Using GeoMesa on top of Apache Accumulo, HBase, Cassandra, and big data file formats for massive geospatial data

ApacheCon 2019

James Hughes

CCRi

DATA TO KNOWLEDGE

James Hughes

- CCRi's Director of Open Source Programs
- Working in geospatial software on the JVM for the last 7 years
- GeoMesa core committer / product owner
- SFCurve project lead
- JTS committer
- Contributor to GeoTools and GeoServer
- Committer representative to the LocationTech Steering Committee

Talk outline

- Background
  - LocationTech Projects
  - Foundational LocationTech Projects
- GeoMesa
  - Distributed Databases (HBase/Accumulo)
  - Apache File formats
  - Streaming with Kafka
  - Analysis with Spark
- Other LocationTech big-data projects
  - GeoTrellis
  - GeoWave
  - RasterFrames

What is LocationTech?

LocationTech is a working group of the Eclipse Software Foundation

eclipse

Working Groups

LocationTech — Advanced Geospatial Software

iot eclipse.org — Internet of Things

SCIENCE eclipse.org — Scientific Research

LTS ECLIPSE.ORG — Long Term Support

PolarSys — Embedded Systems

CCRi
DATA TO KNOWLEDGE

LocationTech Projects

- **Foundational Libraries**
  - **JTS**
  - **Spatial4j**
  - **SFCurve**
  - Proj4j
  - ImageN
  - libspatialindex
- **Big Data Projects**
  - **GeoMesa**
  - **GeoTrellis**
  - **GeoWave**
  - **RasterFrames**
- Other
  - GeoGig
  - GeoPeril

geomesa

GeoWave

Foundational Libraries

Geometry
Topology
Indexing

- JTS,
- Spatial4J
- SFCurve

LocationTech JTS

The JTS Topology Suite is a Java library for creating and manipulating vector geometry.

Provides implementations of

- OGC Geometry classes
- Basic relationships
- Topology operations

LocationTech JTS License

LocationTech JTS is available under a dual-license:

- Eclipse Public License 1.0
- Eclipse Distribution License 1.0 (BSD style)

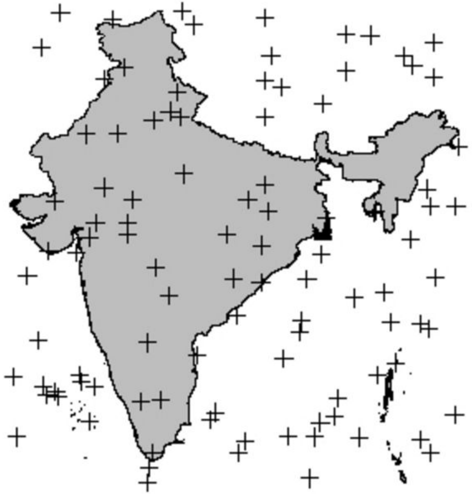
“The Eclipse Foundation makes available all content in this project ("Content"). Unless otherwise indicated below, the Content is provided to you under the terms and conditions of either the Eclipse Public License 1.0 ("EPL") or the Eclipse Distribution License 1.0 (a BSD Style License). For purposes of the EPL, "Program" will mean the Content.”

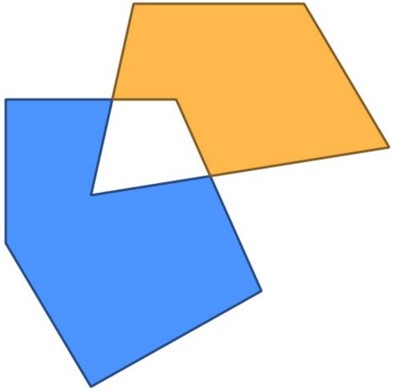
JTS Data Types

Representations:

OGC Simple Features

- Point
- LineString
- LinearRing
- Polygon
- MultiPoint
- MultiLineString
- MultiPolygon
- GeometryCollection





JTS Supported IO Formats

IO:

