaartenBreddels

PyData - Paris - Meetup - 2017

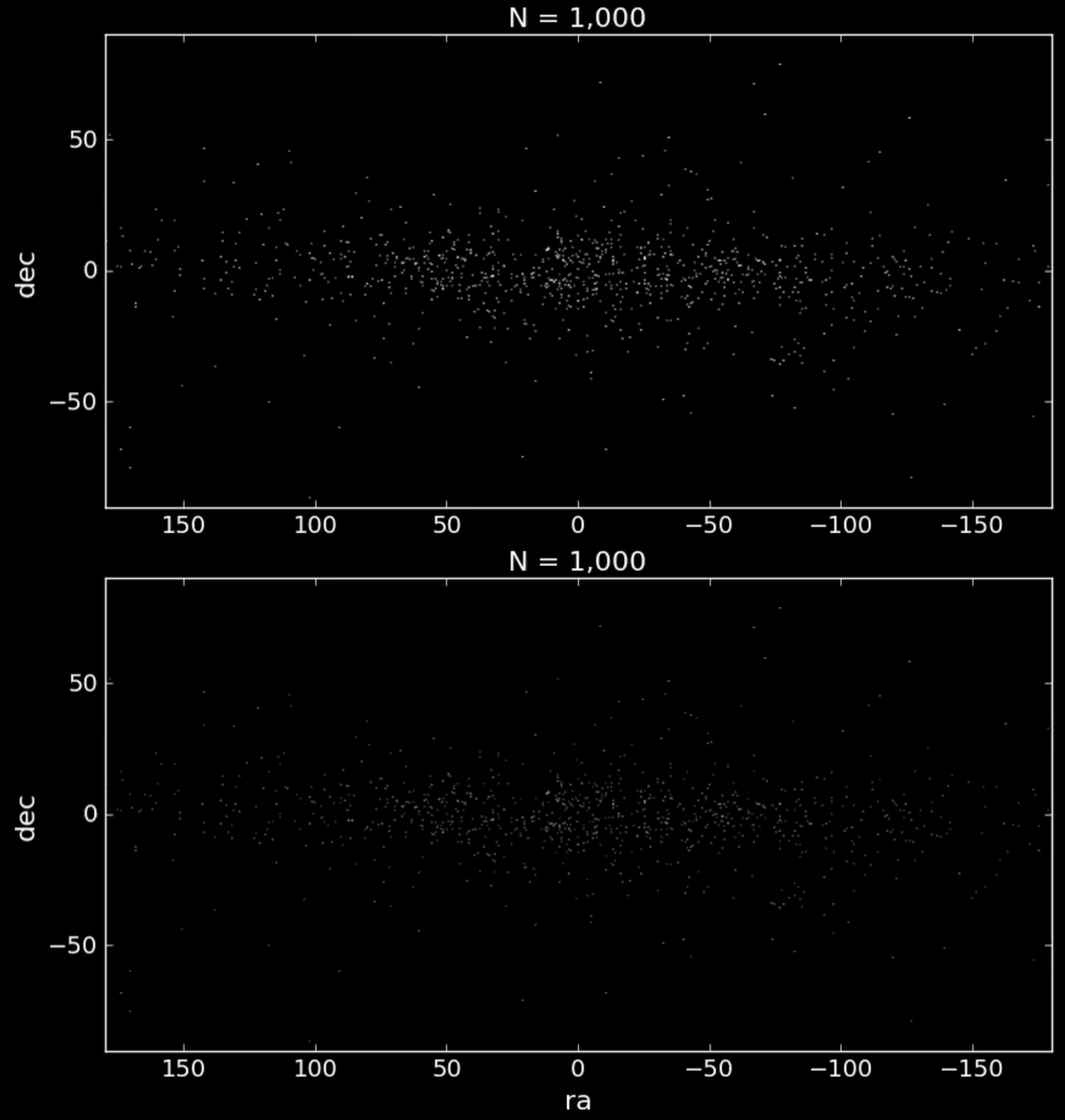
university of groningen / faculty of mathematics and natural sciences / kapteyn astronomical institute

