
# GeoMesa on Spark SQL

## Extracting Location Intelligence from Data

Anthony Fox
Director of Data Science, Commonwealth Computer Research Inc
GeoMesa Founder and Technical Lead

anthony.fox@ccri.com
linkedin.com/in/anthony-fox-ccri
twitter.com/algoriffic
www.ccri.com

# Satellite AIS

exactEarth

SPARK SUMMIT 2017

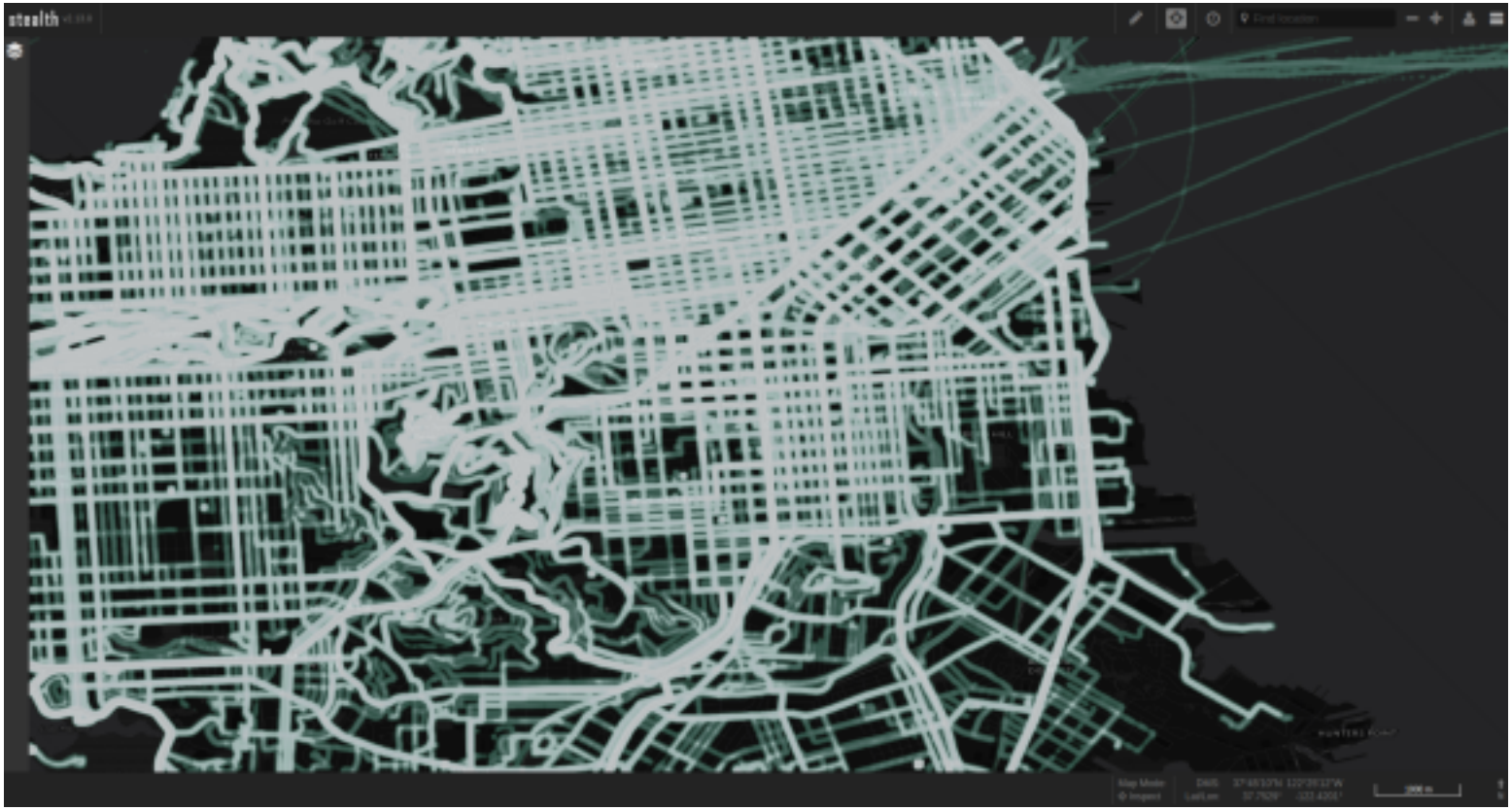
# ADS-B

FlightAware
Live Flight Tracking

SPARK SUMMIT 2017

# Mobile Apps

STRAVA



SPARK SUMMIT 2017

# Mobile Apps

STRAVA


SPARK SUMMIT 2017

**Intro to Location Intelligence and GeoMesa**

Spatial Data Types, Spatial SQL

Extending Spark Catalyst for Optimized Spatial SQL

Density of activity in San Francisco

Speed profile of San Francisco

Intro to Location Intelligence and GeoMesa

**Spatial Data Types, Spatial SQL**

Extending Spark Catalyst for Optimized Spatial SQL

Density of activity in San Francisco

Speed profile of San Francisco

Intro to Location Intelligence and GeoMesa

