# Spatial Data Systems

By: Mohamed Sarwat

 @MoSarwat

# Spatial Data Applications

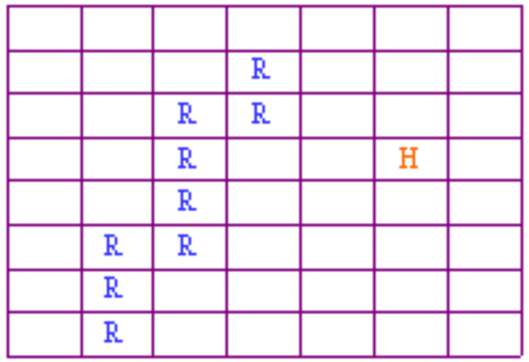
- GIS applications (maps):
  - Urban planning, route optimization, fire or pollution monitoring, utility networks, etc
- Other applications:
  - VLSI design, CAD/CAM, model of human brain, etc
- Traditional applications:
  - Multidimensional records

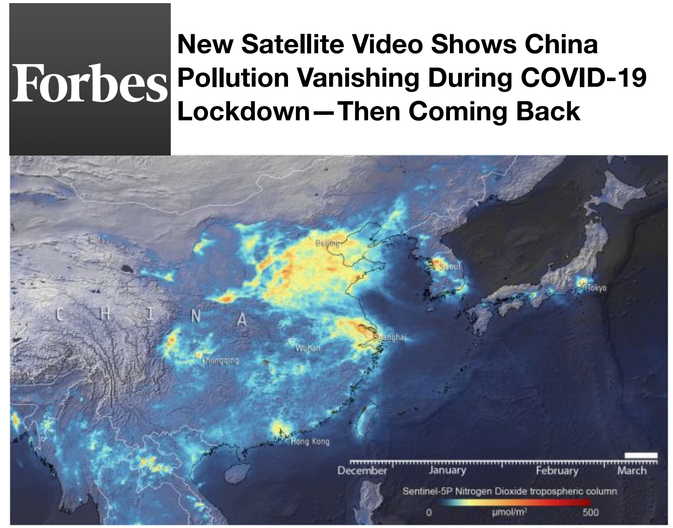
# What Is a Spatial Database?

- A SDBMS is a DBMS
- It offers spatial data types/data models/ query language
  - Support spatial properties/operations
- It supports spatial data types in its implementation
  - Support spatial indexing, algorithms for spatial selection and join

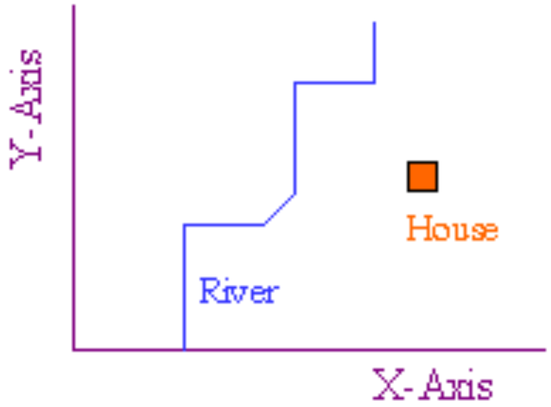
# Spatial Representation

- Raster model:




- Vector model:

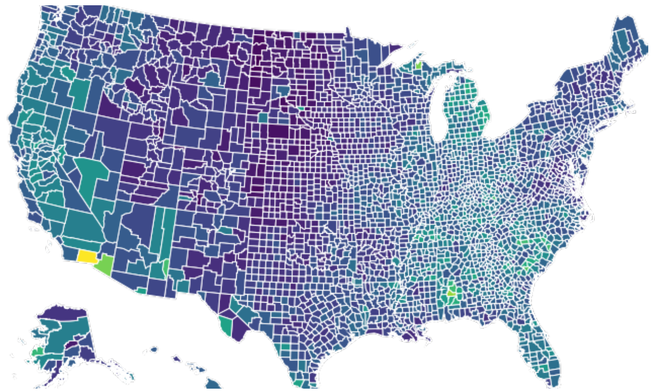



# Spatial Data Types


- Point : 2 real numbers
- Line : sequence of points
- Region : area included inside n-points

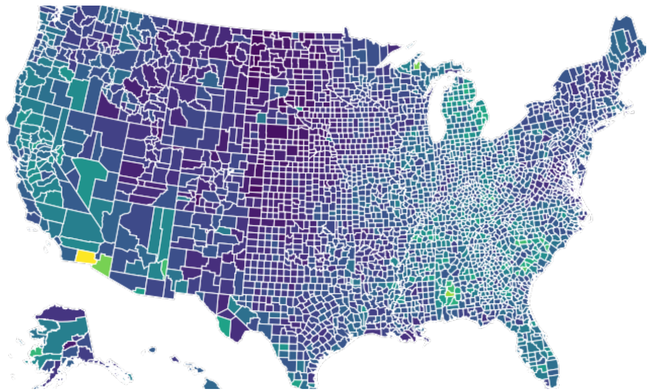
# Why Spatial is Special (1)

- Properties:
  - More than one dimension
  - Geometrical properties
  - Graph Properties



# Why Spatial is Special (1)

- Properties:
  - More than one dimension
  - Geometrical properties
  - Graph Properties



- **Both Data Intensive and Compute Intensive**
- **Sorting is difficult**

# Why Spatial is Special (2)

- Need query language extension
- Need new indexing and storage layouts
- New query optimization / processing operators
- New machine learning algorithms
- New data Visualization Techniques

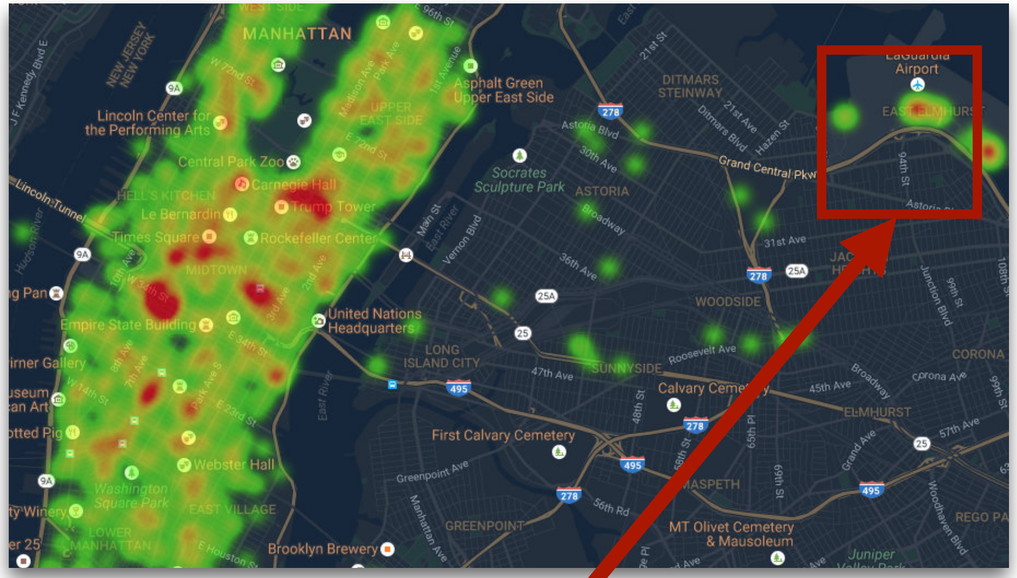
# Spatial Relationships

- Topological relationships:
  - adjacent, inside, disjoint, etc
- Direction relationships:
  - Above, below, north_of, etc
- Metric relationships:
  - “distance < 100”
- And operations to express the relationships