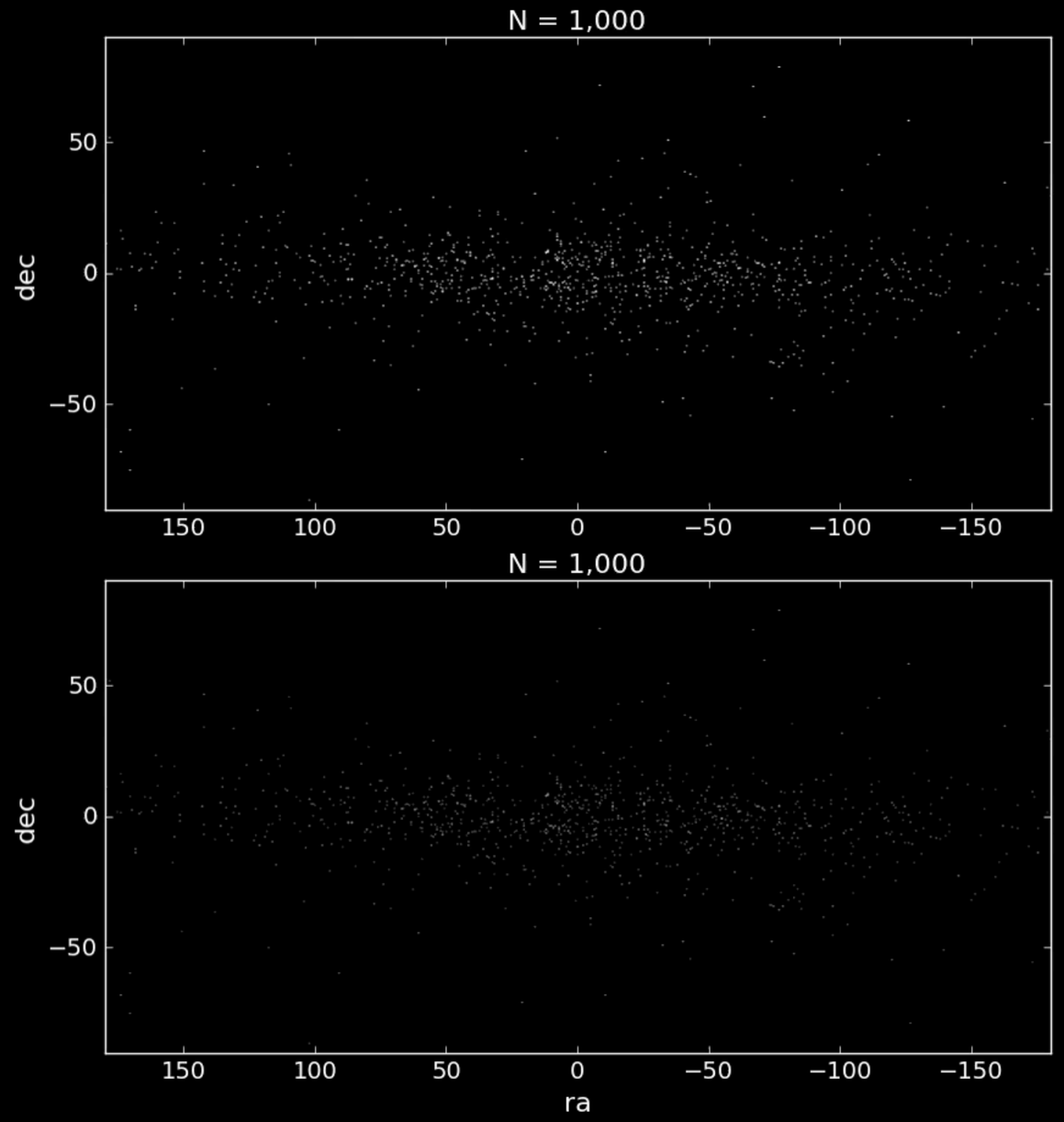
Agenda

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

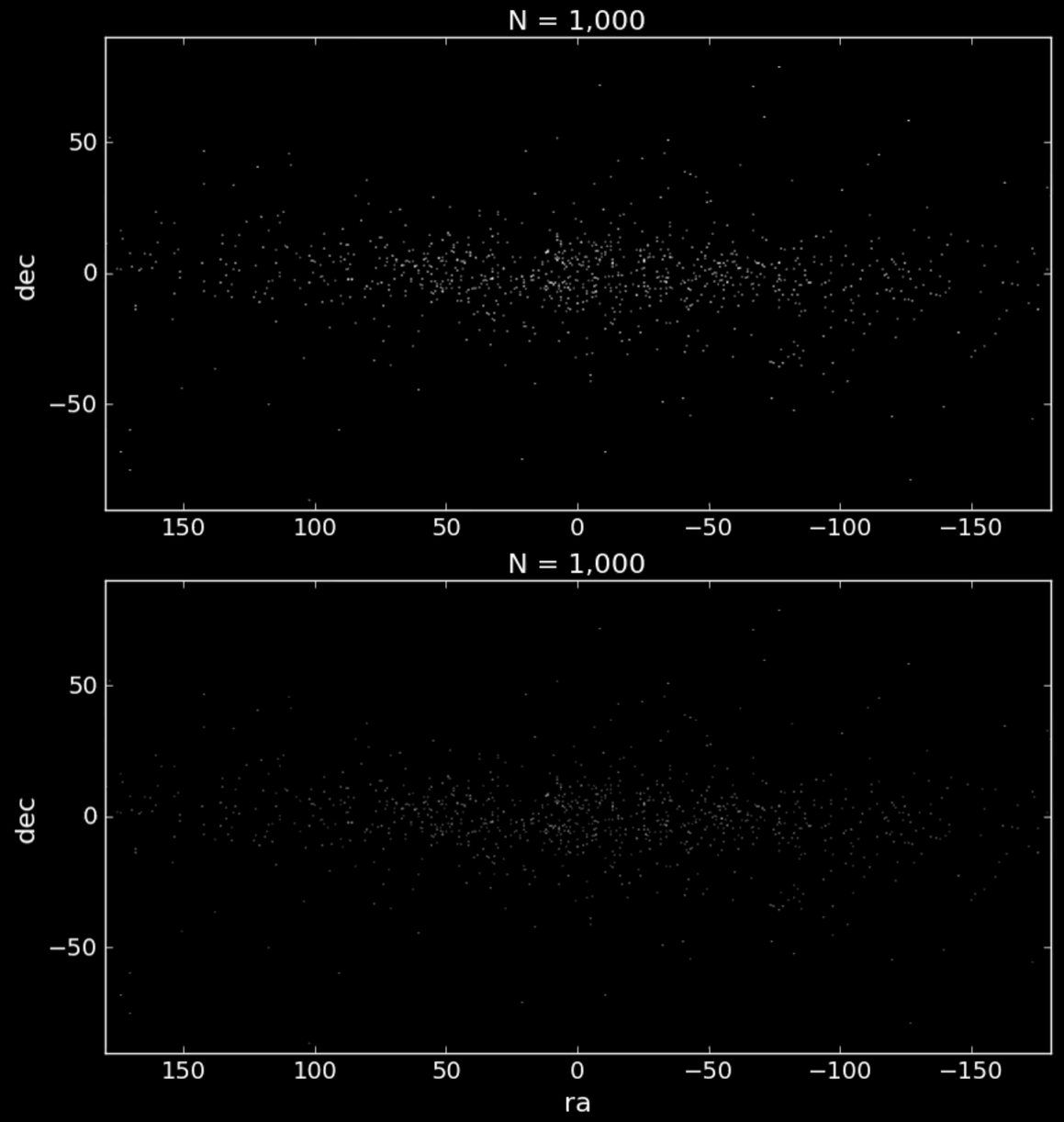
Motivation: Gaia

>1,000,000,000 stars/objects


N = 1,000
50
0
−50
dec
150
100
50
0
−50
−100
−150
N = 1,000
50
0
−50
dec
150
100
50
0
−50
−100
−150
ra


N = 1,000
50
0
−50
dec
150
100
50
0
−50
−100
−150
N = 1,000
50
0
−50
dec
150
100
50
0
−50
−100
−150
ra