Spatial Data Types, Spatial SQL

**Extending Spark Catalyst for Optimized Spatial SQL**

Density of activity in San Francisco

Speed profile of San Francisco

Intro to Location Intelligence and GeoMesa

Spatial Data Types, Spatial SQL

Extending Spark Catalyst for Optimized Spatial SQL

**Density of activity in San Francisco**

**Speed profile of San Francisco**

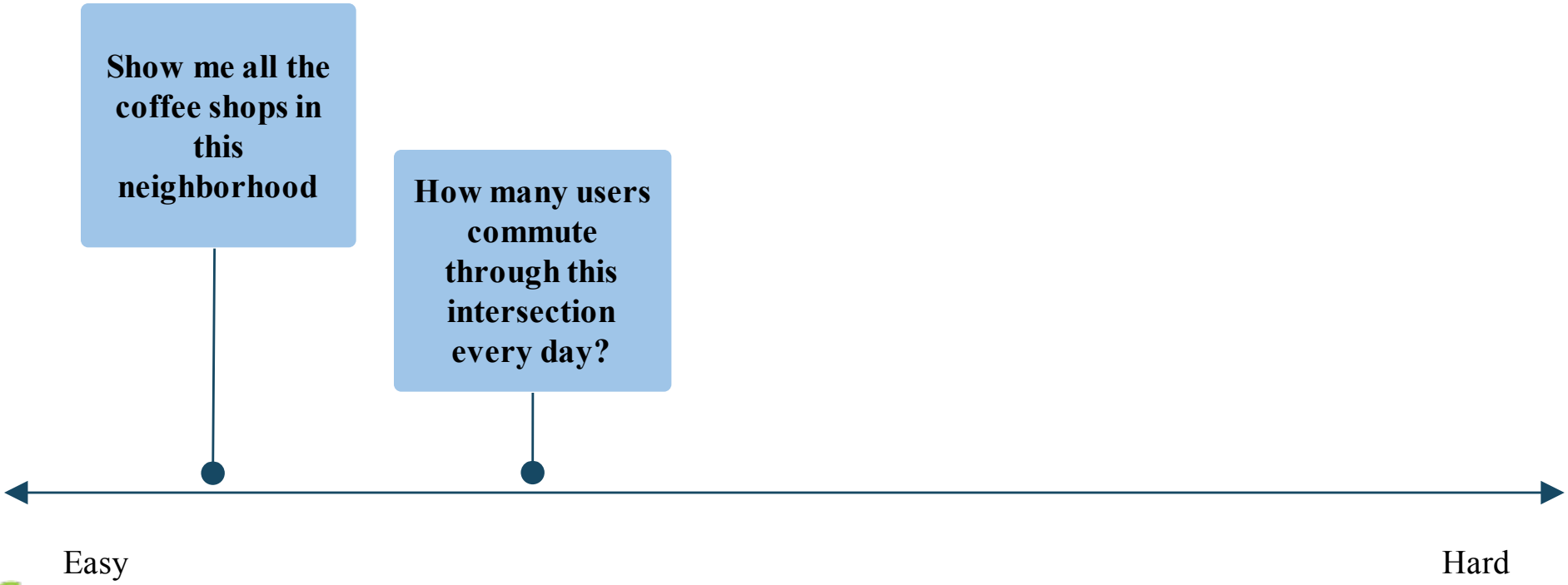
# Location Intelligence

Easy

Hard

# Location Intelligence

# Location Intelligence

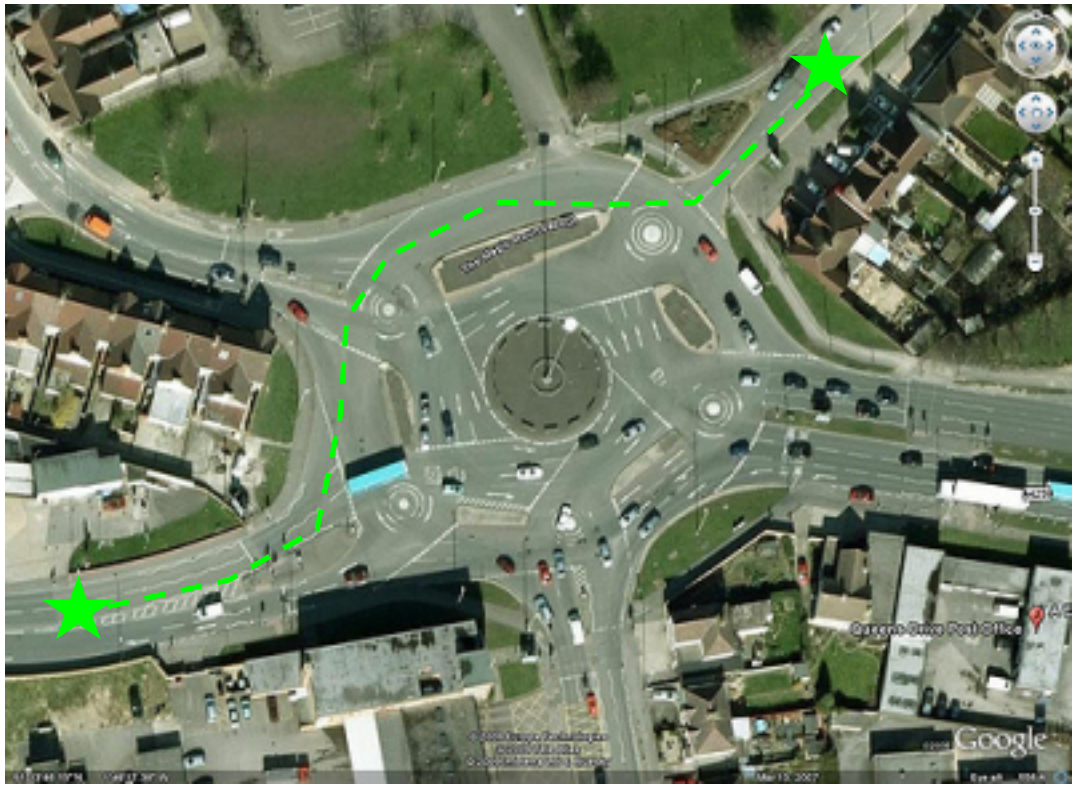


# Location Intelligence

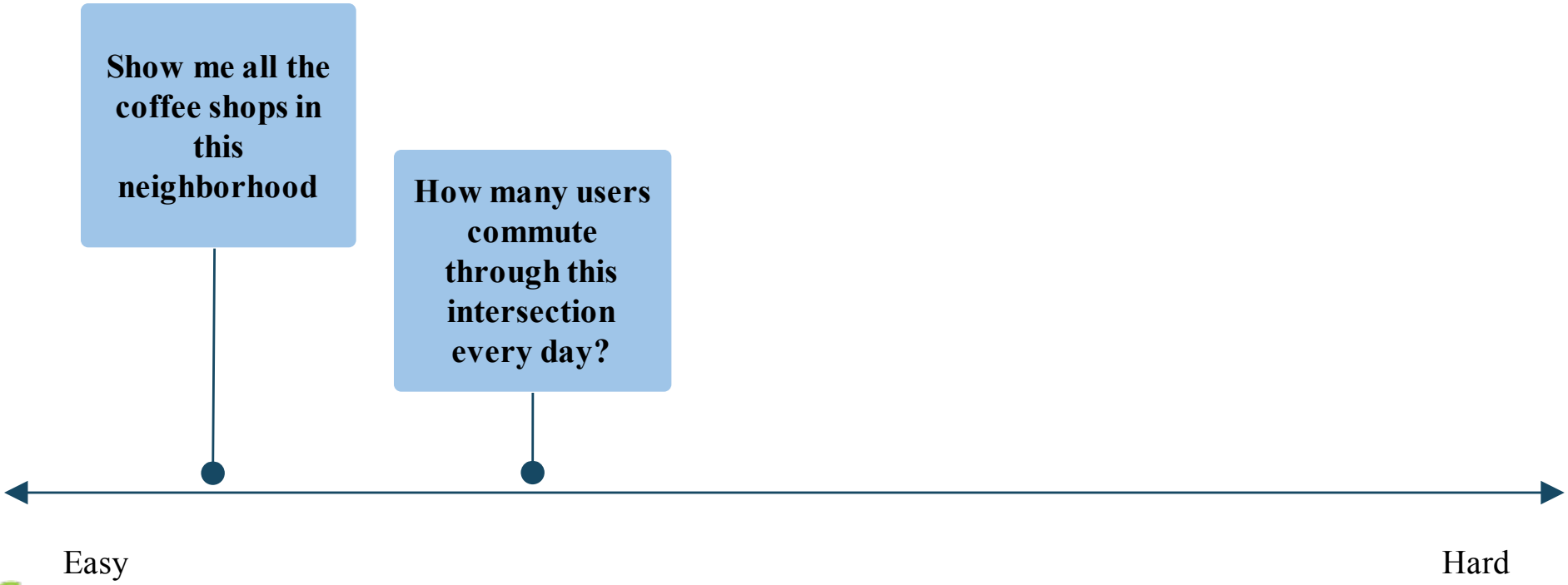
# Location Intelligence

# Location Intelligence

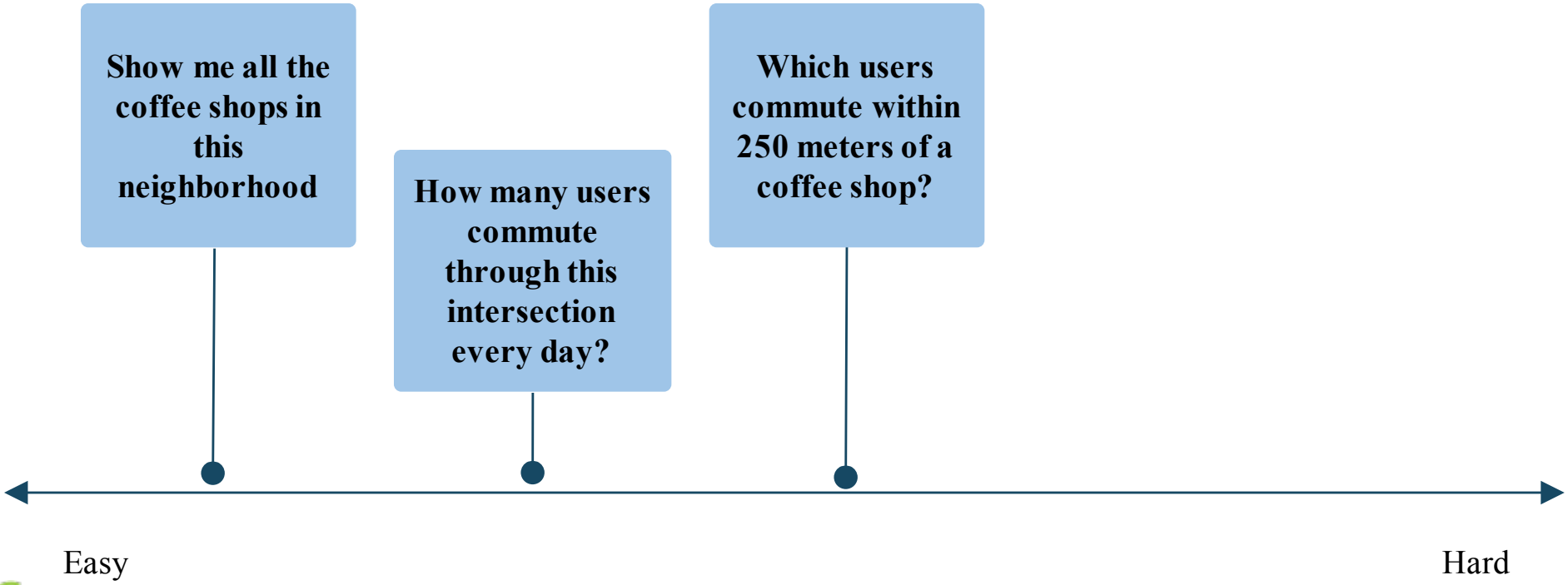
# Location Intelligence



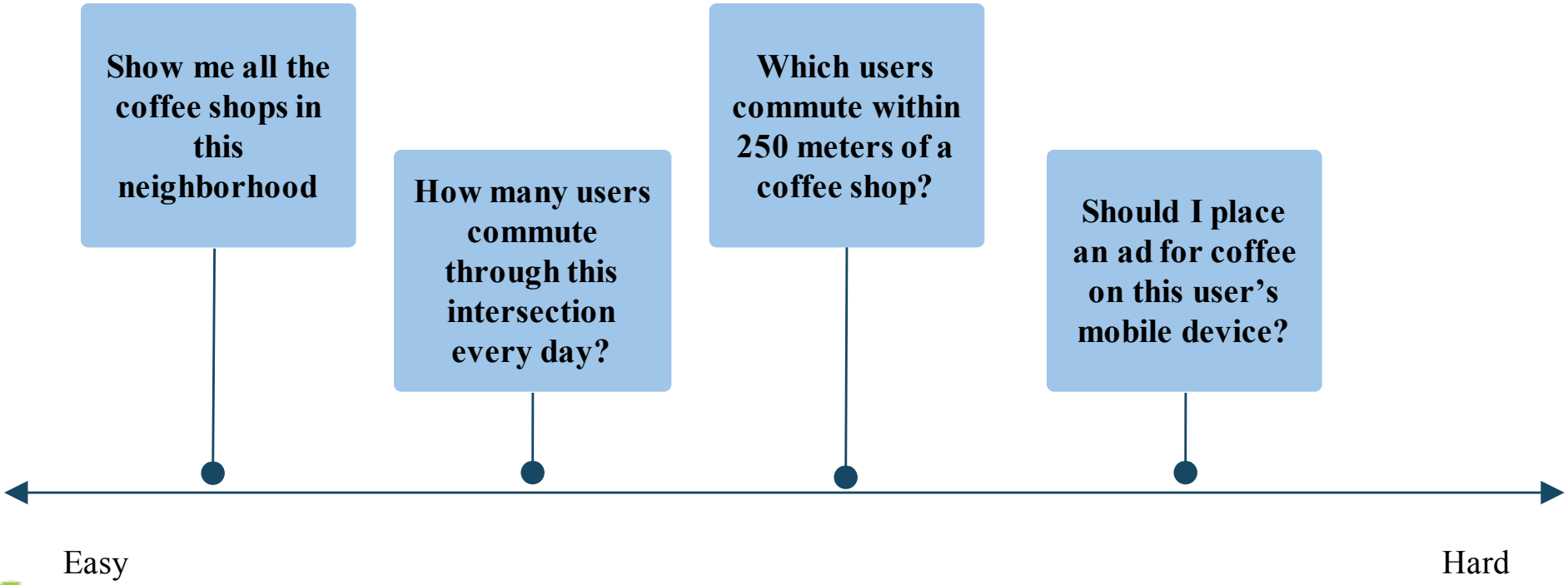
# Location Intelligence



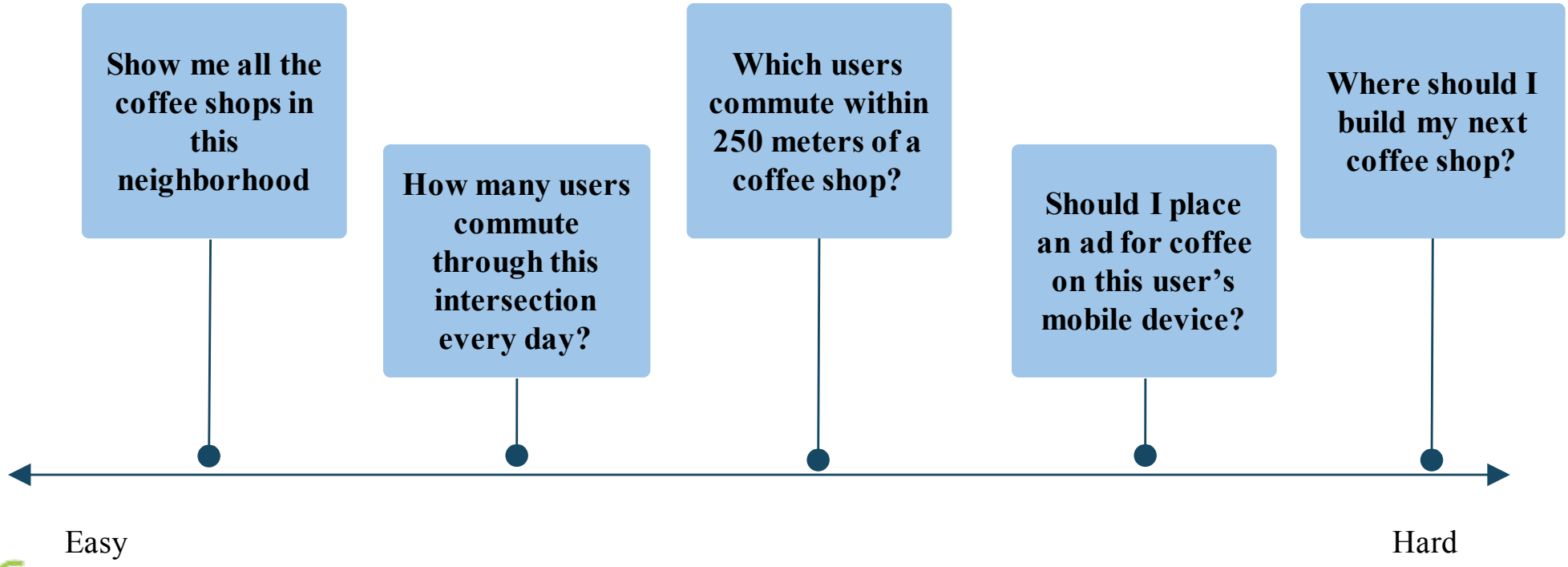
# Location Intelligence



# Location Intelligence



# Location Intelligence



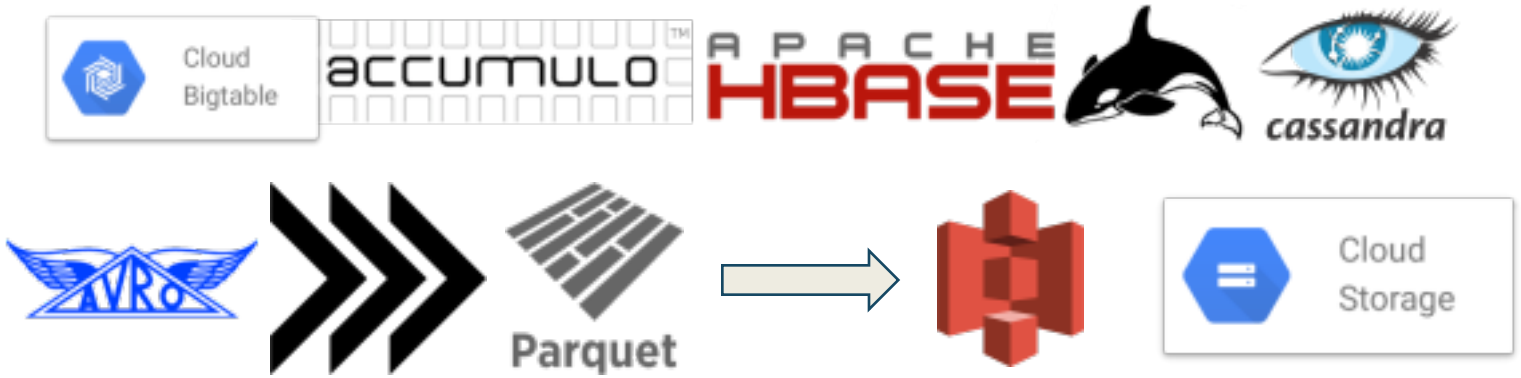
# What is GeoMesa?