- WKT
- WKB
- GeoJSON
- KML
- GML2

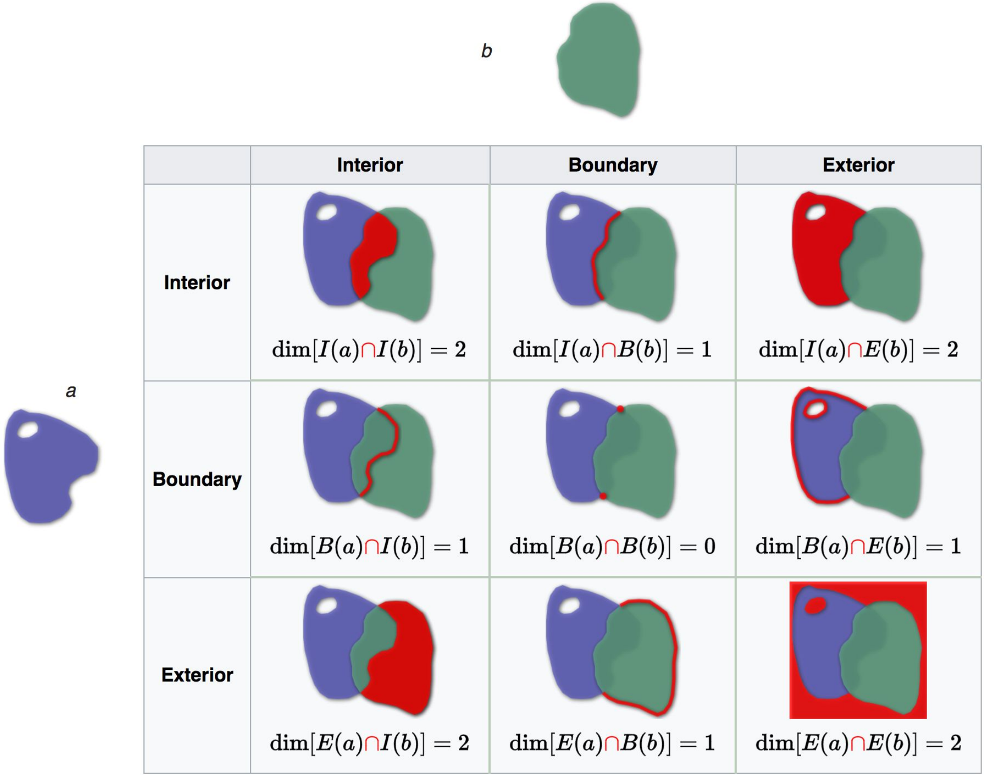
```
wkt_geom
Polygon ((-105.03792611059080286
39.78014782225491786, -105.04818400099962616
39.75856265597848704, -105.02284438556741009
39.75418720873850731, -105.01231287864754904
39.76789982851657612, -105.01364722199988933
39.78389171288461768, -105.03792611059080286
39.78014782225491786))
```

