High-Speed Access for an NVO Data Grid Node

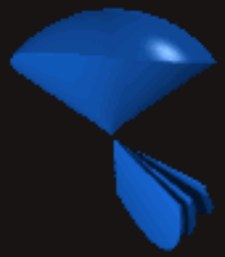
María A. Nieto-Santisteban, Aniruddha R. Thakar, Alex Szalay, Tanu Malik,

The Johns Hopkins University

Jim Gray

Microsoft Research

Sloan Digital Sky Survey

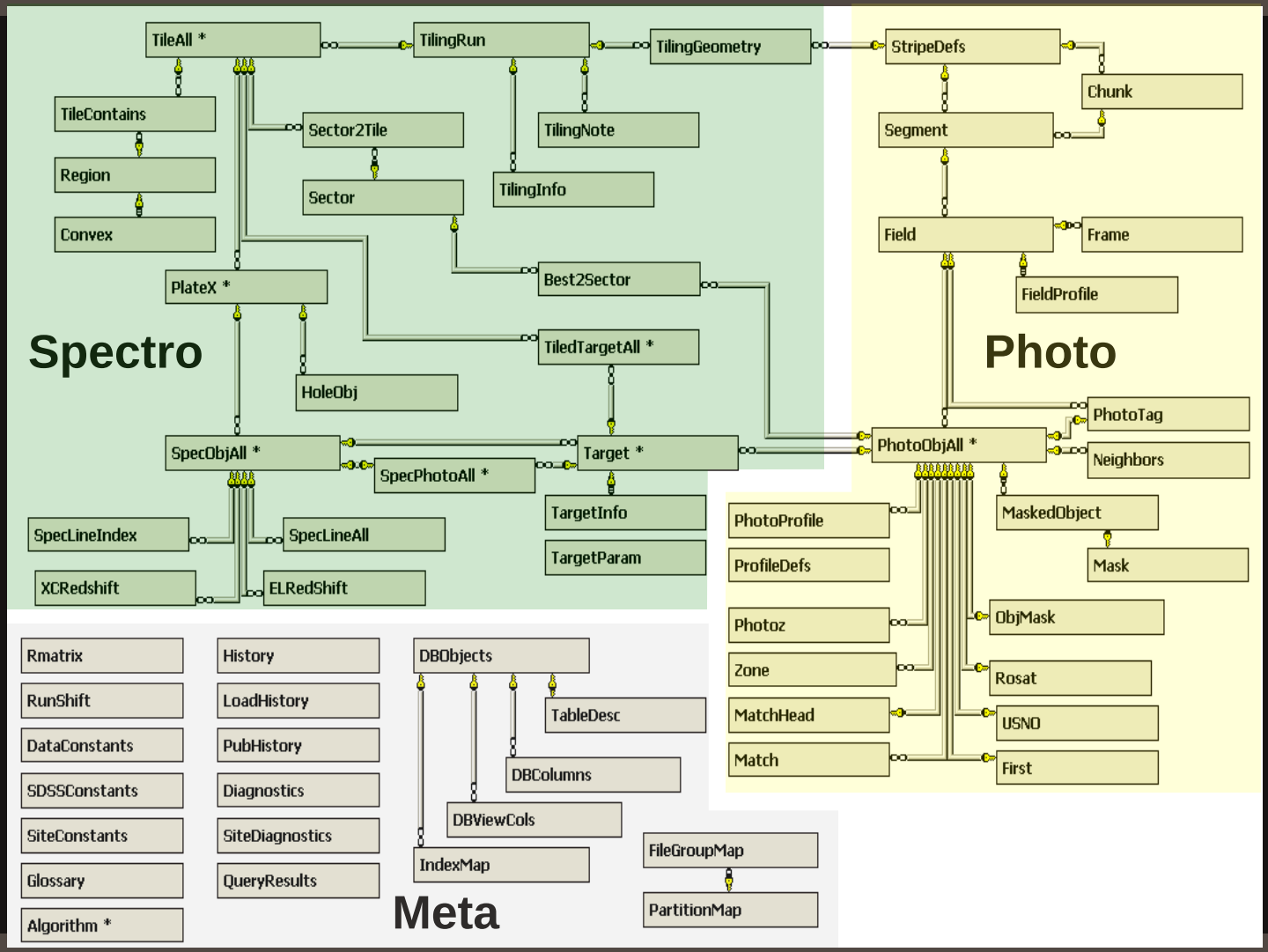


- Digital map in 5 spectral bands covering ¼ of the sky.
- Will obtain 40 TB of raw pixel data.
- Photometric catalog with more than 200 million objects.
- Spectra of 1 million objects.
- Data Release One – DR1: 6 TB of images, 200 k spectra.

