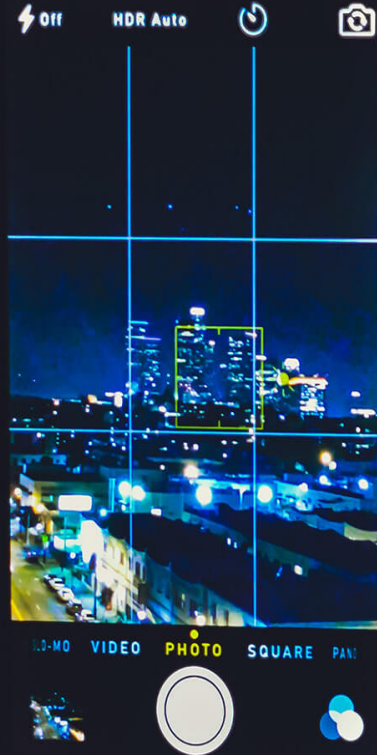


Geospatial and MongoDB



Agenda

MongoDB Geospatial Features
Query Examples
Optimizations

Norberto Leite

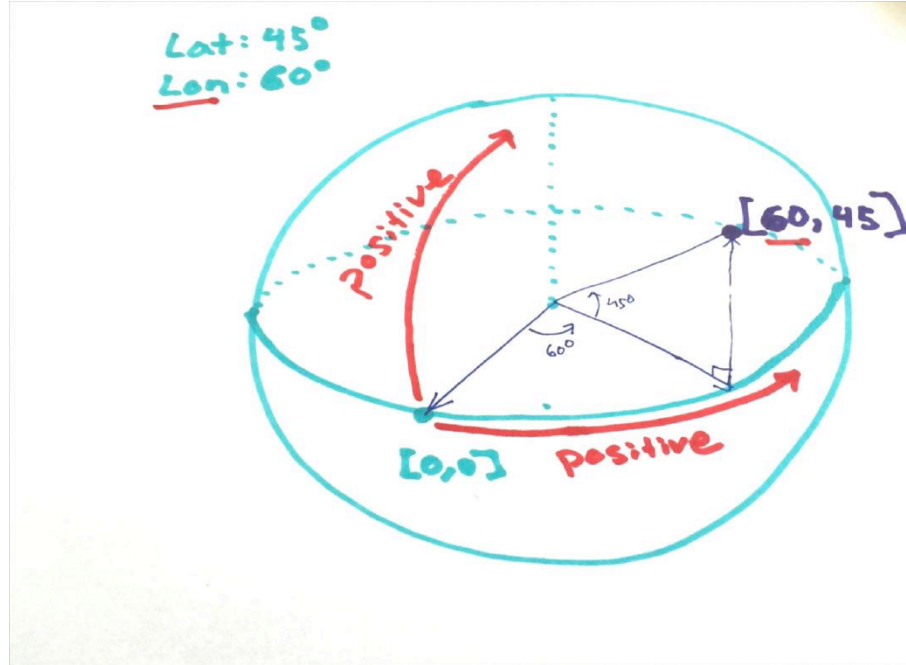
Developer Advocate
Curriculum Engineer

Twitter: @nleite
norberto@mongodb.com



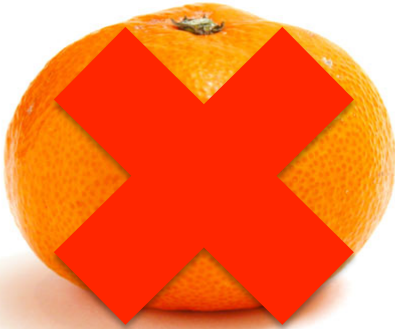
The Basics

[Longitude, Latitude]

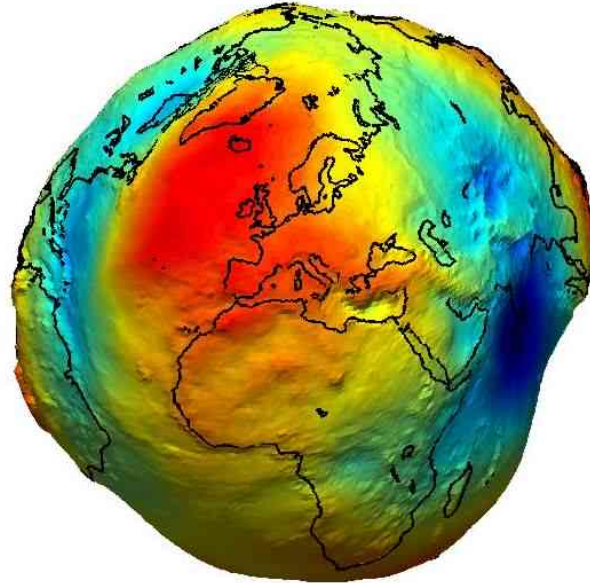


Quiz Time!

Which of these shapes is the most similar with Planet Earth?



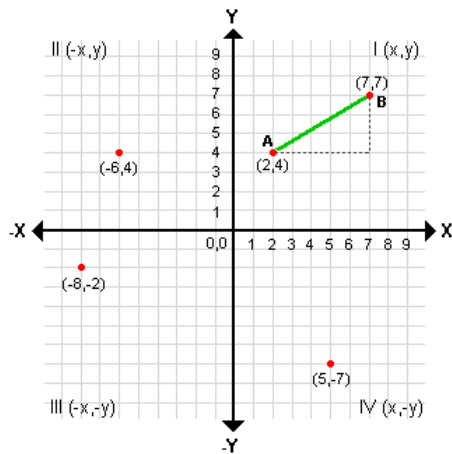
Stripped version of Earth: Geoid



<http://www.ngs.noaa.gov/GEOID/>

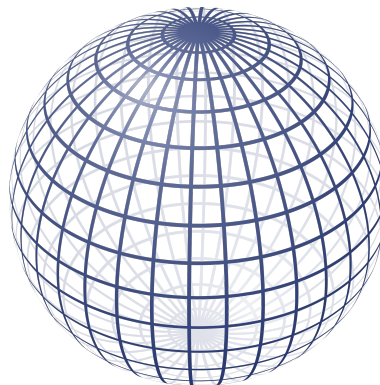
Surface Types

Flat

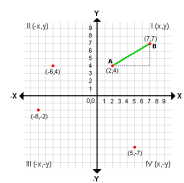


2d Indexes

Spherical



2dsphere Indexes



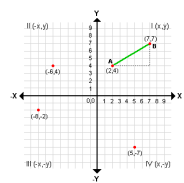
2D Indexes

```
var place = {  
  type: "building",  
  name: "AW1 - Building"  
  location: [4.380717873573303, 50.81219570880462]  
}
```

Defined by [lon,lat] arrays

```
var checkin = {  
  type: "checkin",  
  message: "this place is awesome!"  
  location: {  
    lng: 4.348099529743194,  
    lat: 50.850980167854615  
  }  
}
```

Or use an embedded
document

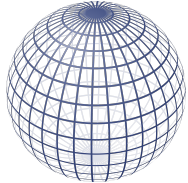


2D Indexes

//index creation

```
db.col.createIndex( {'location': '2d'})
```

```
db.col.createIndex( {'location': '2d'}, {'sparse': true})
```



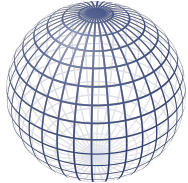
Spherical Surface

```
var place = {  
  type: "building",  
  name: "AW1",  
  location: {  
    //Line, MultiLine, Polygon, MultiPolygon, GeometryCollection  
    type: "Point",  
    coordinates: [4.380717873573303, 50.81219570880462]  
  }  
}
```

Defined by a
subdocument – GeoJSON

Requires a type

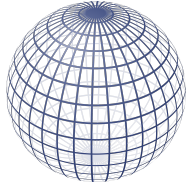
Coordinates array



Spherical Surface

```
var place = {  
  type: "building",  
  name: "AW1",  
  location: {  
    type: "Polygon",  
    coordinates: [  
      [  
        [  
          4.380406737327576,  
          50.812253331704625  
        ],  
        [  
          4.380889534950256,  
          50.81239569385869  
        ],  
        [  
          4.381093382835388,  
          50.812134696244804  
        ],  
        [  
          4.380605220794678,  
          50.81198894369594  
        ],  
        [  
          4.380406737327576,  
          50.812253331704625  
        ]  
      ]  
    ]  
  }  
}
```

<http://geojson.org/>



2dsphere Indexes

//index creation

```
db.collection.createIndex( { location : "2dsphere" } )
```

//compound with more than 2 members

```
db.collection.createIndex( { location : "2dsphere", name: 1, type: 1 } )
```

2d vs 2dsphere

2d index	2dsphere
Legacy	Spatial Support
Coordinates Pair	GeoJson Format
Manual Earth-like geometry calculations	WGS84 Datum
Single extra field for compound indexes	Multiple fields

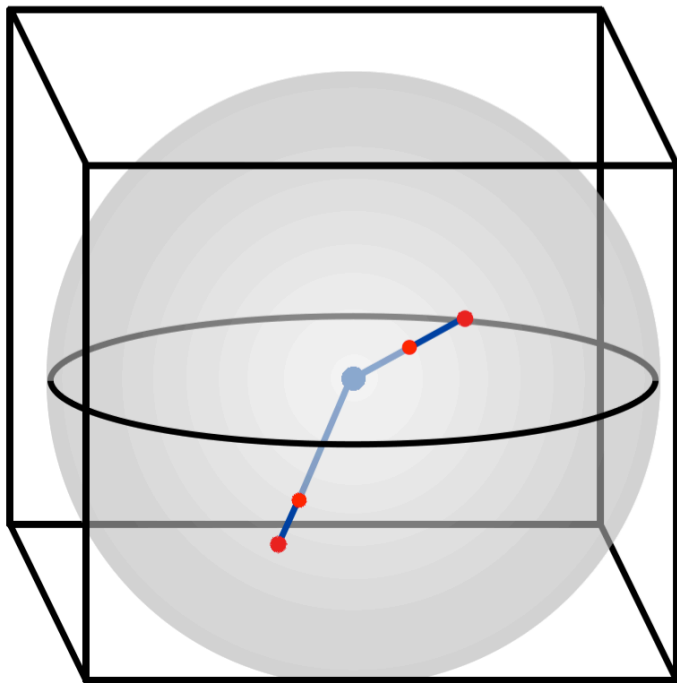
Indexation

Points to Index Keys

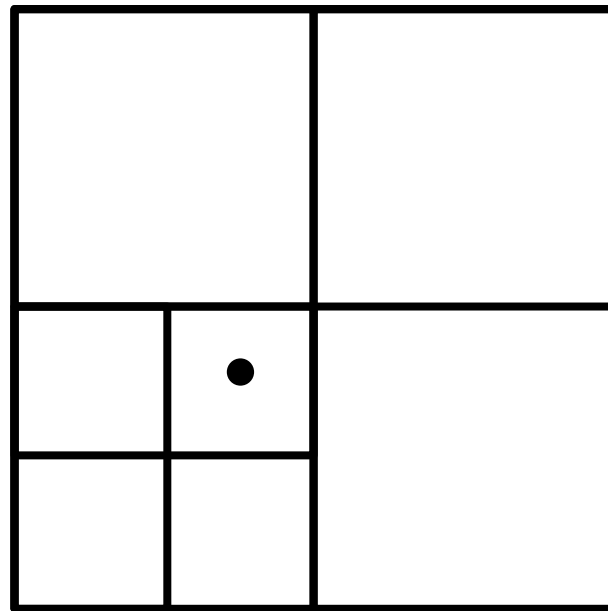
```
db.places.insert({  
  name: "Starbucks"  
  loc: {  
    type: "Point",  
    coordinates: [1.3, 45]  
  }  
})  
db.places.createIndex({loc: "2dsphere"})
```

How does MongoDB generate index keys?

Points to Index Keys



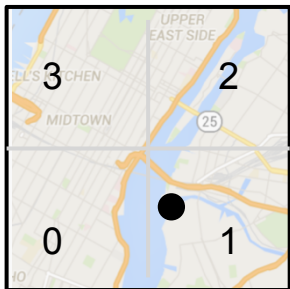
Project spherical point to bounding cube



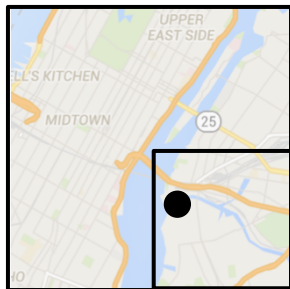
Each face is a Quadtree

Points to Index Keys

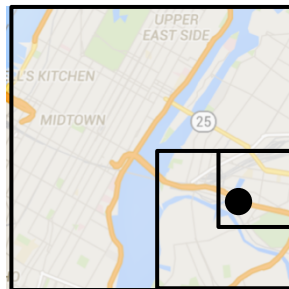
Face 5



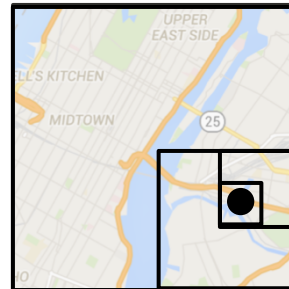
Key 5F



5F1



5F12



5F120

...

