

FileDB

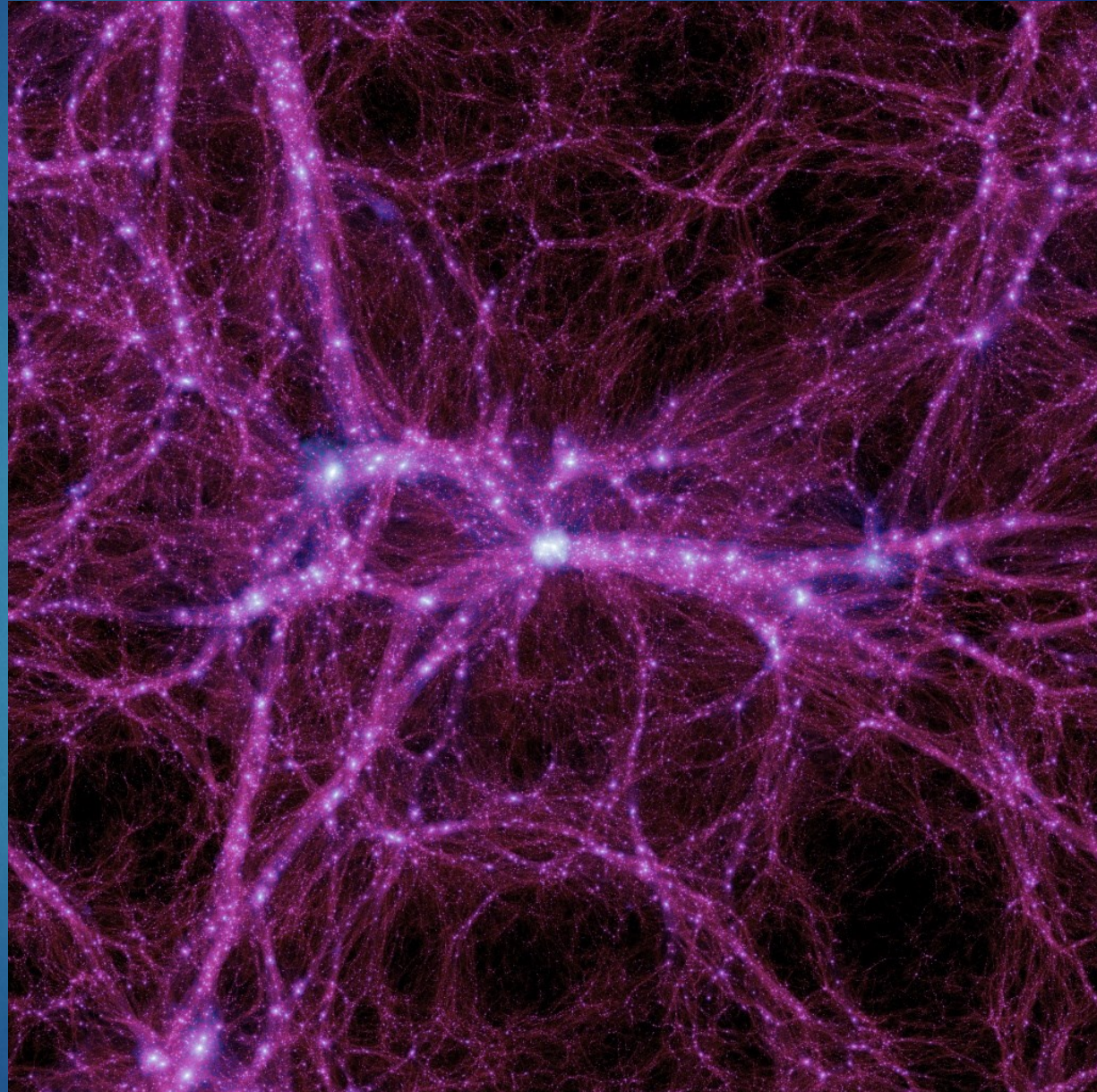
ACCESSING (COSMOLOGICAL) SIMULATIONS FROM THE DATABASE



Millennium-II Simulation

- 100 Mpc/h
- 10^{10} particles
- $6.9 \cdot 10^6 M_{\text{sun}}/h$
- ~ 10 million halos
- $\sim 300\text{GB/snapsho}$
†