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [
      -122.65335738658904,
      45.512083676585156
    ]
  },
  "properties": {
    "name": "Hungry Heart Cupcakes",
    "address": "1212 SE Hawthorne Boulevard",
    "website": "http://www.hungryheartcupcakes.com",
    "gluten free": "no"
  }
}
```

JTS Topological Predicate Support

Predicates (DE-9IM)

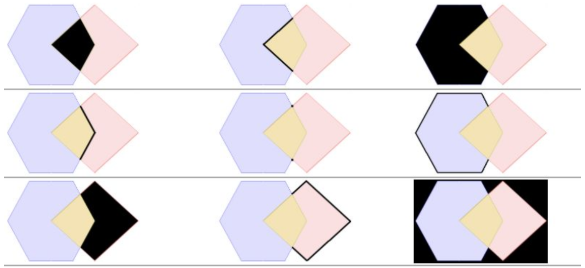
- Equals
- Disjoin
- Intersects
- Touches
- Crosses
- Within
- Contains
- Overlaps
- Covers
- CoveredBy

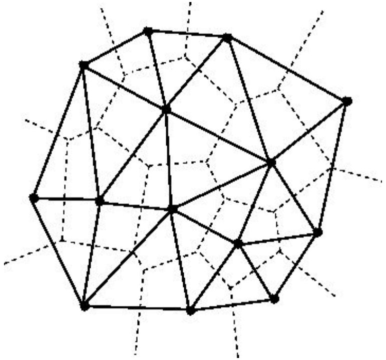


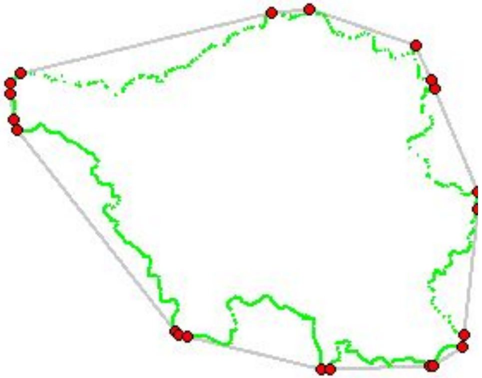
JTS Topology Operations

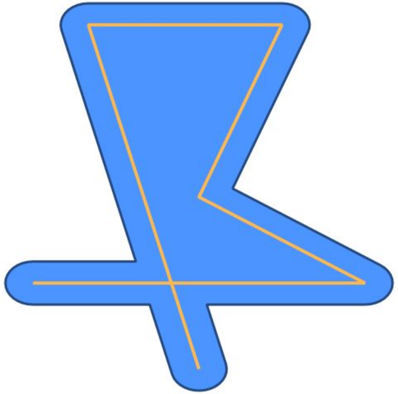
Algorithms

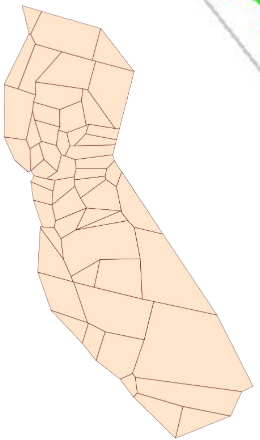
- Convex Hull
- Buffer
- Validation
- Dissolve
- Overlay operations
- Polygonization
- Simplification
- Triangulation
- Voronoi
- Linear Referencing
- and more...