A suite of tools for persisting, querying, analyzing, and streaming spatio-temporal data at scale


LocationTech

# What is GeoMesa?

A suite of tools for **persisting**, querying, analyzing, and streaming spatio-temporal data at scale



# What is GeoMesa?

A suite of tools for persisting, **querying**, analyzing, and streaming spatio-temporal data at scale

# What is GeoMesa?

A suite of tools for persisting, querying, **analyzing**, and streaming spatio-temporal data at scale




# What is GeoMesa?

A suite of tools for persisting, querying, analyzing, and **streaming** spatio-temporal data at scale

Intro to Location Intelligence and GeoMesa

**Spatial Data Types, Spatial SQL**

Extending Spark Catalyst for Optimized Spatial SQL

Density of activity in San Francisco

Speed profile of San Francisco

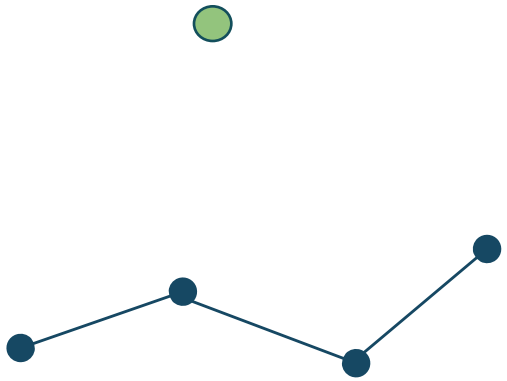
# Spatial Data Types

# Spatial Data Types

| | |
|---|---|
| Points | Locations<br>Events<br>Instantaneous Positions |

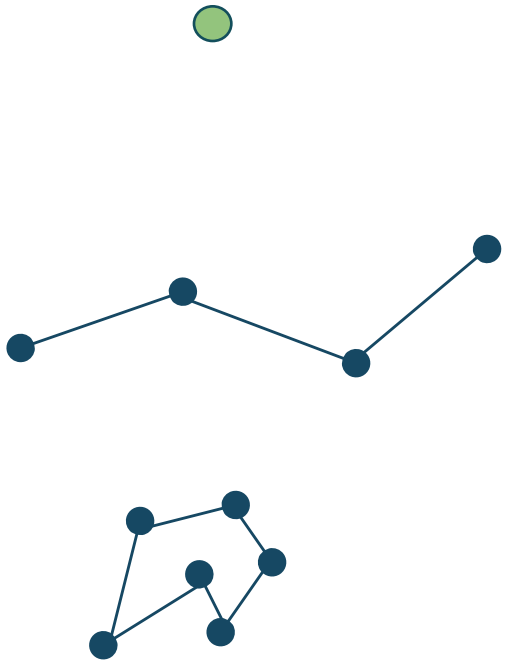
# Spatial Data Types

| | |
|---|---|
| Points | Locations<br>Events<br>Instantaneous Positions |
| Lines | Road networks<br>Voyages<br>Trips<br>Trajectories |



# Spatial Data Types

| | |
|---|---|
| Points | Locations<br>Events<br>Instantaneous Positions |
| Lines | Road networks<br>Voyages<br>Trips<br>Trajectories |
| Polygons | Administrative Regions<br>Airspaces |



# Spatial Data Types

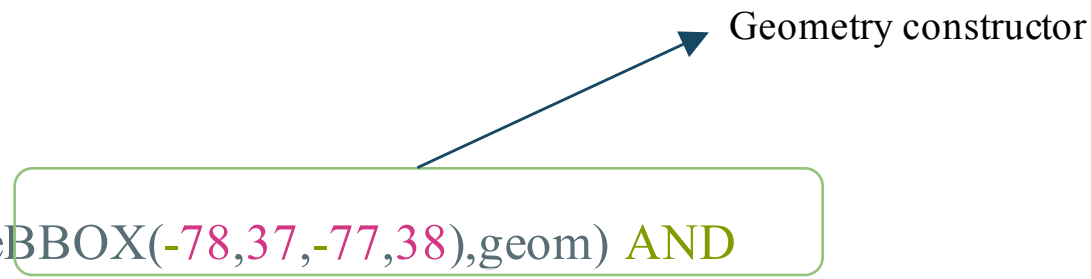
| | | |
|---|---|---|
| Points | Locations<br>Events<br>Instantaneous Positions | PointUDT<br>MultiPointUDT |
| Lines | Road networks<br>Voyages<br>Trips<br>Trajectories | LineUDT<br>MultiLineUDT |
| Polygons | Administrative Regions<br>Airspaces | PolygonUDT<br>MultiPolygonUDT |

# Spatial SQL

```
SELECT
  activity_id,user_id,geom,dtg
FROM
  activities
WHERE
  st_contains(st_makeBBOX(-78,37,-77,38),geom) AND
  dtg > cast('2017-06-01' as timestamp)        AND
  dtg < cast('2017-06-05' as timestamp)
```

# Spatial SQL

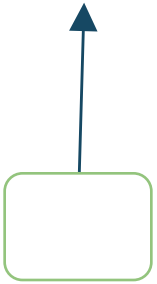
```
SELECT
  activity_id,user_id,geom,dtg
FROM
  activities
WHERE
  st_contains(st_makeBBOX(-78,37,-77,38),geom) AND
  dtg > cast('2017-06-01' as timestamp)        AND
  dtg < cast('2017-06-05' as timestamp)
```



# Spatial SQL

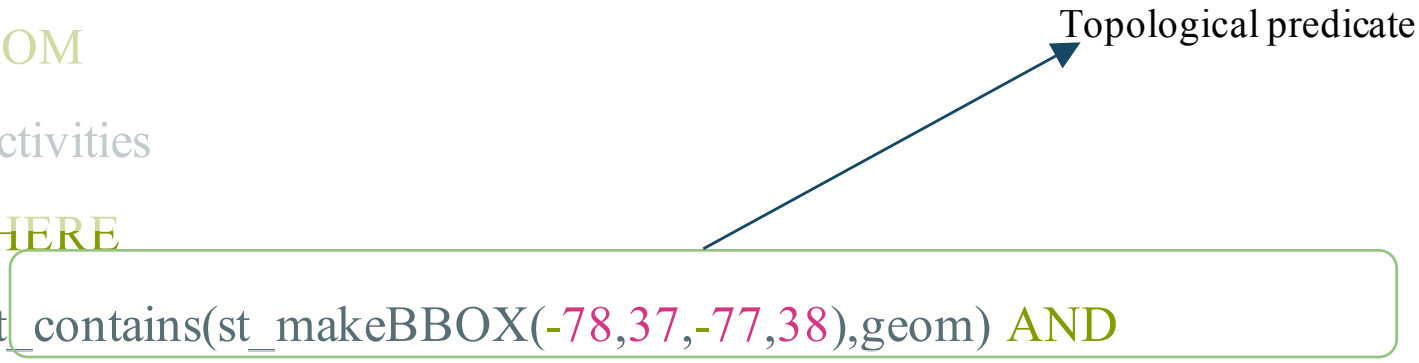
```
SELECT
  activity_id,user_id,geom,dtg
FROM
  activities
WHERE
  st_contains(st_makeBBOX(-78,37,-77,38),geom) AND
  dtg > cast('2017-06-01' as timestamp)        AND
  dtg < cast('2017-06-05' as timestamp)
```

Spatial column in schema



# Spatial SQL

```
SELECT
  activity_id,user_id,geom,dtg
FROM
  activities
WHERE
  st_contains(st_makeBBOX(-78,37,-77,38),geom) AND
  dtg > cast('2017-06-01' as timestamp)        AND
  dtg < cast('2017-06-05' as timestamp)
```



# Sample Spatial UDFs: Geometry Constructors

| | | |
|---|---|---|
| st_geomFromWKT | Create a point, line, or polygon from a WKT | st_geomFromWKT('POINT(-122.40,37.78)') |
| st_makeLine | Create a line from a sequence of points | st_makeLine(collect_list(geom)) |
| st_makeBBOX | Create a bounding box from (left, bottom, right, top) | st_makeBBOX(-123,37,-121,39) |
| … | | |

# Sample Spatial UDFs: Topological Predicates

| | | |
|---|---|---|
| st_contains | Returns true if the second argument is contained within the first argument | st_contains(<br>  st_geomFromWKT('POLYGON…'),<br>  geom<br>) |
| st_within | Returns true if the second argument geometry is entirely within the first argument | st_within(<br>  st_geomFromWKT('POLYGON…'),<br>  geom<br>) |
| st_dwithin | Returns true if the geometries are within a specified distance from each other | st_dwithin(geom1, geom2, 100) |

# Sample Spatial UDFs: Processing

| | | |
|---|---|---|
| st_bufferPoint | Create a buffer around a point for *distance within* type queries | st_bufferPoint(geom, 10) |
| st_envelope | Extract the envelope of a geometry | st_envelope(geom) |
| st_geohash | Encode the geometry using a Z-Order space filling curve. Useful for grid analysis. | st_geohash(geom, 35) |
| st_closestpoint | Find the point on the target geometry that is closest to the given geometry | st_closestpoint(geom1, geom2) |
| st_distanceSpheroid | Find the great circle distance using the WGS84 ellipsoid | st_distanceSpheroid(geom1, geom2) |

Intro to Location Intelligence and GeoMesa

Spatial Data Types, Spatial SQL

**Extending Spark Catalyst for Optimized Spatial SQL**

Density of activity in San Francisco

Speed profile of San Francisco

# Optimizing Spatial SQL

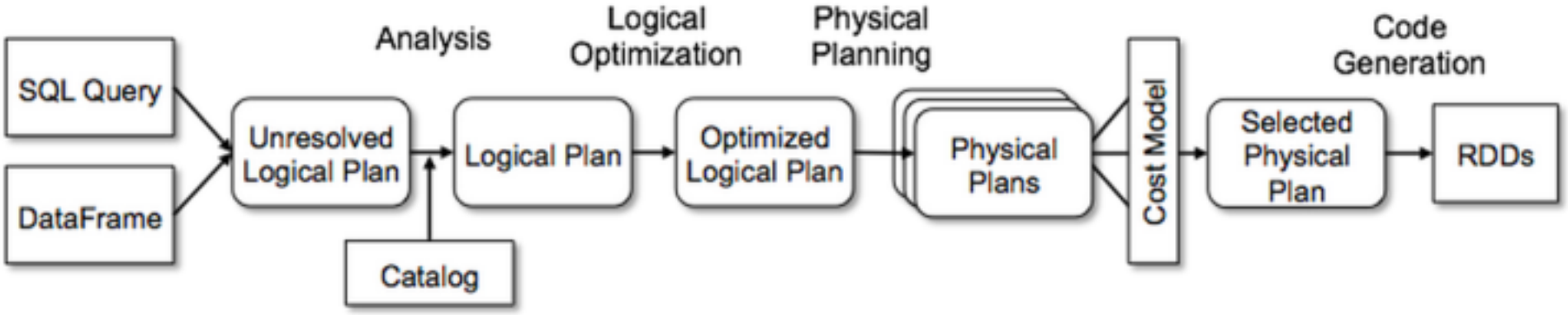
```
SELECT
  activity_id,user_id,geom,dtg
FROM
  activities
WHERE
  st_contains(st_makeBBOX(-78,37,-77,38),geom) AND
  dtg > cast('2017-06-01' as timestamp)        AND
  dtg < cast('2017-06-05' as timestamp)
```

# Optimizing Spatial SQL

```
SELECT
  activity_id,user_id,geom,dtg
FROM
  activities
WHERE
  st_contains(st_makeBBOX(-78,37,-77,38),geom) AND
  dtg > cast('2017-06-01' as timestamp)        AND
  dtg < cast('2017-06-05' as timestamp)
```

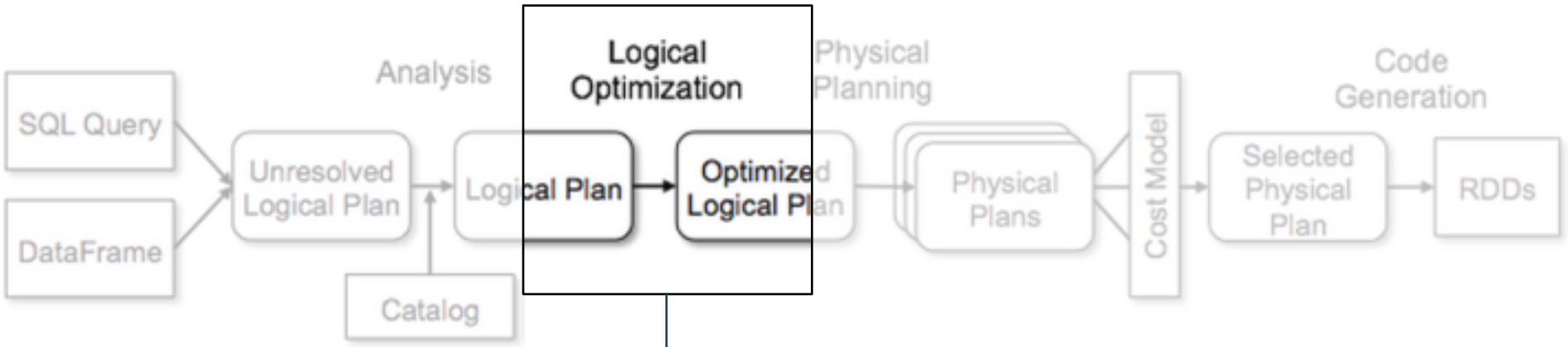
**Only load partitions that have records that intersect the query geometry.**

# Extending Spark's Catalyst Optimizer




https://databricks.com/blog/2015/04/13/deep-dive-into-spark-sqls-catalyst-optimizer.html

# Extending Spark's Catalyst Optimizer




https://databricks.com/blog/2015/04/13/deep-dive-into-spark-sqls-catalyst-optimizer.html

Catalyst exposes hooks to insert optimization rules in various points in the query processing logic.