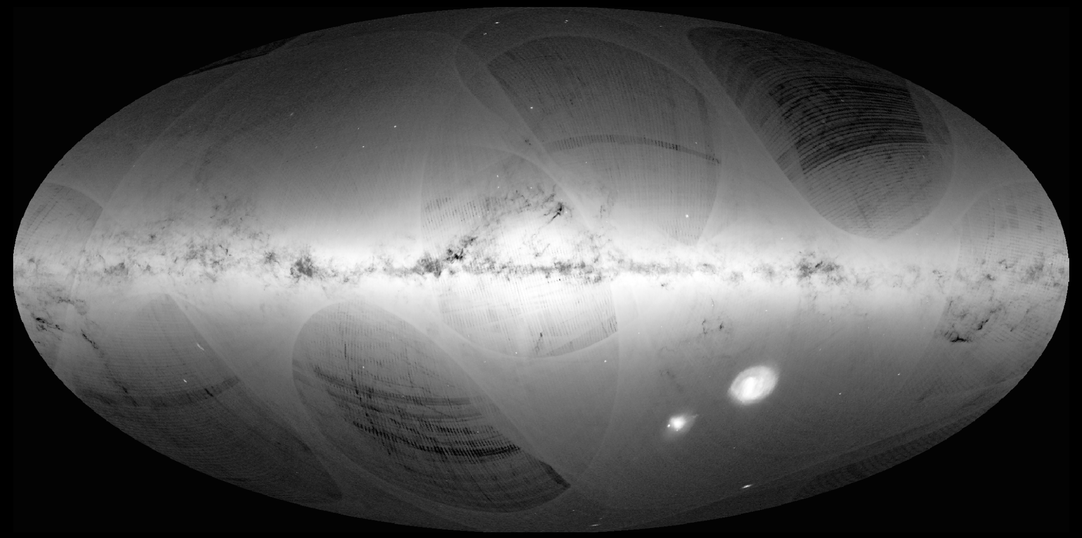
scatter



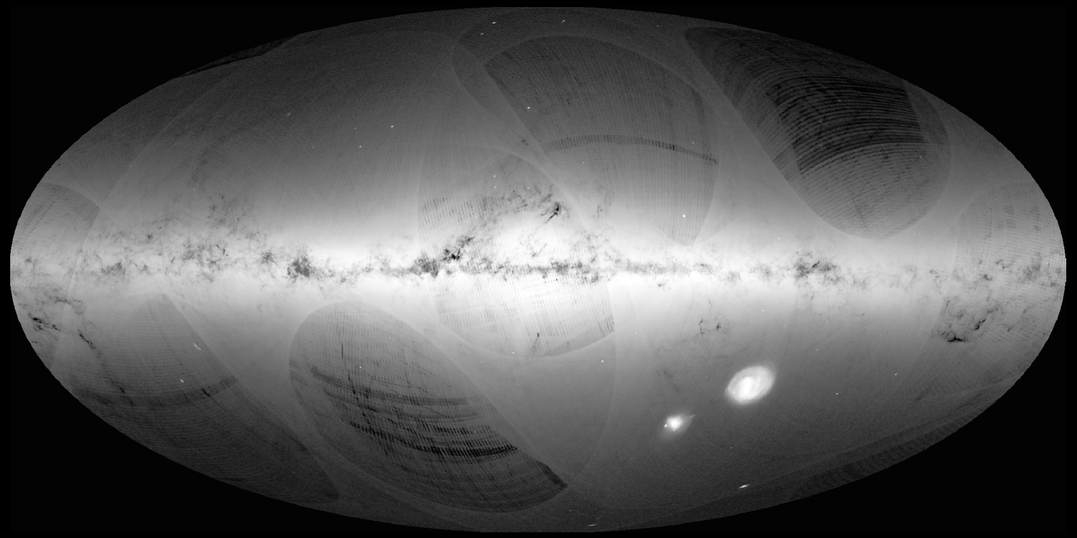
scatter

density

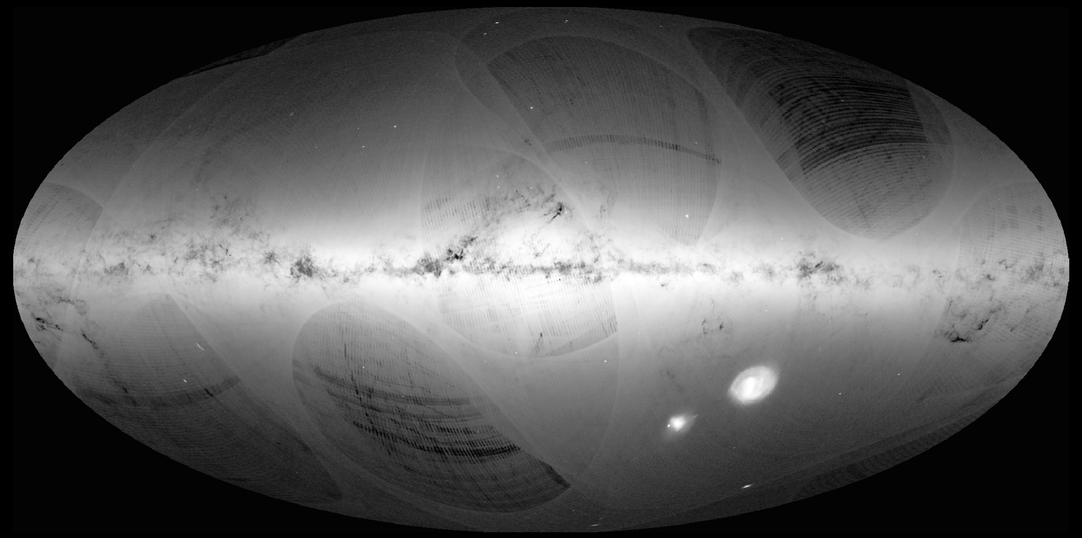
- How fast can it be done?



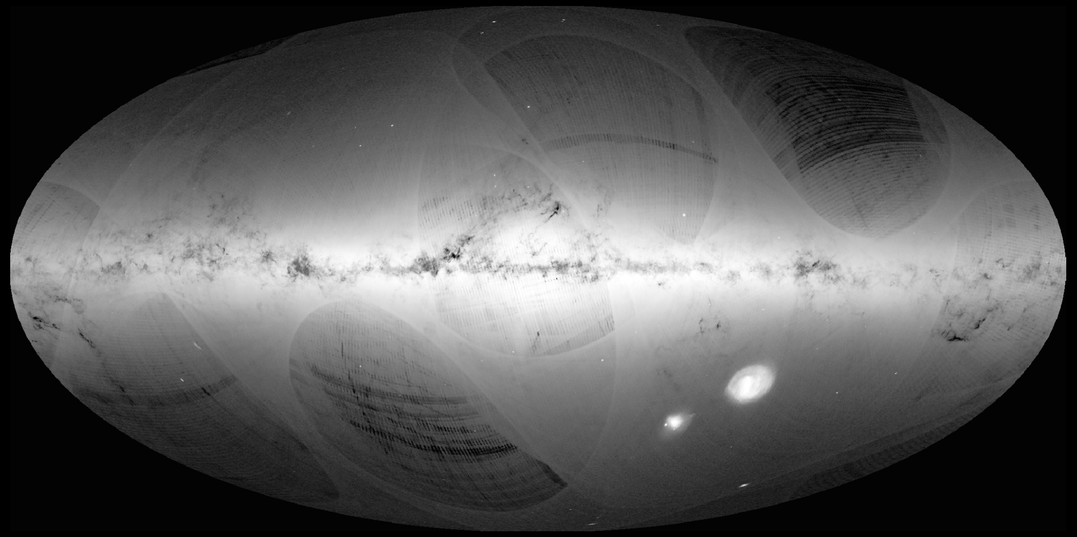
- How fast can it be done?
  - 10^9 * 2 * 8 bytes = 15 GiB (double is 8 bytes)



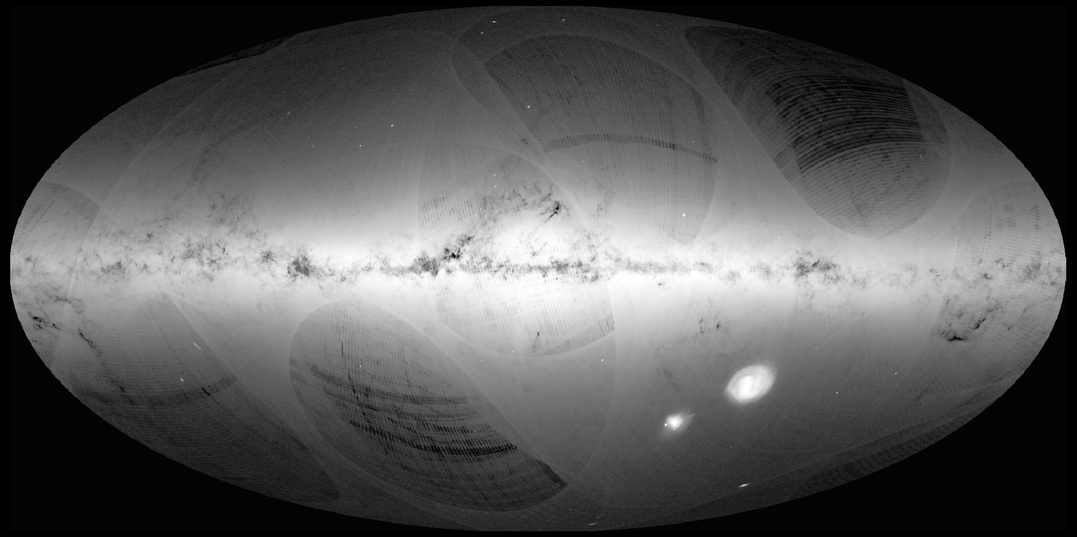
- How fast can it be done?
  - 10^9 * 2 * 8 bytes = 15 GiB (double is 8 bytes)
  - Memory bandwidth: 10-20 GiB/s: ~1 second