# Examples

- A database:
  - Relation states(sname: string, area: region, spop: int)
  - Relation cities(cname: string, center: point; ext: region)
  - Relation rivers(rname: string, route:line)
- SELECT * FROM rivers WHERE route intersects R
- SELECT cname, sname FROM cities, states WHERE center inside area
- SELECT rname, length(intersection(route, California)) FROM rivers WHERE route intersects California

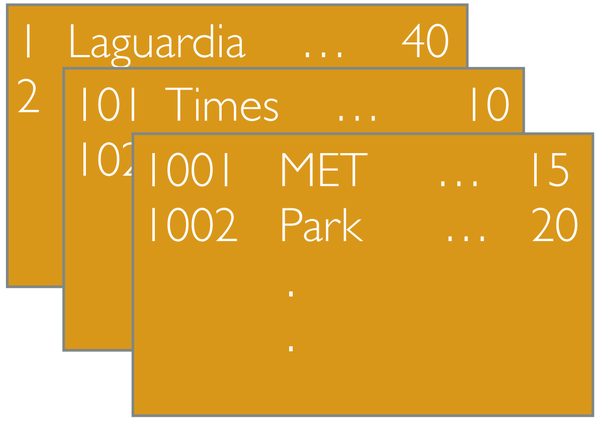
# Spatial Queries




| Trip Time Stamp | Pick up location | …. | Fare amount |
|---|---|---|---|
| 1 | Laguardia | … | $40 |
| 2 | JFK | … | $30 |
| 3 | <lat,long> | … | $… |
| … | … | … | … |

# RANGE QUERY

Return all the NYC taxi trips for which the pick up location is within the Laguardia Airport area



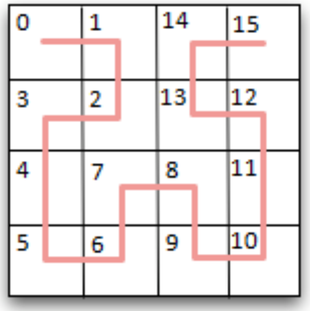

Data Pages

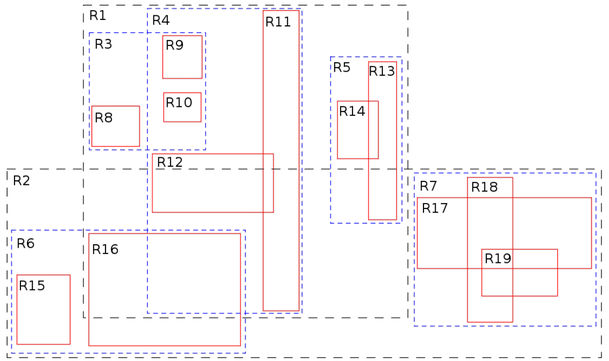
# Spatial Queries

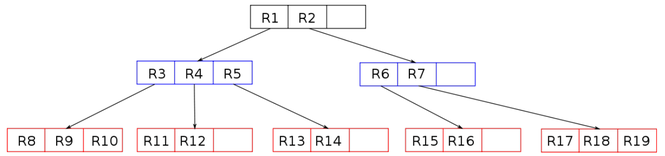
- Selection queries: “Find all objects inside query q”, inside-> intersects, north
- Nearest Neighbor-queries: “Find the closets object to a query point q”, k-closest objects
- Spatial join queries: Two spatial relations S1 and S2, find all pairs: {x in S1, y in S2, and x rel y= true}, rel= intersect, inside, etc

# Access Methods

- Point Access Methods (PAMs):
  - Index methods for 2 or 3-dimensional points (k-d trees, Z-ordering, grid-file)
- Spatial Access Methods (SAMs):
  - Index methods for 2 or 3-dimensional regions and points (R-trees)

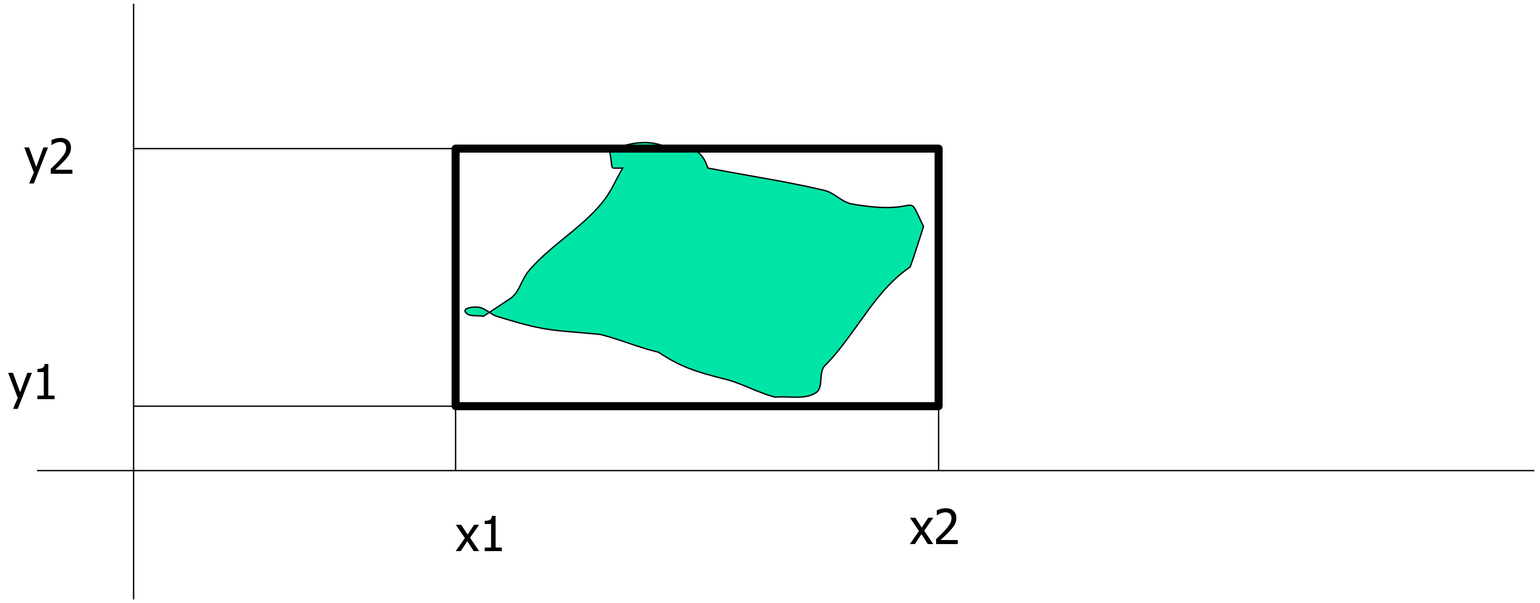


# Indexing Using SAMs

- Approximate each region with a simple shape: usually Minimum Bounding Rectangle (MBR) = [(x1, x2), (y1, y2)]