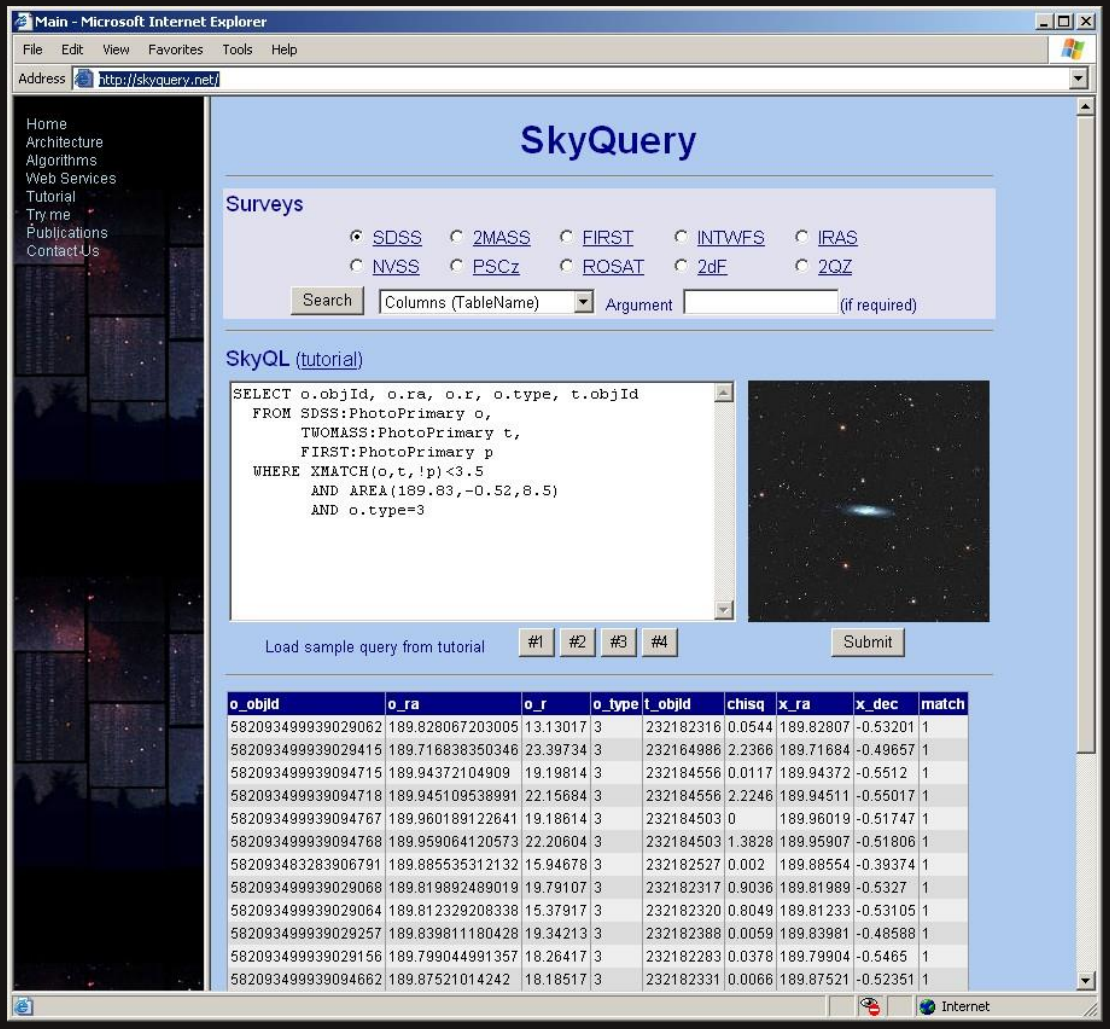
The SkyServer Database

- Processed data is stored into a relational database, **SkyServer**.
- Allows fast exploration and analysis of the data (Data Mining).
- DR1 data base (Best + Target): ~ 1TB and 6TB for final release.
- Heavily indexed to speed up access.
- Short queries can run interactively.
- Long queries (> 1 hour) require a custom Batch Query System.
- DBMS, Microsoft SQL Server 2000.