Boylan-Kolchin et al
2009

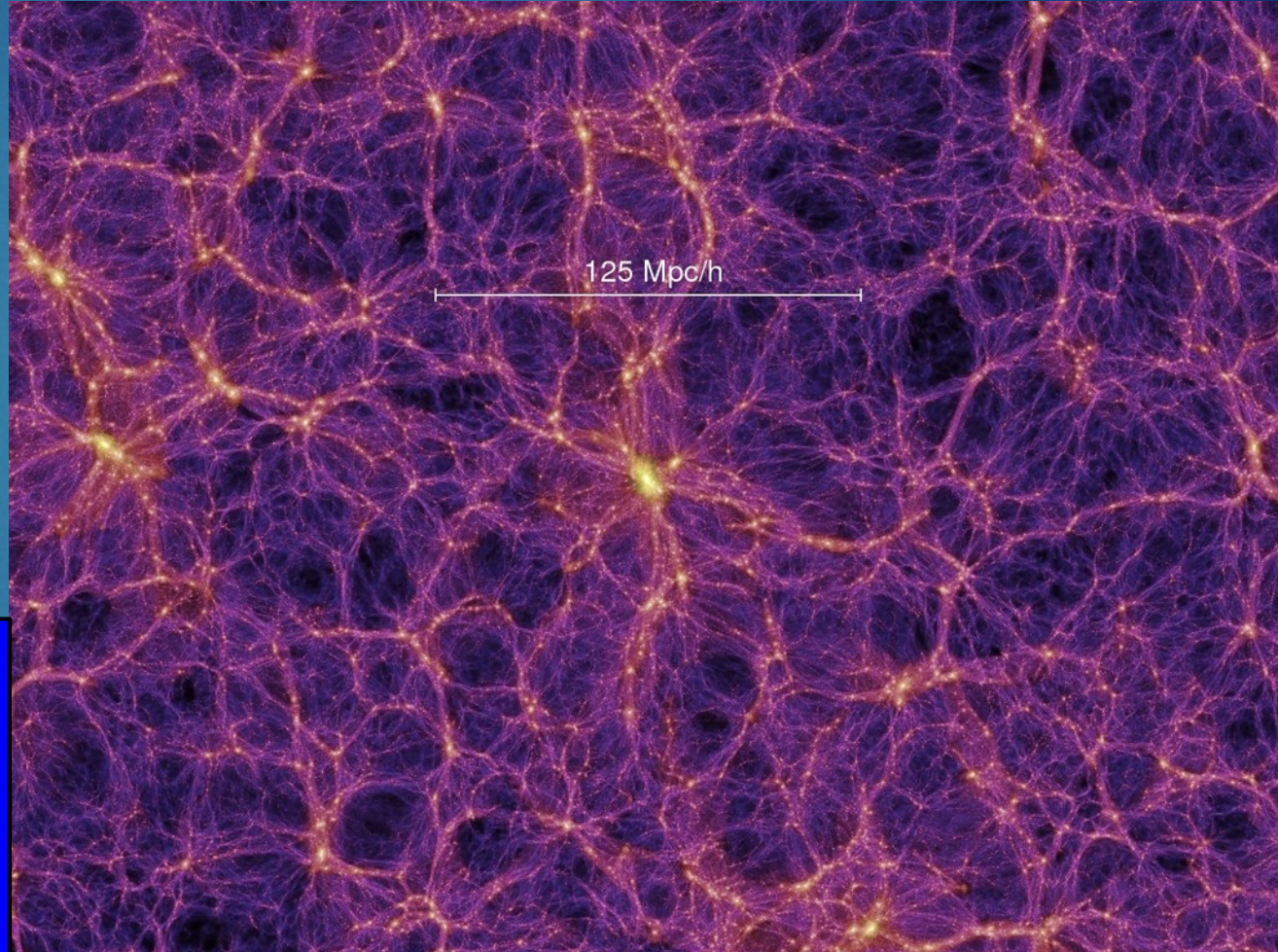


Millennium Simulation

- 500 Mpc/h
- 10^{10} particles
- $8.6 \cdot 10^8 M_{\text{sun}}/h$
- ~18 million halos
- ~300GB/snaps hot

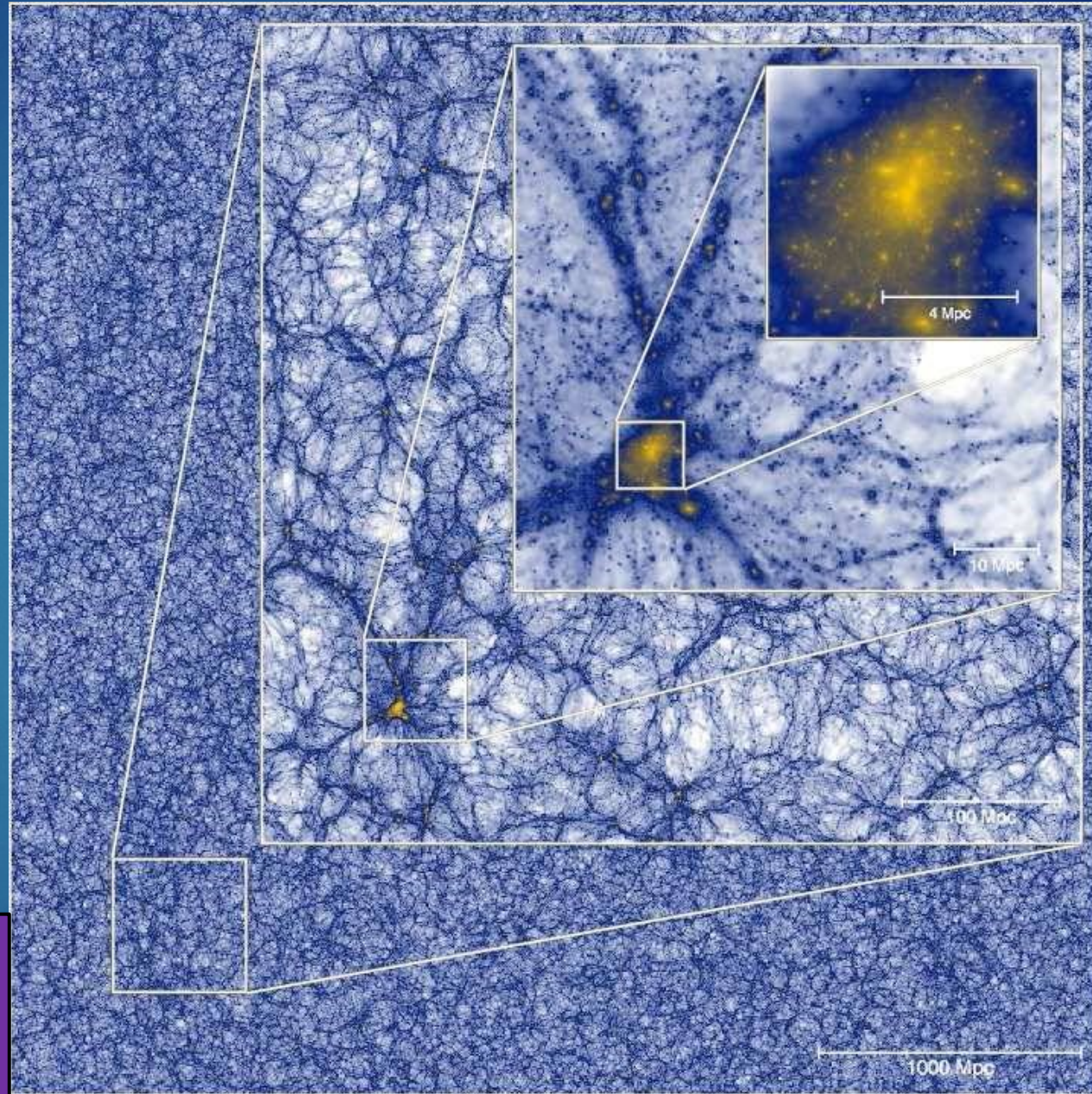
Springel et al 2005

MR II



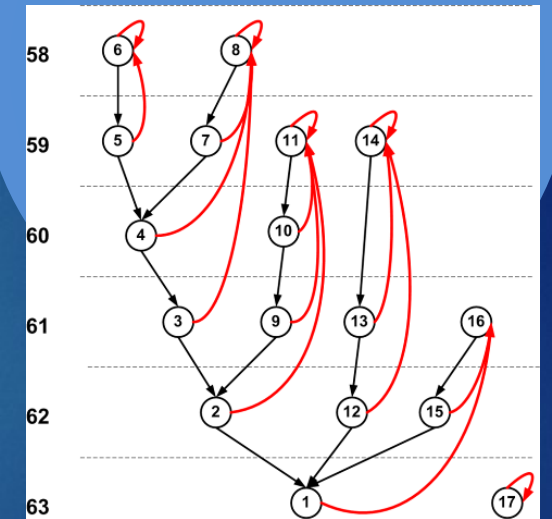
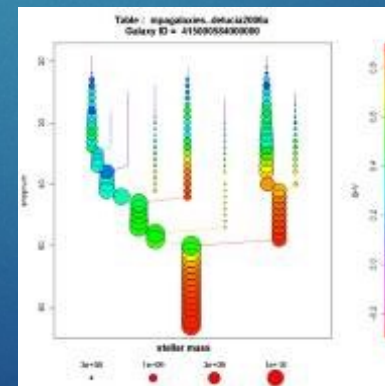
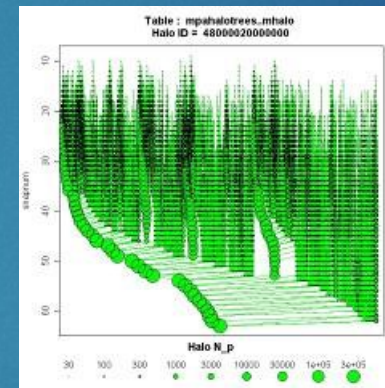
MR-XXL