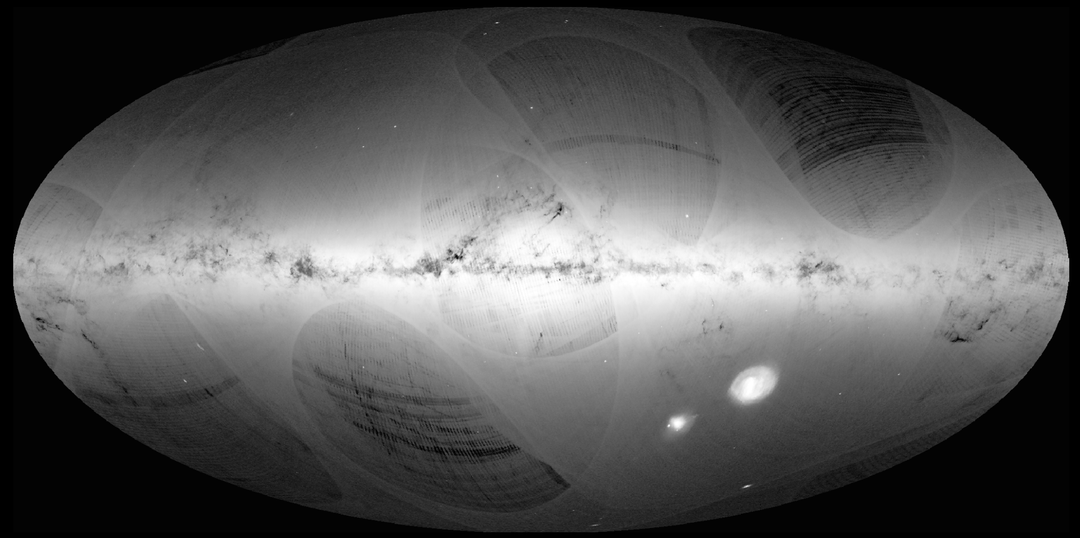
- How fast can it be done?
  - 10^9 * 2 * 8 bytes = 15 GiB (double is 8 bytes)
  - Memory bandwidth: 10-20 GiB/s: ~1 second
  - CPU: 3 Ghz (but multicore, say 4-8): 12-24 cycles/second

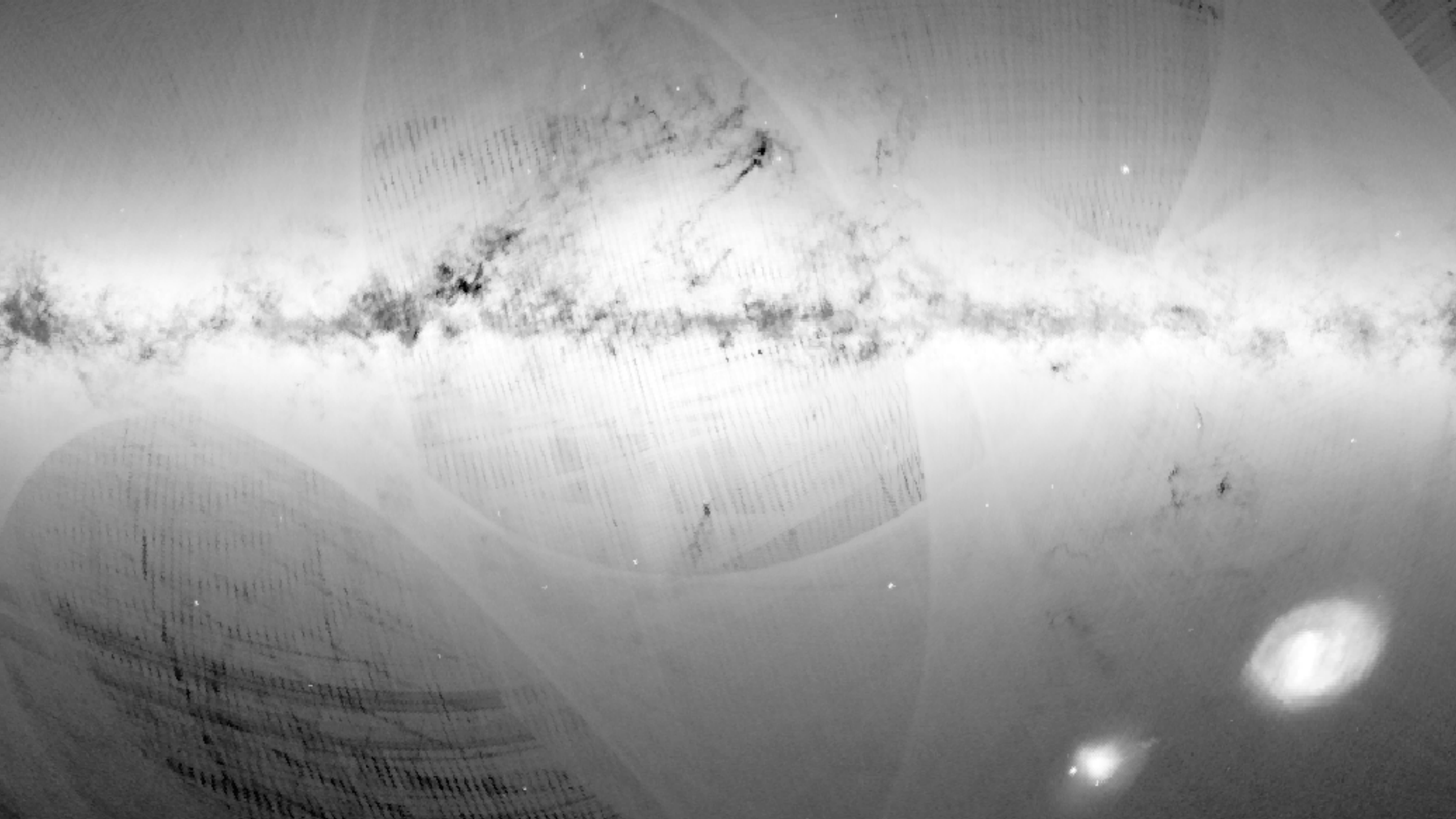


- How fast can it be done?
  - 10^9 * 2 * 8 bytes = 15 GiB (double is 8 bytes)
  - Memory bandwidth: 10-20 GiB/s: ~1 second
  - CPU: 3 Ghz (but multicore, say 4-8): 12-24 cycles/second
  - Few cycles per row/object, simple algorithm



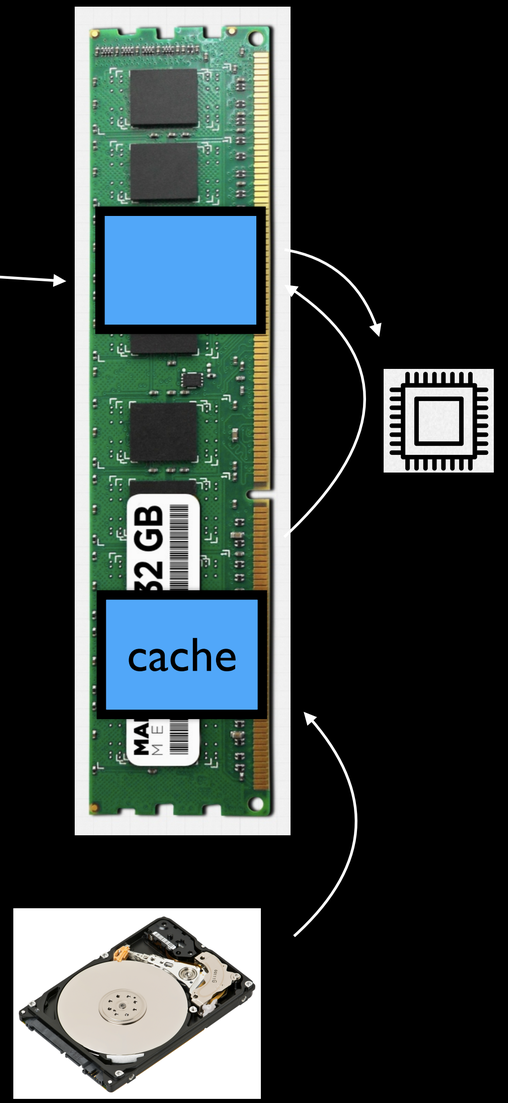
- How fast can it be done?
  - 10^9 * 2 * 8 bytes = 15 GiB (double is 8 bytes)
  - Memory bandwidth: 10-20 GiB/s: ~1 second
  - CPU: 3 Ghz (but multicore, say 4-8): 12-24 cycles/second
  - Few cycles per row/object, simple algorithm
  - Histograms/Density grids