# Extending Spark's Catalyst Optimizer

```
/**
  * :: Experimental ::
  * A collection of methods that are considered experimental, but can be used to hook into
  * the query planner for advanced functionality.
  *
  * @group basic
  * @since 1.3.0
  */
@Experimental
@transient
@InterfaceStability.Unstable
def experimental: ExperimentalMethods = sparkSession.experimental
```

# SQL optimizations for Spatial Predicates

```
SELECT
  activity_id,user_id,geom,dtg
FROM
  activities
WHERE
  st_contains(st_makeBBOX(-78,37,-77,38),geom) AND
  dtg > cast('2017-06-01' as timestamp)        AND
  dtg < cast('2017-06-05' as timestamp)
```

# SQL optimizations for Spatial Predicates

```
SELECT
  activity_id,user_id,geom,dtg
FROM
  activities
WHERE
  st_contains(st_makeBBOX(-78,37,-77,38),geom) AND
  dtg > cast('2017-06-01' as timestamp)        AND
  dtg < cast('2017-06-05' as timestamp)
```

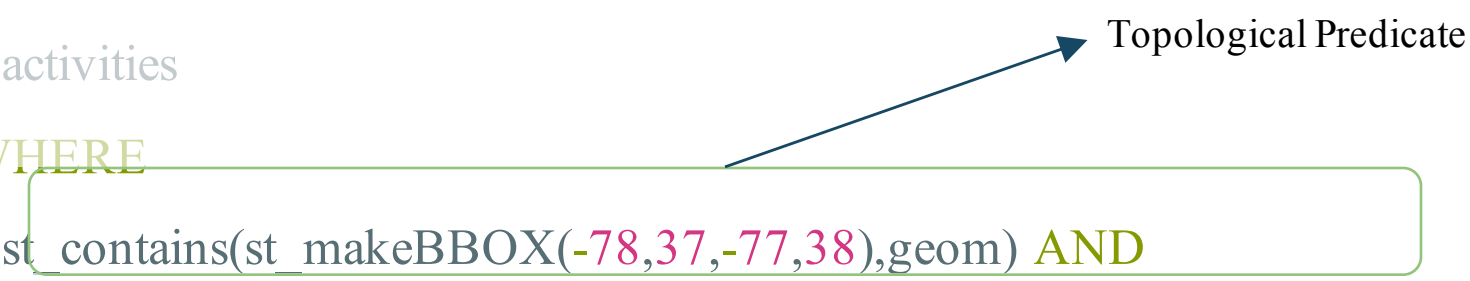
# SQL optimizations for Spatial Predicates

```
SELECT
   activity_id,user_id,geom,dtg
FROM
   activities
WHERE
   st_contains(st_makeBBOX(-78,37,-77,38),geom) AND
   dtg > cast('2017-06-01' as timestamp)        AND
   dtg < cast('2017-06-05' as timestamp)
```

Relational Projection

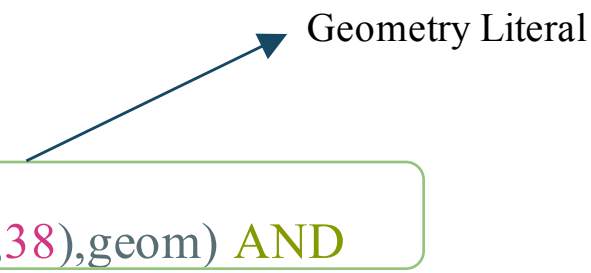
# SQL optimizations for Spatial Predicates

```
SELECT
  activity_id,user_id,geom,dtg
FROM
  activities
WHERE
  st_contains(st_makeBBOX(-78,37,-77,38),geom) AND
  dtg > cast('2017-06-01' as timestamp)        AND
  dtg < cast('2017-06-05' as timestamp)
```



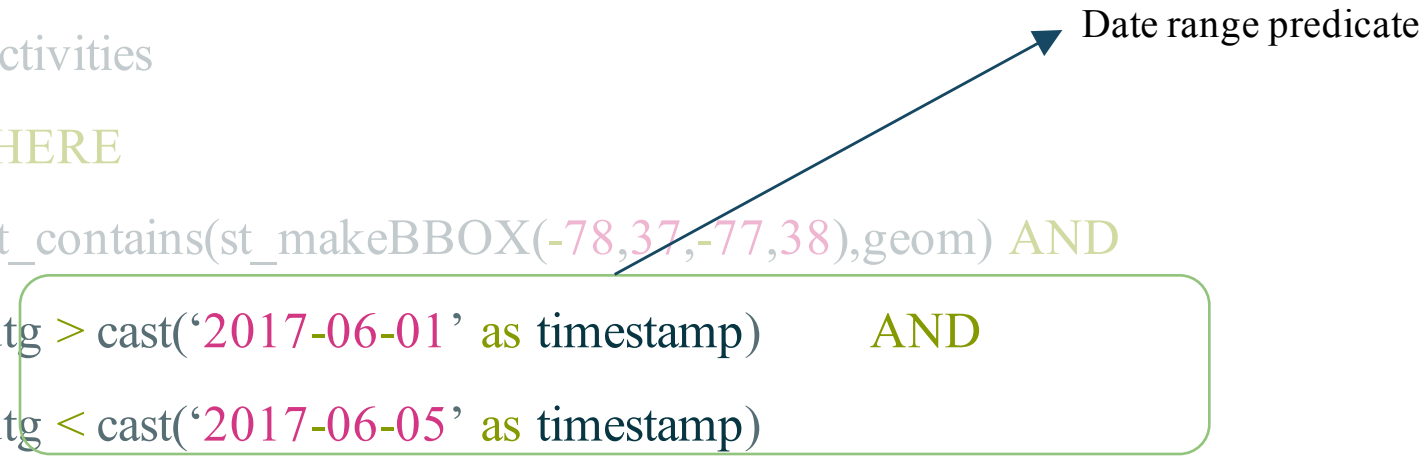
# SQL optimizations for Spatial Predicates

```
SELECT
  activity_id,user_id,geom,dtg
FROM
  activities
WHERE
  st_contains(st_makeBBOX(-78,37,-77,38),geom) AND
  dtg > cast('2017-06-01' as timestamp)        AND
  dtg < cast('2017-06-05' as timestamp)
```



# SQL optimizations for Spatial Predicates

```
SELECT
  activity_id,user_id,geom,dtg
FROM
  activities
WHERE
  st_contains(st_makeBBOX(-78,37,-77,38),geom) AND
  dtg > cast('2017-06-01' as timestamp)        AND
  dtg < cast('2017-06-05' as timestamp)
```



# SQL optimizations for Spatial Predicates

```
object STContainsRule extends Rule[LogicalPlan] with PredicateHelper {
  override def apply(plan: LogicalPlan): LogicalPlan = {
    plan.transform {
      case filt @ Filter(f, lr@LogicalRelation(gmRel: GeoMesaRelation, _, _)) =>
          …
          val relation = gmRel.copy(filt = ff.and(gtFilters :+ gmRel.filt))
          lr.copy(expectedOutputAttributes = Some(lr.output),
                relation = relation)
    }
}
```

# SQL optimizations for Spatial Predicates

```
object STContainsRule extends Rule[LogicalPlan] with PredicateHelper {
  override def apply(plan: LogicalPlan): LogicalPlan = {
    plan.transform {
      case filt @ Filter(f, lr@LogicalRelation(gmRel: GeoMesaRelation, _, _)) =>
        …
        val relation = gmRel.copy(filt = ff.and(gtFilters :+ gmRel.filt))
        lr.copy(expectedOutputAttributes = Some(lr.output),
          relation = relation)
    }
  }
```

Intercept a Filter on a GeoMesa Logical Relation

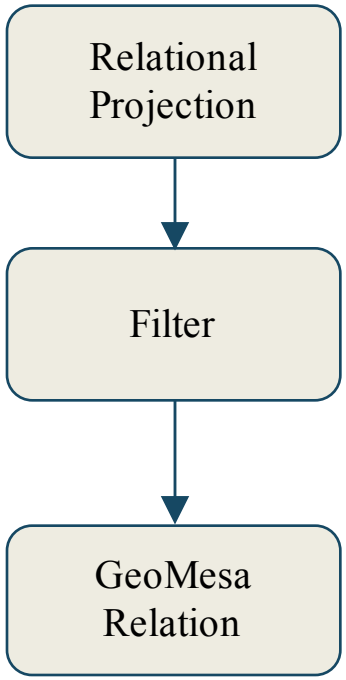
# SQL optimizations for Spatial Predicates

```
object STContainsRule extends Rule[LogicalPlan] with PredicateHelper {
  override def apply(plan: LogicalPlan): LogicalPlan = {
    plan.transform {
      case filt @ Filter(f, lr@LogicalRelation(gmRel: GeoMesaRelation, _, _)) =>
        ...
        val relation = gmRel.copy(filt = ff.and(gtFilters :+ gmRel.filt))
        lr.copy(expectedOutputAttributes = Some(lr.output),
          relation = relation)
    }
  }
```

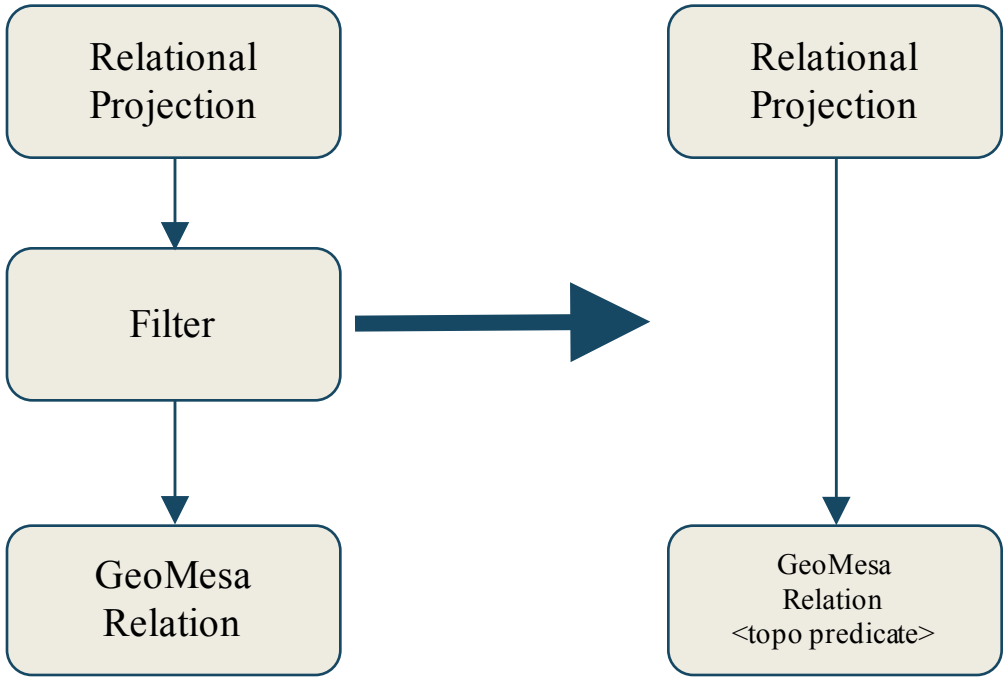
Extract the predicates that can be handled by GeoMesa, create a new GeoMesa relation with the predicates pushed down into the scan, and return a modified tree with the new relation and the filter removed.