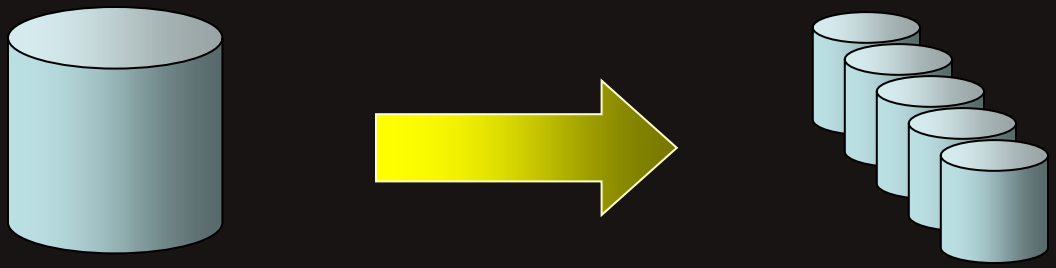
SkyServer Database Schema



SkyServer and the NVO



Partitioning and Parallelization



Goals

- Speed up query execution:
  - Queries looking at different parts of the sky are distributed among servers.
  - Queries covering wide areas are executed in parallel by different servers. (Sequential scans fall naturally in this range)
  - Neighborhood queries are "isolated" and processed in parallel. (Gravitational lenses and Galaxy clusters)
- Speed up cross-match requests from other NVO data nodes.

Partitioning Facts

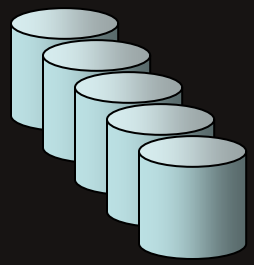
- *Partitioning works well if tables in the database are naturally divisible into similar partitions where most of the rows accessed by any SQL statement can be placed on the same member server.*
- *Partitioning is most effective if the tables in the database can be partitioned symmetrically. (Not exactly our case)*
- *Related data should be placed on the same member server so most SQL statements routed to a member will require minimum data from other servers.*
- *Data should be partitioned uniformly across the member servers.*

Designing Partitions. Microsoft SQL Server Books Online

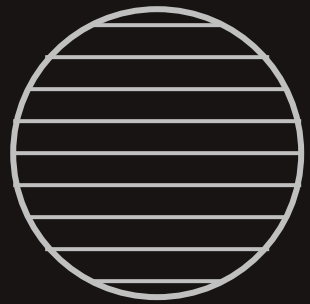
Partitioning Strategy

- Two-Step Process

1. Distribute data homogenously among servers.
   - Each server has roughly the same amount of objects.
   - Objects inside servers are spatially related.
   - Balances the workload among servers.
   - Queries redirected to the server holding the data.



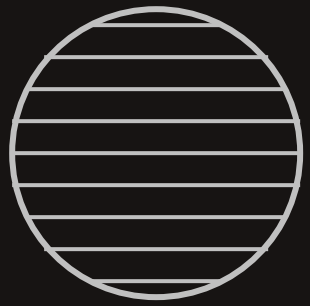
2. (Re)Define zones inside each server dynamically.
   - Zones are defined according to some search radius to solve specific problems:
     - Finding Galaxy Cluster,
     - Gravitational Lenses, etc.
   - Facilitates cross-match queries from other NVO data nodes.