- 3Gpc/h
- 3×10^{11} particles
- 750 million halos /snapshot
- 9TB/snapshot
- browse



Millennium Database

- ▶ Particle clusters
- ▶ Merger trees
- ▶ SAM galaxies
- ▶ Light cones



Access to raw data

- ▶ 10-300 billion particles
- ▶ Too large to ingest in database
- ▶ Would still be nice to query directly



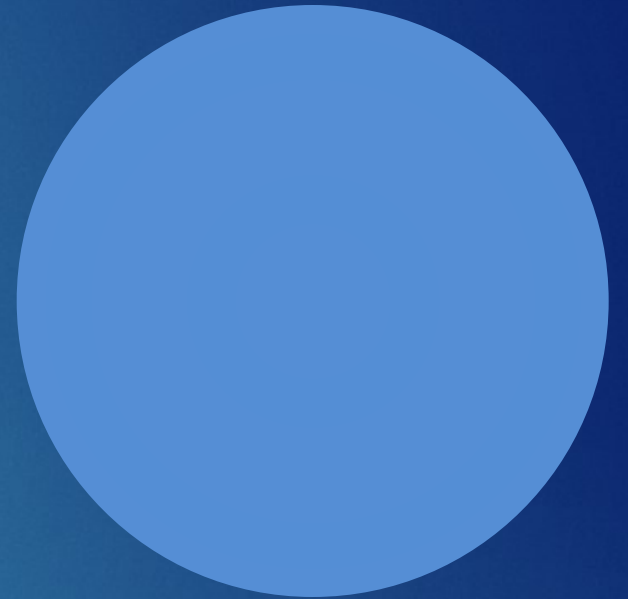
FileDB

- ▶ Accessing data outside of db ...
- ▶ ... from within db
- ▶ Using custom user defined table-valued functions
- ▶ Written in C#, deployed in MS SQL Server database

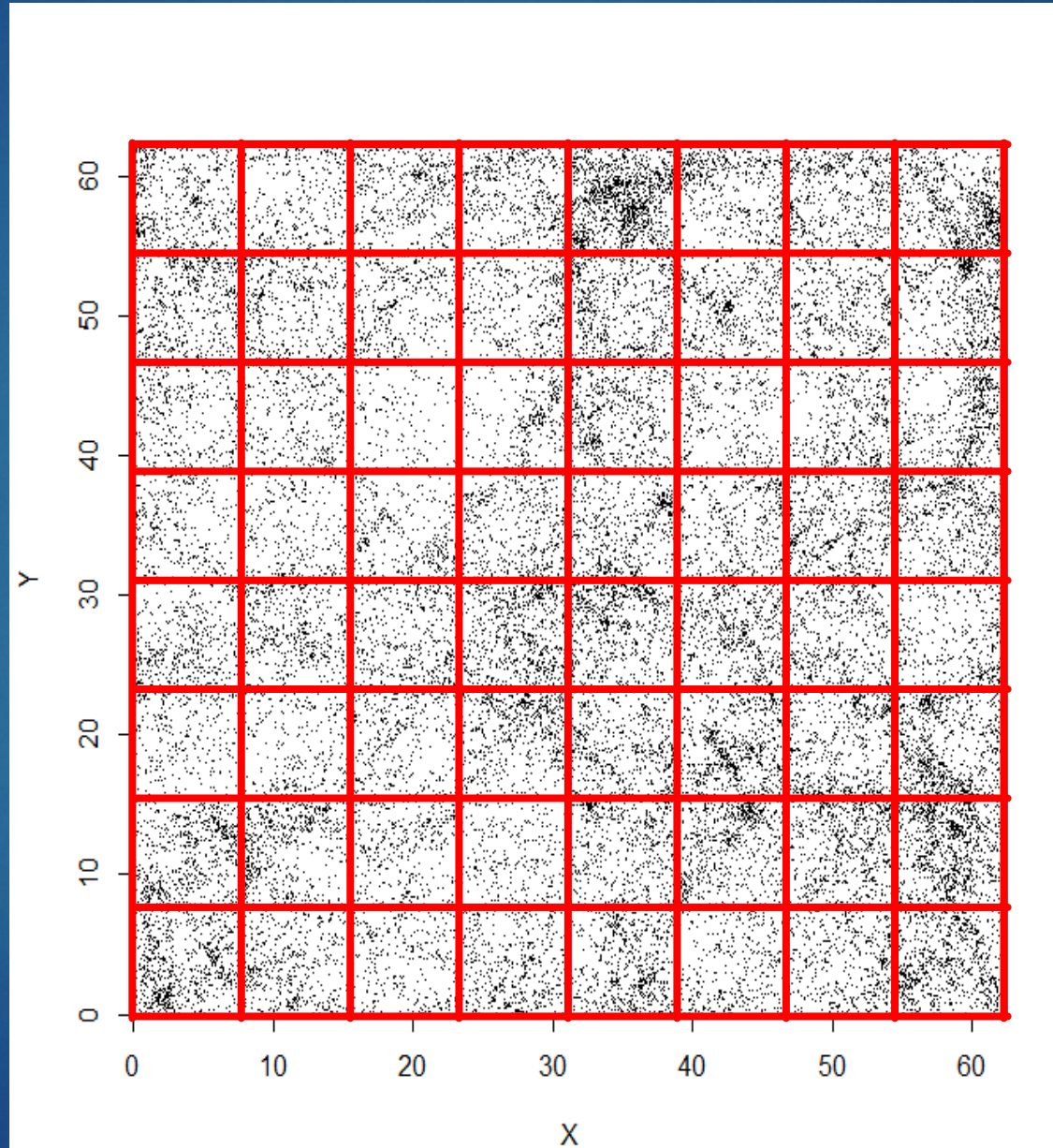
```
SELECT *  
FROM MillenniumParticles(63,'SPHERE[0,0,0,10]')
```


How to make this fast?