JTS is EVERYWHERE

Net Topology Suite

JTS

GEOS

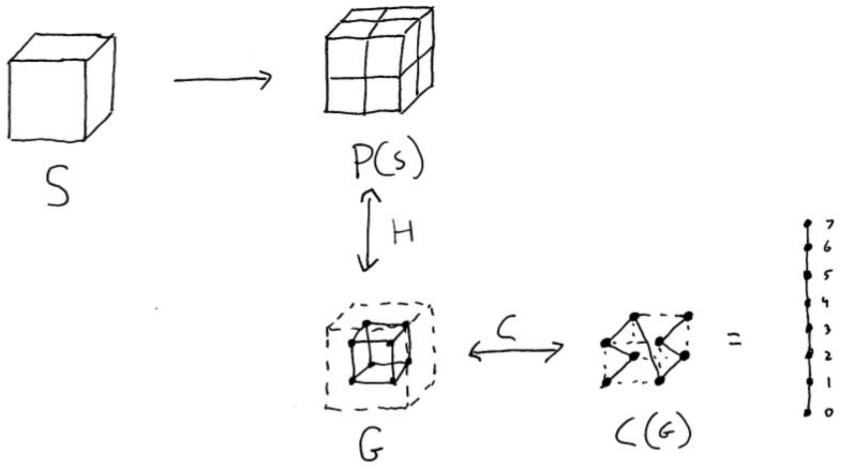
JSTS

Spatial4j

- Spatial4j is a general purpose spatial / geospatial ASL licensed open-source Java library. Capabilities:
  - provide common shapes that can work in Euclidean and geodesic world models
  - provide distance calculations and other math
  - read & write shapes from formats like WKT and GeoJSON
- Came out of need to add spatial indexing to Apache Lucene

SFCurve

- This library represents a collaborative attempt to create a solid, robust and modular library for dealing with space filling curves on the JVM.
- Implementations of space-filling curves:
  - Hilbert (2D)
  - Z-order (2D, 3D, nD)



LocationTech GeoMesa

- GeoMesa Overview
- Reference Architecture
- Live Demo

What is GeoMesa?

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