# Indexing Using SAMs (Cont.)

Two steps:

- Filtering step: Find all the MBRs (using the SAM) that satisfy the query
- Refinement step:For each qualified MBR, check the original object against the query

# Spatial Indexing

- Point Access Methods (PAMs) vs Spatial Access Methods (SAMs)
- PAM: index only point data
  - Hierarchical (tree-based) structures
  - Multidimensional Hashing
  - Space filling curve
- SAM: index both points and regions
  - Transformations
  - Overlapping regions
  - Clipping methods

## Data Retrieval/Processing/Preparation




## Spatial Data Analysis Tools






# Apps Generate tons of Geospatial Data

Available geospatial data is huge, growing at a staggering rate, and comes from many heterogeneous sources

# How does a data scientist drive business insights from geospatial data?

### Fraud and Abuse

Detect patterns of fraud and collusion (e.g. claims fraud, credit card fraud)

### Retail

Site selection, urban planning, foot traffic analysis

### Financial Services


Economic distribution, loan risk analysis, predicting sales at retail investments

### Healthcare


Identifying disease epicenters, environmental impact on health, planning care

### Disaster Recovery

Flood surveys, earthquake mapping, response planning

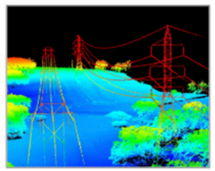
### Defense and Intel

Reconnaissance, threat detection, damage assessment

### Infrastructure

Transportation planning, agriculture management, housing development

### Energy



Climate change analysis, energy asset inspection, oil discovery

## Data Processing/Preparation

Preparation queries

Prepared data

PostgreSQL

PostGIS

mongoDB.

MySQL

Raw Data

- ESRI ShapeFiles
- IoT Sensor Data
- GeoJSON Docs
- Geo-Tagged Social Media
- NASA HDF/ NetCDF
- Vegetation Indices
- GPS Raw Data
- Map / Road Network Data

**Data Scientist**

## Analyzing the Data

Issue the Analysis

Analysis Results

ArcGIS

QGIS

python™

tableau

MATLAB

**80% of the time spent here**

## Data Processing/Preparation

Preparation queries

Prepared data

APACHE Spark™

Raw Data

- ESRI ShapeFiles
- IoT Sensor Data
- GeoJSON Docs
- Geo-Tagged Social Media
- NASA HDF/ NetCDF
- Vegetation Indices
- GPS Raw Data
- Map / Road Network Data

Data Scientist

## Analyzing the Data

Issue the Analysis

Analysis Results

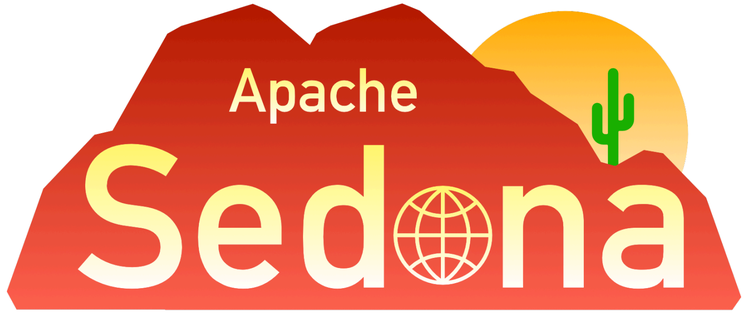
ArcGIS QGIS python™ tableau® MATLAB

**80% of the time spent here**

# Apache Spark back then...

- **Did not provide a geospatial data processing API**
  - Geospatial was treated as yet another attribute
  - The Geospatial data was only perceived as two extra columns (X,Y)
  - A programmer had to write 1000s of lines of code to implement simple geospatial operations like Spatial Join
- **Could not efficiently optimize geospatial queries**
  - No spatial proximity-aware load balancing
  - No spatial indexes
  - No spatial data compression techniques

# Apache Sedona (GeoSpark)



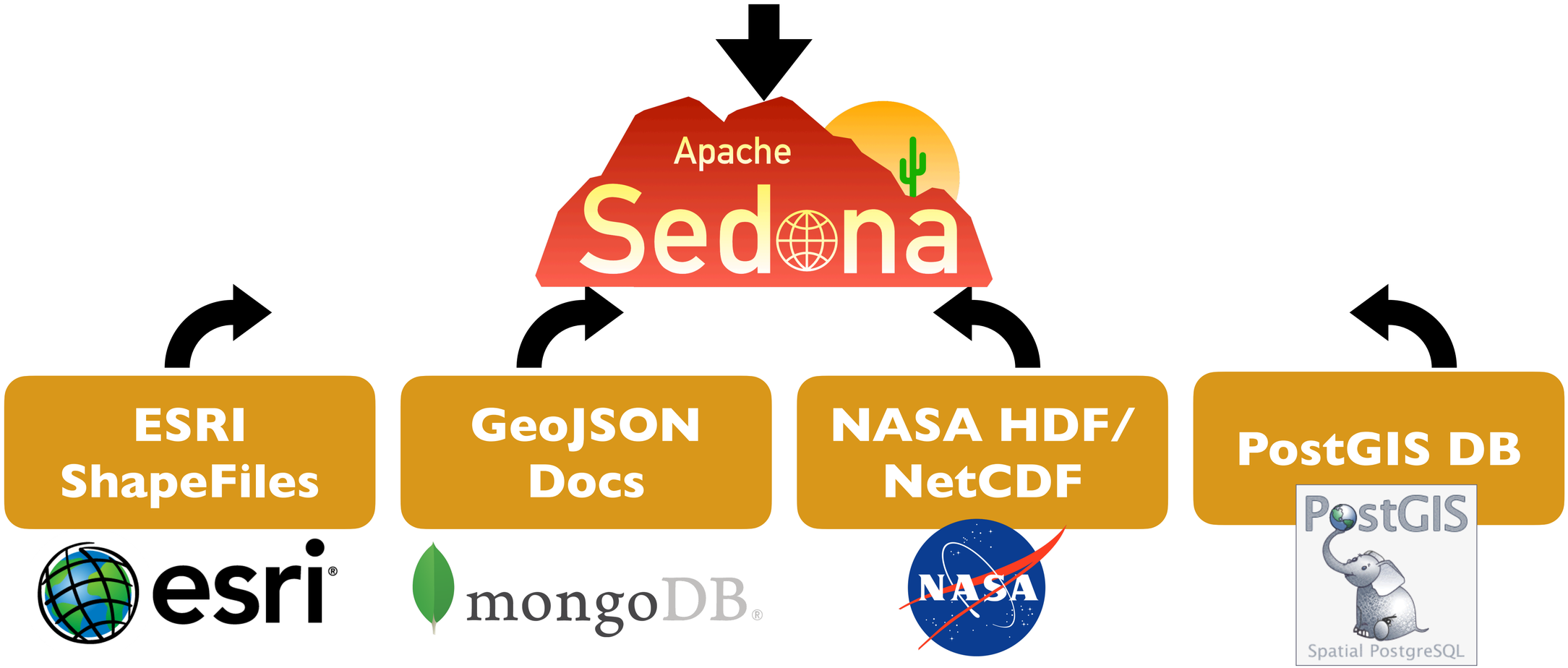

Treats location, distance, and spatial interaction as core aspects of the data processing / preparation system