Every cm² can be represented with 30 levels

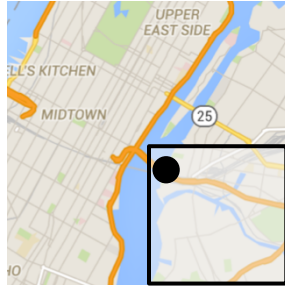
S2 Library

Level	Min Area	Max Area
0	85,011,012 km ²	85,011,012 km ²
1	21,252,753 km ²	21,252,753 km ²
12	3.31 km ²	6.38 km ²
30	0.48 cm ²	0.93 cm ²

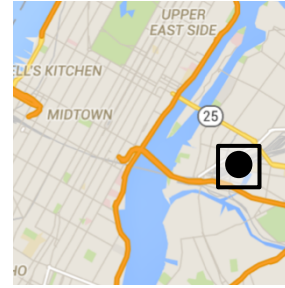


smallest cell

Points to Index Keys



Key **5F1**



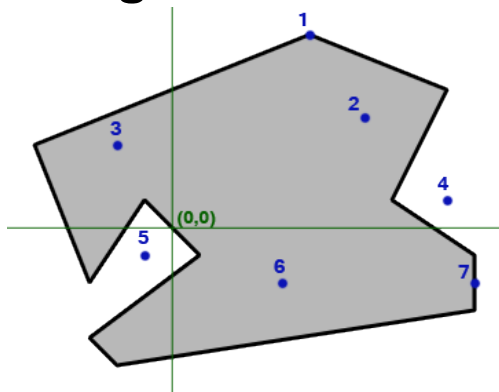
5F120

A key is a prefix of another iff it is a parent cell

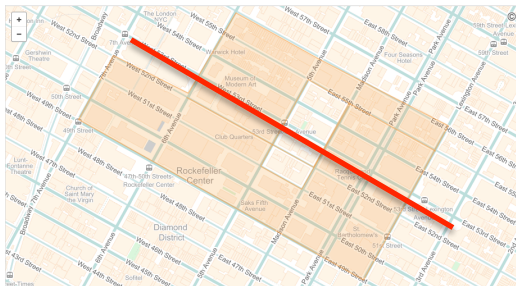
Query Examples

Geospatial operators

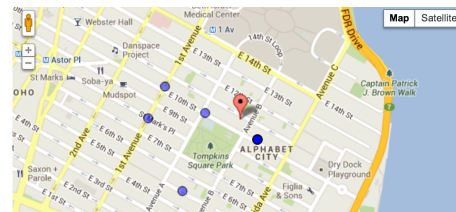
\$geoWithin



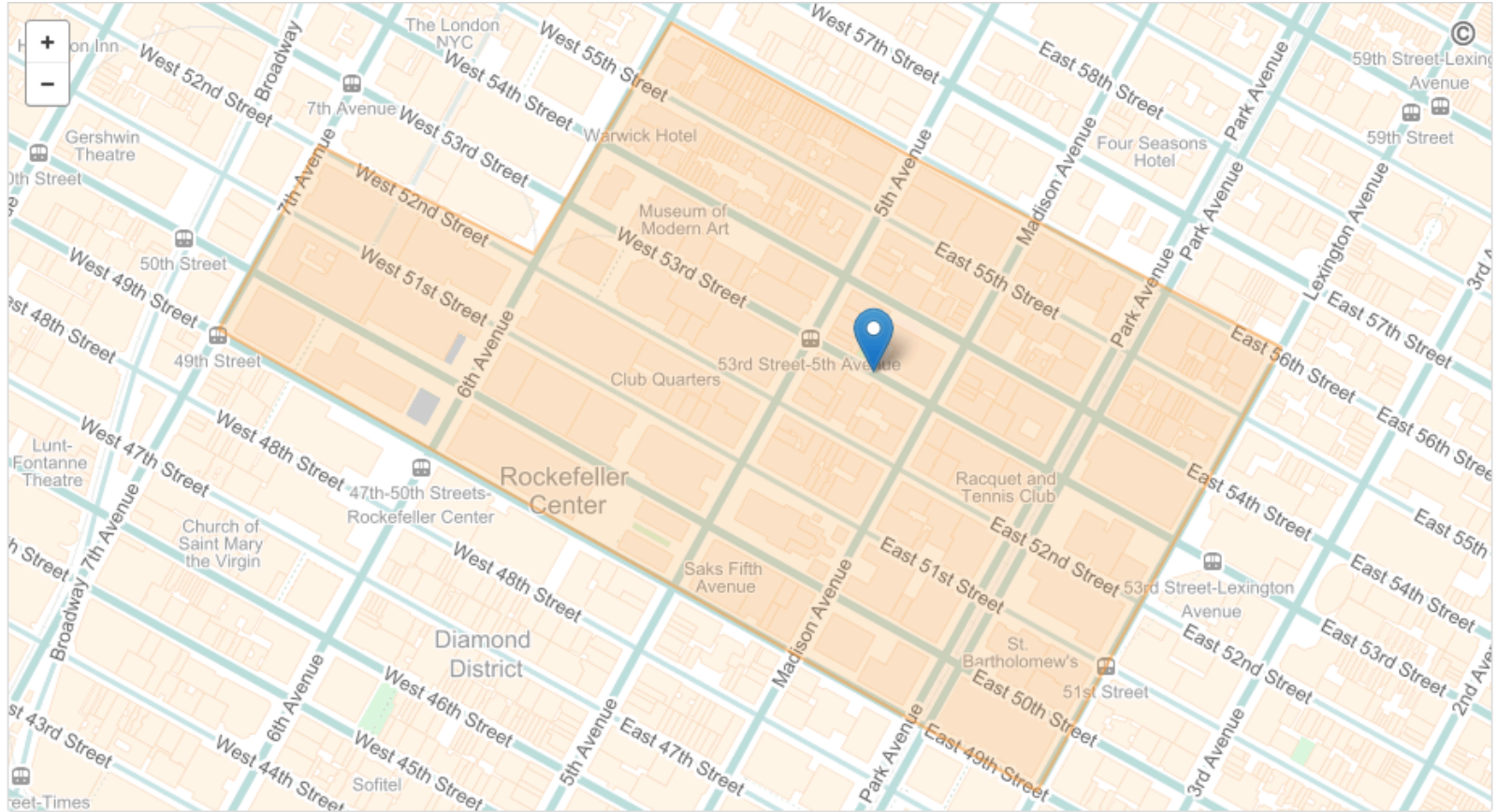
\$geoIntersects



\$near/\$nearSphere

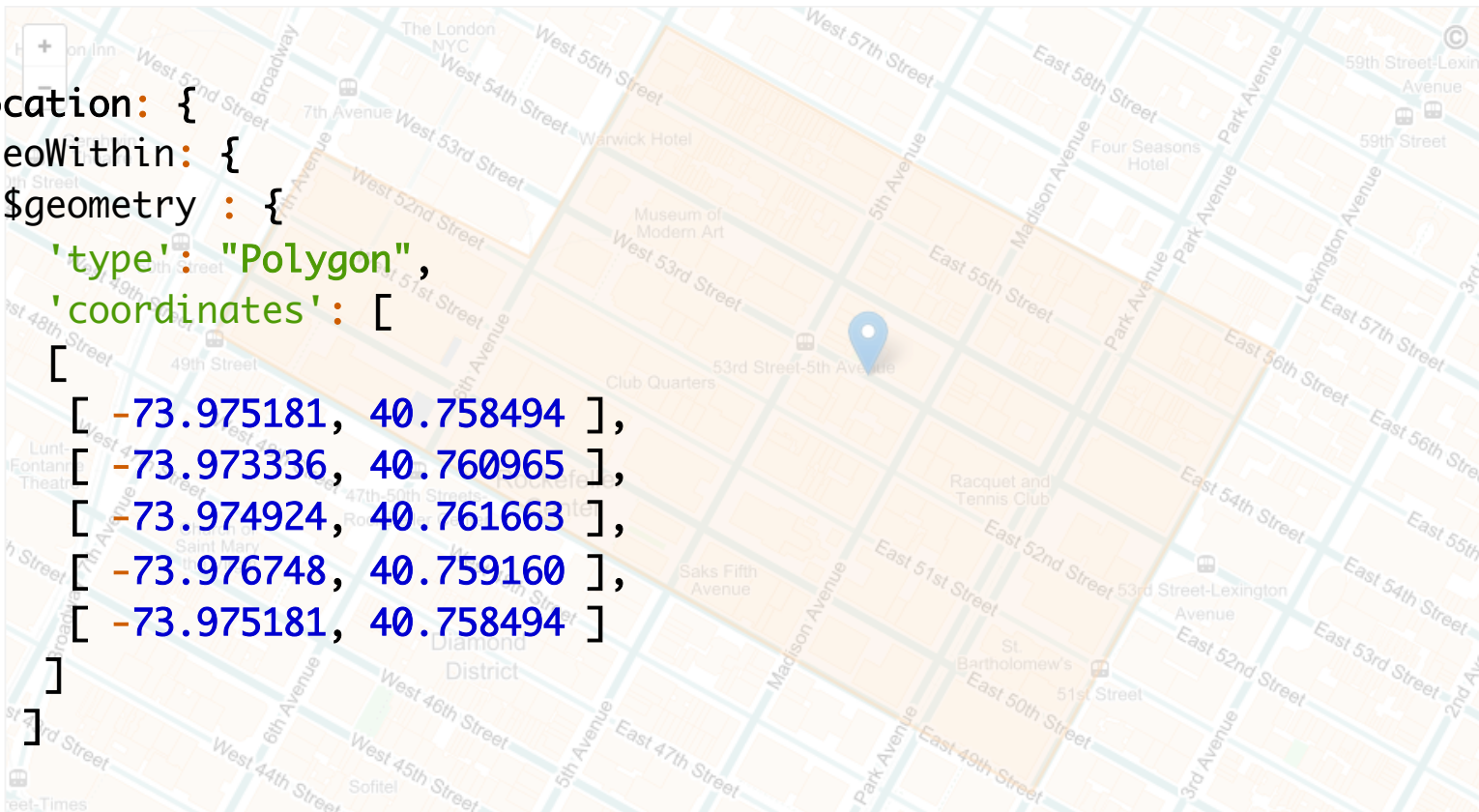


\$geoWithin

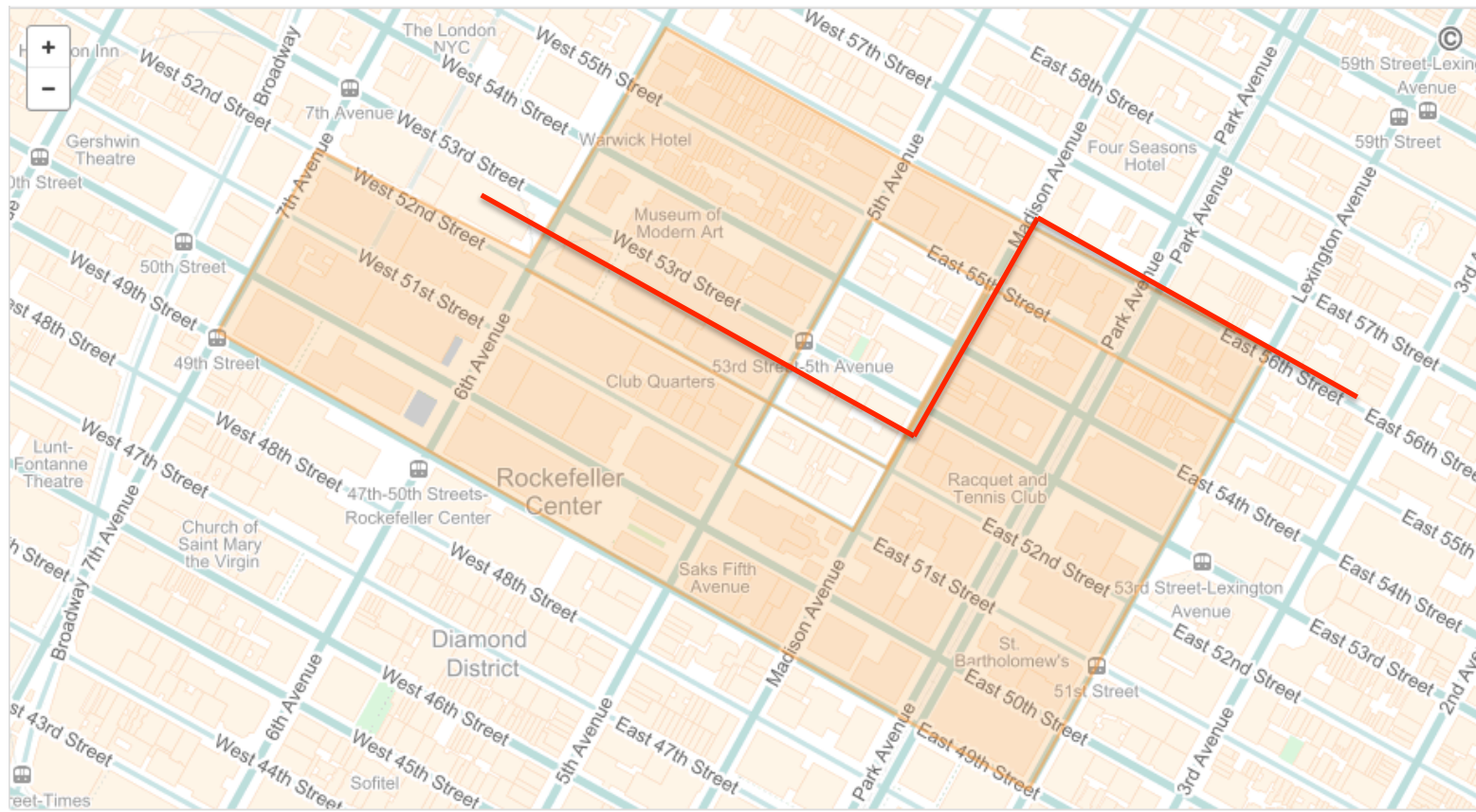


\$geoWithin

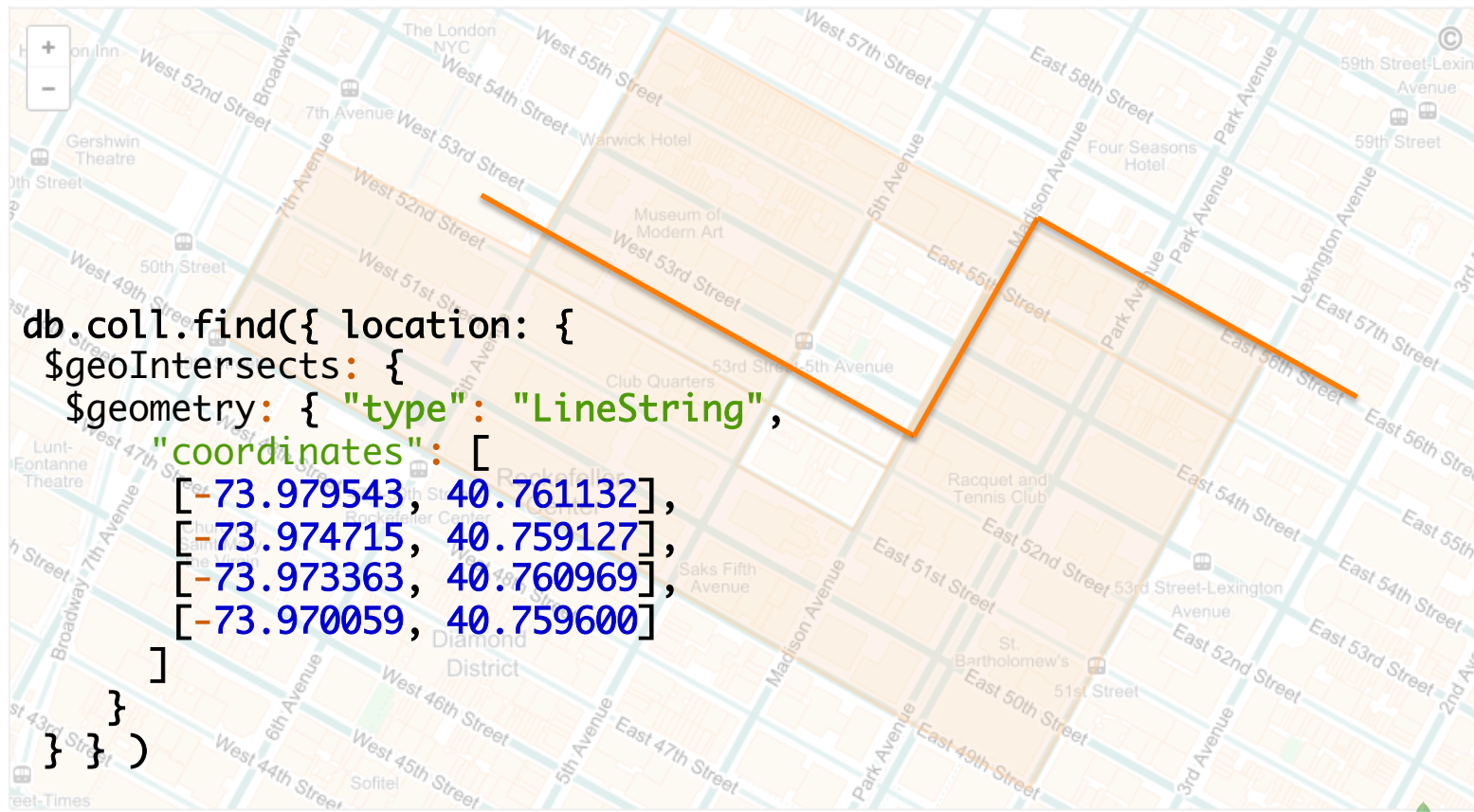
```
{ location: {  
  $geoWithin: {  
    $geometry: {  