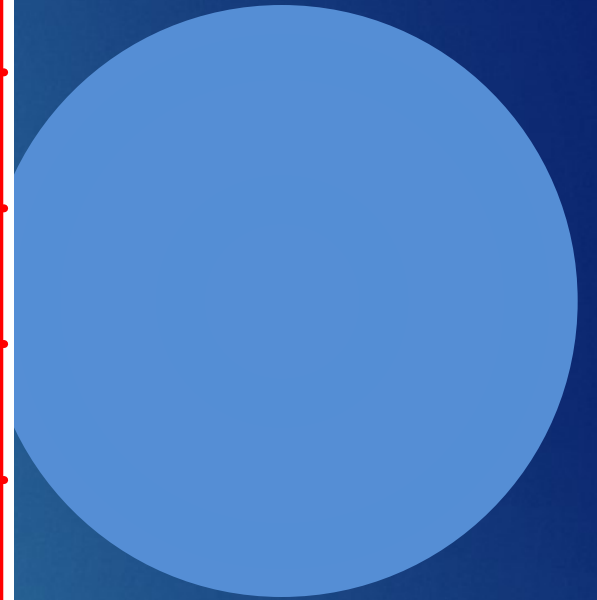
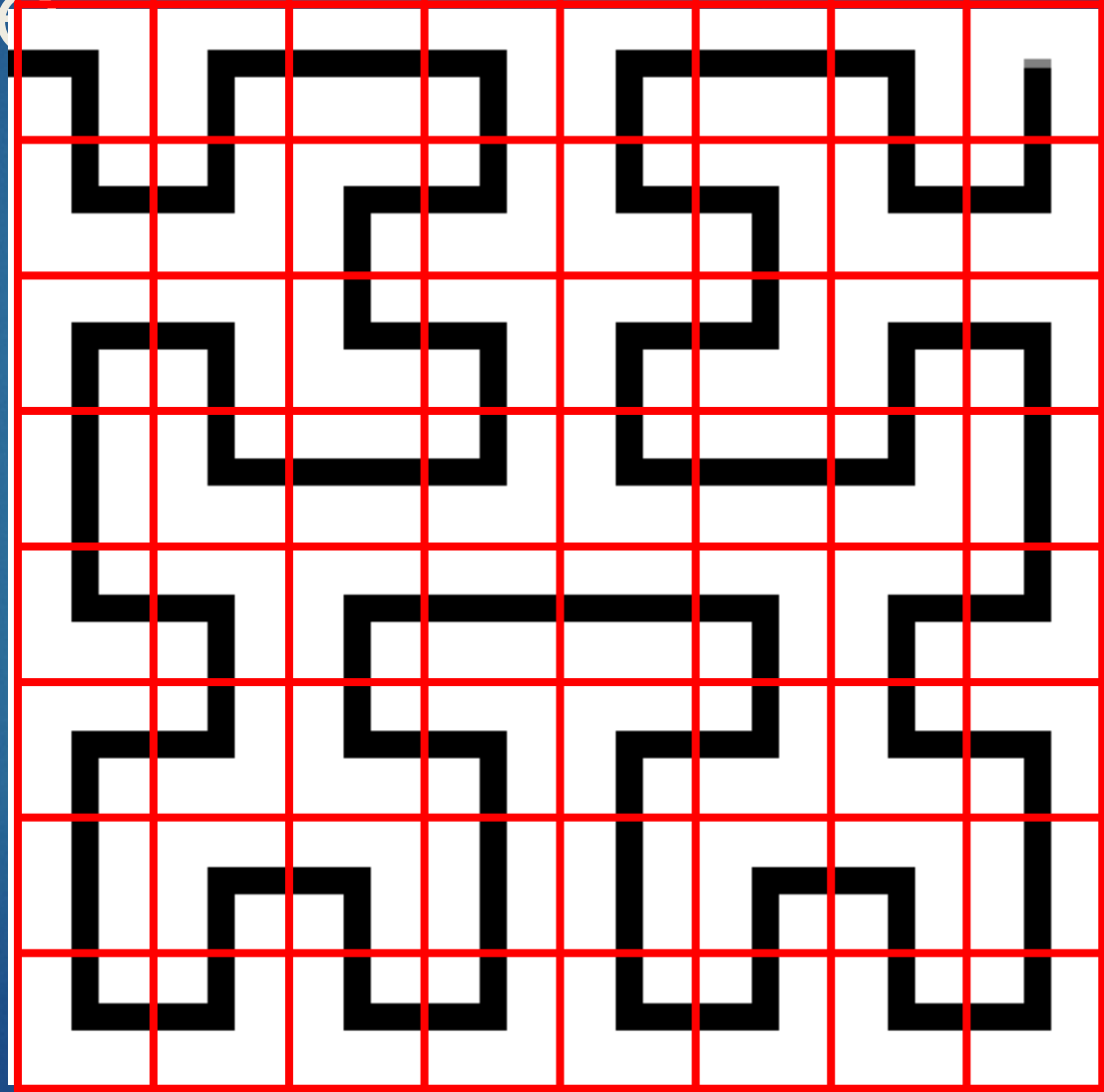
- ▶ Need 3D spatial index
- ▶ Use space-filling curve
 - ▶ Morton z-curve
 - ▶ Peano-Hilbert curve