Mapping the Sphere into Zones

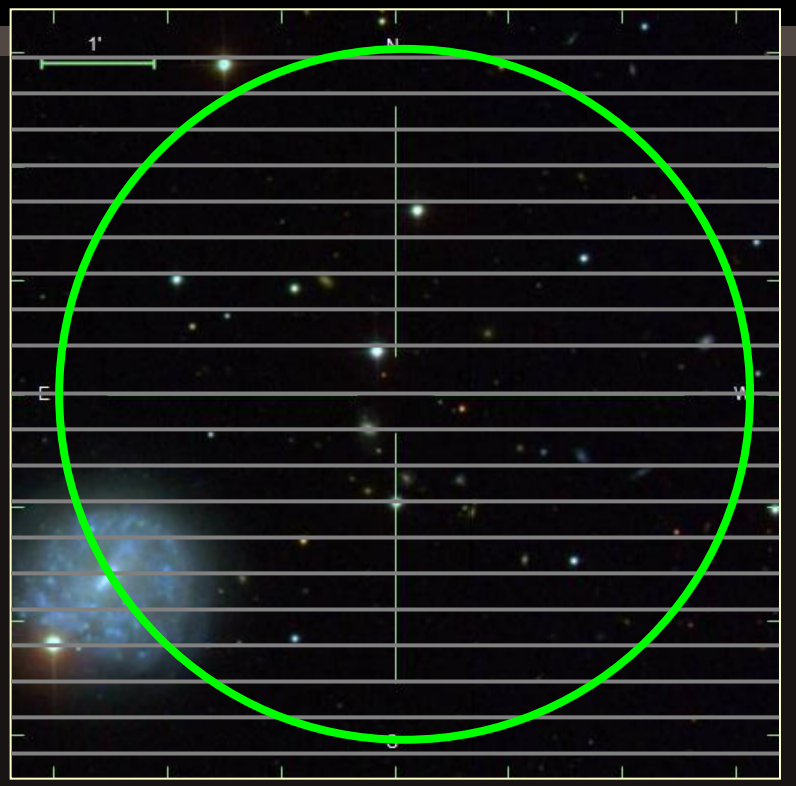
- Each Zone is a declination stripe of height h.
- In principle, h can be any number. In practice, 30 arcsec. (DR1 => 8000 ZONES)
- South-pole zone = Zone 0.
- Each object belongs to one Zone:

$$\text{ZoneID} = \text{floor}\left((\text{dec} + 90) / h \right)$$



- Each server holds N "contiguous" Zones.
- N is determined by the number of objects that each Zone contains and the number of servers in the cluster.
  - Not all servers contain the same number of zones.
  - Not all servers cover the same declination range.
- Straightforward mapping between queries and servers.

Cone Searches using Zones



- ConeSearch (ra, dec, r)
  - Need to search only on zones between:

maxZone = ceiling ((dec + 90 + r)/ h)

minZone = floor ((dec + 90 – r)/ h)

  - Restrict search on dec to

dec ∈ [(dec – r), (dec + r)]