      'type': "Polygon",  
      'coordinates': [  
        [  
          [-73.975181, 40.758494 ],  
          [-73.973336, 40.760965 ],  
          [-73.974924, 40.761663 ],  
          [-73.976748, 40.759160 ],  
          [-73.975181, 40.758494 ]  
        ],  
        ]  
      ]  
    }  
  }  
}
```



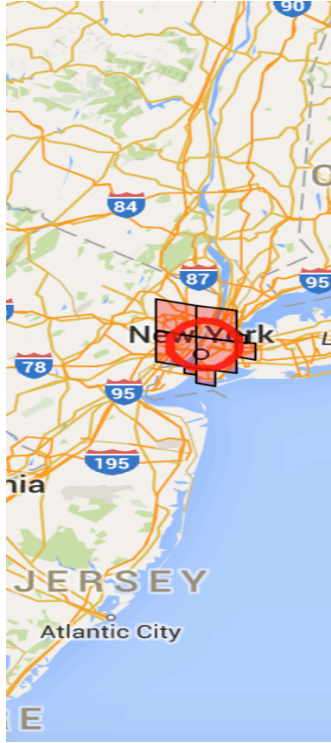
\$geoIntersects



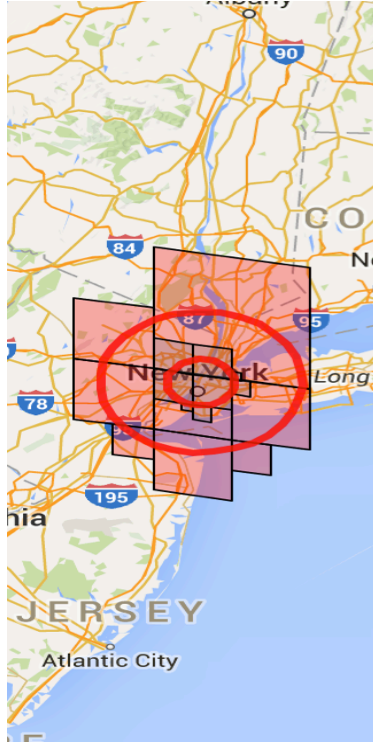
\$geoIntersects



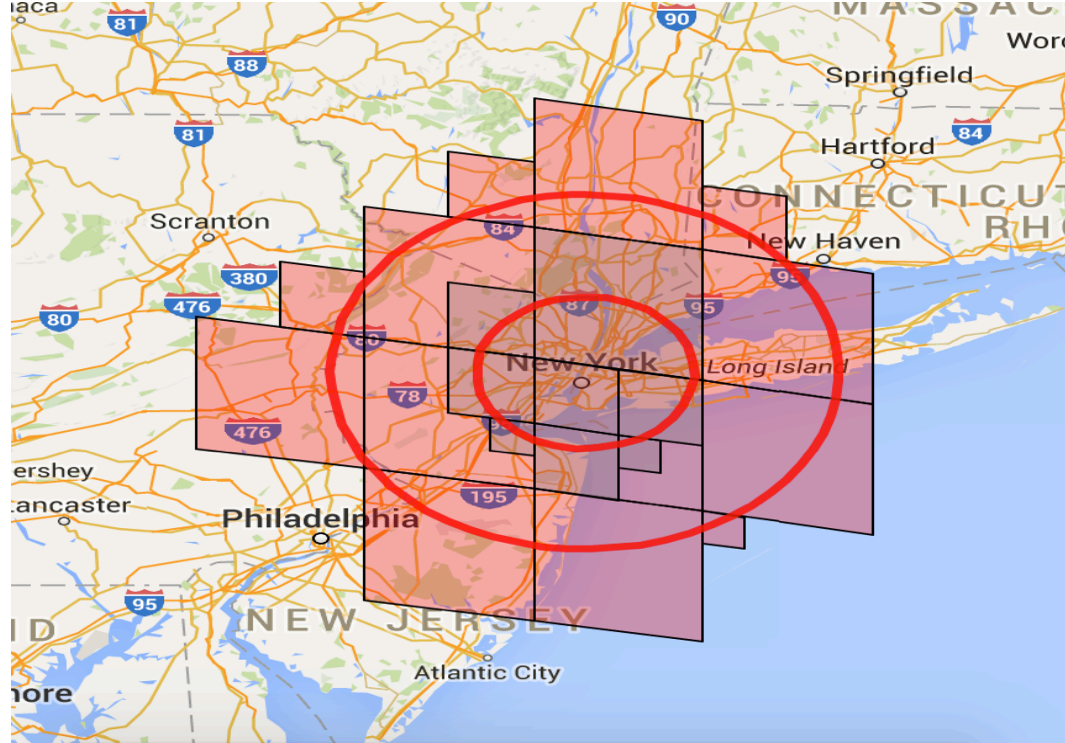
\$near



Stage 1



Stage 2



Stage 3

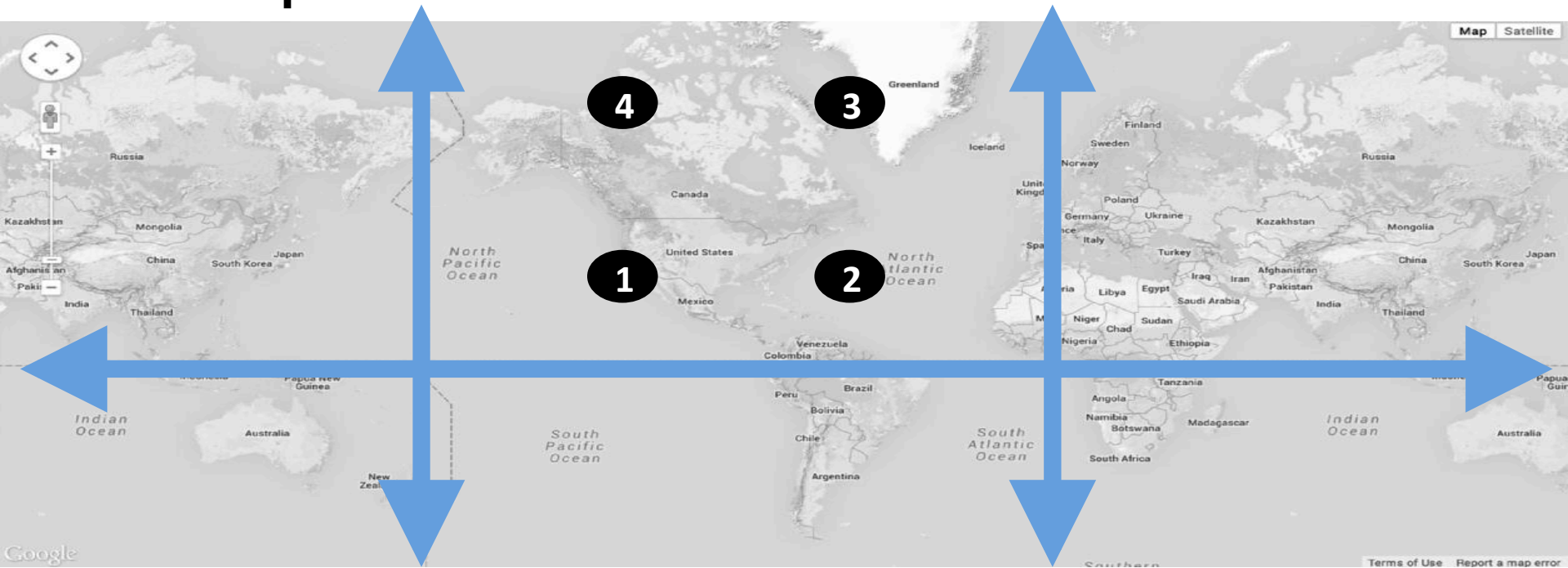
Polygons

A world map showing longitude lines from 90°W to 90°E. The map is divided into four quadrants by the Prime Meridian (0°) and the Equator (0°). Longitude values are displayed in boxes along the Equator: 90, 135, 180, -180, -120, -45, -10, 10, 45, 90. The map includes labels for major countries and oceans.