What is GeoMesa?

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

What is GeoMesa?

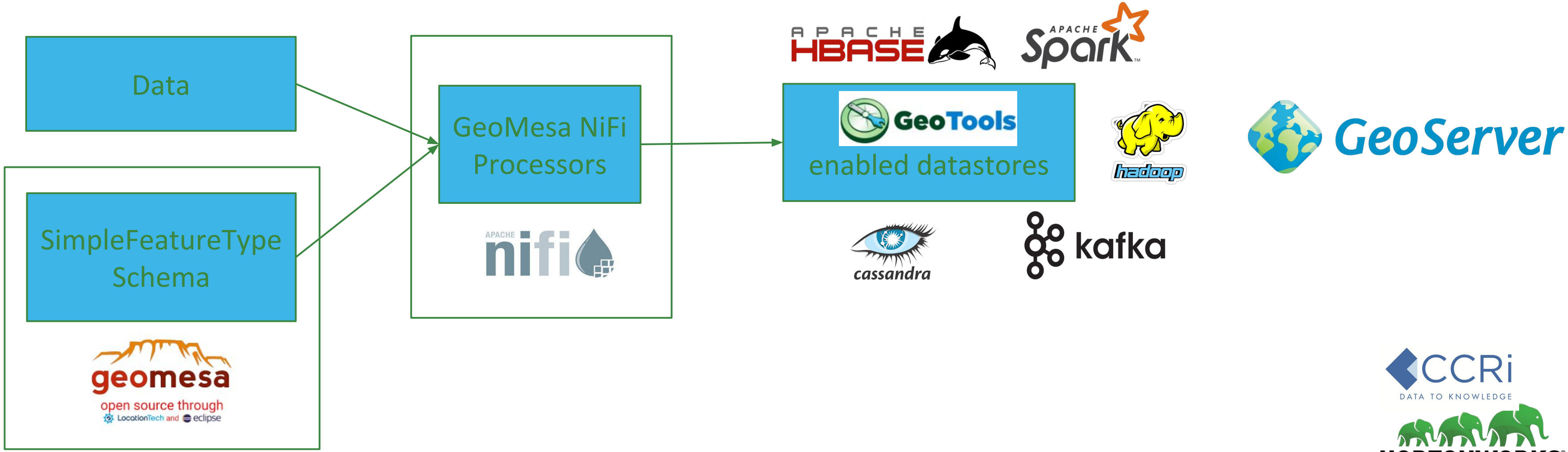
A suite of tools for streaming, **persisting**, managing, and analyzing spatio-temporal data at scale

What is GeoMesa?

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

GeoMesa NiFi

- GeoMesa-NiFi allows you to ingest data into GeoMesa straight from NiFi by leveraging custom processors.
- NiFi allows you to ingest data into GeoMesa from every source GeoMesa supports and more.



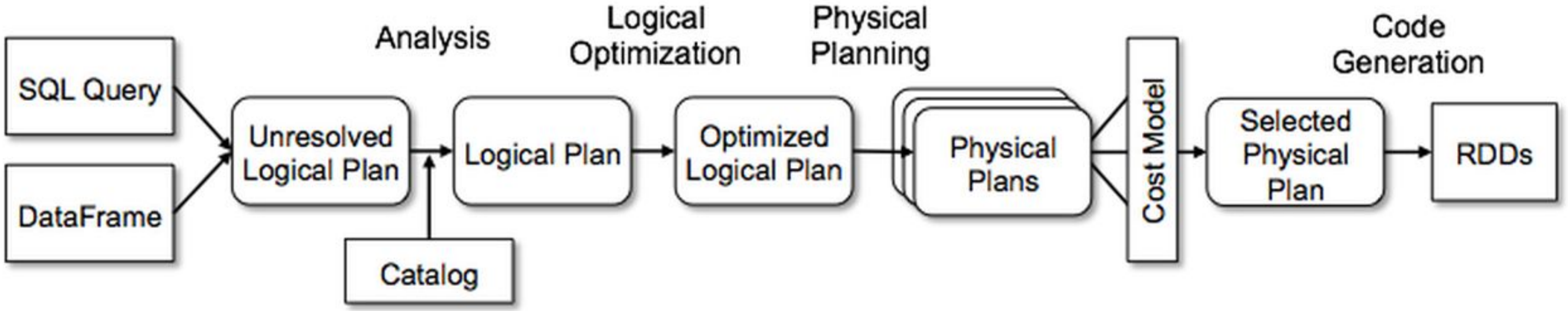
CCRi DATA TO KNOWLEDGE

HORTONWORKS

What is GeoMesa?

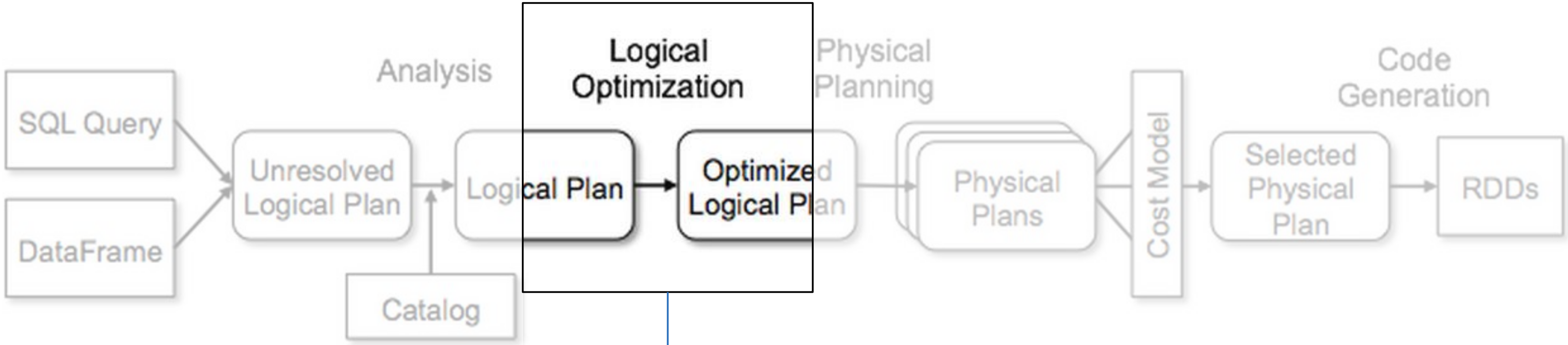
A suite of tools for streaming, persisting, managing, and **analyzing** spatio-temporal data at scale

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.

CCRi

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)
```