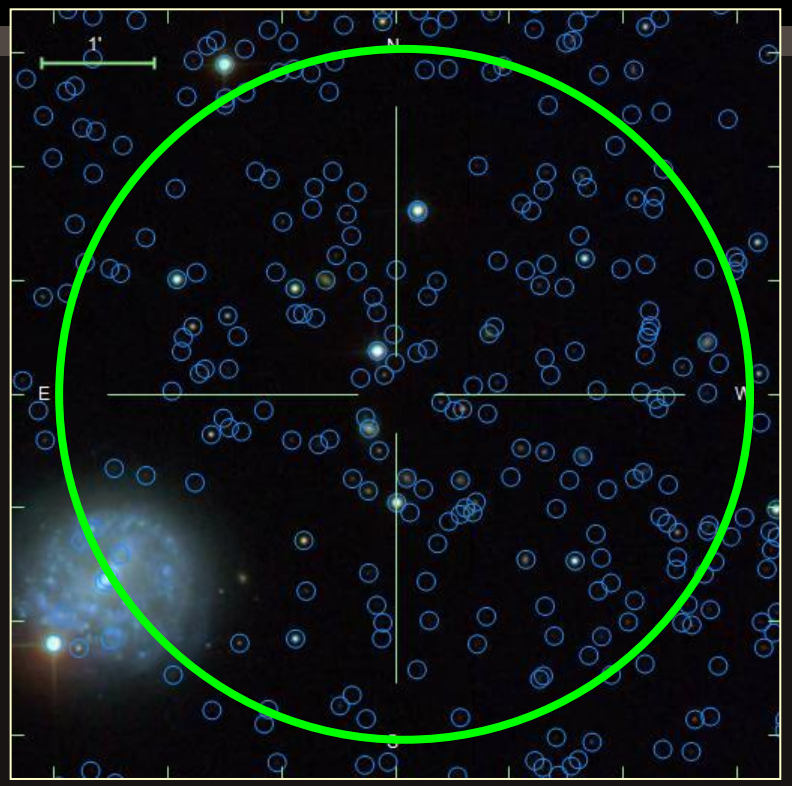
  - Restrict search on ra to

ra ∈ [(ra–r)/(cos(|dec|) + ε), (ra+r)/(cos(|dec|) + ε)] ε = 1 e–6

  - Filter on distance

$$\sqrt{((cx - x)^2 + (cy - y)^2 + (cz - z)^2)} < r$$

Cone Searches using Zones



- ConeSearch (ra, dec, r)
  - Need to search only on zones between:

maxZone = ceiling ((dec + 90 + r)/ h)

minZone = floor ((dec + 90 – r)/ h)

  - Restrict search on dec to

dec ∈ [(dec – r), (dec + r)]

  - Restrict search on ra to

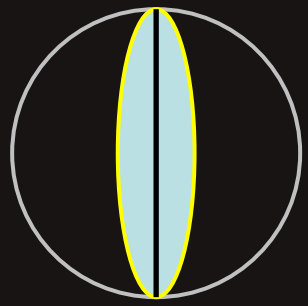
ra ∈ [(ra–r)/(cos(|dec|) + ε), (ra+r)/(cos(|dec|) + ε)] ε = 1 e–6

  - Filter on distance

$$\sqrt{((cx - x)^2 + (cy - y)^2 + (cz - z)^2)} < r$$

Margins & Buffers

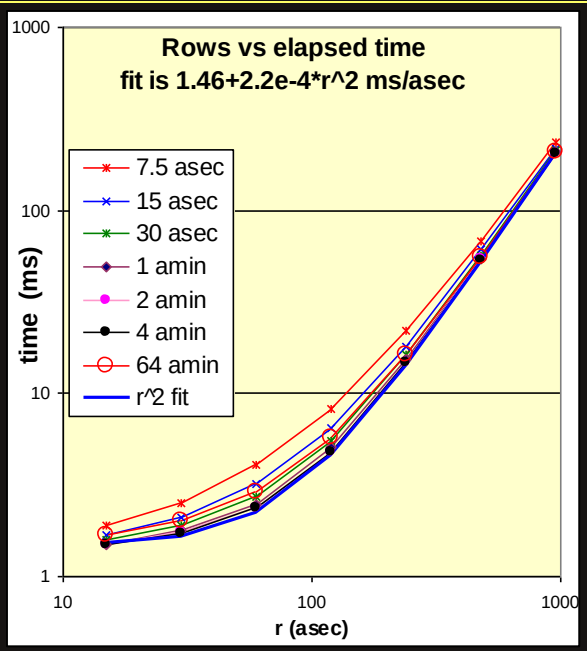
- To improve queries around Ra = 0 (or 360):
  - Duplicate objects inside Ra = [-1, 0) and Ra = (360, 361] facilitates searches.
  - Objects in the margin area are marked as Margin.