90

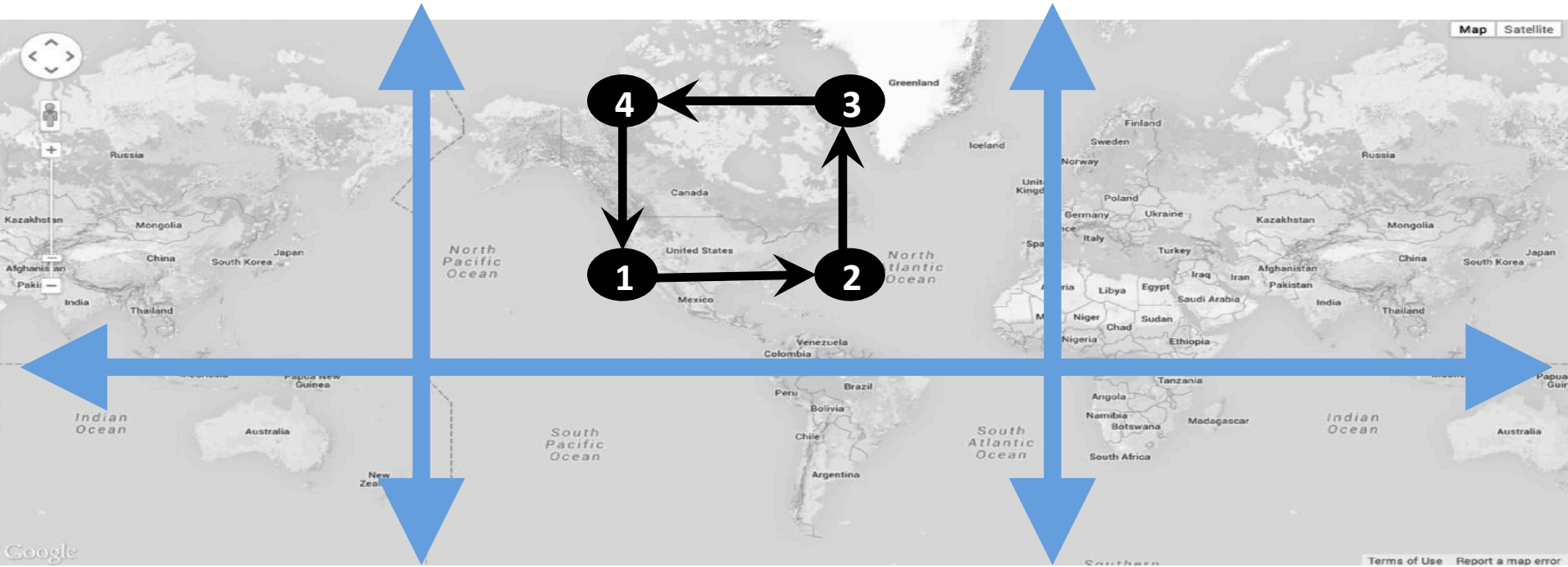
Map Satellite

Define a polygon by specifying 4 points

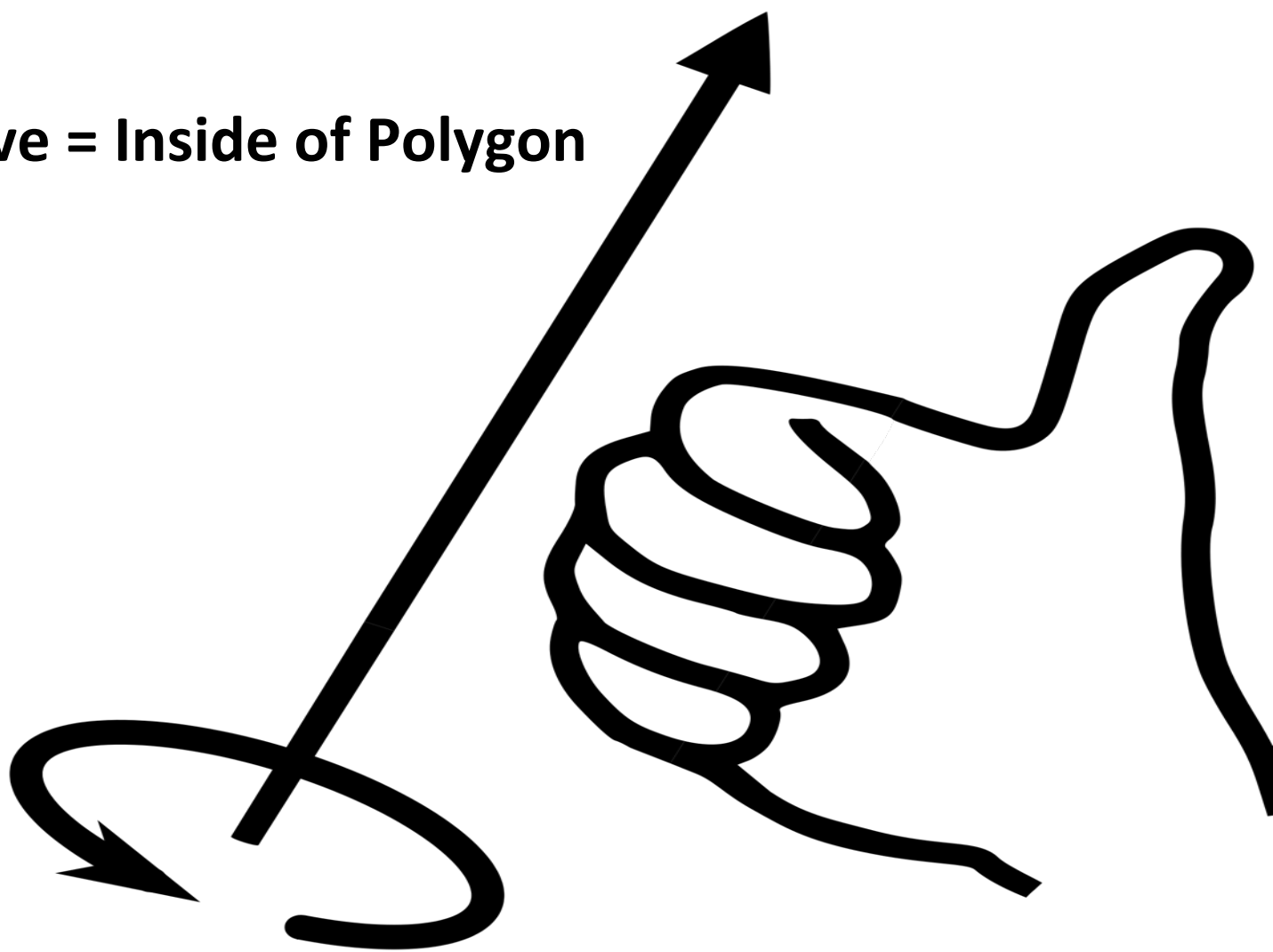


But what's the inside of the polygon?

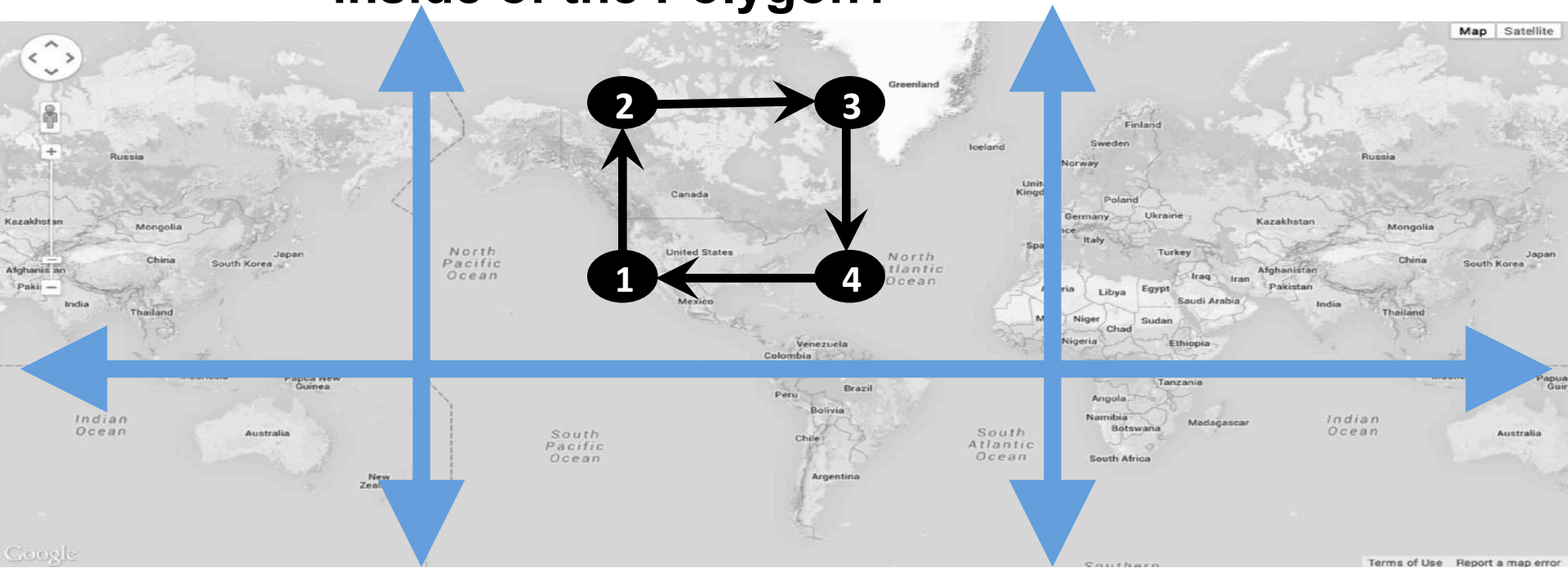
Convention for deciding “inside”: Winding Order + Right Hand Rule



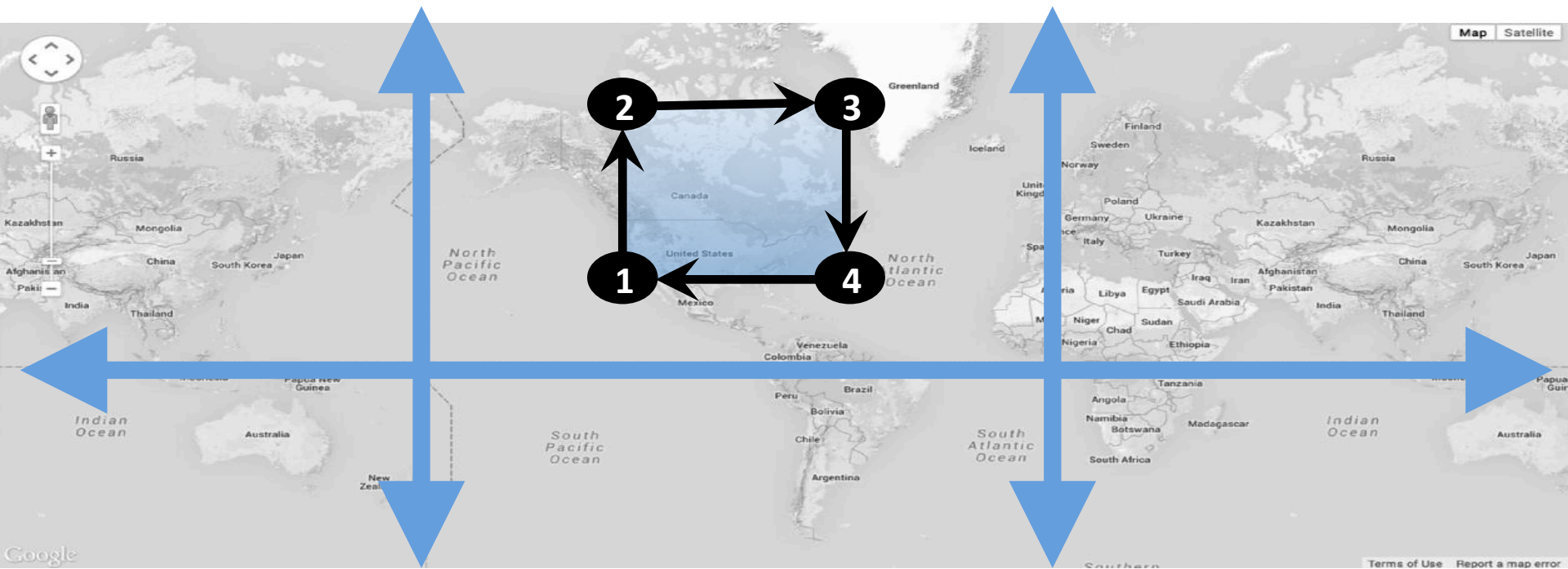
Positive = Inside of Polygon



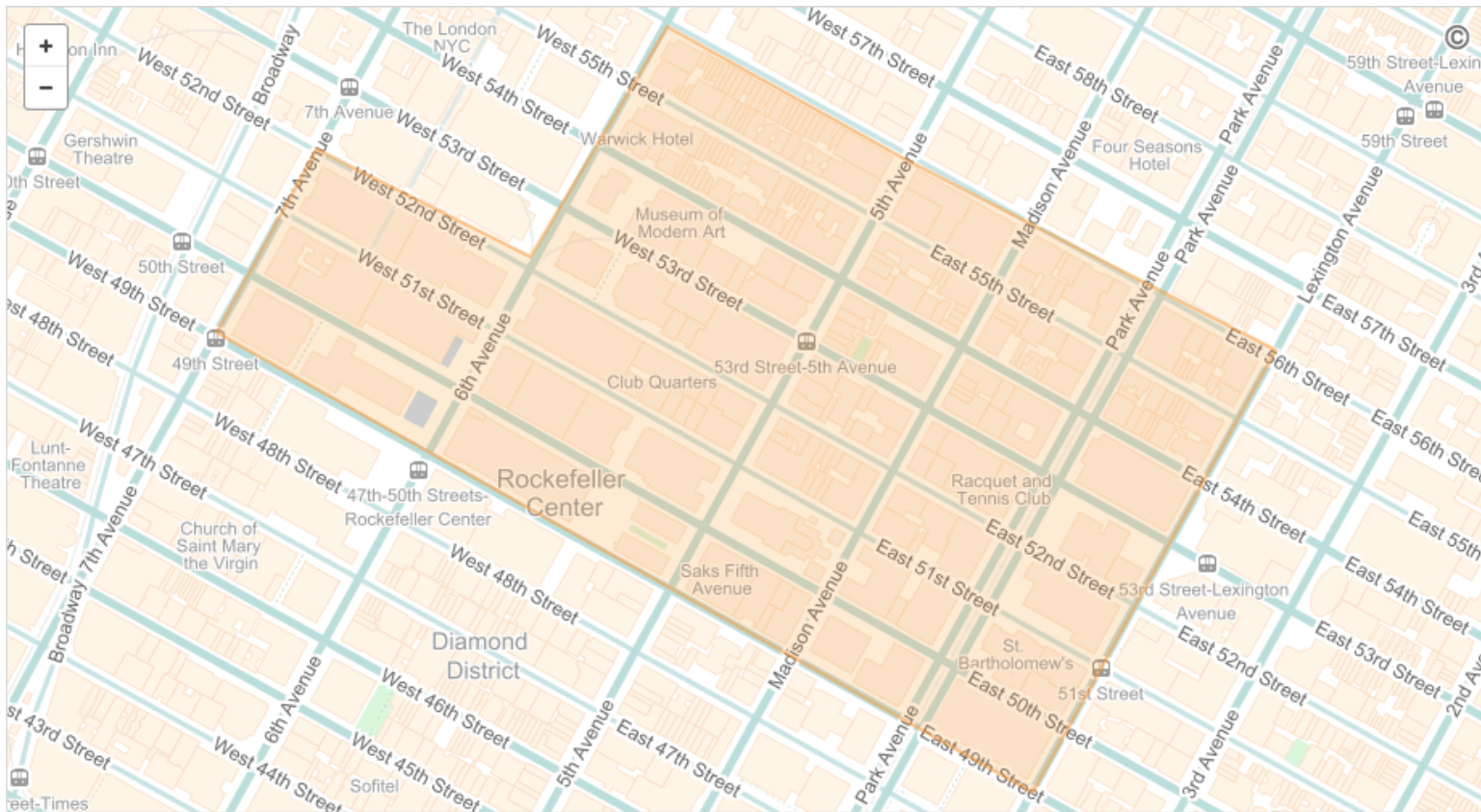
But how does MongoDB pick the inside of the Polygon?