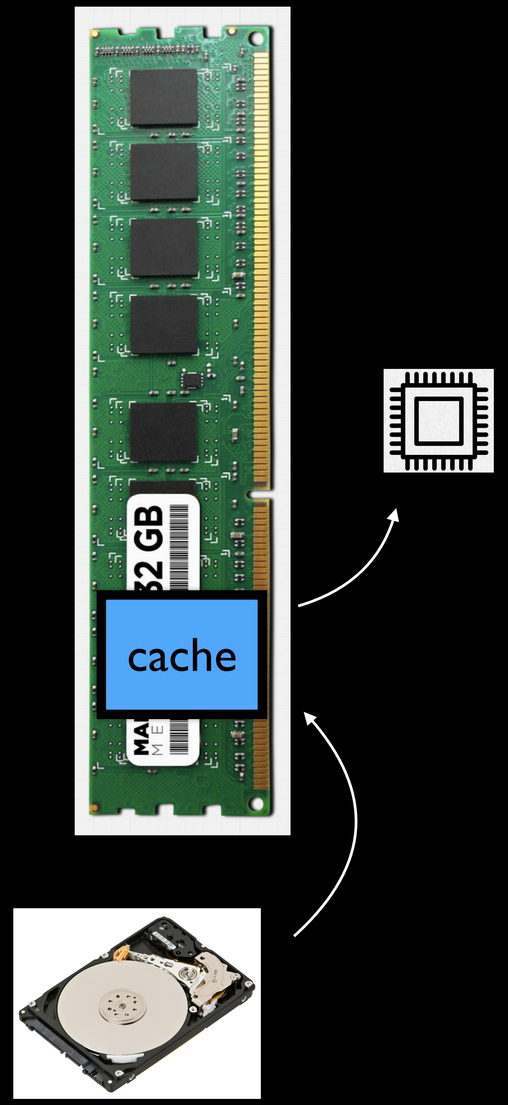
How to store and read the data

- 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 is you want to use the file system cache



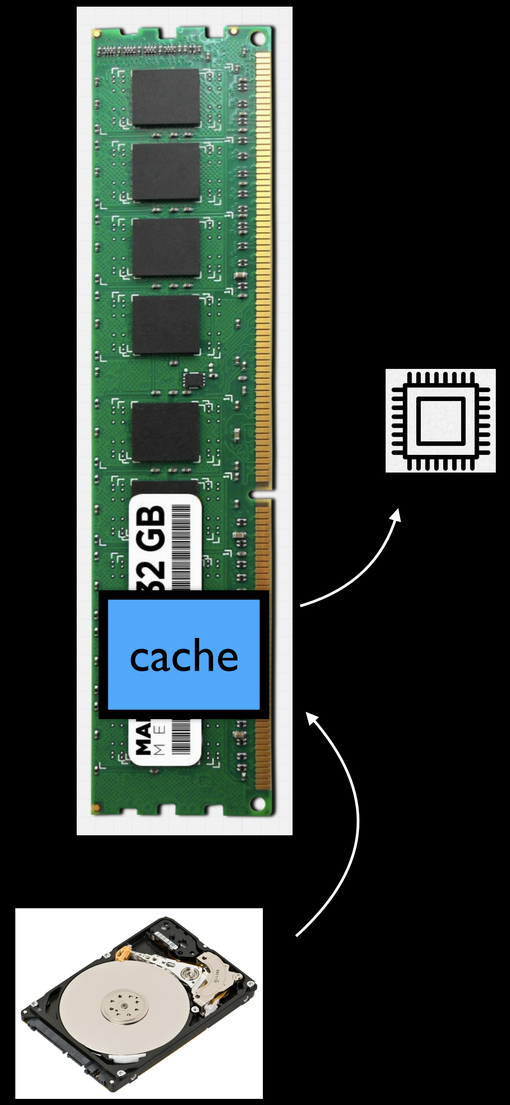
How to store and read the data

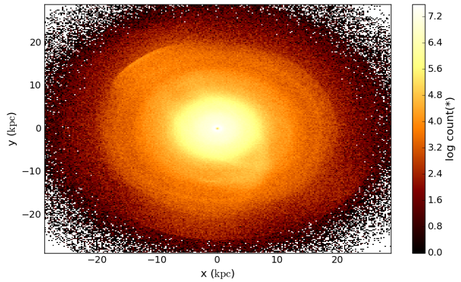
- Storage: native, column based (hdf5)
- Normal (POSIX read) method:
  - Allocate memory
  - read from disk to memory
  - Actually: from disk, to OS cache, to memory
  - Wastes memory/cache
  - 15 GB data , requires 30 GB is you want to use the file system cache



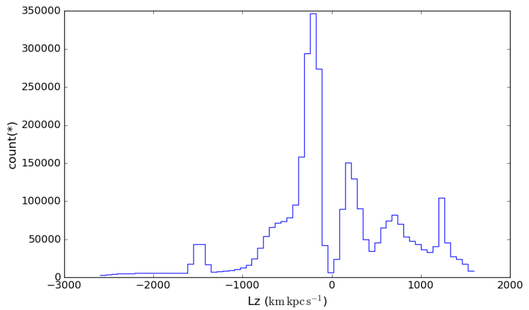
How to store and read the data

- 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 is 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)

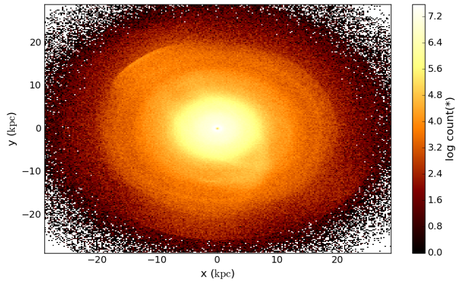



20
10
0
−10
−20
y (kpc)
−20
−10
0
10
20
x (kpc)
7.2
6.4
5.6
4.8
4.0
3.2
2.4
1.6
0.8
0.0
log count(+)

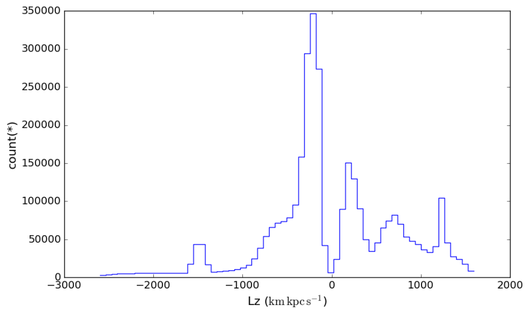
1d


350000
300000
250000
200000
150000
100000
50000
0
count(*)
−3000
−2000
−1000
0
1000
2000
Lz (km kpc s^{-1})

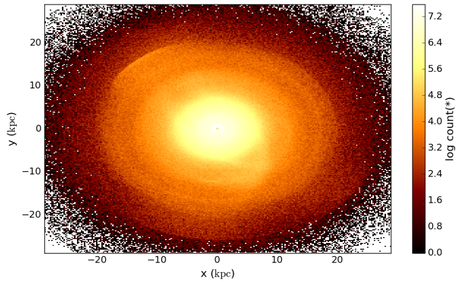
2d


20
10
0
−10
−20
y (kpc)
−20
−10
0
10
20
x (kpc)
7.2
6.4
5.6
4.8
4.0
3.2
2.4
1.6
0.8
0.0
log count(*)