*Jia Yu, Zongsi Zhang, ,M. Sarwat, "Spatial data management in apache spark: the GeoSpark perspective and beyond."* ***GeoInformatica 2019***

*Jia Yu, Jinxuan Wu, ,M. Sarwat, "GeoSpark: a cluster computing framework for processing large-scale spatial data"* ***ACM SIGSPATIAL GIS 2015***

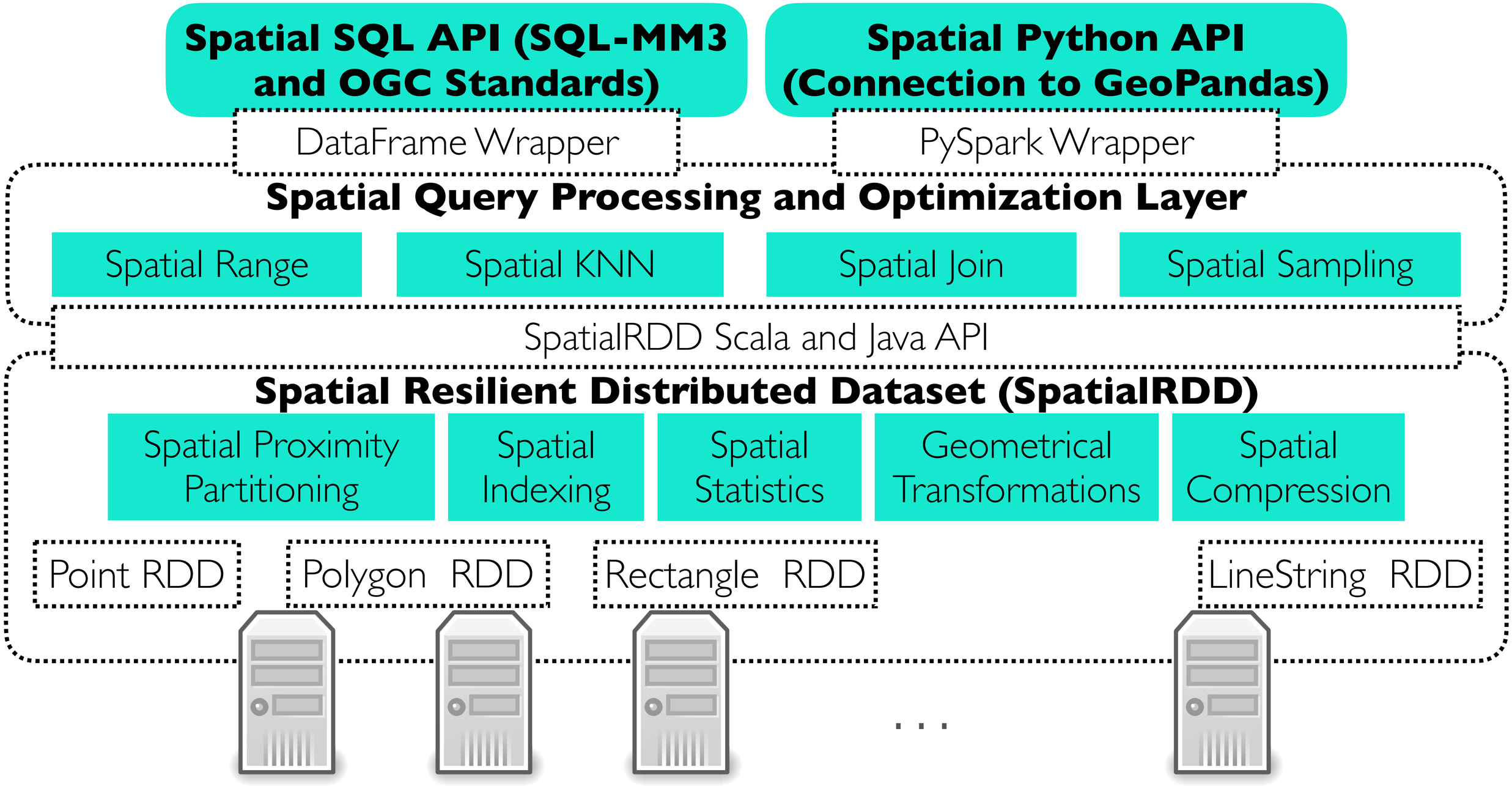
*Jia Yu, Jinxuan Wu, Mohamed Sarwat: A demonstration of GeoSpark: A cluster computing framework for processing big spatial data.* ***IEEE ICDE 2016***

# User write geospatial data processing tasks




- Website: http://sedona.apache.org
- Github link: https://github.com/apache/incubator-sedona
- Twitter: @ApacheSedona

# Overview

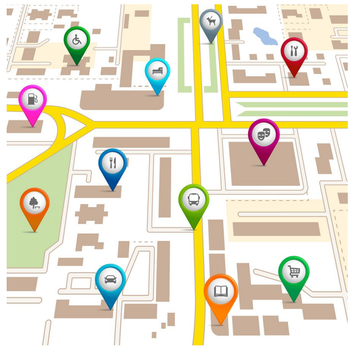

Spatial SQL API (SQL-MM3 and OGC Standards)
Spatial Python API (Connection to GeoPandas)
DataFrame Wrapper
PySpark Wrapper
Spatial Query Processing and Optimization Layer
Spatial Range
Spatial KNN
Spatial Join
Spatial Sampling
SpatialRDD Scala and Java API
Spatial Resilient Distributed Dataset (SpatialRDD)
Spatial Proximity Partitioning
Spatial Indexing
Spatial Statistics
Geometrical Transformations
Spatial Compression
Point RDD
Polygon RDD
Rectangle RDD
LineString RDD
. . .