MongoDB 2.6 behavior



Small Areas Polygon



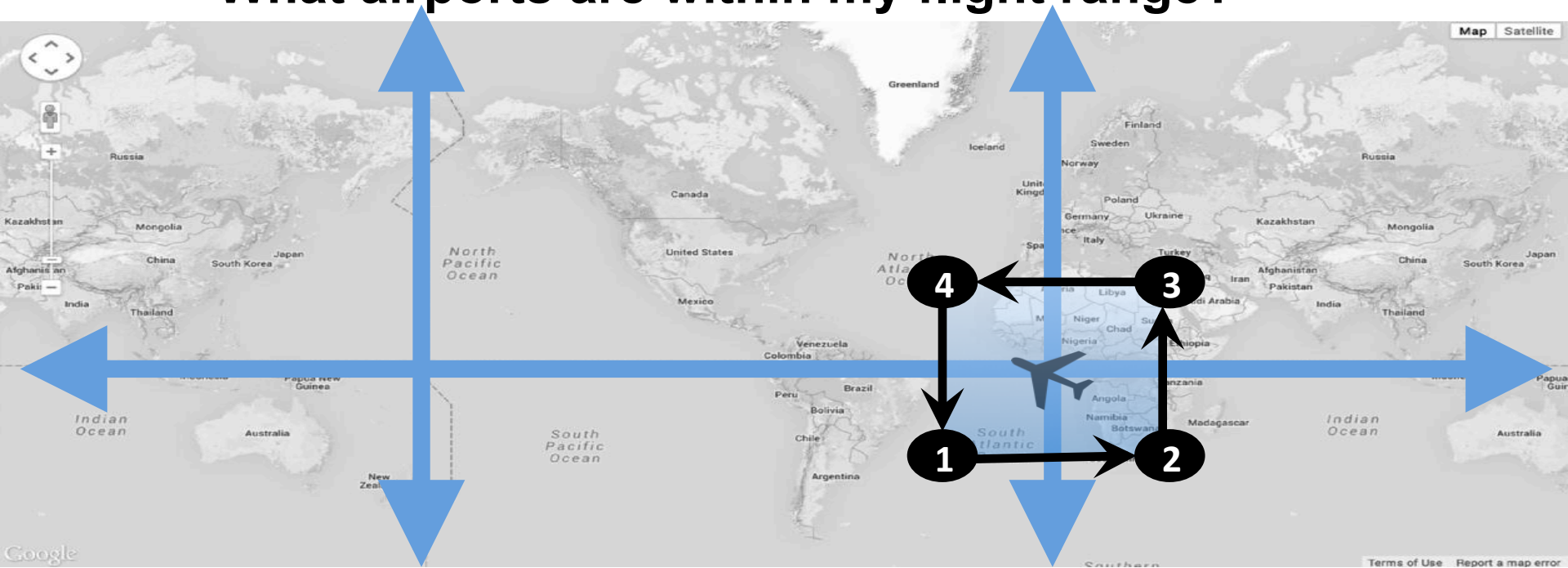
Polygon

```
var polygon = {  
  "type": "Polygon",  
  "coordinates": [  
    [  
      [-73.969581, 40.760331],  
      [-73.974487, 40.762245],  
      [-73.977692, 40.763598],  
      [-73.979508, 40.761269],  
      [-73.982364, 40.762358],  
      [-73.983692, 40.760497],  
      [-73.972821, 40.755861],  
      [-73.969581, 40.760331]  
    ],  
  ]  
}
```

Defines a business service area

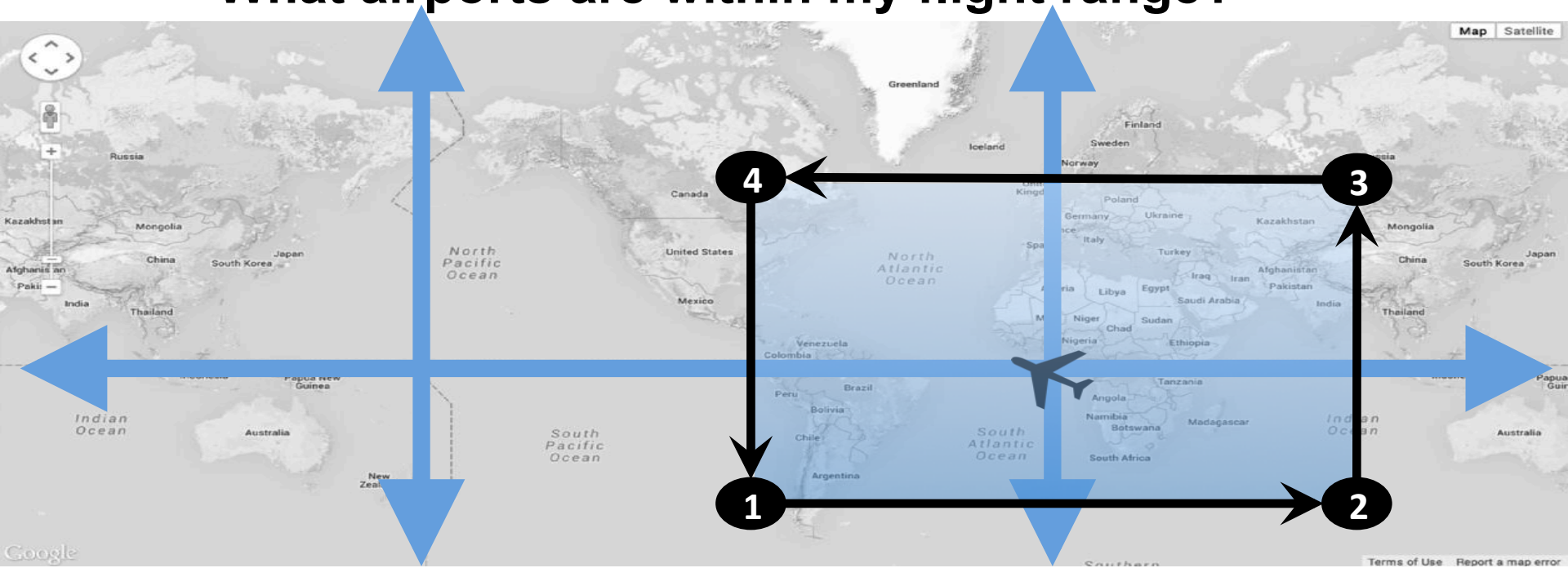
Big Polygon

**I am an airplane at [0, 0]
What airports are within my flight range?**



Start with a plane with a medium sized flight range polygon

I am an airplane at [0, 0]
What airports are within my flight range?



If it's a longer range plane, that polygon gets *bigger*

I am an airplane at [0, 0]
What airports are within my flight range?



Eventually polygon get so *big* it covers more than 50% of the planet

urn:x-mongodb:crs:strictwinding:EPSG:4326

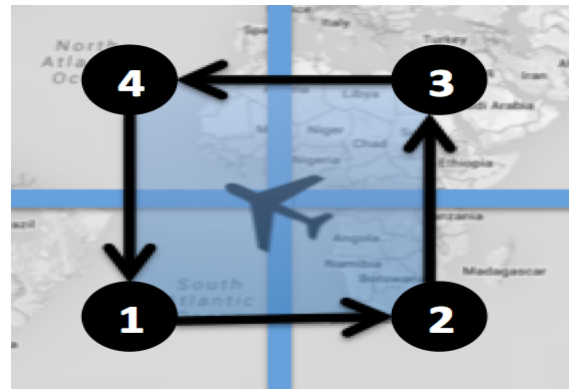
- urn
 - Uniform resource name
- x-mongodb
 - MongoDB extension
- strictwinding
 - Enforces explicit “counter-clockwise” winding
 - a.k.a.
 - anticlockwise,
 - right hand rule
 - the correct way
- EPSG:4326
 - Another name for WGS84
(the standard web geo coordinate system)

How do I do that?

```
// build some geo JSON
var crs = "urn:x-mongodb:crs:strictwinding:EPSG:4326"
var bigCRS = { type : "name", properties : { name : crs } };
var bigPoly = {   type : "Polygon",
                  coordinates : [
                    [[-10.0, -10.0],
                    [10.0, -10.0],
                    [10.0, 10.0],
                    [-10.0, 10.0],
                    [-10.0, -10.0]],
                  crs : bigCRS
                };
```

```
var cursor = db.<collection>.find({
  loc : { $geoWithin : { $geometry : bigPoly } }
});
```

```
var cursor = db.<collection>.find({
  loc : { $geoIntersects : { $geometry : bigPoly } }
});
```