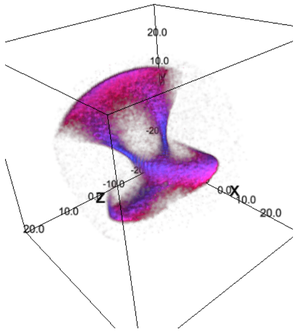
1d



2d

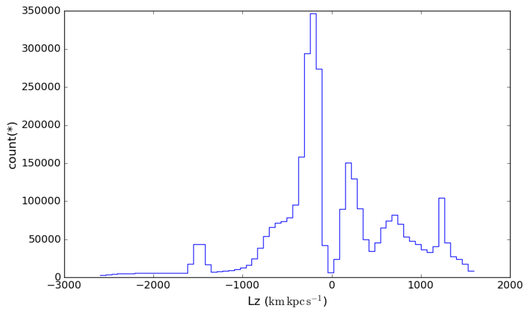


3d

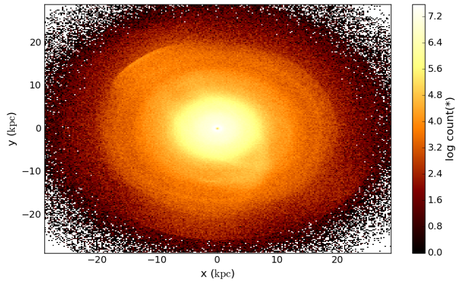


0d 330,000 rows

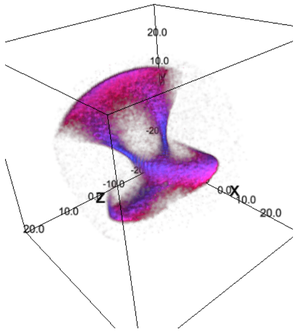
1d



2d

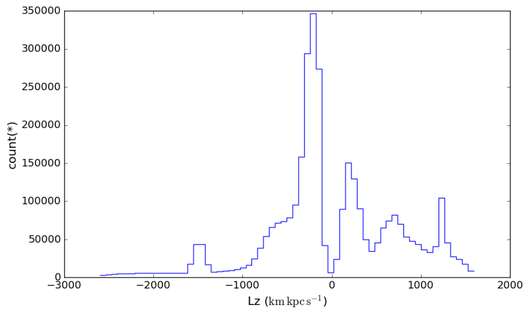


3d

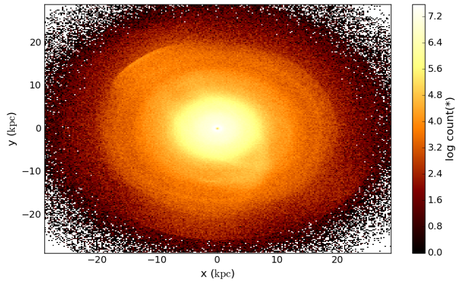


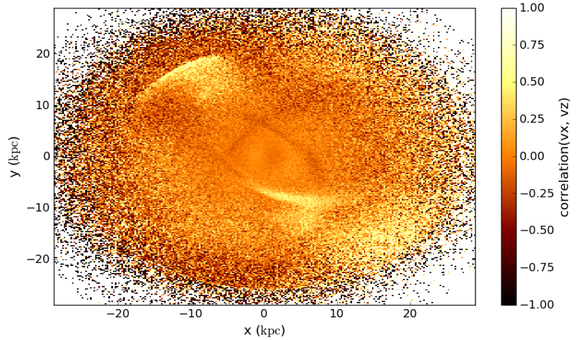
0d 330,000 rows

1d

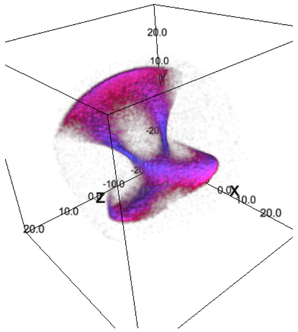


2d



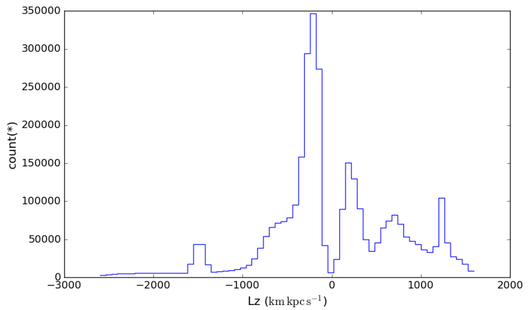


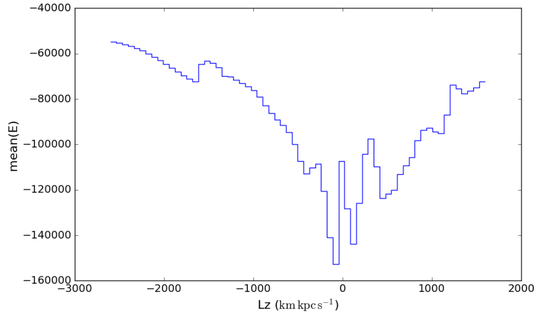
3d



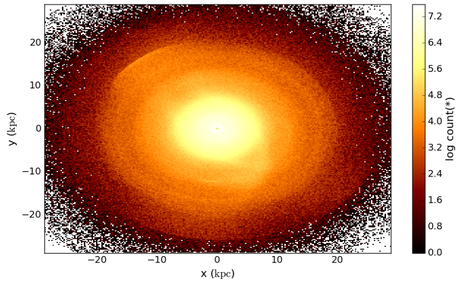
0d 330,000 rows

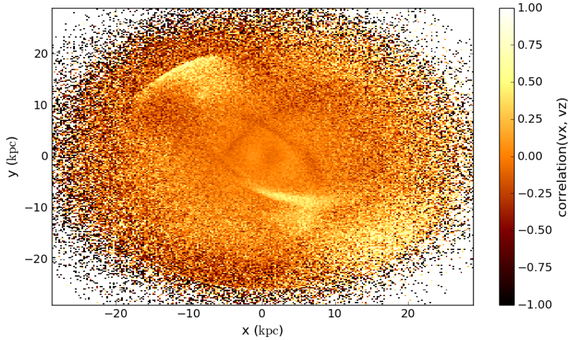
1d



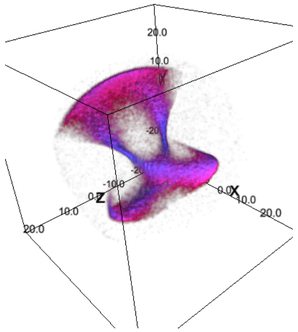


2d



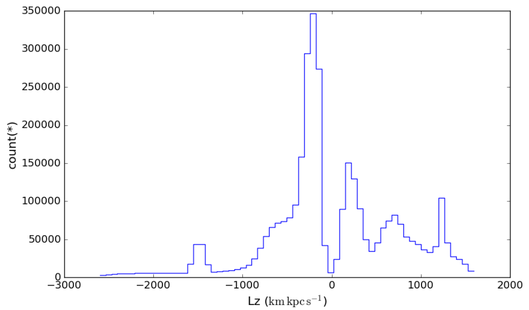


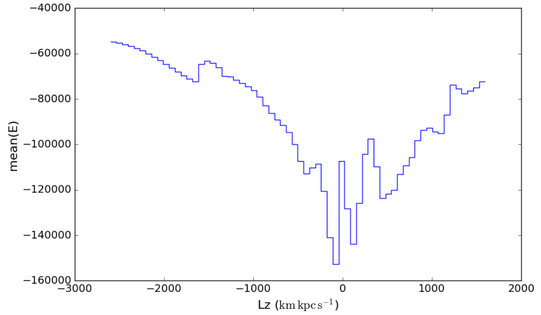
3d



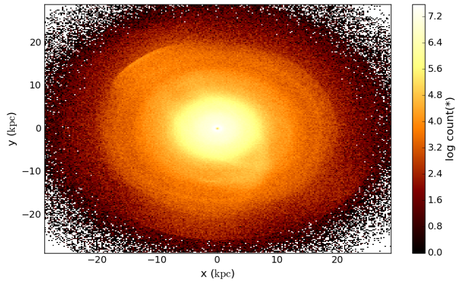
0d 330,000 rows mean: -0.083

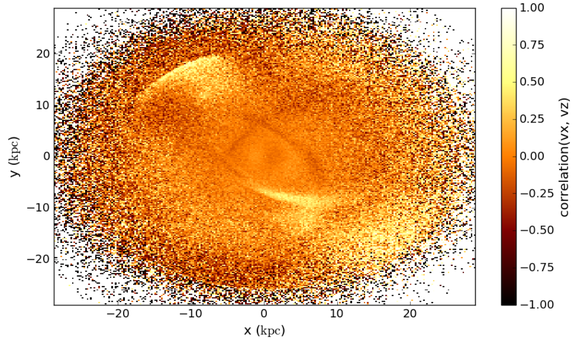
1d



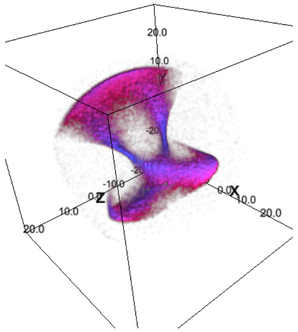