# Spatial Resilient Distributed Dataset (SpatialRDD)

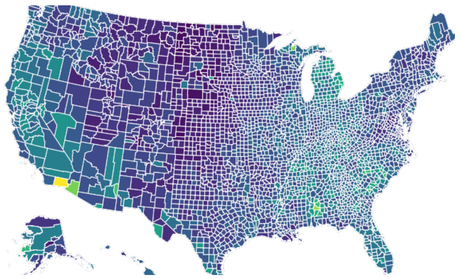
# Representing Geospatial Data Using SpatialRDDs

```
val pointRDDInputLocation = "/Download/checkin.csv"
val pointRDDOffset = 0 // The point long/lat starts from Column 0
val pointRDDSplitter = FileDataSplitter.CSV
val carryOtherAttributes = true // Carry Column 2 (hotel, gas, bar...)
var objectRDD = new PointRDD(sc, pointRDDInputLocation, pointRDDOffset, pointRDDSplitter, carryOtherAttributes)
```

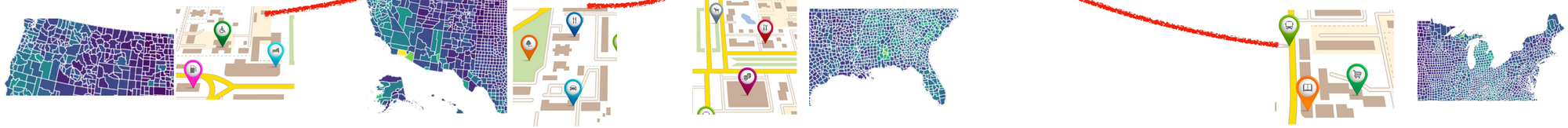
PointRDD



PolygonRDD



```
val polygonRDDInputLocation = "/Download/checkinshape.csv"
val polygonRDDStartOffset = 0 // The coordinates start from Column 0
val polygonRDDEndOffset = 9 // The coordinates end at Column 9
val polygonRDDSplitter = FileDataSplitter.CSV
val carryOtherAttributes = true // Carry Column 10 (hotel, gas, bar...)
var objectRDD = new PolygonRDD(sc, polygonRDDInputLocation, polygonRDDStartOffset, polygonRDDEndOffset, polygonRDDSplitter, carryOtherAttributes)
```

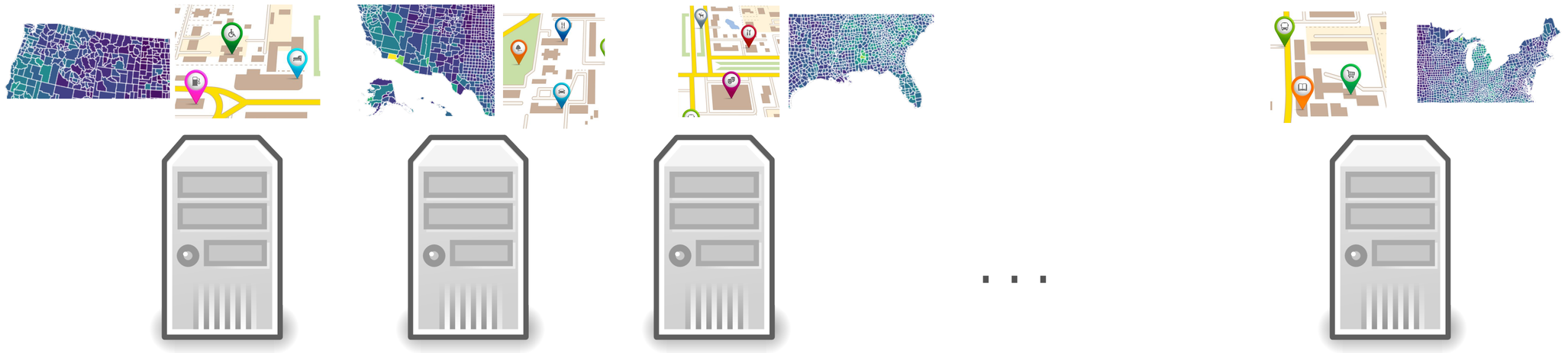


. . .

# SpatialRDD: Local Memory Footprint

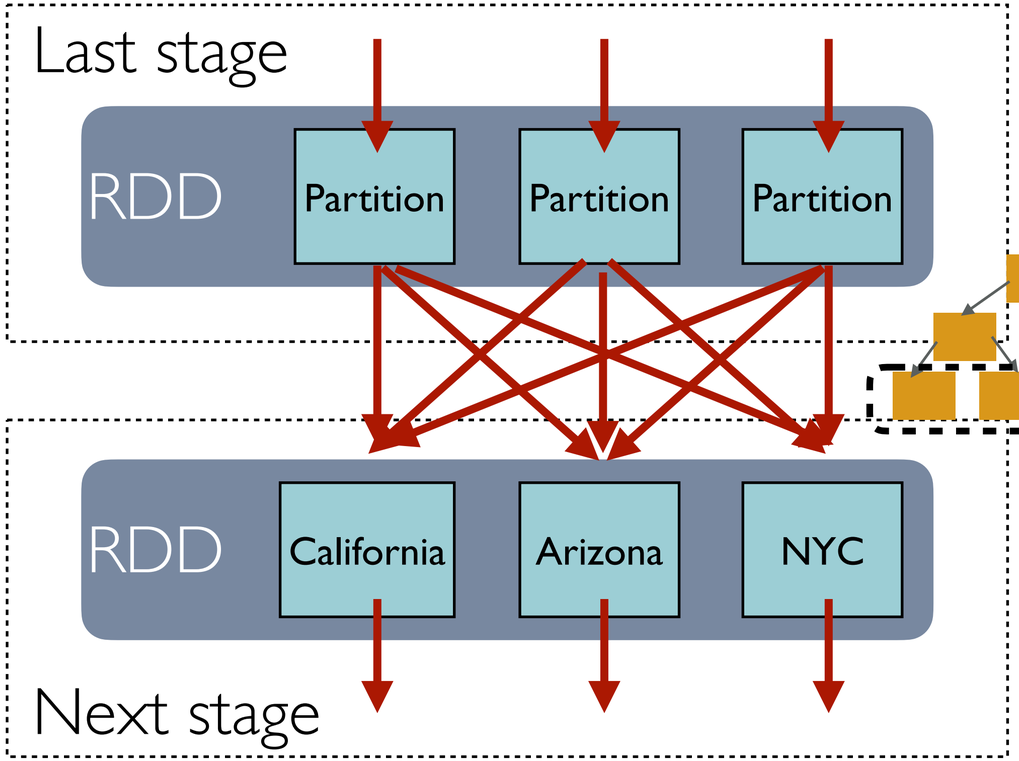
- Compact In-Memory Representation
- Support Heterogeneous Spatial Data types
- Geometrical Computation Library at each node

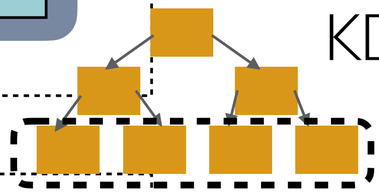
| 1 | 2 | 3 | 4 | 5-12 | 13-20 | 21-28 | 29-36 | ... | ... | ... | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Object type | Sub-obj num | Sub-obj1 type | Coord num | longitude | latitude | longitude | latitude | ... | Sub-obj2 type | Coord num | ... |



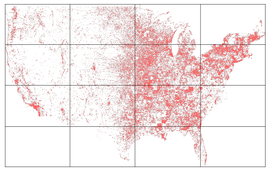
# SpatialRDD: Partitioning

- Sample the RDD
- Build a KD-Tree/ Quad-Tree/R-Tree on the sample
- Take the leaf nodes of the tree as the global partition file
- Re-partition the RDD according to the partition file