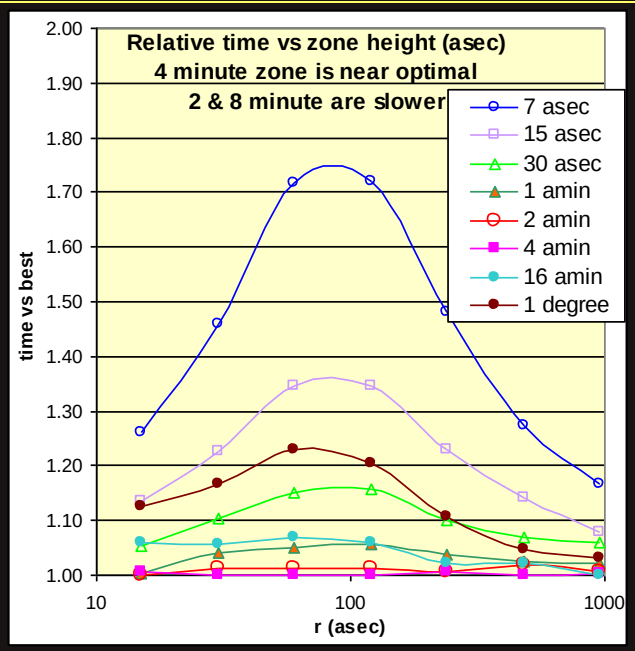
- To guarantee that neighboring searches can be fully satisfied inside a single server some zones are replicated:
  - Each server adds 2 extra buffer regions of height R_M
  - R_M, is the maximum neighboring distance we assume (1 degree).
  - Objects in buffers are "marked" as Visitors to the Server.

Zoning Performance

- Time vs Search Radius for Cone Search searches



Any small zone height is adequate



h of 4 arcminutes is near-optimal

- 7x faster than using external calls to the HTM functions

Partitioning Process

- **Generate partitions (n_servers, n_buffers)**
  1. Calculate the number of objects included on each Zone, N_z.
  2. Compute the accumulated distribution of objects , A, for each Zone. A_{Zi} = Sum (N_{zi}), i = 1 .. i
  3. Assign the 100/n_servers % of objects to each server.
  4. Add to each server the buffer zones.
  5. Add margin objects.

  Output:
  - ServerZones (ServerId, ZoneID, objID, ra, dec, x, y, z, wrap, native, …) Indexed by ZoneID and objID for fast access!
  - Servers (ServerID, nObj, minZoneID, maxZoneID, overlapMinZoneID, overlapMaxZoneID, minDEC, maxDEC)
- Transfer data from main server to cluster members.

Data Transfer: Main Server - Nodes

1. Replicate the database schema on each server to maintain relationships between tables.
2. Member servers pull data from the main server using the ServerZones table.
   - Can be done in parallel.
   - Easier for the transaction manager.
3. Asymmetric partitioning:
   - Replicate most of the tables on each server. It makes it easier and faster.
   - Partition PhotoObjAll and SpecObjAll … maybe others.
4. Rebuild indexes.

Routing Rules Definition

- Determine where to send a query.
- Need a parser to capture regions requests like POINT, CIRCLE, REC, POLY … etc. (Reuse parser from SkyQuery.)
- Once we have a declination, (or x,y,z)

```
server =   SELECT ServerID
           FROM   Servers
           WHERE ( obj.dec  BETWEEN minDEC AND maxDEC )
```

- Queries without positional constrains mean full table scans and have to be sent to all nodes to be processed in parallel.

Building Neighborhoods

- Computing neighborhoods is computationally-intensive.
- Nested loop where for each object all neighbors inside some radius are computed.
- For completeness @deltazone = {-1, 0, 1}

```
insert      neighbors                                          -- insert one zone's neighbors
select      o1.objID as objID,                                 -- object pairs
            o2.objID as NeighborObjID,

from        zone o1 join zone o2                               -- join 2 zones
        on  o1.zoneID - @deltaZone = o2.zoneID                 -- using zone number and ra
        and o2.ra between o1.ra - @r and o1.ra + @r            -- points near  ra
where                                                          -- elided margin logic
  and o2.dec between o1.dec - @r and o1.dec + @r               -- quick filter on dec
  and sqrt  ( power(o1.x - o2.x, 2) + power(o1.y - o2.y, 2) + power(o1.z - o2.z, 2))
        < @r                                                   -- careful filter on distance
```