2d



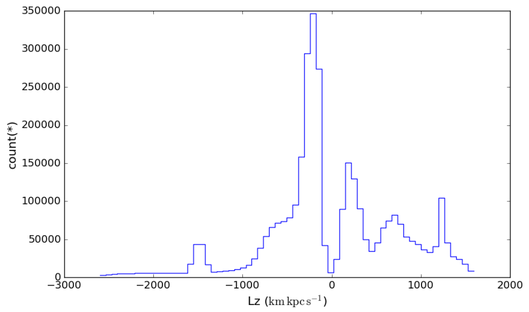


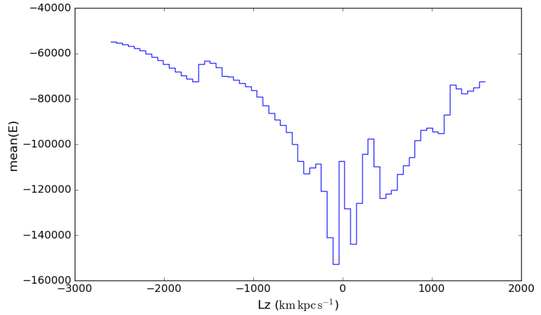
3d



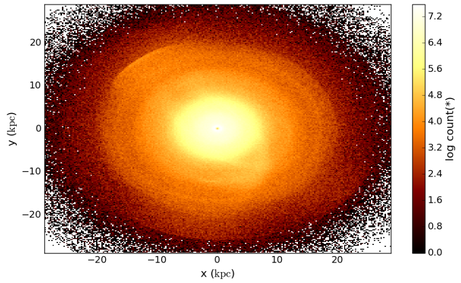
0d 330,000 rows mean: -0.083

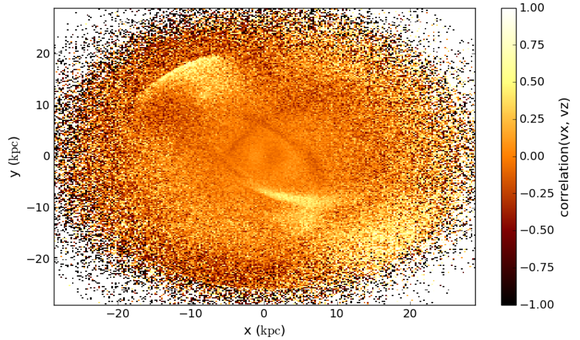
1d



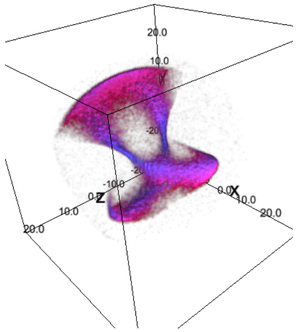


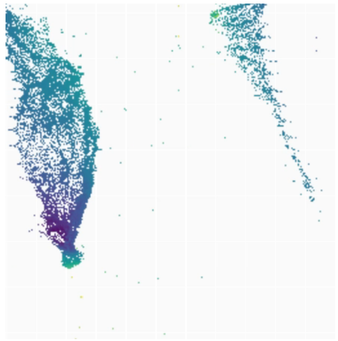
2d





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
- Focusses mostly on statistics on N-d grids (count/mean/max/std/…)

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

vaex

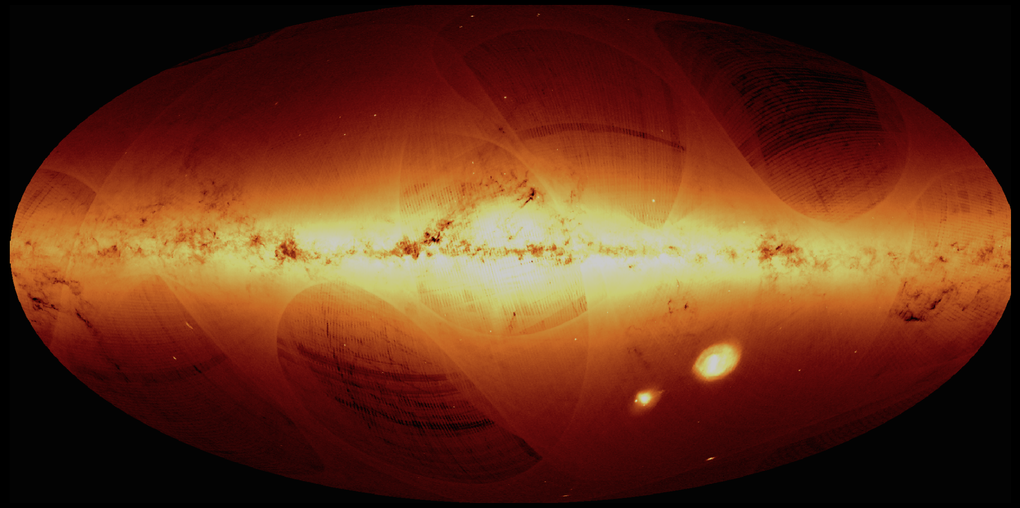
- Python library (conda/pip installable)
- Think pandas-like but for large datasets
- Focusses mostly on statistics on N-d grids (count/mean/max/std/…)
- >1 billion rows / sec on a `decent` desktop (quad core 3Gz)
  - >50x faster than scipy.stats.binned_statistic_2d
- 0.1-0.2 billion rows on this 2yr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