GeoMesa will compute the minimal ranges necessary to cover the query region.

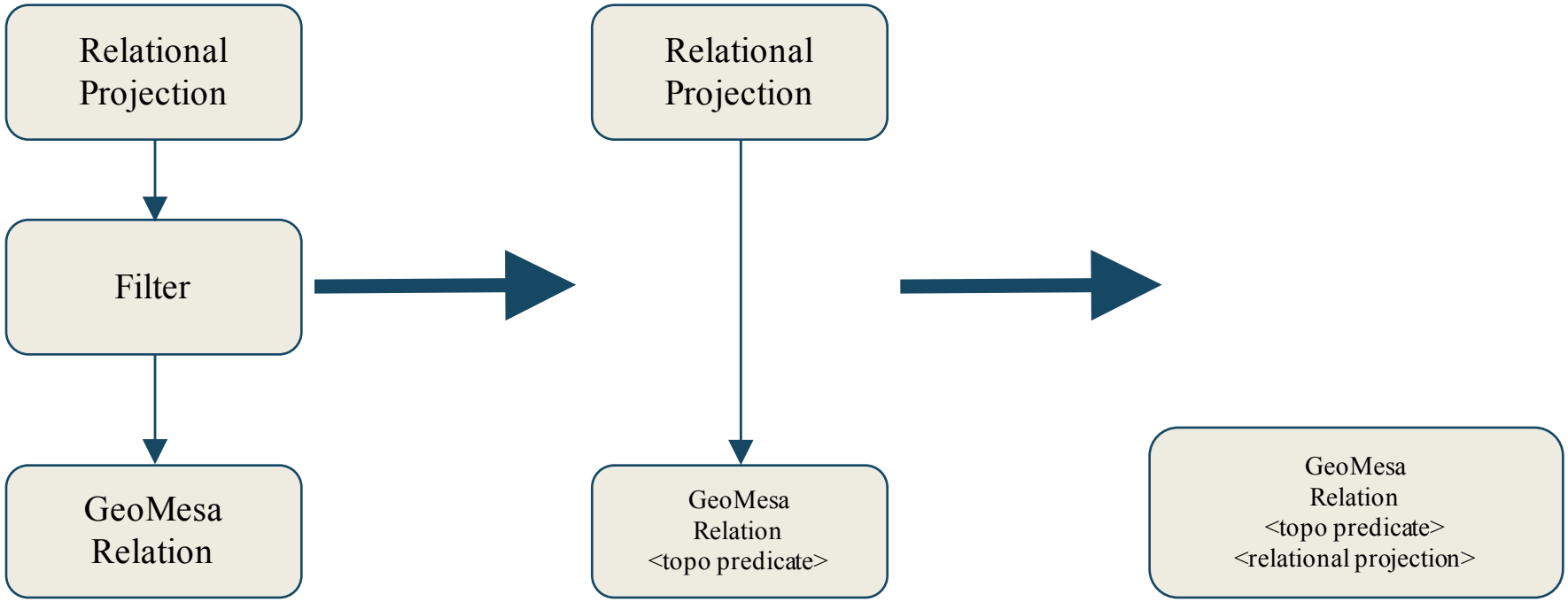
# SQL optimizations for Spatial Predicates



# SQL optimizations for Spatial Predicates



# SQL optimizations for Spatial Predicates



# SQL optimizations for Spatial Predicates

```
SELECT
  activity_id,user_id,geom,dtg
FROM
  activities
WHERE
  st_contains(st_makeBBOX(-78,37,-77,38),geom) AND
  dtg > cast('2017-06-01' as timestamp)        AND
  dtg < cast('2017-06-05' as timestamp)
```

# SQL optimizations for Spatial Predicates

```
SELECT
  *
FROM
  activities<pushdown filter and projection>
```

# SQL optimizations for Spatial Predicates

```
SELECT
  *
FROM
  activities<pushdown filter and projection>
```

**Reduced I/O, reduced network overhead, reduced compute load - faster Location Intelligence answers**

Intro to Location Intelligence and GeoMesa

Spatial Data Types, Spatial SQL

Extending Spark Catalyst for Optimized Spatial SQL

**Density of activity in San Francisco**

**Speed profile of San Francisco**

# Density of Activity in San Francisco

1. Constrain to San Francisco
2. Snap location to 35 bit geohash
3. Group by geohash and count records per geohash

```
SELECT
        geohash,
        count(geohash) as count
FROM (
  SELECT st_geohash(geom,35) as geohash
  FROM sf
  WHERE
   st_contains(st_makeBBOX(-122.4194-1,37.77-1,-122.4194+1,37.77+1),
          geom)
)
GROUP BY geohash
```

# Density of Activity in San Francisco

1. **Constrain to San Francisco**
2. Snap location to 35 bit geohash
3. Group by geohash and count records per geohash

```
SELECT
        geohash,
        count(geohash) as count
FROM (
  SELECT st_geohash(geom,35) as geohash
  FROM sf
  WHERE
    st_contains(st_makeBBOX(-122.4194-1,37.77-1,-122.4194+1,37.77+1),
      geom)
)
GROUP BY geohash
```

# Density of Activity in San Francisco

1. Constrain to San Francisco
2. **Snap location to 35 bit geohash**
3. Group by geohash and count records per geohash

```
SELECT
        geohash,
        count(geohash) as count
FROM (
  SELECT st_geohash(geom,35) as geohash
  FROM sf
  WHERE
   st_contains(st_makeBBOX(-122.4194-1,37.77-1,-122.4194+1,37.77+1),
        geom)
)
GROUP BY geohash
```

# Density of Activity in San Francisco

1. Constrain to San Francisco
2. Snap location to 35 bit geohash
3. **Group by geohash and count records per geohash**

```
SELECT
        geohash,
        count(geohash) as count
FROM (
  SELECT st_geohash(geom,35) as geohash
  FROM sf
  WHERE
   st_contains(st_makeBBOX(-122.4194-1,37.77-1,-122.4194+1,37.77+1),
            geom)
)
GROUP BY geohash
```

# Density of Activity in San Francisco

1. Constrain to San Francisco
2. Snap location to 35 bit geohash
3. Group by geohash and count records per geohash

```
+-------+-----+
|geohash|count|
+-------+-----+
|9q8zhzw|75769|
|9q8zkf1|  777|
|9q8z9fu|  894|
|9q8zexz|16077|
|9q8zckh| 8043|
|9q8zspq|55027|
|9q8ztb8|28296|
|9q8z5rq|20482|
|9q8zn0k|15062|
|9q8zxy1| 2289|
|9q8ymy4|24343|
|9q8ytst| 3466|
|9q8yt5f| 4569|
|9q8yye9|22025|
|9q8yv8v|18613|
|9q8yvnc|44084|
|9q8yyzh| 8856|
|9q8yu7t| 9208|
|9q8zduj| 1906|
|9q8zjh5|  176|
+-------+-----+
only showing top 20 rows
```

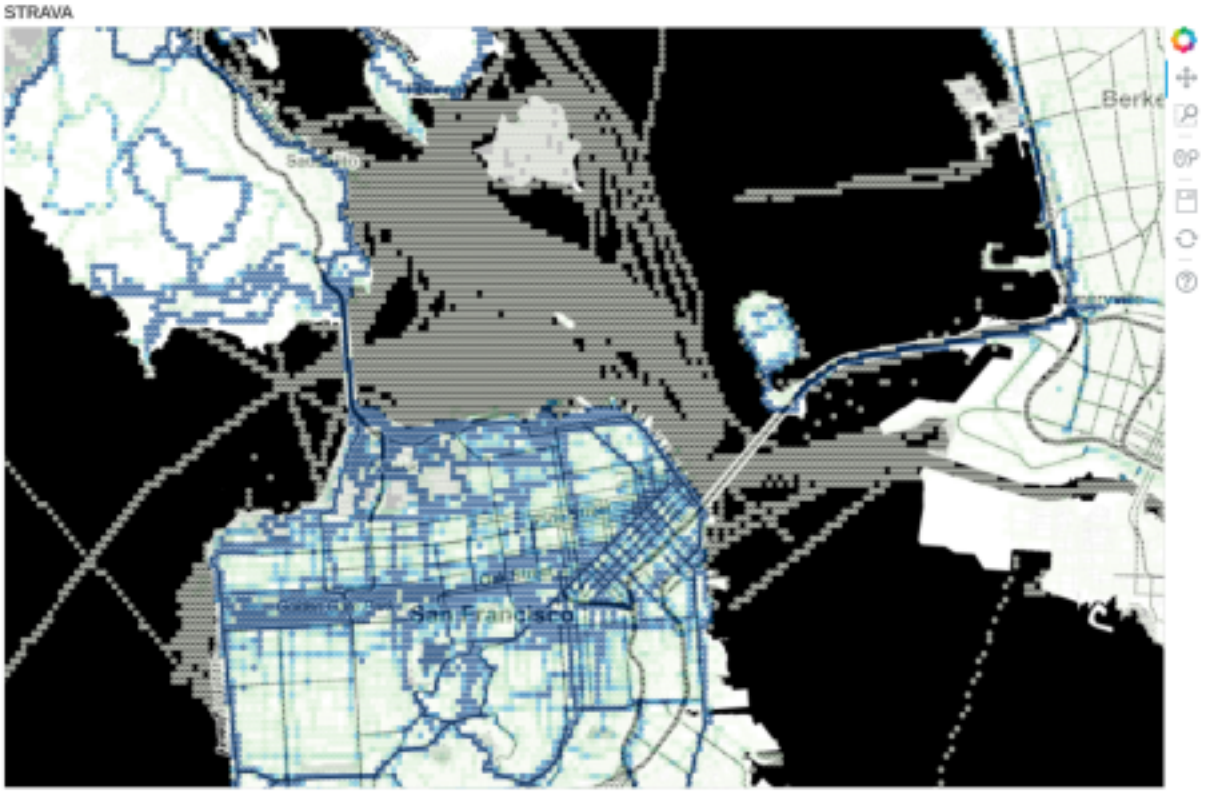
# Density of Activity in San Francisco

## Visualize using Jupyter and Bokeh

```
p = figure(title="STRAVA",
        plot_width=900, plot_height=600,
        x_range=x_range,y_range=y_range)
p.add_tile(tonerlines)
p.circle(x=projecteddf['px'],
       y=projecteddf['py'],
       fill_alpha=0.5,
       size=6,
       fill_color=colors,
       line_color=colors)
show(p)
```

# Density of Activity in San Francisco

**Visualize using Jupyter and Bokeh**



# Speed Profile of a Metro Area

## Inputs

### STRAVA Activities

An activity is sampled once per second

Each observation has a location and time


```
{
 "type": "Feature",
 "geometry": { "type": "Point", "coordinates": [-122.40736,37.807147] },
 "properties": {
  "activity_id": "**********",
  "athlete_id": "**********",
  "device_type": 5,
  "activity_type": "Ride",
  "frame_type": 2,
  "commute": false,
  "date": "2016-11-02T23:58:03",
  "index": 0
 },
 "id": "6a9bb90497be6f64eae009e6c760389017bc31db:0"
}
```

# Speed Profile of a Metro Area

1. **Select all activities within metro area**
2. Sort activity by dtg ascending
3. Window over each set of consecutive samples
4. Create a temporary table

```
SELECT
  activity_id,
  index,
  geom as s,
  lead(geom) OVER (PARTITION BY activity_id ORDER by dtg asc) as e,
  dtg as start,
  lead(dtg) OVER (PARTITION BY activity_id ORDER by dtg asc) as end
FROM activities
WHERE
  activity_type = 'Ride' AND
  st_contains(
        st_makeBBOX(-122.4194-1,37.77-1,-122.4194+1,37.77+1),
        geom)
ORDER BY dtg ASC
```