Building Neighborhood Performance

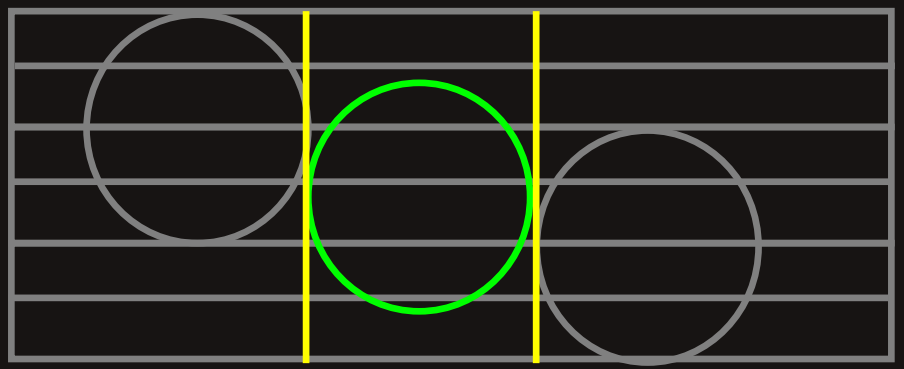
- Results for Personal SkyServer (154k rows)

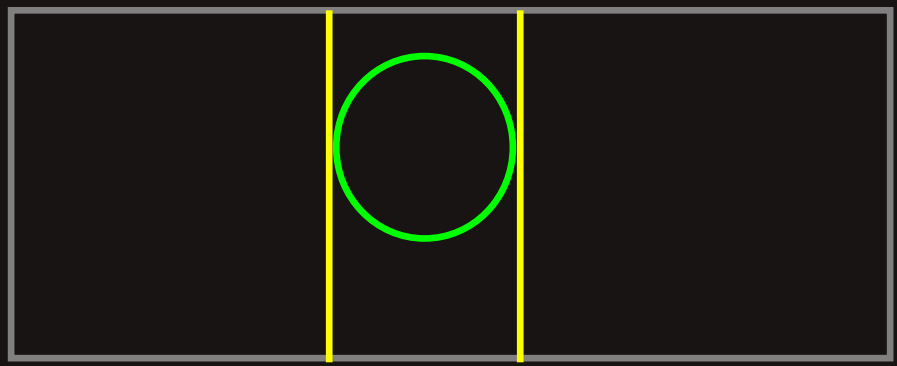
– Build Zone table:	9.483 s	
– Join to Zone –1:	10.487 s	generated 128,469 rows
– Join to Zone 0:	16.513 s	generated 389,157 rows
– Join to Zone 1:	9.433 s	generated 126,104 rows
– Add mirror rows:	10.723 s	Total = 1,287,460 rows
– Create the index:	7.563 s	
Total time	64.203 s	

- For DR1, computing the neighbor table (30") took 2 days instead of 2 weeks.
- The overall improvement has been 32x faster than using external calls to the HTM functions.
- Can be done in Parallel!

Neighborhoods best Performance

- Building Neighborhoods performs best when the zone height is equal to the radius of the neighborhood.



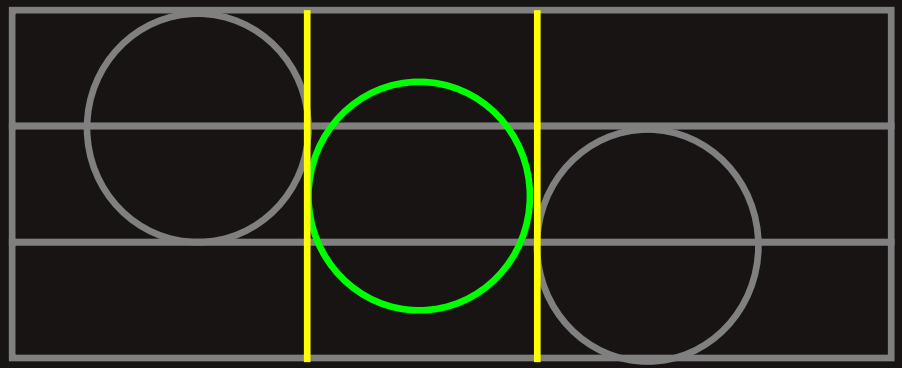


small radius imply joins with two or more northern zones and two or more southern neighbors.

Tall zones require many more pairs and the work rise quadraticly.