CCRi

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)
```

GeoMesa Relation

CCRi

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)
```

Topological Predicate

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)
```

Geometry Literal

CCRi

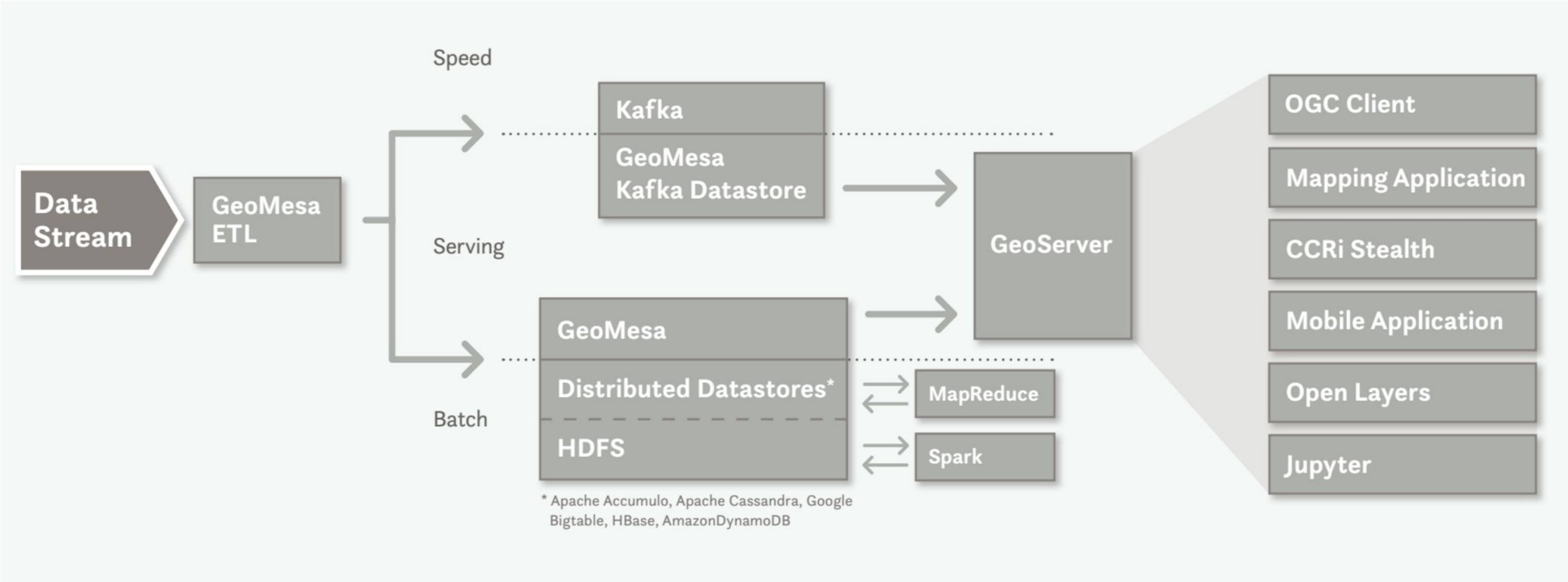
SQL optimizations for Spatial Predicates