# Speed Profile of a Metro Area

1. Select all activities within metro area
2. **Sort activity by dtg ascending**
3. Window over each set of consecutive samples
4. Create a temporary table

```
SELECT
  activity_id,
  index,
  geom as s,
  lead(geom) OVER (PARTITION BY activity_id ORDER by dtg asc) as e,
  dtg as start,
  lead(dtg) OVER (PARTITION BY activity_id ORDER by dtg asc) as end
FROM activities
WHERE
  activity_type = 'Ride' AND
  st_contains(
        st_makeBBOX(-122.4194-1,37.77-1,-122.4194+1,37.77+1),
        geom)
ORDER BY dtg ASC
```

# Speed Profile of a Metro Area

1. Select all activities within metro area
2. Sort activity by dtg ascending
3. **Window over each set of consecutive samples**
4. Create a temporary table

```
SELECT
  activity_id,
  index,
  geom as s,
  lead(geom) OVER (PARTITION BY activity_id ORDER by dtg asc) as e,
  dtg as start,
  lead(dtg) OVER (PARTITION BY activity_id ORDER by dtg asc) as end
FROM activities
WHERE
  activity_type = 'Ride' AND
  st_contains(
        st_makeBBOX(-122.4194-1,37.77-1,-122.4194+1,37.77+1),
        geom)
ORDER BY dtg ASC
```

# Speed Profile of a Metro Area

1. Select all activities within metro area
2. Sort activity by dtg ascending
3. **Window over each set of consecutive samples**
4. Create a temporary table

```
SELECT
  activity_id,
  index,
  geom as s,
  lead(geom) OVER (PARTITION BY activity_id ORDER by dtg asc) as e,
  dtg as start,
  lead(dtg) OVER (PARTITION BY activity_id ORDER by dtg asc) as end
FROM activities
WHERE
  activity_type = 'Ride' AND
  st_contains(
        st_makeBBOX(-122.4194-1,37.77-1,-122.4194+1,37.77+1),
        geom)
ORDER BY dtg ASC
```

# Speed Profile of a Metro Area

1. Select all activities within metro area
2. Sort activity by dtg ascending
3. Window over each set of consecutive samples
4. **Create a temporary table**

```
spark.sql(“””
SELECT
  activity_id,
  index,
  geom as s,
  lead(geom) OVER (PARTITION BY activity_id ORDER by dtg asc) as e,
  dtg as start,
  lead(dtg) OVER (PARTITION BY activity_id ORDER by dtg asc) as end
FROM activities
WHERE
  activity_type = 'Ride' AND
  st_contains(
        st_makeBBOX(-122.4194-1,37.77-1,-122.4194+1,37.77+1),
        geom)
ORDER BY dtg ASC
“””).createOrReplaceTempView(“segments”)
```

# Speed Profile of a Metro Area

1. Select all activities within metro area
2. Sort activity by dtg ascending
3. Window over each set of consecutive samples
4. Create a temporary table

```
In [25]: spark.sql(
"""
SELECT
   geom as s,
   lead(geom) OVER (PARTITION BY activity_id ORDER by dtg asc) as e,
   dtg as start,
   lead(dtg) OVER (PARTITION BY activity_id ORDER by dtg asc) as end
FROM sf
WHERE activity_type = 'Ride'
ORDER BY dtg asc
""").show(truncate=False)
```

```
+--------------------------+--------------------------+--------------------+--------------------+
|s                         |e                         |start               |end                 |
+--------------------------+--------------------------+--------------------+--------------------+
|POINT (-122.403497 37.796432)|POINT (-122.403441 37.796163)|2016-10-01 07:01:25.0|2016-10-01 07:01:29.0|
|POINT (-122.403441 37.796163)|POINT (-122.40343 37.79611)  |2016-10-01 07:01:29.0|2016-10-01 07:01:30.0|
|POINT (-122.40343 37.79611)  |POINT (-122.403421 37.796062)|2016-10-01 07:01:30.0|2016-10-01 07:01:31.0|
|POINT (-122.403421 37.796062)|POINT (-122.403456 37.796017)|2016-10-01 07:01:31.0|2016-10-01 07:01:32.0|
|POINT (-122.403456 37.796017)|POINT (-122.403451 37.795981)|2016-10-01 07:01:32.0|2016-10-01 07:01:33.0|
|POINT (-122.403451 37.795981)|POINT (-122.403447 37.795944)|2016-10-01 07:01:33.0|2016-10-01 07:01:34.0|
|POINT (-122.403447 37.795944)|POINT (-122.403442 37.795908)|2016-10-01 07:01:34.0|2016-10-01 07:01:35.0|
|POINT (-122.403442 37.795908)|POINT (-122.403435 37.795873)|2016-10-01 07:01:35.0|2016-10-01 07:01:36.0|
```

# Speed Profile of a Metro Area

5. **Compute the distance between consecutive points**
6. Compute the time difference between consecutive points
7. Compute the speed
8. Snap the location to a grid based on a GeoHash
9. Create a temporary table

```
SELECT
  st_geohash(s,35) as gh,
st_distanceSpheroid(s, e)/
    cast(cast(end as long)-cast(start as long) as double)
    as meters_per_second
FROM segments
```

# Speed Profile of a Metro Area

5. Compute the distance between consecutive points
6. **Compute the time difference between consecutive points**
7. Compute the speed
8. Snap the location to a grid based on a GeoHash
9. Create a temporary table

```
SELECT
  st_geohash(s,35) as gh,
  st_distanceSpheroid(s, e)/
    cast(cast(end as long)-cast(start as long) as double)
    as meters_per_second
FROM segments
```

# Speed Profile of a Metro Area

5. Compute the distance between consecutive points
6. Compute the time difference between consecutive points
7. **Compute the speed**
8. Snap the location to a grid based on a GeoHash
9. Create a temporary table

```
SELECT
  st_geohash(s,35) as gh,
  st_distanceSpheroid(s, e)/
    cast(cast(end as long)-cast(start as long) as double)
    as meters_per_second
FROM segments
```

# Speed Profile of a Metro Area

5. Compute the distance between consecutive points
6. Compute the time difference between consecutive points
7. Compute the speed
8. **Snap the location to a grid based on a GeoHash**
9. Create a temporary table

```
SELECT
 st_geohash(s,35) as gh,
 st_distanceSpheroid(s, e)/
   cast(cast(end as long)-cast(start as long) as double)
   as meters_per_second
FROM segments
```

# Speed Profile of a Metro Area

5. Compute the distance between consecutive points
6. Compute the time difference between consecutive points
7. Compute the speed
8. Snap the location to a grid based on a GeoHash
9. **Create a temporary table**

```
spark.sql(“””
SELECT
  st_geohash(s,35) as gh,
  st_distanceSpheroid(s, e)/
    cast(cast(end as long)-cast(start as long) as double)
    as meters_per_second
FROM segments
“””).createOrReplaceTempView(“gridspeeds”)
```

# Speed Profile of a Metro Area

5. Compute the distance between consecutive points
6. Compute the time difference between consecutive points
7. Compute the speed
8. Snap the location to a grid based on a GeoHash
9. Create a temporary table

```
In [33]: spark.sql(
         """
         SELECT
            st_geohash(s,30) as gh,
            st_distanceSpheroid(s, e)/
              cast(cast(end as long)-cast(start as long) as double) as meters_per_second
         FROM segments
         """).show()
```

```
+------+------------------+
|    gh| meters_per_second|
+------+------------------+
|9q8zn8|5.9618546994330694|
|9q8zn8| 4.489081320908792|
|9q8zn8|3.4220339047269968|
|9q8zn8|4.8532700664059565|
|9q8zn8| 6.724161438827997|
|9q8zn8|  8.13151025443169|
|9q8zn8| 7.078674201646255|
|9q8zn8| 9.972934192494485|
|9q8zn8| 5.228541182757107|
|9q8zn8| 8.637458644329197|
|9q8zn8| 6.238981935255453|
|9q8zn8|5.1476384306689145|
|9q8zn8| 4.614274118319755|
|9q8zn8|   3.7926167585289|
|9q8yyx|3.3697452165753523|
|9q8yyx| 5.002785203836417|
|9q8yyx| 3.008499247612676|
|9q8yyx| 2.6455242826351539|
|9q8yyx|2.7413673292367142|
|9q8yyx|   3.3396971902483|
+------+------------------+
only showing top 20 rows
```

# Speed Profile of a Metro Area

10. **Group the grid cells**
11. For each grid cell, compute the median and standard deviation of the speed
12. Extract the location of the grid cell

```
SELECT
  st_centroid(st_geomFromGeoHash(gh,35)) as p,
  percentile_approx(meters_per_second,0.5) as avg_meters_per_second,
  stddev(meters_per_second) as std_dev
FROM gridspeeds
GROUP BY gh
```

# Speed Profile of a Metro Area

10. Group the grid cells
11. **For each grid cell, compute the median and standard deviation of the speed**
12. Extract the location of the grid cell

```
SELECT
  st_centroid(st_geomFromGeoHash(gh,35)) as p,
  percentile_approx(meters_per_second,0.5) as med_meters_per_second,
  stddev(meters_per_second) as std_dev
FROM gridspeeds
GROUP BY gh
```

# Speed Profile of a Metro Area

10. Group the grid cells
11. For each grid cell, compute the median and standard deviation of the speed
12. **Extract the location of the grid cell**

```
SELECT
  st_centroid(st_geomFromGeoHash(gh,35)) as p,
  percentile_approx(meters_per_second,0.5) as avg_meters_per_second,
  stddev(meters_per_second) as std_dev
FROM gridspeeds
GROUP BY gh
```

# Speed Profile of a Metro Area

10. Group the grid cells
11. For each grid cell, compute the median and standard deviation of the speed
12. Extract the location of the grid cell

```
In [36]: spark.sql(
"""
SELECT
    st_centroid(st_geoeFromGeoHash(gh,30)) as p,
    percentile_approx(meters_per_second,0.5) as avg_meters_per_second,
    stddev(meters_per_second) as std_dev
FROM gridspeeds
GROUP BY gh
"""),show(truncate=False)
```

```
+------------------------------------------+--------------------+--------------------+
|p                                         |avg_meters_per_second|std_dev             |
+------------------------------------------+--------------------+--------------------+
|POINT (-118.0096435546875 33.74725341796875)|29.676260037729447  |1.0053242287423414  |
|POINT (-117.9656982421875 33.71978759765625)|32.61891249560786   |NaN                 |
|POINT (-117.9327392578125 33.69232177734375)|31.956967217626826  |0.13457036607757904 |
|POINT (-117.9437255859375 33.70330810546875)|32.085216085692075  |NaN                 |
|POINT (-117.9327392578125 33.69781494140625)|32.17472724029561   |0.5588892829379065  |
|POINT (-117.9217529296875 33.68682861328125)|32.355755540079784  |0.8587985234919129  |
|POINT (-117.8887939453125 33.68682861328125)|31.714696790235276  |NaN                 |
|POINT (-117.8778076171875 33.68682861328125)|29.00784194247733   |NaN                 |
|POINT (-117.8558349609375 33.68133544921875)|28.67192319503053   |NaN                 |
|POINT (-117.8338623046875 33.67034912109375)|29.53143535560496   |3.0560498712375535  |
|POINT (-117.8228759765625 33.67034912109375)|18.685081378756342  |8.9707213036407     |
|POINT (-117.8338623046875 33.67584228515625)|5.992025859887013   |4.542280862928457 5 |
|POINT (-117.8228759765625 33.67584228515625)|8.927158515626157   |NaN                 |
|POINT (-117.8228759765625 33.68133544921875)|8.920866027442344   |1.6888776937045932  |
|POINT (-117.8118896484375 33.68133544921875)|0.08661693036809194 |4.018956714477015   |
|POINT (-117.8118896484375 33.68682861328125)|0.022758042751552857|NaN                 |
|POINT (-121.4044189453125 35.84564208984375)|18.838227935605982  |NaN                 |
|POINT (-121.4154052734375 35.85662841796875)|21.886604822583904  |2.489453904376773   |
|POINT (-121.4044189453125 35.85113525390625)|18.488256245614935  |1.2892779684179 8   |
|POINT (-121.7449951171875 36.20819091796875)|16.528008862649042  |NaN                 |
+------------------------------------------+--------------------+--------------------+
only showing top 20 rows
```