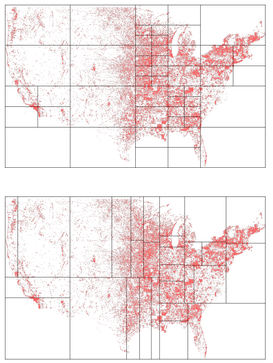


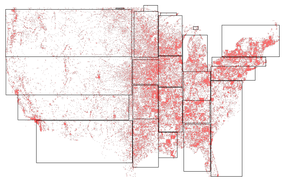
Uniform grids



KD-Tree



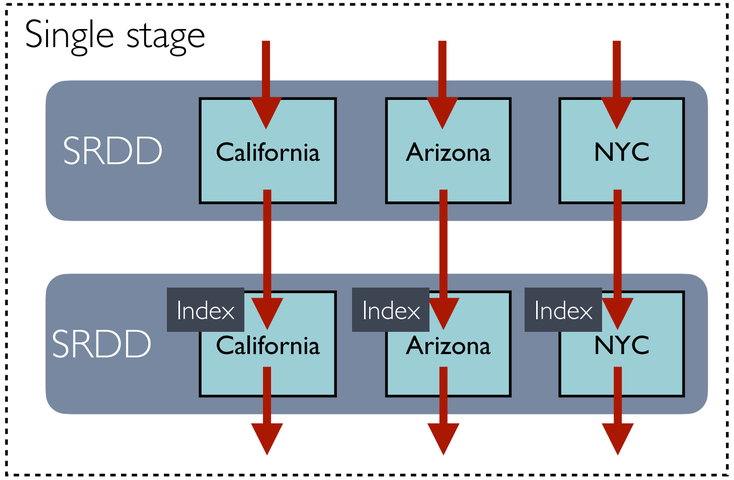
Quad-Tree

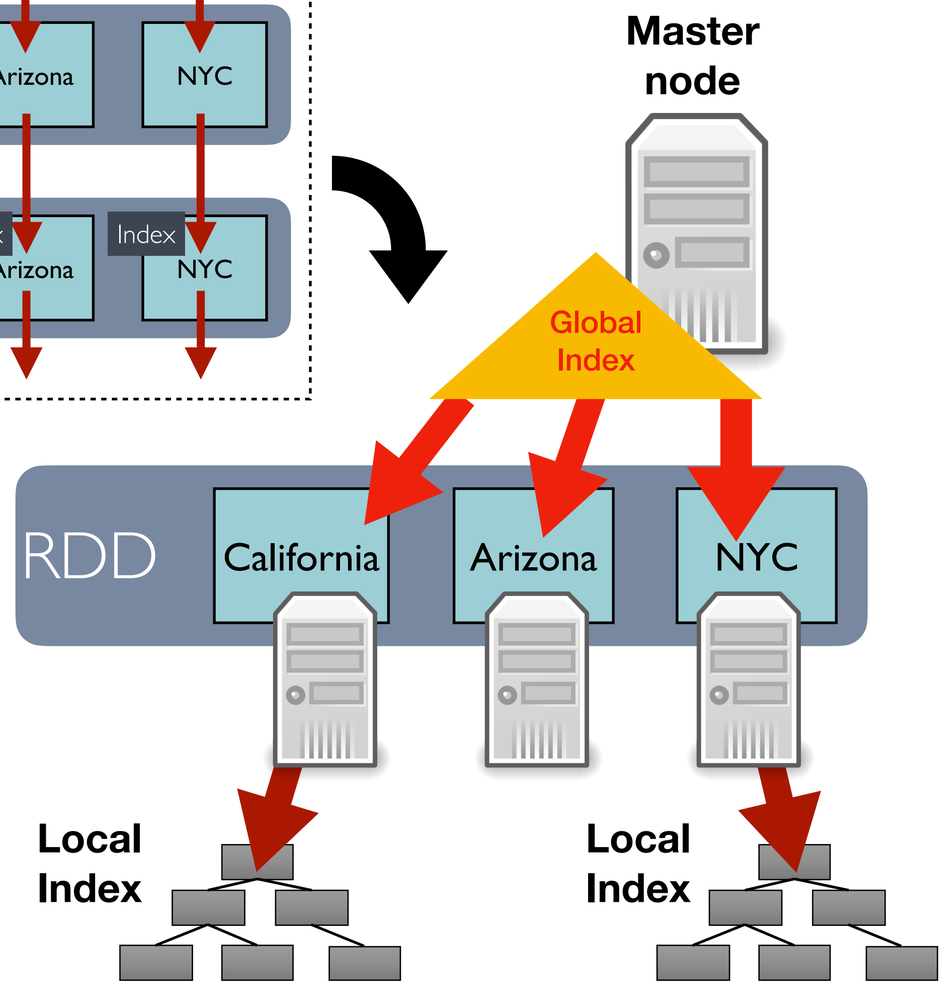


R-Tree

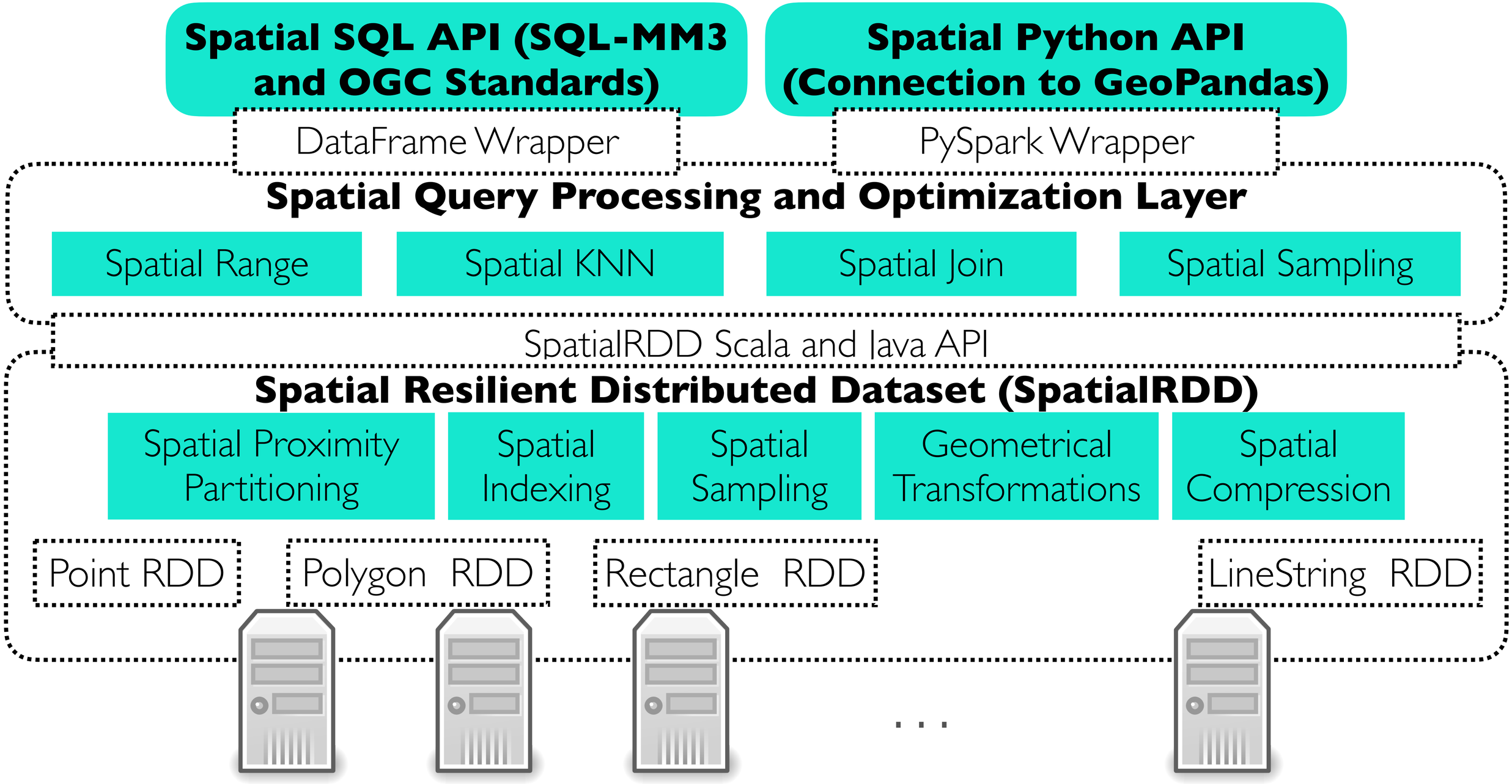
# SpatialRDD: Indexing

- **Global index**
  - Index the bounding box of each SpatialRDD partition
  - Lightweight, on the master machine
  - No entries for individual records



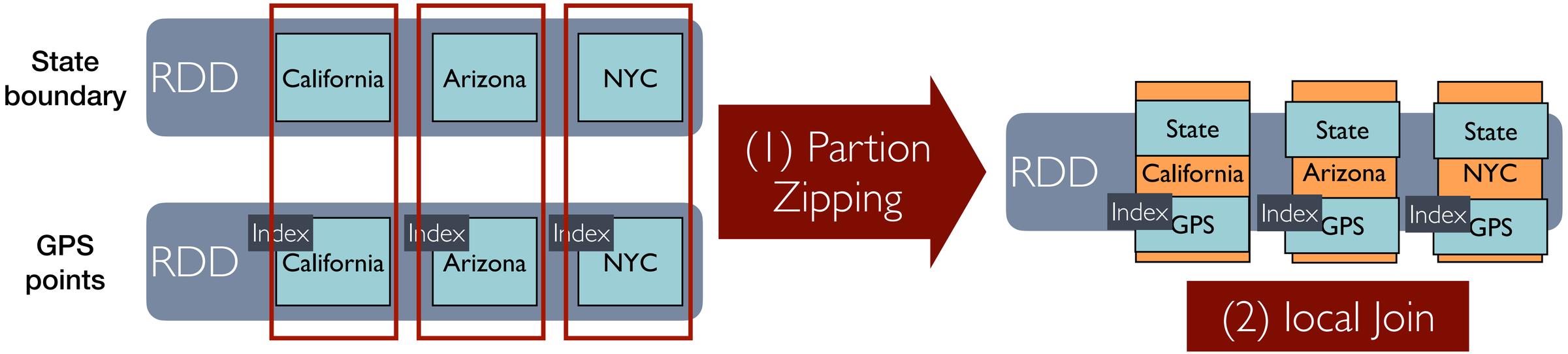




- **Local indexing**
  - On each SpatialRDD partition: RTree, QuadTree,…
  - Has entries for individual records
  - Operations that use the spatial index requires a refinement phase based on the real shapes of objects


Spatial SQL API (SQL-MM3 and OGC Standards)
Spatial Python API (Connection to GeoPandas)
DataFrame Wrapper
PySpark Wrapper
Spatial Query Processing and Optimization Layer
Spatial Range
Spatial KNN
Spatial Join
Spatial Sampling
SpatialRDD Scala and Java API
Spatial Resilient Distributed Dataset (SpatialRDD)
Spatial Proximity Partitioning
Spatial Indexing
Spatial Sampling
Geometrical Transformations
Spatial Compression
Point RDD
Polygon RDD
Rectangle RDD
LineString RDD
. . .

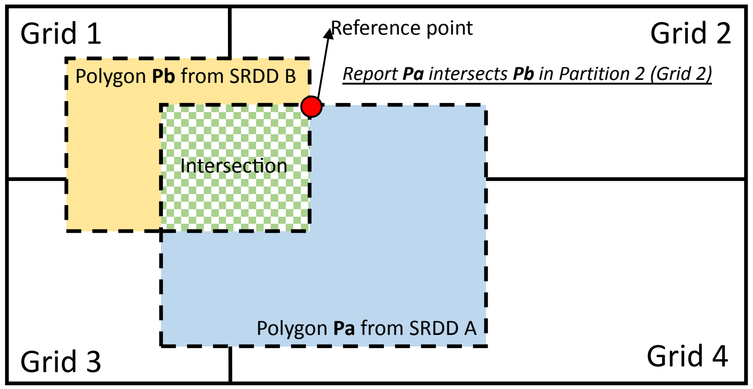
# Spatial Join Operator in Apache Sedona

# Spatial Join Operator(S)




(3) Duplicates Removal