vaex

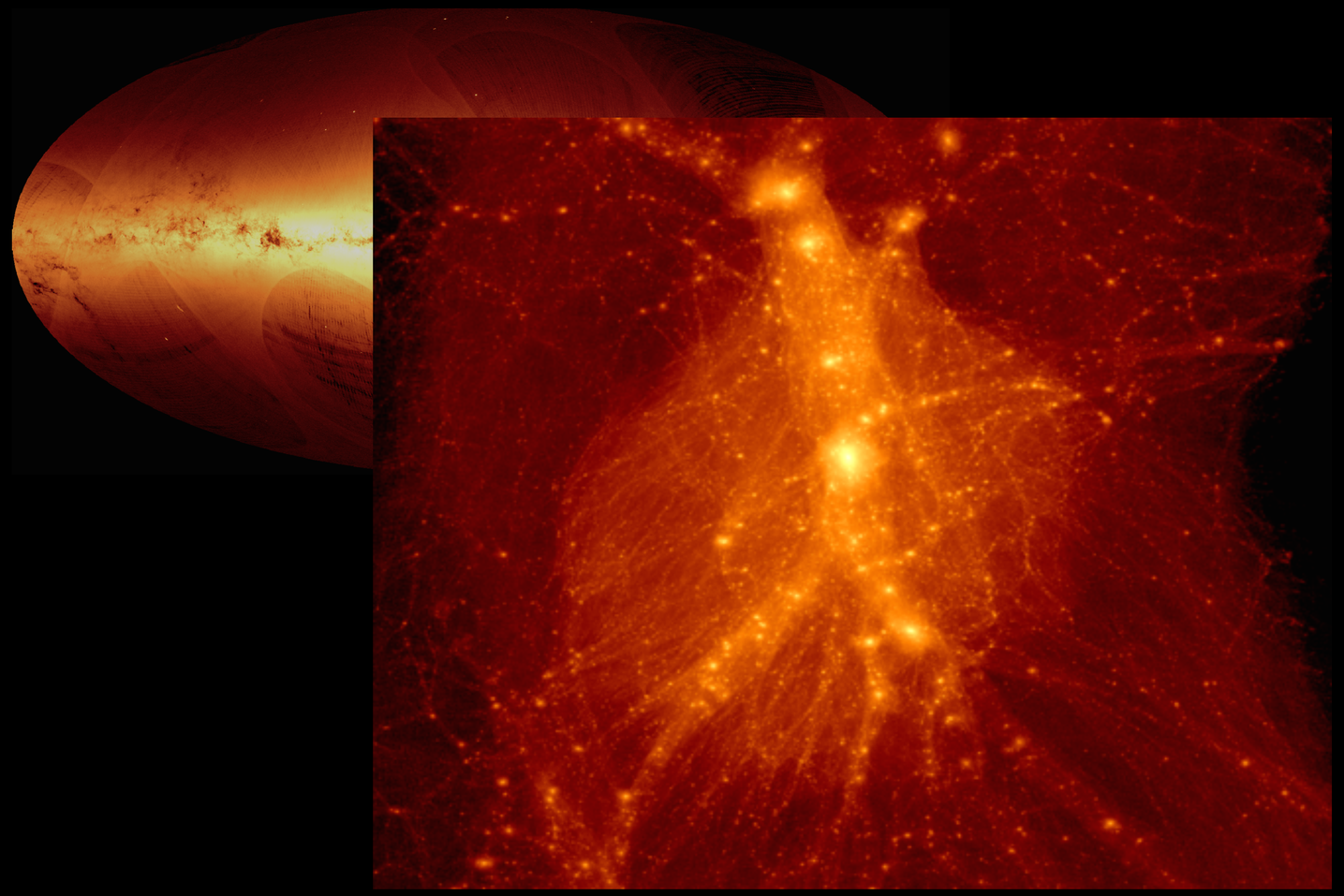
- Python library (conda/pip installable)
- Think pandas-like but for large datasets
- Focusses mostly on statistics on N-d grids (count/mean/max/std/…)
- >1 billion rows / sec on a `decent` desktop (quad core 3Gz)
  - >50x faster than scipy.stats.binned_statistic_2d
- 0.1-0.2 billion rows on this 2yr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

What kind of data?

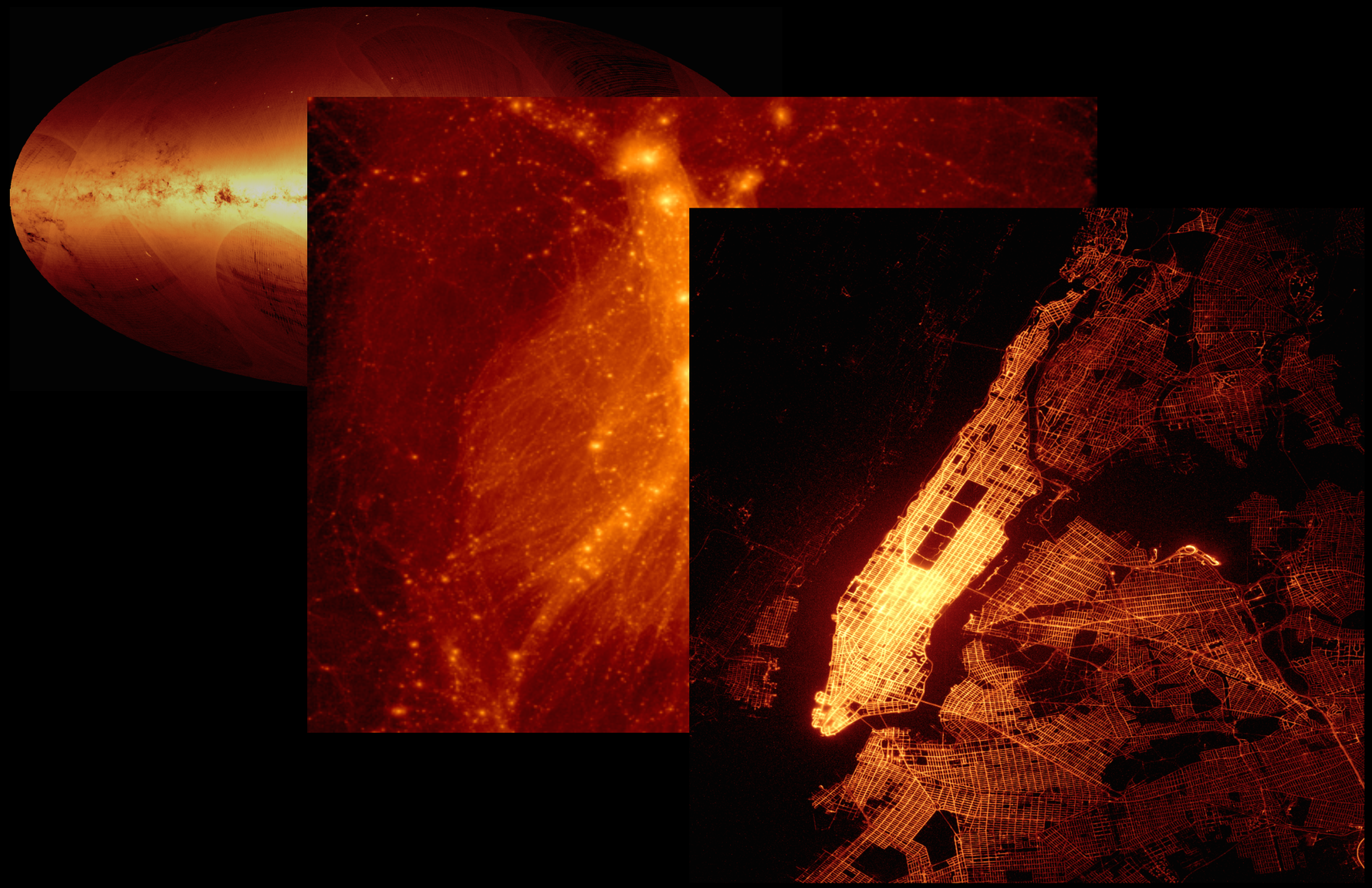
What kind of data?



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
- 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/talk-pydata-meetup-paris-2017