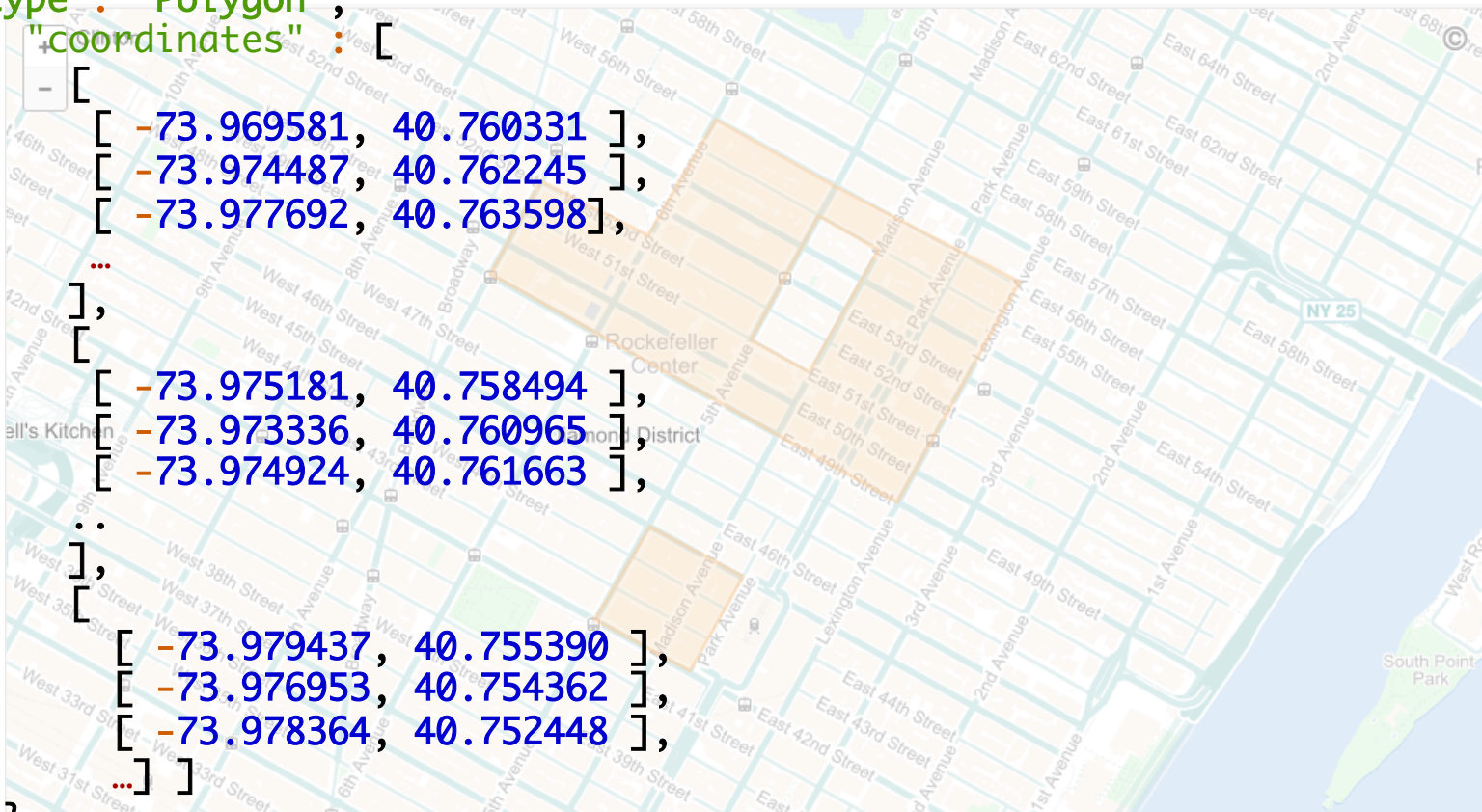


Complex Polygons



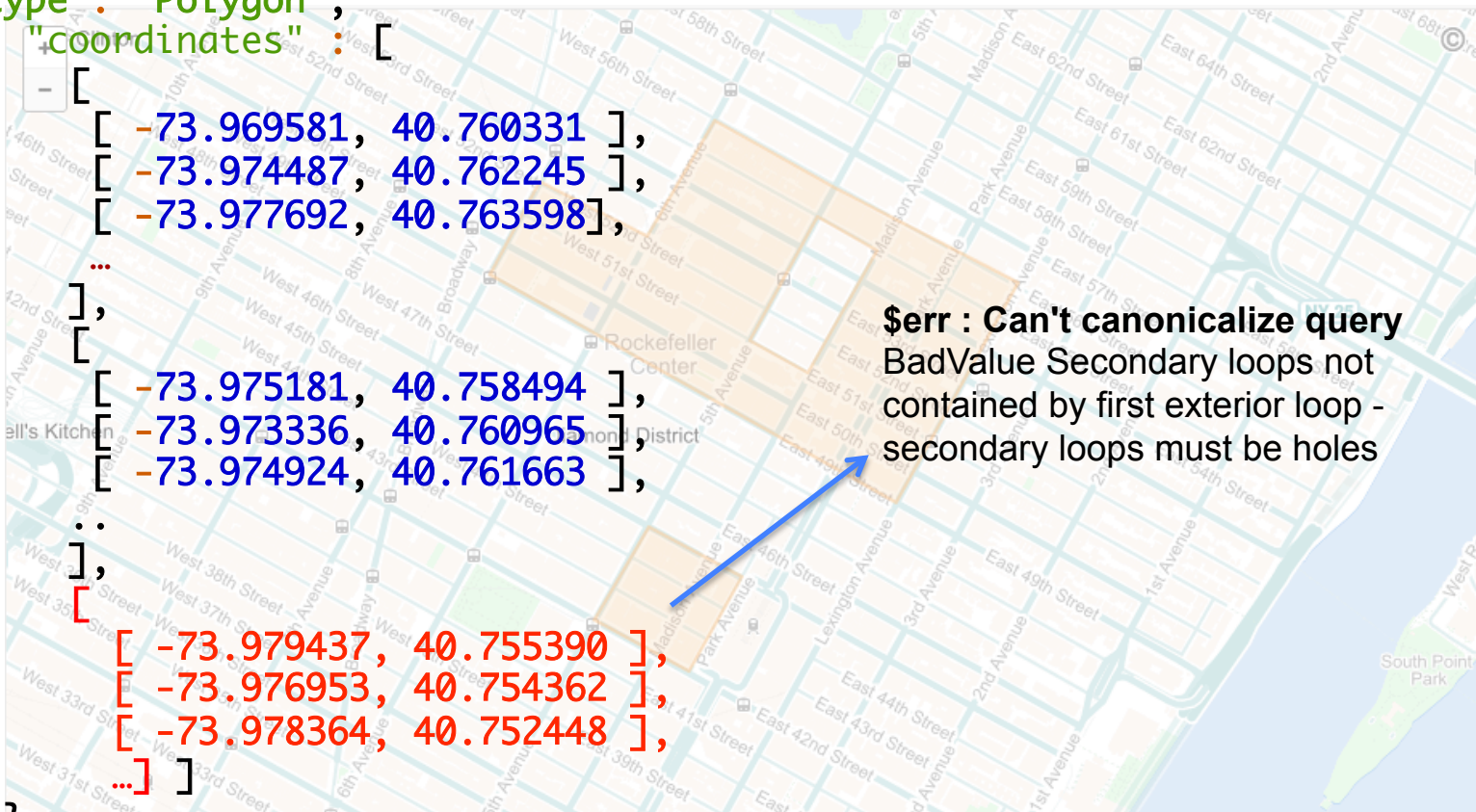
Complex Polygons

```
{ "type": "Polygon",  
  "coordinates": [  
    [  
      [-73.969581, 40.760331 ],  
      [-73.974487, 40.762245 ],  
      [-73.977692, 40.763598 ],  
      ...  
    ],  
    [  
      [-73.975181, 40.758494 ],  
      [-73.973336, 40.760965 ],  
      [-73.974924, 40.761663 ],  
      ...  
    ],  
    [  
      [-73.979437, 40.755390 ],  
      [-73.976953, 40.754362 ],  
      [-73.978364, 40.752448 ],  
      ...  
    ]  
  ]  
}
```



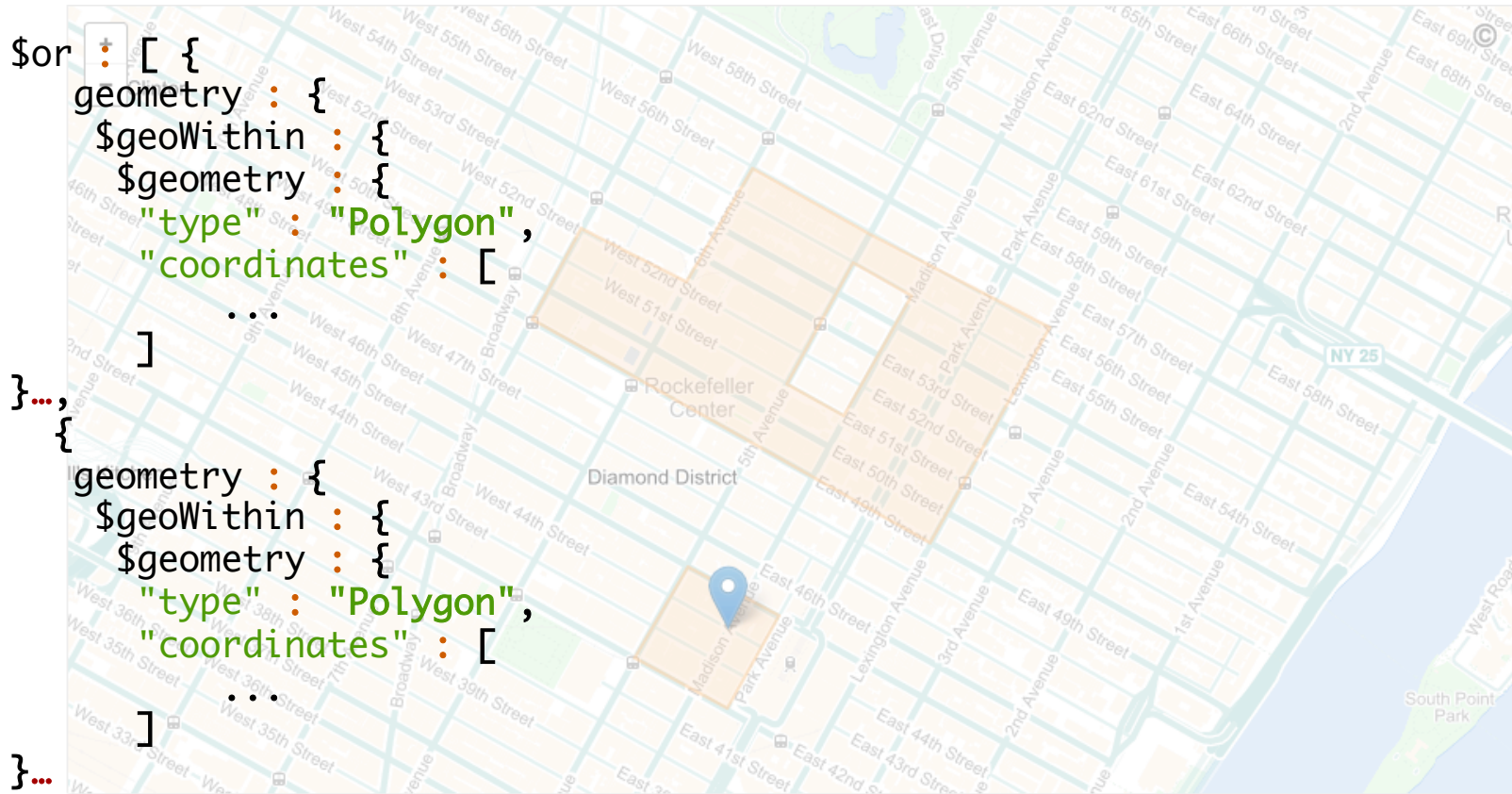
Complex Polygons

```
{ "type": "Polygon",  
  "coordinates": [  
    [  
      [-73.969581, 40.760331 ],  
      [-73.974487, 40.762245 ],  
      [-73.977692, 40.763598 ],  
      ...  
    ],  
    [  
      [-73.975181, 40.758494 ],  
      [-73.973336, 40.760965 ],  
      [-73.974924, 40.761663 ],  
      ...  
    ],  
    [  
      [-73.979437, 40.755390 ],  
      [-73.976953, 40.754362 ],  
      [-73.978364, 40.752448 ],  
      ...  
    ]  
  ]  
}
```



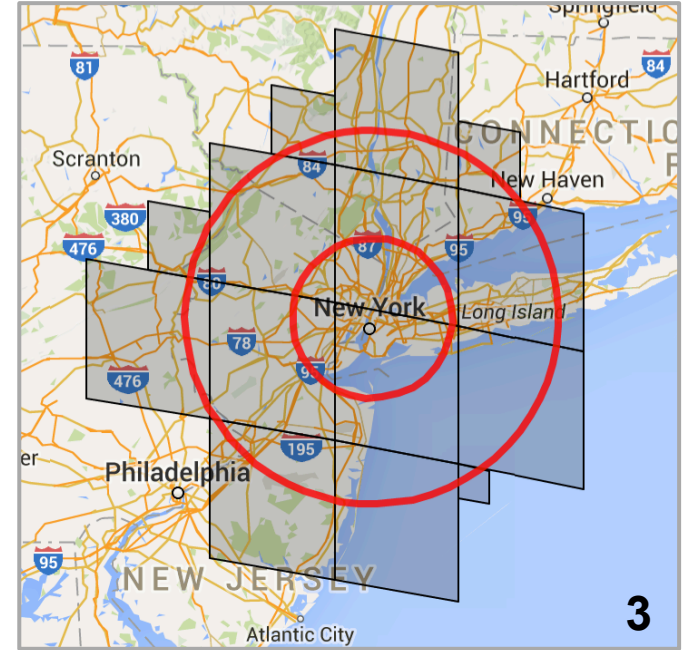
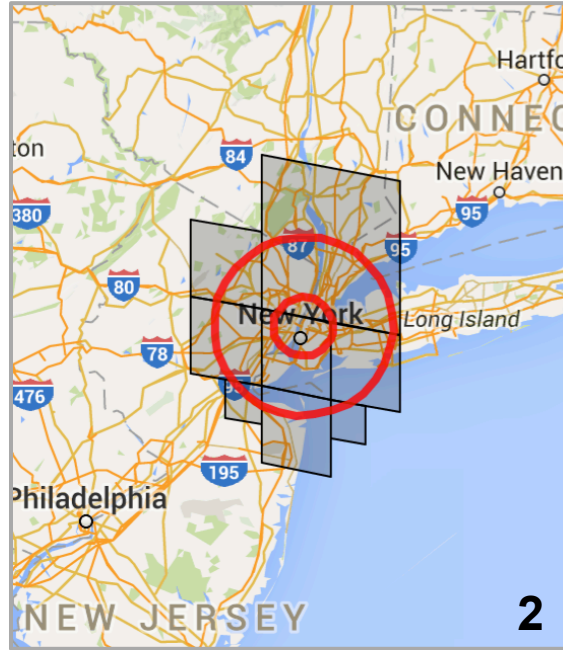
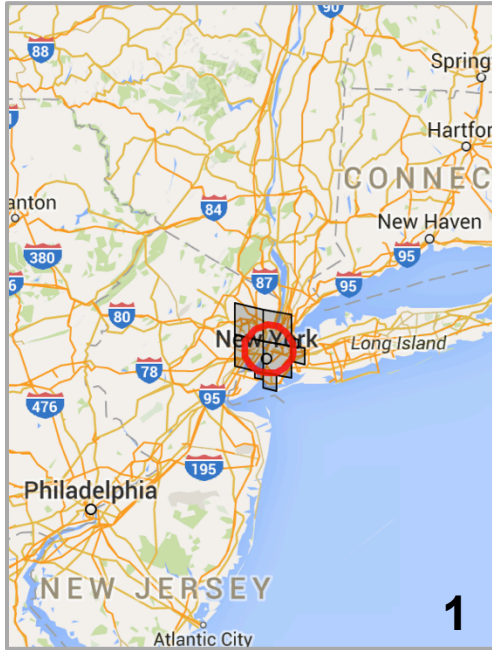
\$err : Can't canonicalize query
BadValue Secondary loops not
contained by first exterior loop -
secondary loops must be holes