- Query results with duplicates (Pa, Pb) (Pa, Pb) (Pa, Pb) (Pa, Pb)
- Compute the intersection of Pa and Pb
- Take Reference Point(maxX, maxY) of intersection
- Report (Pa, Pb) in a partition only if reference point is within the boundary of this partition



# Spatial Join Operator(S)

GeoSpark performs **spatial join** on
**4 billion** GPS point data and
**200 thousands** polygon data in
~**3 mins** on 4 commodity machines

# SQL API and Dataframe Wrapper in Apache Sedona

# Spatial Data Frames

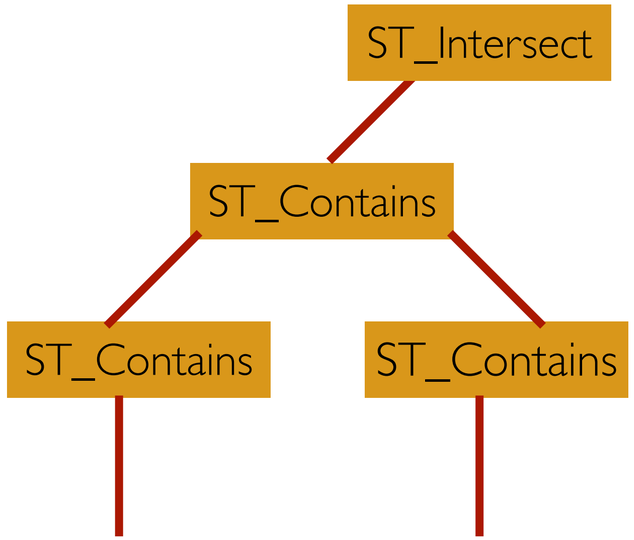
```
var spatialDf = sparkSession.sql(
  """
    |SELECT ST_GeomFromWKT(_c0) AS countyshape, _c1, _c2
    |FROM rawdf
  """.stripMargin)
spatialDf.createOrReplaceTempView("spatialdf")
spatialDf.show()
```

```
|                countyshape|_c1|_c2|     _c3|  _c4|           _c5|                  _c6|
+---------------------------+---+---+--------+-----+--------------+---------------------+
|POLYGON ((-97.019...| 31|039|00835841|31039|        Cuming|          Cuming County|
|POLYGON ((-123.43...| 53|069|01513275|53069|     Wahkiakum|       Wahkiakum County|
|POLYGON ((-104.56...| 35|011|00933054|35011|       De Baca|         De Baca County|
|POLYGON ((-96.910...| 31|109|00835876|31109|     Lancaster|       Lancaster County|
```