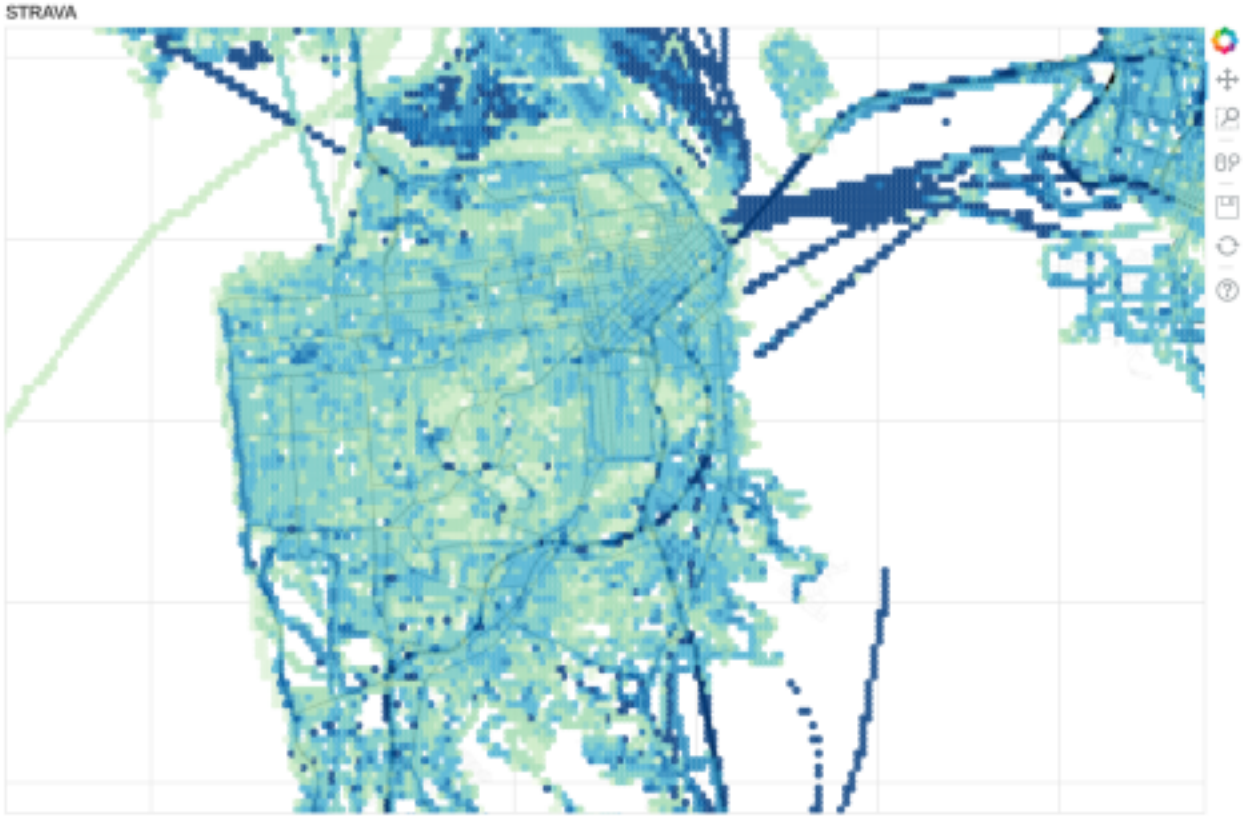
SPARK SUMMIT 2017

# Speed Profile of a Metro Area

## Visualize using Jupyter and Bokeh

```
p = figure(title="STRAVA",
        plot_width=900,plot_height=600,
        x_range=x_range,y_range=y_range)
p.add_tile(tonerlines)
p.circle(x=projecteddf['px'],
       y=projecteddf['py'],
       fill_alpha=0.5,
       size=6,
       fill_color=colors,
       line_color=colors)
show(p)
```

# Speed Profile of a Metro Area



# Speed Profile of a Metro Area


SPARK SUMMIT 2017


# Thank You.

geomesa.org
github.com/locationtech/geomesa
twitter.com/algoriffic
linkedin.com/in/anthony-fox-ccri
anthony.fox@ccri.com
www.ccri.com

# Indexing Spatio-Temporal Data in Bigtable

Moscone Center coordinates
**37.7839° N, 122.4012° W**

# Indexing Spatio-Temporal Data in Bigtable

- Bigtable clones have a single dimension lexicographic sorted index

Moscone Center coordinates
**37.7839° N, 122.4012° W**

# Indexing Spatio-Temporal Data in Bigtable

- Bigtable clones have a single dimension lexicographic sorted index
- What if we concatenated latitude and longitude?

Moscone Center coordinates
**37.7839° N, 122.4012° W**

| Row Key |
| --- |
| 37.7839,-122.4012 |

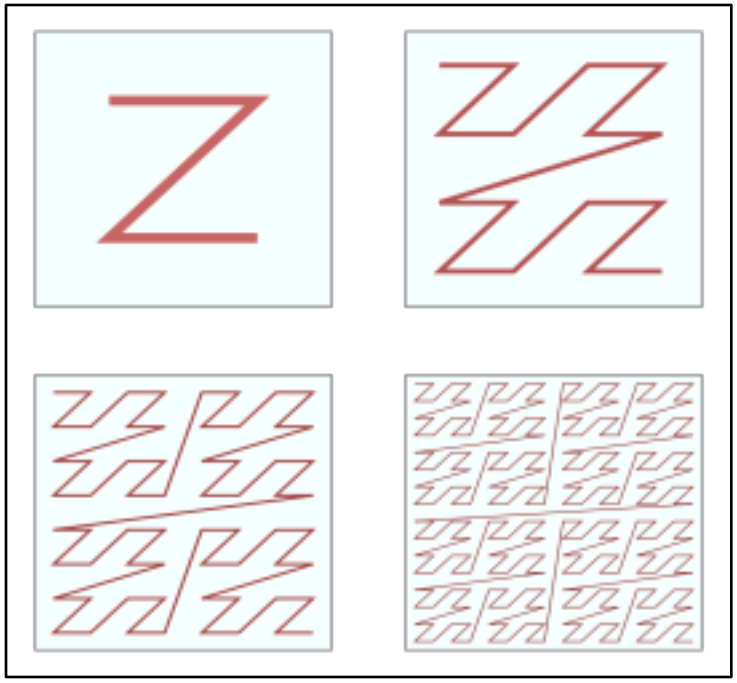
# Indexing Spatio-Temporal Data in Bigtable

- Bigtable clones have a single dimension lexicographic sorted index
- What if we concatenated latitude and longitude?
- Fukushima sorts lexicographically near Moscone Center because they have the same latitude

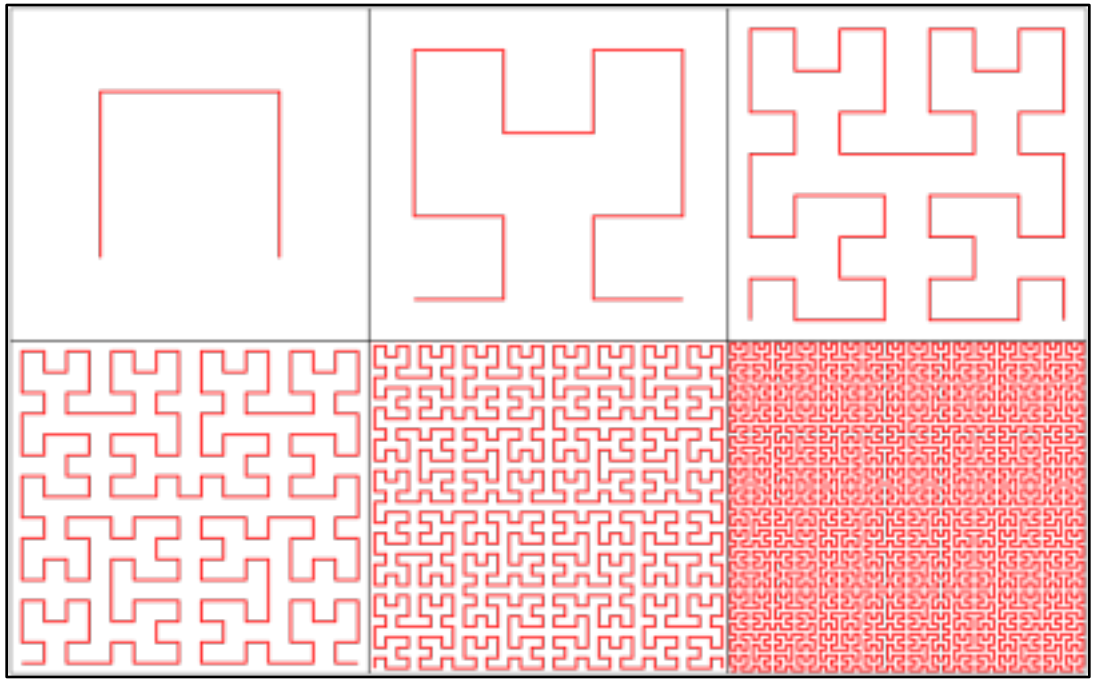
Moscone Center coordinates
**37.7839° N, 122.4012° W**

| Row Key |
| --- |
| 37.7839,-122.4012 |
| 37.7839,140.4676 |

# Space-filling Curves



2-D Z-order Curve



2-D Hilbert Curve

# Space-filling curve example

Encode coordinates to a 32 bit Z

Moscone Center coordinates
**37.7839° N, 122.4012° W**

# Space-filling curve example

Moscone Center coordinates
**37.7839° N, 122.4012° W**

Encode coordinates to a 32 bit Z

1. Scale latitude and longitude to use 16 available bits each
   **scaled_x = (-122.4012 + 180)/360 * 2^16**
   **= 10485**
   **scaled_y = (37.7839 + 90)/180 * 2^16**
   **= 46524**

# Space-filling curve example

Moscone Center coordinates
**37.7839° N, 122.4012° W**

Encode coordinates to a 32 bit Z

1. Scale latitude and longitude to use 16 available bits each
   **scaled_x = (-122.4012 + 180)/360 * 2^16**
   **= 10485**
   **scaled_y = (37.7839 + 90)/180 * 2^16**
   **= 46524**
1. Take binary representation of scaled coordinates
   **bin_x = 0010100011110101**
   **bin_y = 1011010110111100**

# Space-filling curve example

Moscone Center coordinates
**37.7839° N, 122.4012° W**

Encode coordinates to a 32 bit Z

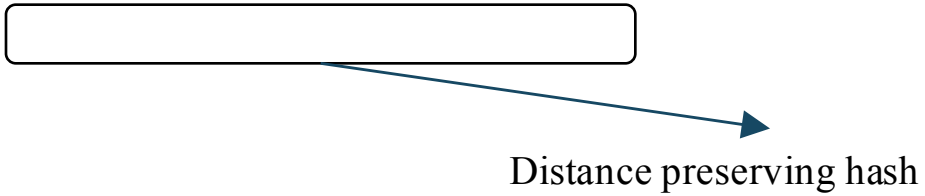
1. Scale latitude and longitude to use 16 available bits each
   **scaled_x = (-122.4012 + 180)/360 * 2^16**
   **= 10485**
   **scaled_y = (37.7839 + 90)/180 * 2^16**
   **= 46524**
1. Take binary representation of scaled coordinates
   **bin_x = 0010100011110101**
   **bin_y = 1011010110111100**
1. Interleave bits of x and y and convert back to an integer
   **bin_z = 01001101100100011110111101110010**
   **z = 1301409650**