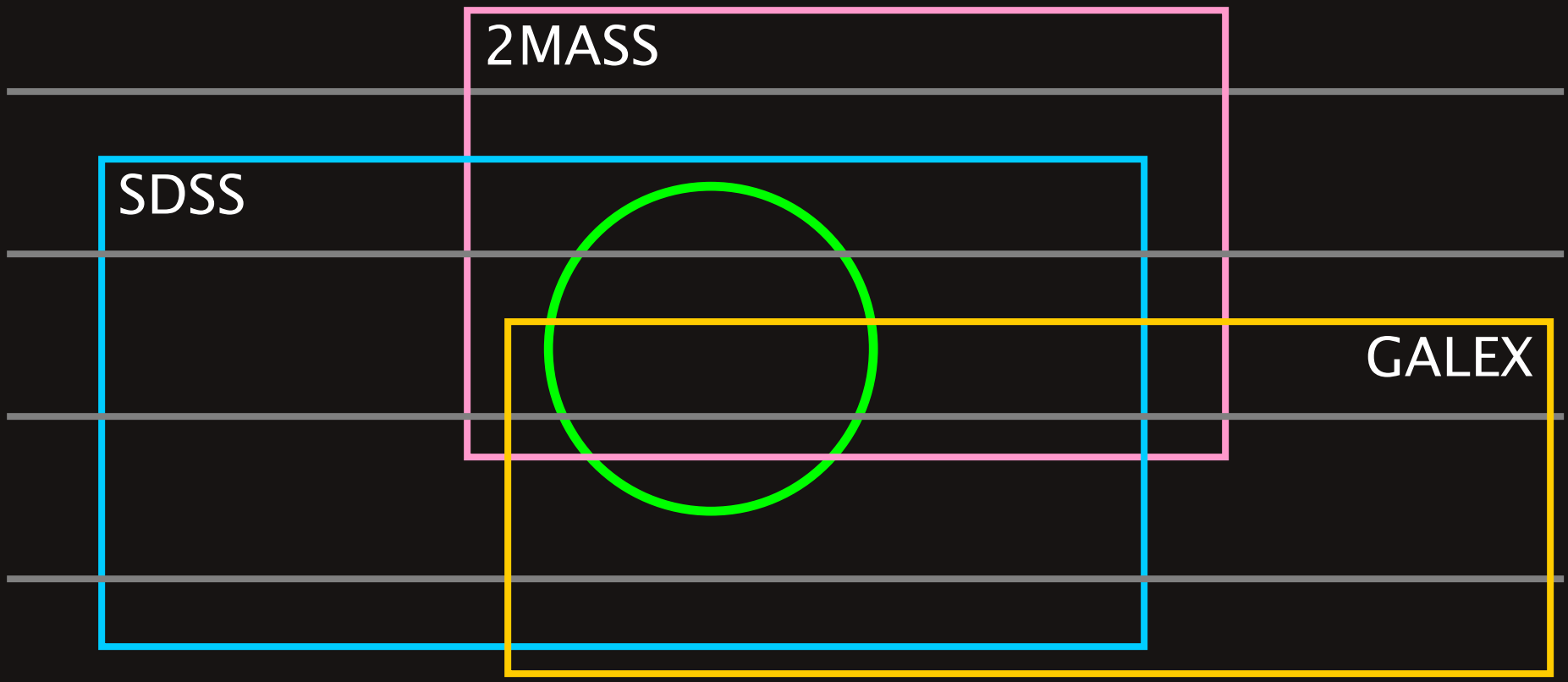
Neighborhoods best Performance

- Building Neighborhoods performs best when the zone height is equal to the radius of the neighborhood.



the center zone requires just a join with the upper and southern zones. A box of 3r x 2r.

Zones and Cross-Match



- Applying a zoning approach to other surveys makes JOINs a faster process.

Work to do

- Do the actual partitioning of SkyServer. Test it and measure performance.
- So far we have “played” with MySkyServer a subset of DR1 with 1.3 GB.
- Test with the finding galaxy cluster algorithm to compare against current grid approach.
- Connect our databases to do the high computational processing on the GRID.

After all ... Why High-Speed Access?

To allow interactive exploration and visualization of the data and do new discoveries!

Any time left for the Image Cutout demo?

It takes 5 minutes.