# Spatial SQL API

- Spatial SQL: SQL-MM3, Simple Feature Access
  - SQL-MM3: PostGIS, GeoSpark
    - *ST_Contains, ST_Within*
  - Simple Feature Access
    - *Contains, Within*
  - Compatible with each other in most cases
  - Implement these functions in Spark expression (not UDF)

- Spark expression, the way Spark writes its own functions
  - Unary, binary, ternary
  - Each AST (Abstract Syntax Tree) node is a Spark expression
  - Allow the following features
    - Code generation
    - Output data type
    - Fuse into the Catalyst optimizer



# Spatial SQL API

```
SELECT trip.*
FROM   NYCtaxi trip, GooglePlaces place
WHERE  place.type = 'Hospital'
         AND   ST_DWITHIN(place.loc, trip.dropoff, 10)
```

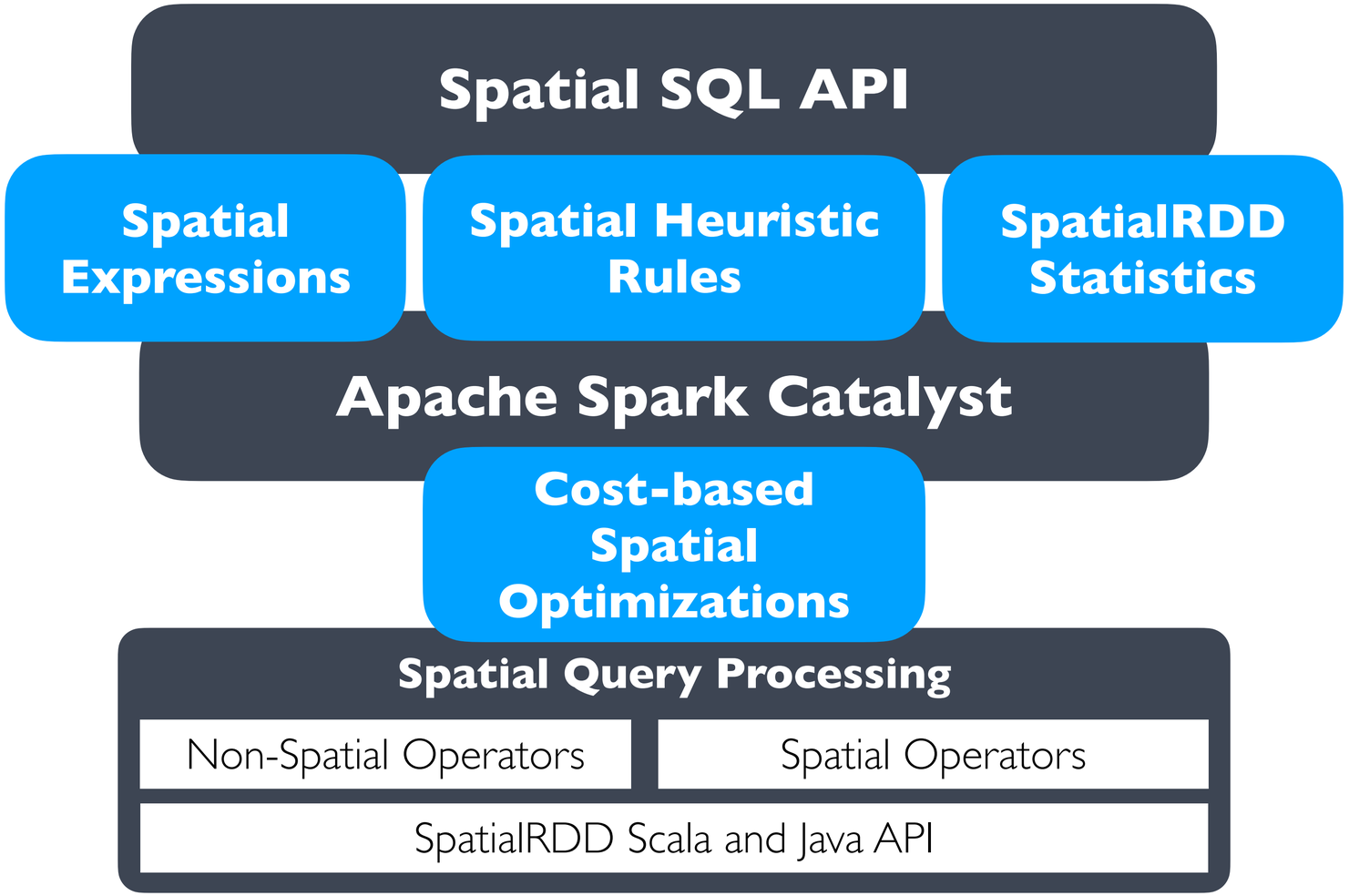

**DataFrame**

| Trip Time Stamp | …. | Fare amount | NumPassengers | Pick up location |
|---|---|---|---|---|
| 1 | … | $40 | 2 | <lat,long> |
| 2 | … | $30 | 1 | <lat,long> |
| 3 | … | $… | 1 | <lat,long> |
| … | … | … | | … |

**DataFrame**

| PlaceID | Placelocation | …. | Place Type |
|---|---|---|---|
| 1 | <lat,long> | … | Hospital |
| 2 | <lat,long> | … | Restaurant |
| 3 | <lat,long> | … | Office |
| … | … | … | … |

# Spatial SQL Optimization




```
SELECT *
FROM polygondf, pointdf
WHERE ST_Contains(polygondf.polygonshape,
                  pointdf.pointshape)
```

```
== Physical Plan ==
RangeJoin polygonshape#20: geometry,
pointshape#43: geometry, false
:- Project [st_polygonfromenvelope(XXX) AS
polygonshape#20]
:  +- *FileScan csv
+- Project [st_point(XXX) AS pointshape#43]
   +- *FileScan csv
```

```
== Physical Plan ==
BroadcastJoin polygonshape#20: geometry,
pointshape#43: geometry, false
:- Project [st_polygonfromenvelope(XXX),
mypolygonid) AS polygonshape#20]
:  +- *FileScan csv
+- Project [st_point(XXX) AS pointshape#43]
   +- *FileScan csv
```

# Extra Credit