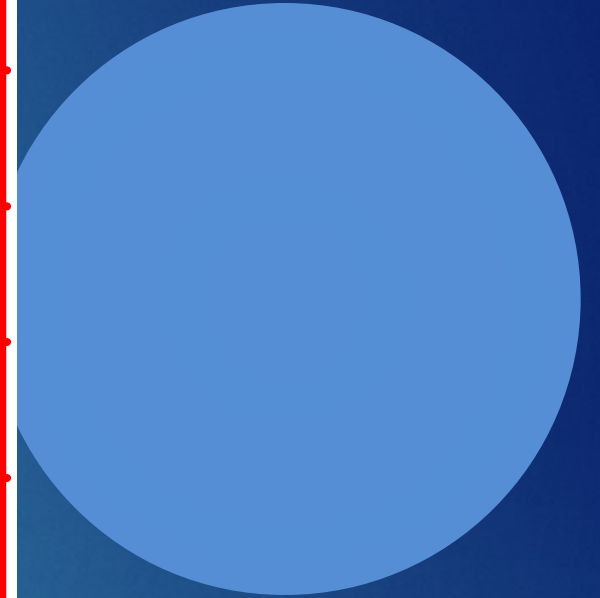
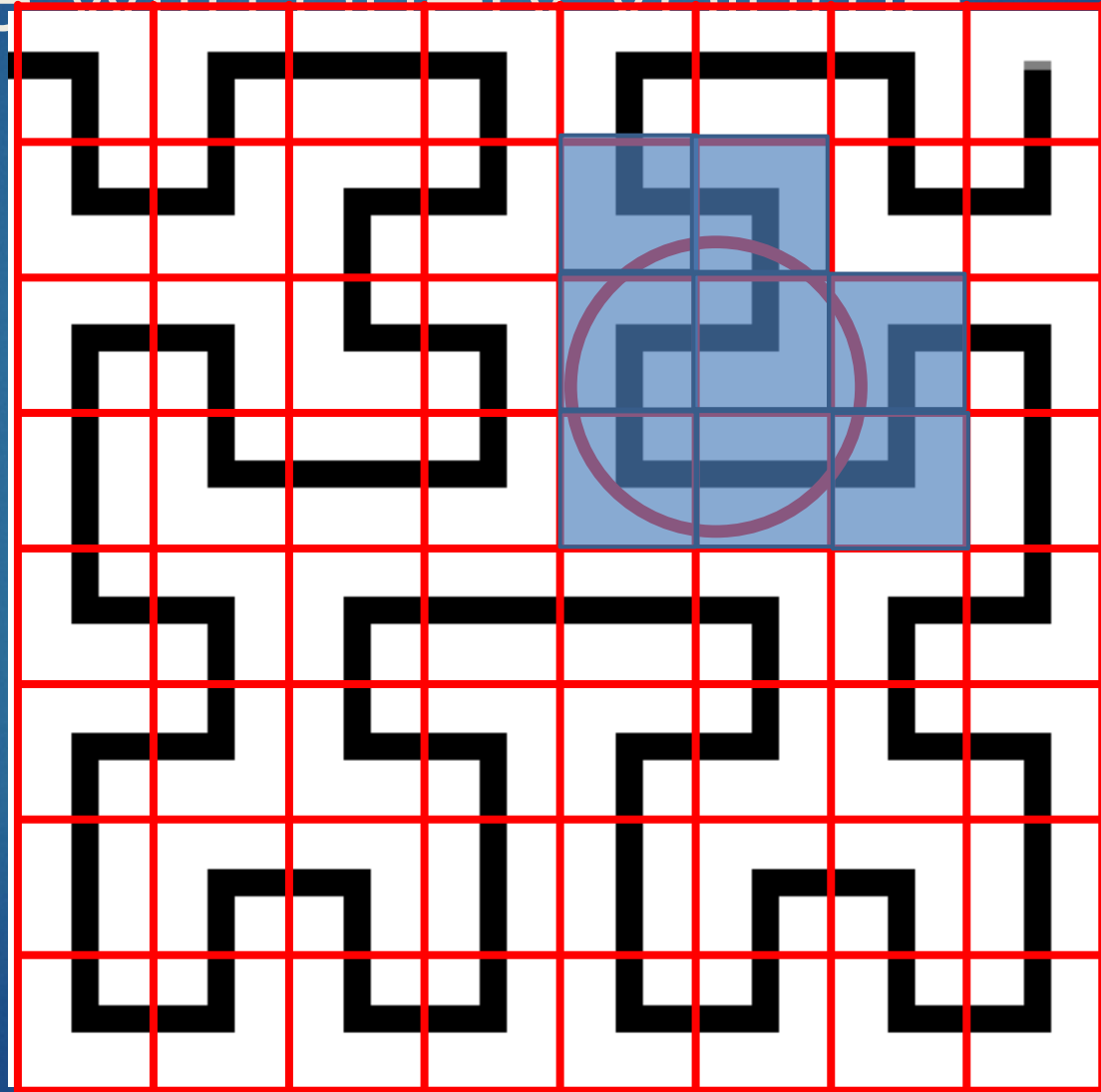
Divide simulation volume in regular grid



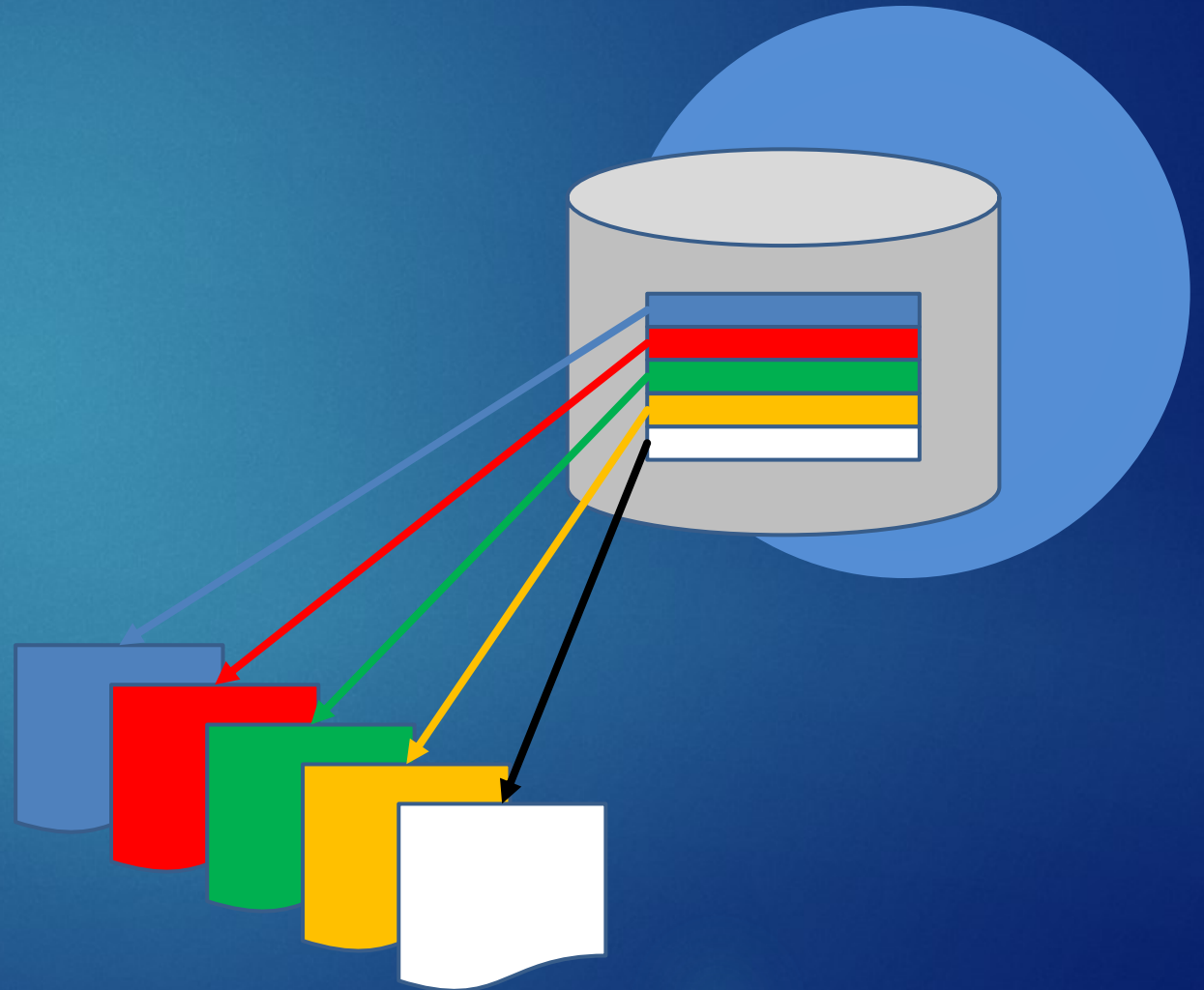
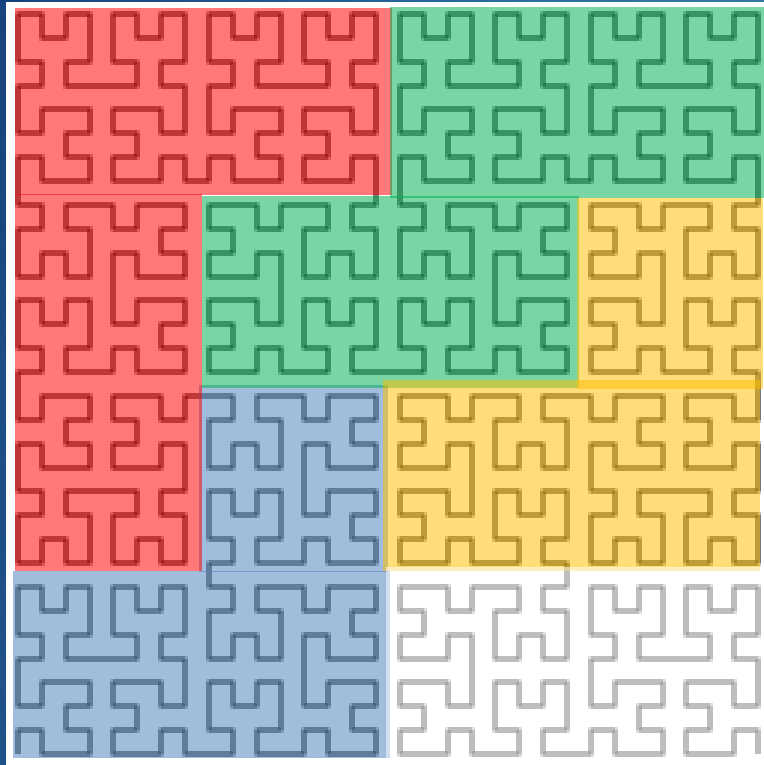
Index cells using space filling curve