Complex Polygons



Optimizations

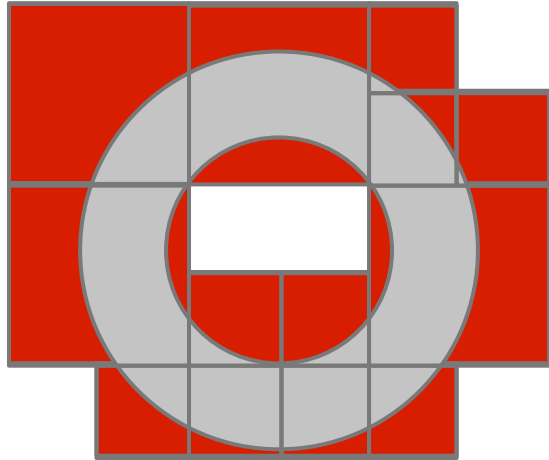
\$geoNear Algorithm



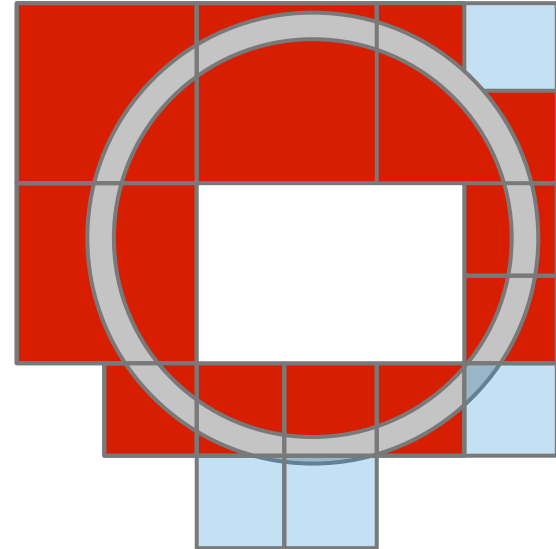
Series of \$geoWithin + sort

Problem 1: Repeated Scans

Stage 2

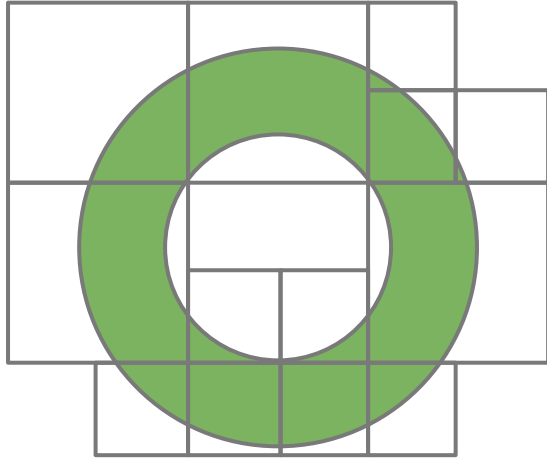


Stage 3

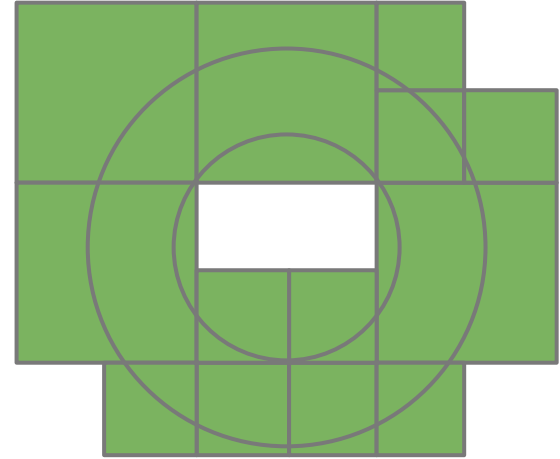


Buffer every document in covering

Original Algorithm

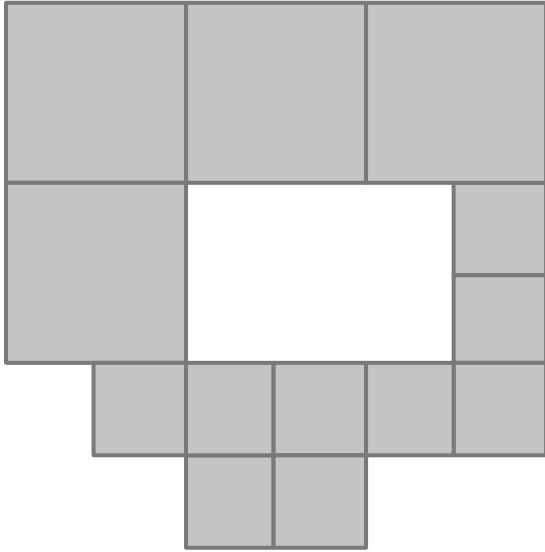


New Algorithm

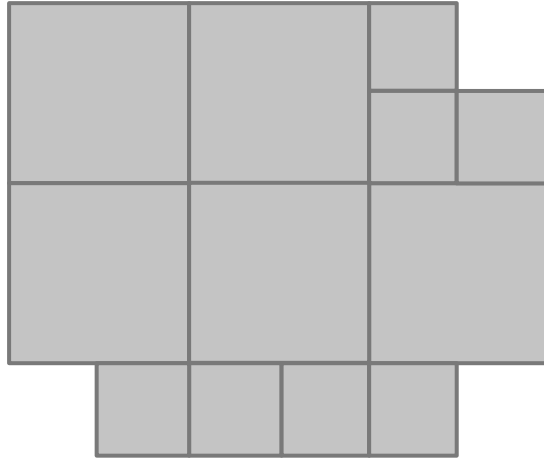


Avoid repeated index scans

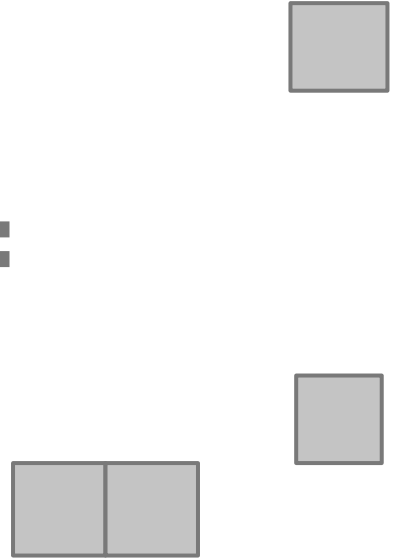
Covering



Visited Cells

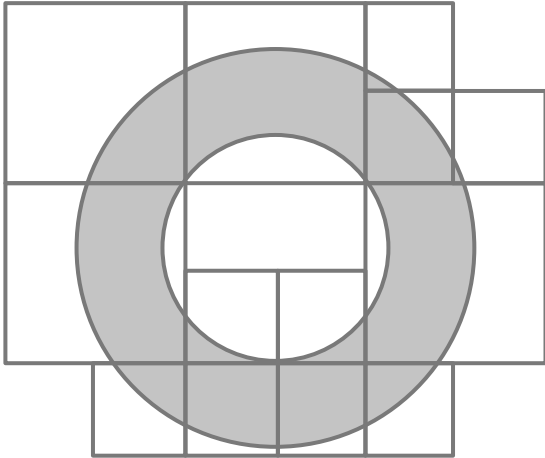


Difference

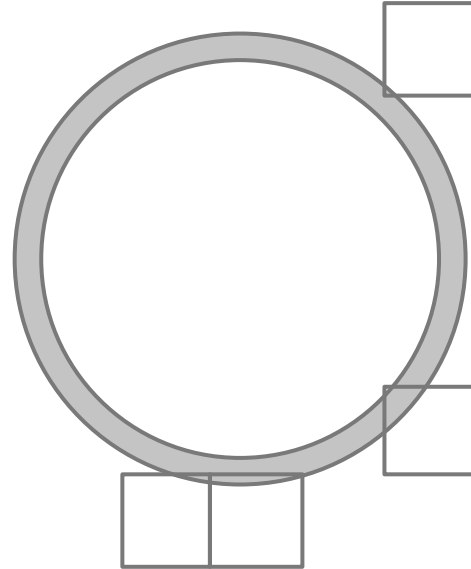


Avoid repeated index scans

Stage 2



Stage 3



Problem 1.1: Unnecessary fetches

Index Scan

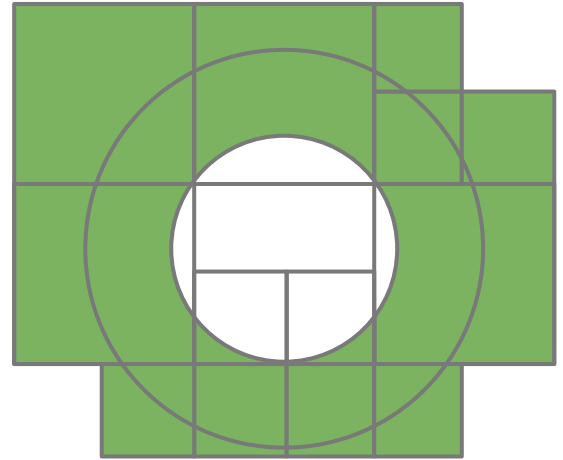


Filter Out Disjoint Keys



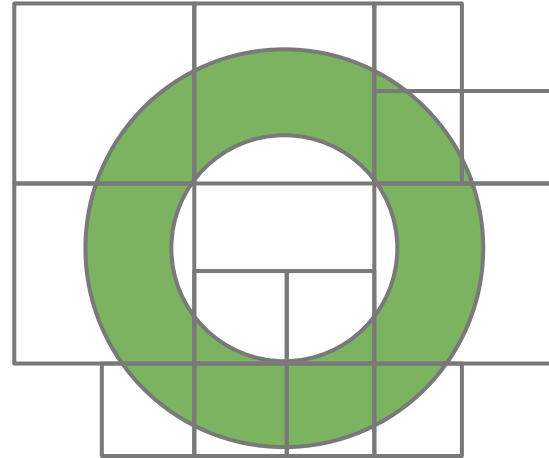
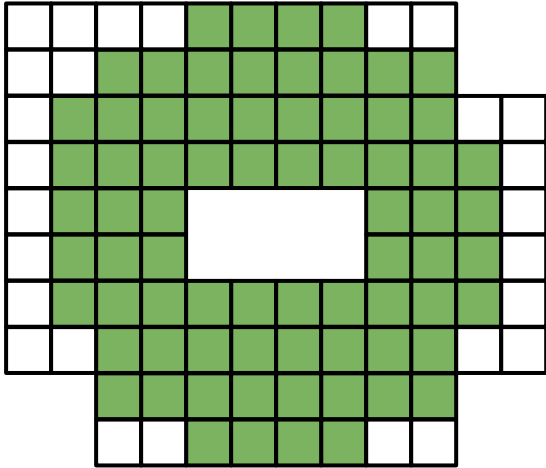
Fetch