# Space-filling curve example

Moscone Center coordinates
**37.7839° N, 122.4012° W**

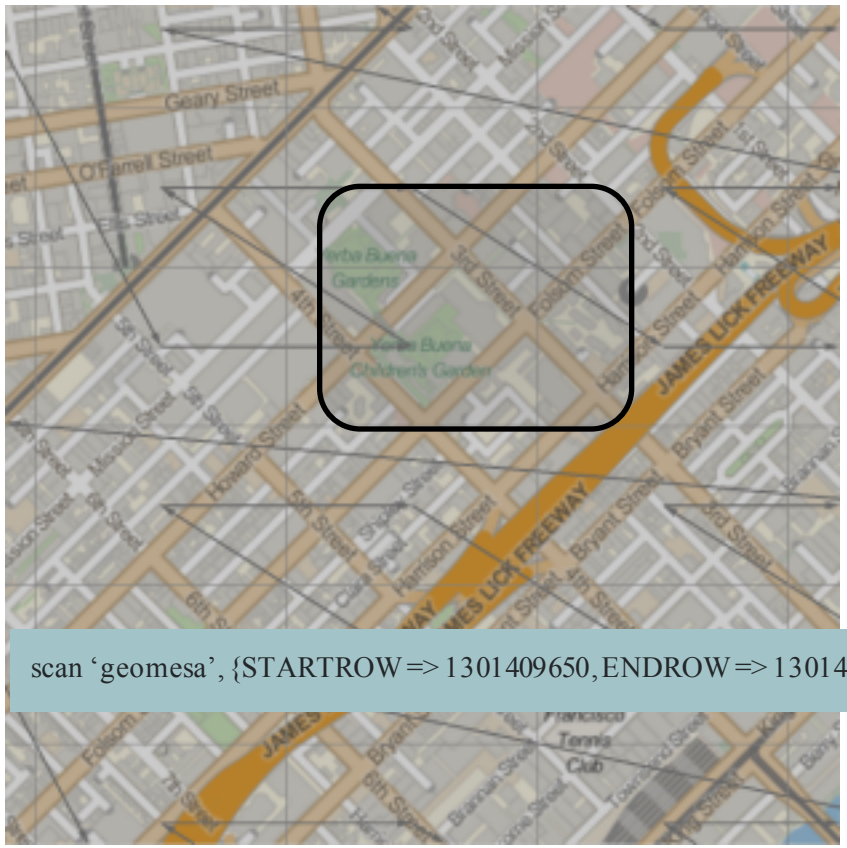
Encode coordinates to a 32 bit Z

1. Scale latitude and longitude to use 16 available bits each
   **scaled_x = (-122.4012 + 180)/360 * 2^16**
   **= 10485**
   **scaled_y = (37.7839 + 90)/180 * 2^16**
   **= 46524**
1. Take binary representation of scaled coordinates
   **bin_x = 0010100011110101**
   **bin_y = 1011010110111100**
1. Interleave bits of x and y and convert back to an integer
   **bin_z = 0100110110010001111011110110010**
   **z = 1301409650**



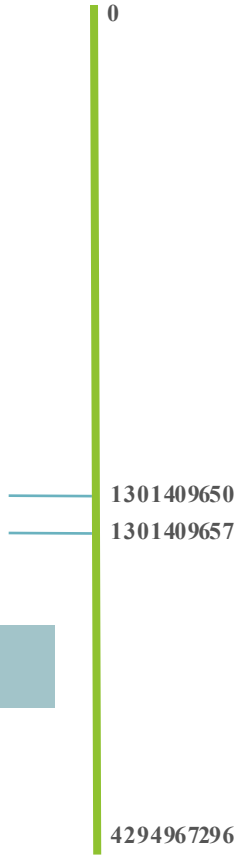
# Space-filling curves linearize a multi-dimensional space

# Regions translate to range scans




scan 'geomesa', {STARTROW => 1301409650, ENDROW => 1301409657}

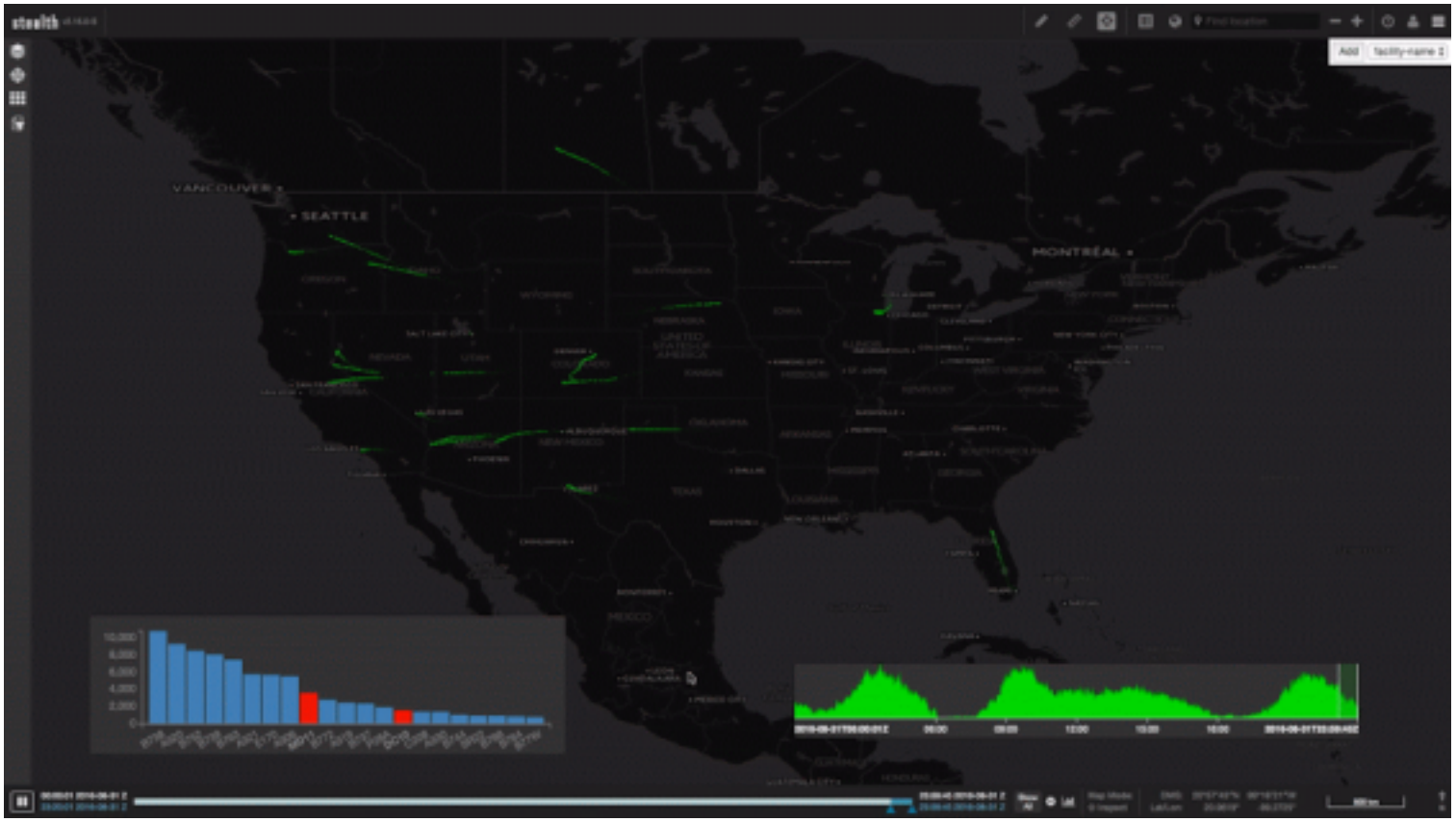
**Bigtable Index**
**[0,2^32]**




SPARK SUMMIT 2017

# ADS-B

FlightAware
Live Flight Tracking




SPARK SUMMIT 2017

# Provisioning Spatial RDDs

# Provisioning Spatial RDDs

Accumulo

```
params = {
    "instanceId": "geomesa",
    "zookeepers": "X.X.X.X",
    "user": "user",
    "password": "******",
    "tableName": "geomesa.strava"
}

spark
    .read
    .format("geomesa")
    .options(**params)
    .option("geomesa.feature", "activities")
    .load()
```

# Provisioning Spatial RDDs

HBase and Bigtable

```
params = {
  "bigtable.table.name": "geomesa.strava"
}

spark
  .read
  .format("geomesa")
  .options(**params)
  .option("geomesa.feature", "activities")
  .load()
```

# Provisioning Spatial RDDs

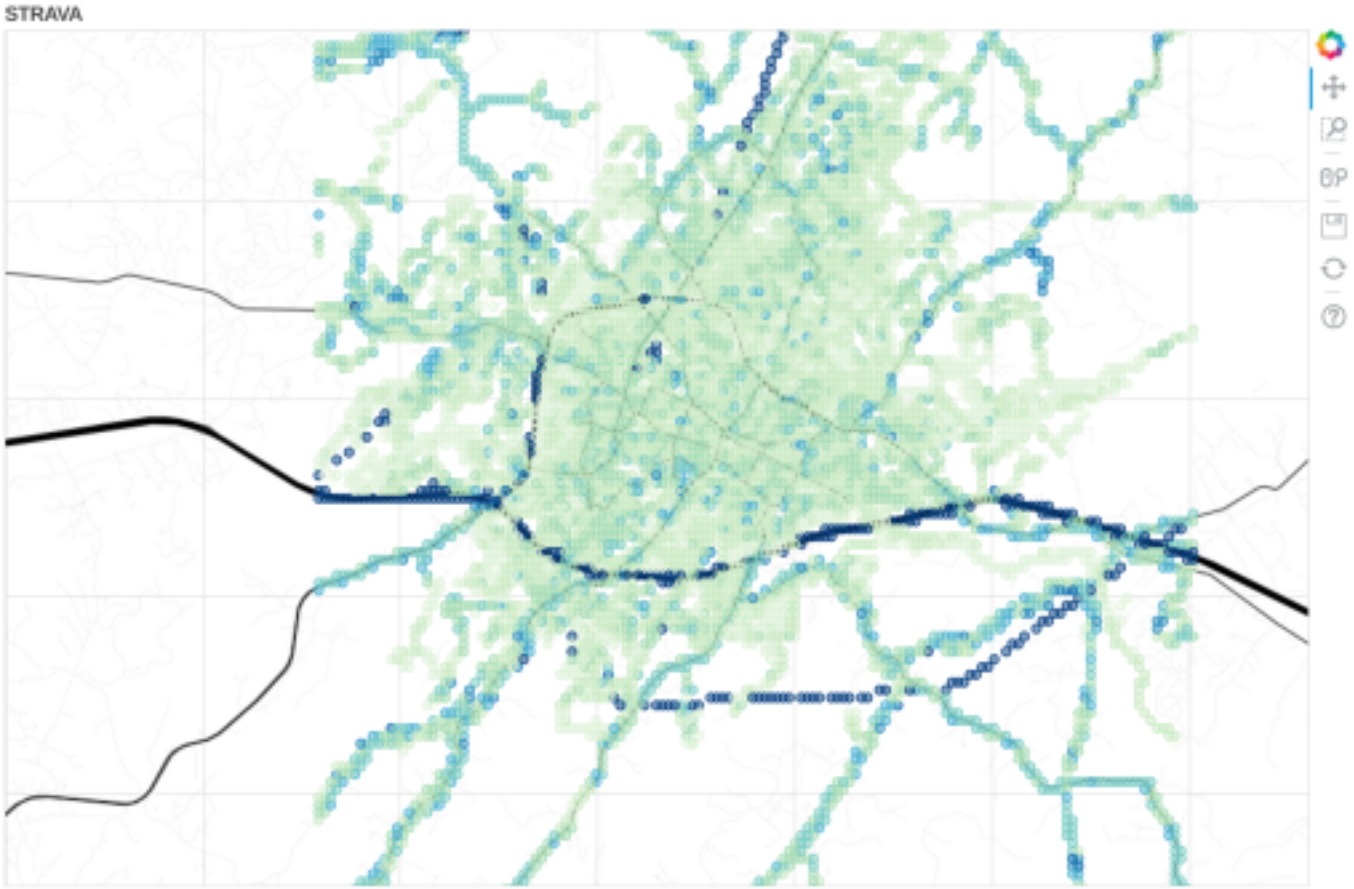
Flat files

```
params = {
    "geomesa.converter": "strava",
    "geomesa.input": "s3://path/to/data/*.json.gz"
}

spark
    .read
    .format("geomesa")
    .options(**params)
    .option("geomesa.feature", "activities")
    .load()
```

# Speed Profile of a Metro Area

# Speed Profile of a Metro Area



# The Dream

# Speed Profile of a Metro Area

Inputs

**Approach**

- Select all activities within metro area
- Sort each activity by dtg ascending
- Window over each set of consecutive samples
- Compute summary statistics of speed
- Group by grid cell
- Visualize