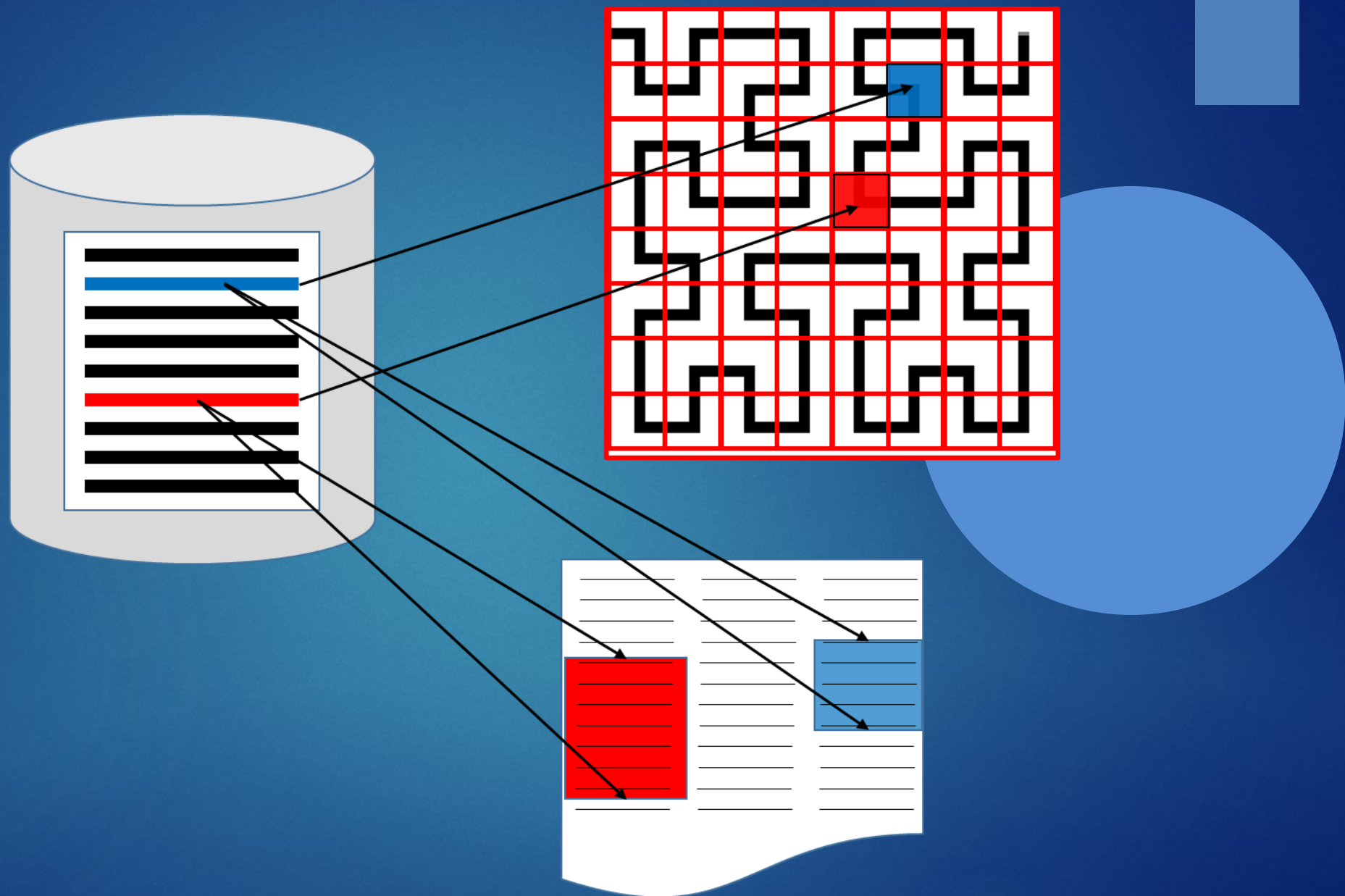


Calculate overlap space filling
curve with query volume



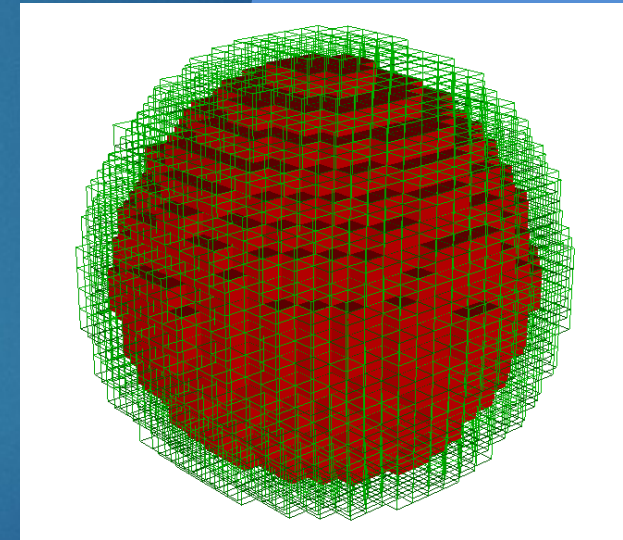
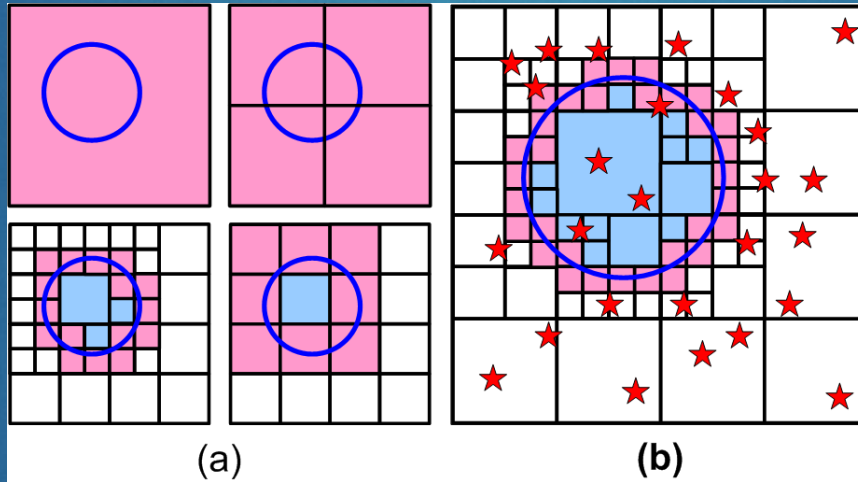
Individual files store intervals on curve; indexed in B





Query particles in sphere/box

- ▶ Divide simulation volume in regular grid
- ▶ Index using space filling curve
- ▶ Calculate overlap space filling curve with query volume



- ▶ Execute as table-valued function from database
- ▶ Technology: C# for SQLCLR table-valued-function

jupyter CosmoUC_5_PlotHalo Last Checkpoint: 11 minutes ago (autosaved)

File Edit View Insert Cell Kernel Help Notebook saved Python 3

Code CellToolBar

Dark-matter Halos in a Cosmological Simulation

```

In [3]: import SciServer.LoginPortal as Login
        token = Login.getToken()
        import SciServer.CasJobs
        import pandas
        import tables
        import numpy as np
        import matplotlib.pyplot as plt
        from mpl_toolkits.mplot3d import Axes3D

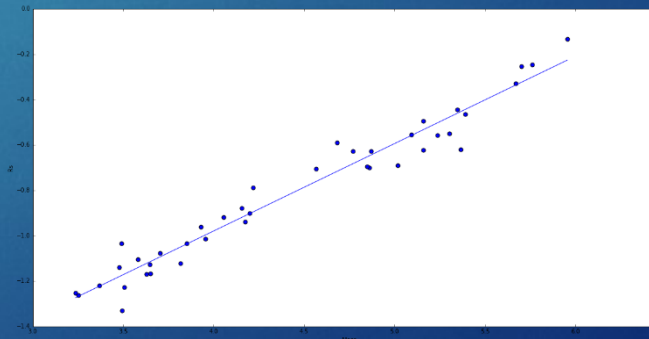
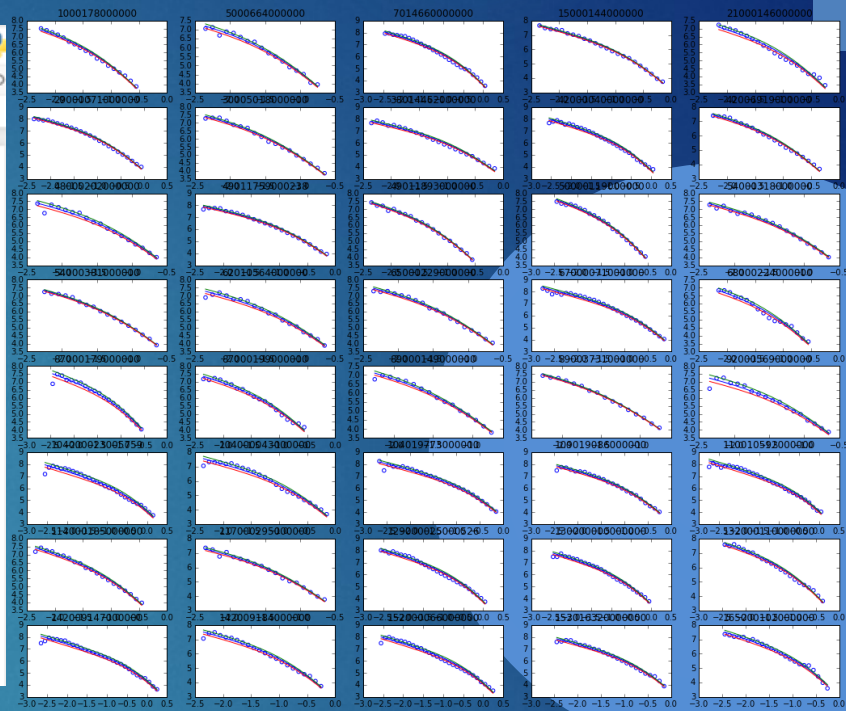