Last Interval

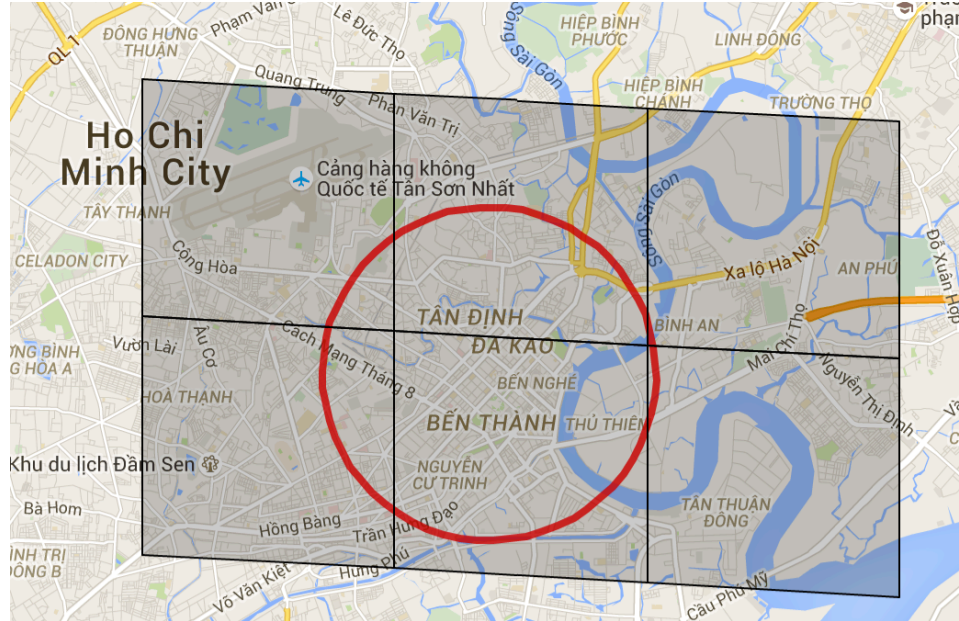


Original \$geoNear Algorithm

Filter out disjoint keys then filter out disjoint docs



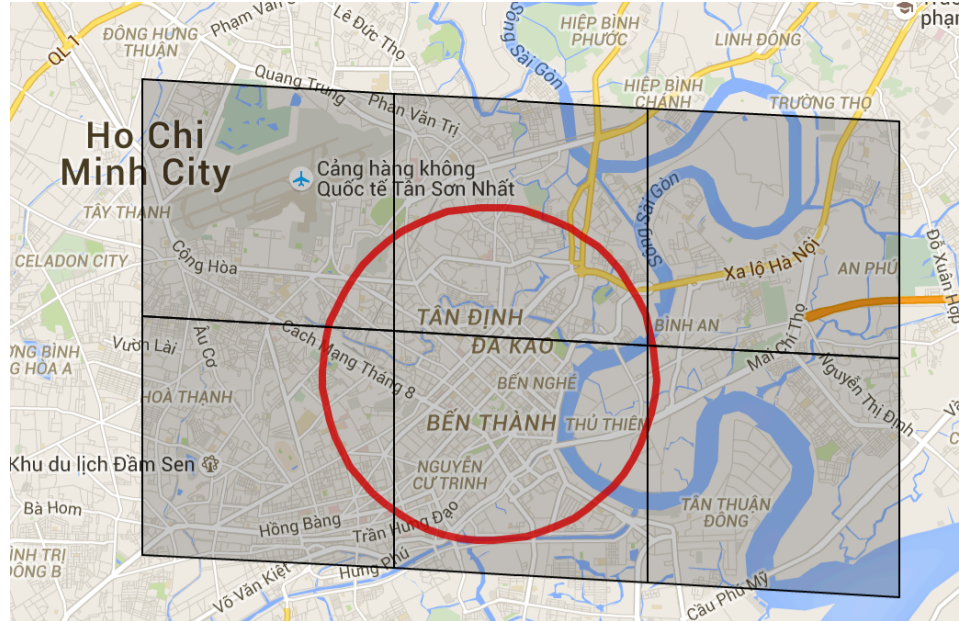
Problem 2: Large Initial Radius



Finding one document 1cm away

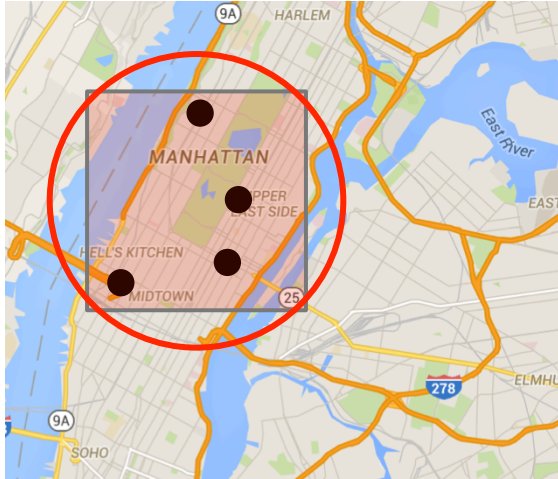
Initial radius is over 1km

Determining the Radius

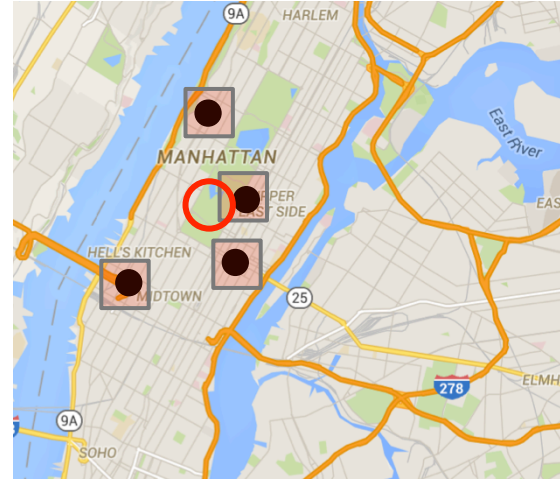


The minimum distance to find documents

Indexing Points to Finest Level



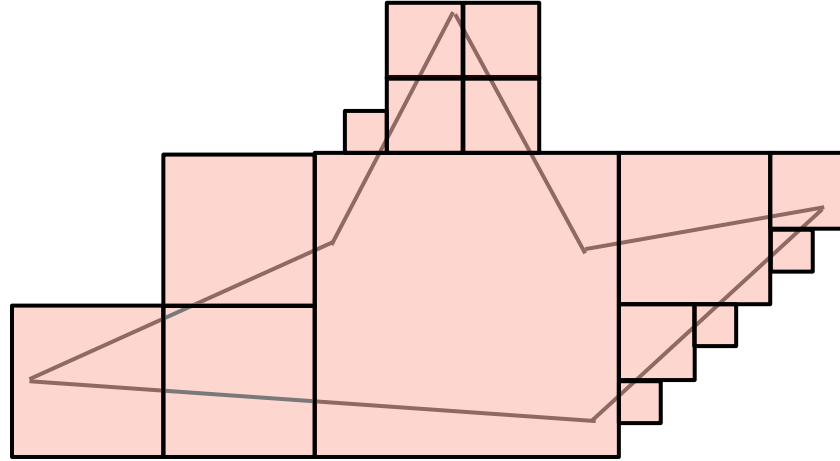
Original indexed level



New indexed level

Bounded by cell size

Why were Points Indexed Coarsely?



Polygons have a tradeoff in storage size

Problem 2.1: New Index Version

- Finer index level means different index keys
- 2dSphere index version 3 introduced

Problem 2.2: Larger Index Size

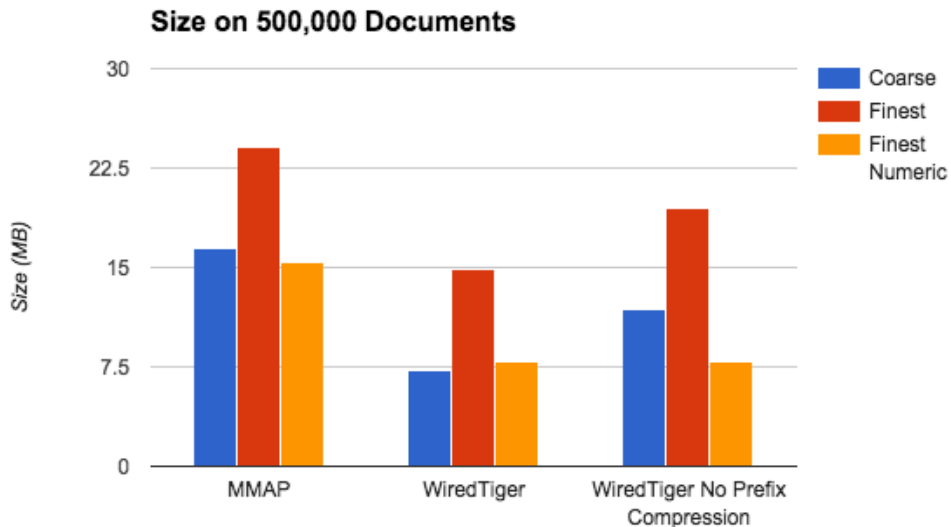
String (v2)

1F12031



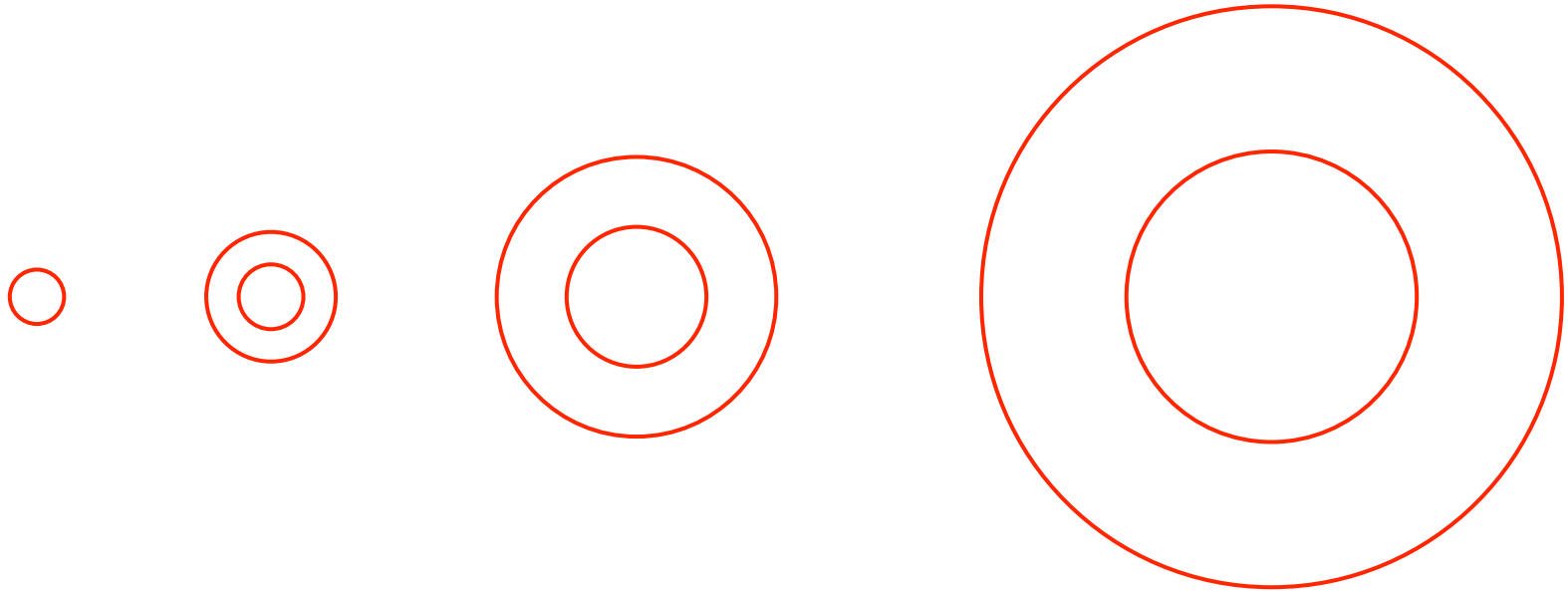
NumberLong (v3)

00101100011011

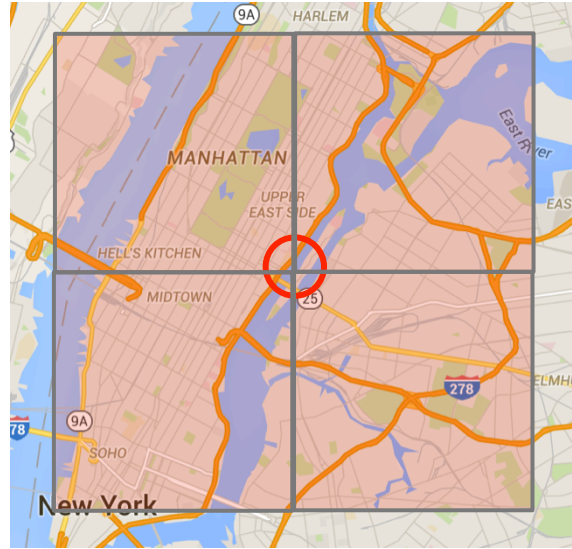


Problem 2.3: More intervals

Because we no longer repeat index scans,
there is little to no performance hit

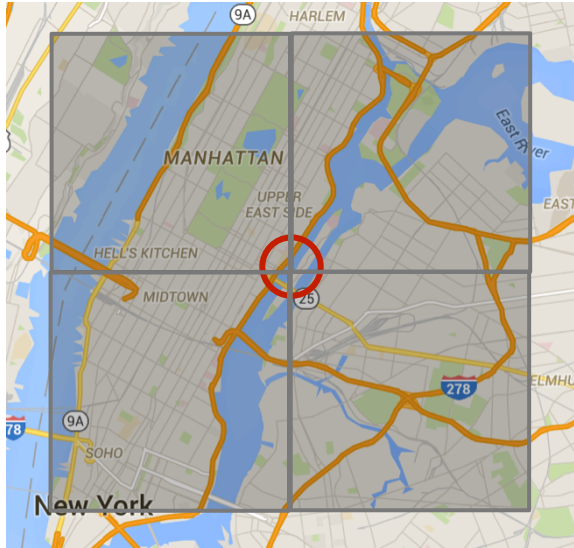


Problem 3: Query Level still Coarse



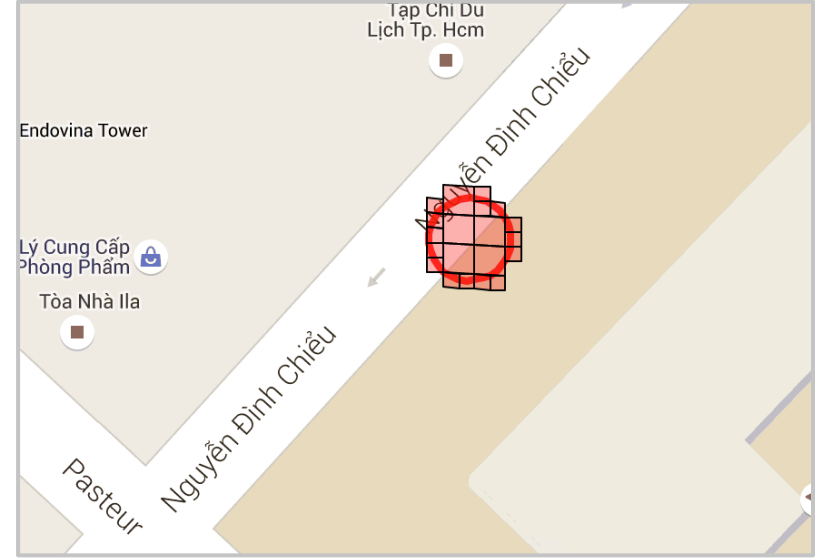
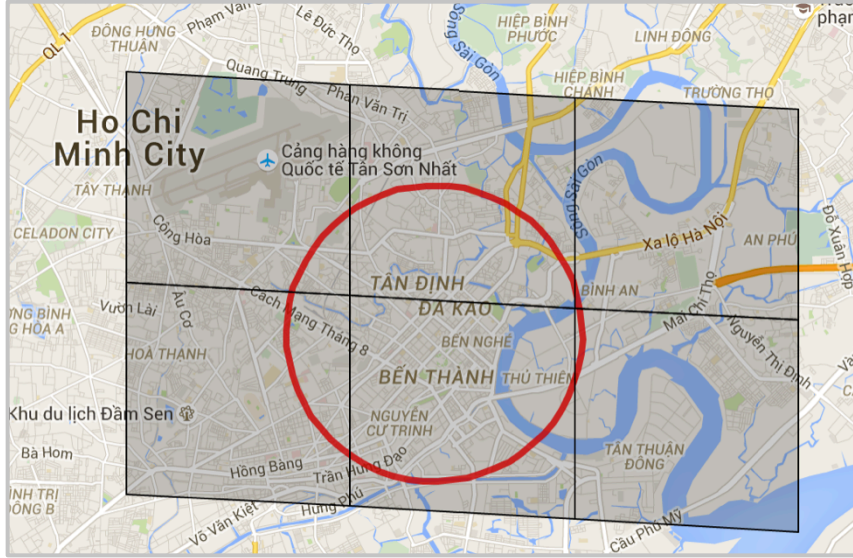
Covering of radius is constrained by index covering levels

Split Index and Query Constraints



Set query maximum to finest level

Before and After



Finding one document 1cm away

Results

	MongoDB 3.0 with 2dsphere Index Version 2	MongoDB 3.1.6 with Index Version
Time (s)	229,550	1571 (0.7% of original)
Keys	183,994,438	429,739 (0.25% of original)
Documents	19,108,349	422,046 (2.2% of original)

Results

Test Case	Ops per second before change	Ops per second after change	Speedup
\$geoWithin queries	1,733	20,198	11.7 x
\$geoNear queries	533	6,783	12.7 x
\$geoNear queries limiting to one document	111	17,211	155 x

More info on Optimization

<https://www.mongodb.com/blog/post/geospatial-performance-improvements-in-mongodb-3-2>



Obrigado!

Norberto Leite
Engineer
norberto@mongodb.com
@nleite

<http://cl.jroo.me/z3/v/D/C/e/a.baa-Too-many-bicycles-on-the-van.jpg>