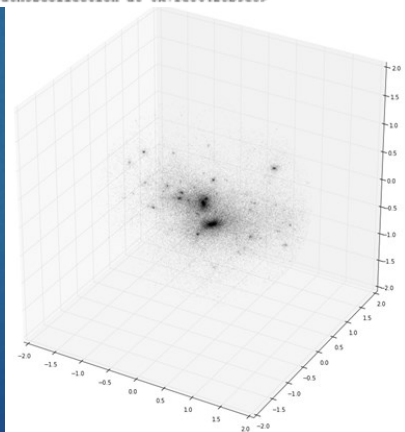
In [10]: %%time
        queryString = """
        select top 100000 p.x-hh.x as x,p.y-hh.y as y,p.z-hh.z as z
        from mpahlotrees.mr hh
        cross apply dbo.MillenniumParticles(hh.snapnum,
        dbo.Sphere::New(hh.x,hh.y,hh.z,3*hh.halfmassradius).ToString()) p
        where hh.haloid=84000007000000 order by newid()
        """
        responseStream = SciServer.CasJobs.executeQuery(queryString, token=token, context="SimulationDB")
        df = pandas.read_csv(responseStream, index_col=None)

        CPU times: user 351 ms, sys: 184 ms, total: 535 ms
        Wall time: 5.27 s

In [13]: fig = plt.figure(figsize=(15, 15))
        ax = fig.add_subplot(111, projection='3d')
        ax.scatter(df.x, df.y, df.z, s=0.001)

Out[13]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x7fe86428b9e8>

```



Also accessible from outside the database

The image displays three sequential JupyterLab file browser views connected by red lines, illustrating a navigation path from a high-level directory to specific data files.

View 1: Root Directory
The path is `/ virgo`. The file list includes:

- ..
- Eagle
- EagleFITS
- Illustris
- Millennium** (highlighted with a red box)
- Millennium2
- millimil
- MRObs
- ScaleFree
- sdss_ml
- yt_samples

View 2: snapdir_040 Subdirectory
The path is `/ virgo / snapdir_040`. The file list includes:

- snapdir_032
- snapdir_033
- snapdir_034
- snapdir_035
- snapdir_036
- snapdir_037
- snapdir_038
- snapdir_039
- snapdir_040** (highlighted with a red box)
- snapdir_041
- snapdir_042

View 3: snap_millennium_040 Files
The path is `/ virgo / snapdir_040 / snap_millennium_040`. The file list includes:

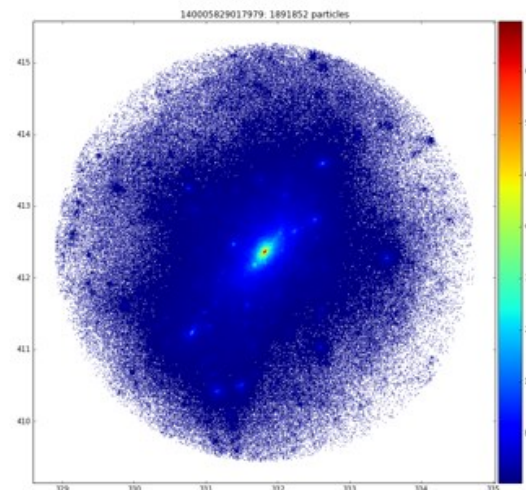
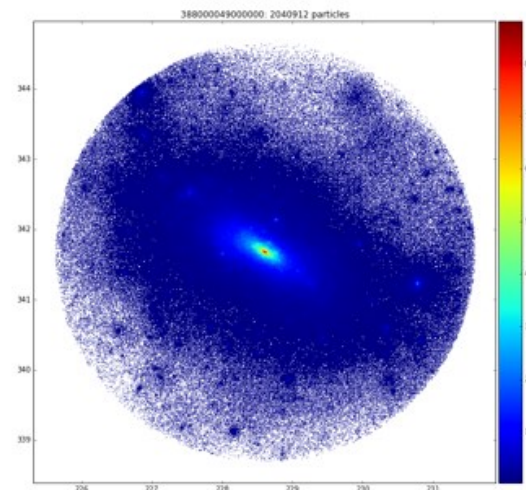
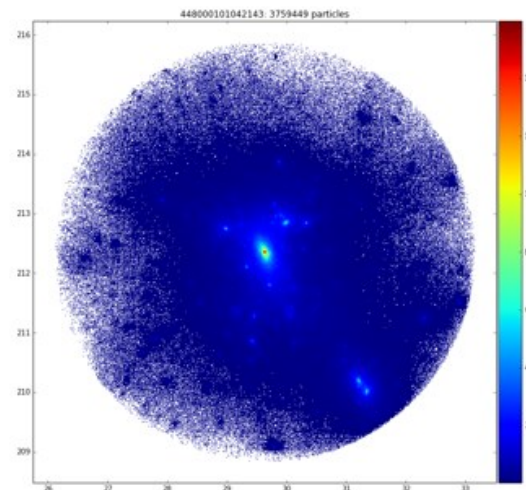
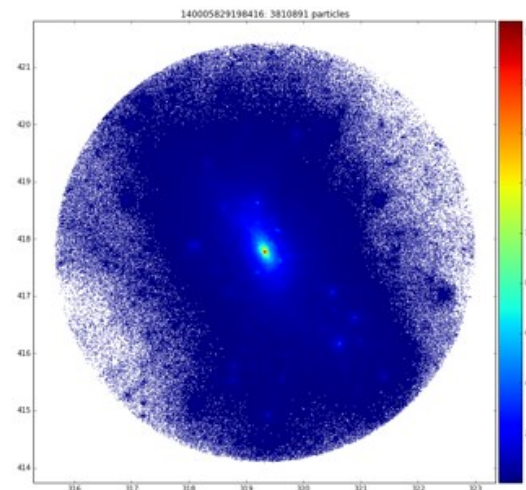
- snap_millennium_040.173
- snap_millennium_040.174
- snap_millennium_040.175
- snap_millennium_040.176
- snap_millennium_040.177
- snap_millennium_040.178
- snap_millennium_040.179
- snap_millennium_040.18
- snap_millennium_040.180
- snap_millennium_040.181
- snap_millennium_040.182
- snap_millennium_040.183
- snap_millennium_040.184
- snap_millennium_040.185

Each file in View 3 is accompanied by a timestamp 'a year ago' and a size in MB.

File Name	Timestamp	Size (MB)
snap_millennium_040.173	a year ago	569 MB
snap_millennium_040.174	a year ago	666 MB
snap_millennium_040.175	a year ago	529 MB
snap_millennium_040.176	a year ago	564 MB
snap_millennium_040.177	a year ago	572 MB
snap_millennium_040.178	a year ago	525 MB
snap_millennium_040.179	a year ago	597 MB
snap_millennium_040.18	a year ago	644 MB
snap_millennium_040.180	a year ago	649 MB
snap_millennium_040.181	a year ago	639 MB
snap_millennium_040.182	a year ago	766 MB
snap_millennium_040.183	a year ago	622 MB
snap_millennium_040.184	a year ago	595 MB
snap_millennium_040.185	a year ago	595 MB


```
ax.set_title(title)
divider = make_axes_locatable(ax)
# Append axes to the right of ax3, with 20% width of ax3
cax = divider.append_axes("right", size="5%", pad=0.05)
# Create colorbar in the appended axes
# Tick locations can be set with the kwarg 'ticks'
# and the format of the ticklabels with kwarg 'format'
cbar = plt.colorbar(im, cax=cax, ticks=MultipleLocator(0.2), format="%0.2f")
```

CPU times: user 17.4 s, sys: 3.87 s, total: 21.3 s
Wall time: 27 s



Thank you