```
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)
```

Date range predicate

CCRi

Proposed Reference Architecture



Live Demo!

- Filtering by spatio-temporal constraints
- Filtering by attributes
- Aggregations
- Transformations

GeoMesa's Persistence

- Distributed Databases
  - Accumulo, HBase, Cassandra
- File formats
  - Arrow, Avro, Orc, Parquet

Indexing Geospatial Data In Key-Value Stores

Accumulo / HBase / Cassandra

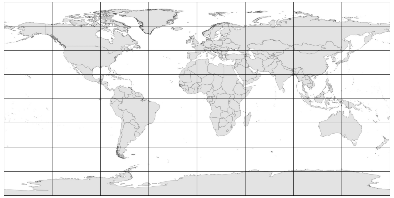
- Key Design using Space Filling Curves

Space Filling Curves (in one slide!)

- **Goal: Index 2+ dimensional data**
- **Approach: Use Space Filling Curves**

Space Filling Curves (in one slide!)

- **Goal: Index 2+ dimensional data**
- **Approach: Use Space Filling Curves**
- First, ‘grid’ the data space into bins.



Space Filling Curves (in one slide!)

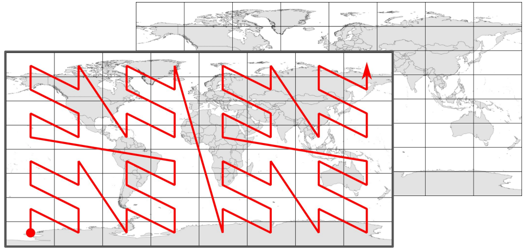
- **Goal: Index 2+ dimensional data**
- **Approach: Use Space Filling Curves**
- First, ‘grid’ the data space into bins.
- Next, order the grid cells with a space filling curve.
    - Label the grid cells by the order that the curve visits the them.
    - Associate the data in that grid cell with a byte representation of the label.

Space Filling Curves (in one slide!)

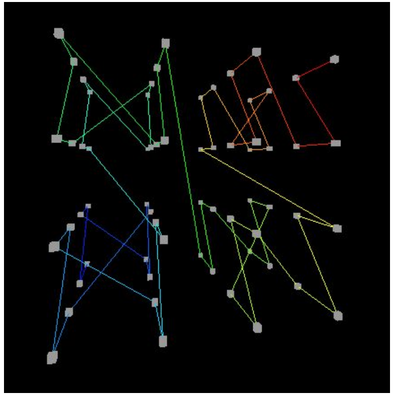
- **Goal: Index 2+ dimensional data**
- **Approach: Use Space Filling Curves**
- First, ‘grid’ the data space into bins.
- Next, order the grid cells with a space filling curve.
    - Label the grid cells by the order that the curve visits the them.
    - Associate the data in that grid cell with a byte representation of the label.
- We prefer “good” space filling curves:
    - Want recursive curves and locality.

Space Filling Curves (in one slide!)

- **Goal: Index 2+ dimensional data**
- **Approach: Use Space Filling Curves**
- First, ‘grid’ the data space into bins.
- Next, order the grid cells with a space filling curve.
    - Label the grid cells by the order that the curve visits the them.
    - Associate the data in that grid cell with a byte representation of the label.
- We prefer “good” space filling curves:
    - Want recursive curves and locality.
- Space filling curves have higher dimensional analogs.



Z2 "GeoHash"



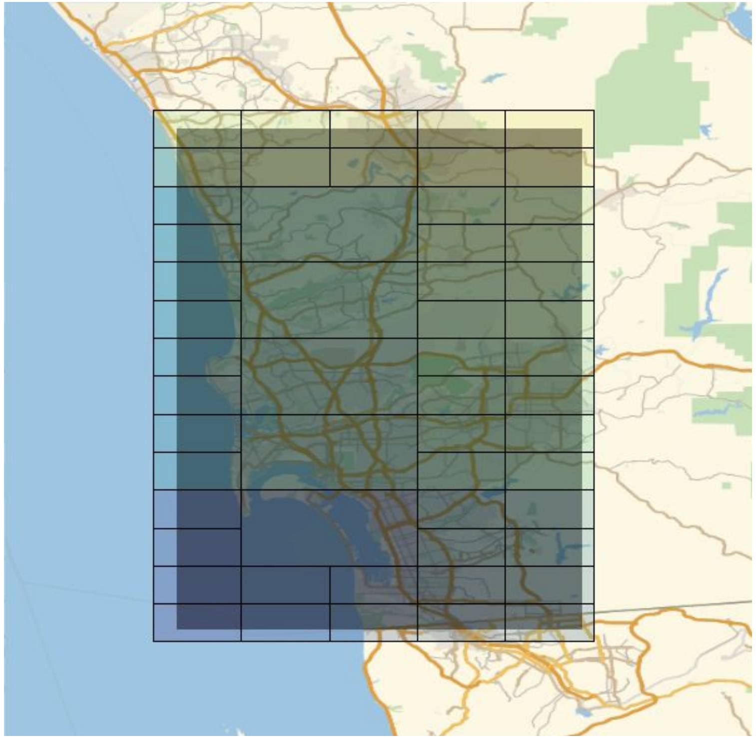
Z3

Query planning with Space Filling Curves

To query for points in the grey rectangle, the query planner enumerates a collection of index ranges which cover the area.

Note: Most queries won't line up perfectly with the gridding strategy.

Further filtering can be run on the tablet/region servers (next section)
or we can return 'loose' bounding box results (likely more quickly).



Server-Side Optimizations

Filtering and transforming records

- Pushing down data filters
  - Z2/Z3 filter
  - CQL Filters
- Projections

Filtering and transforming records overview

Using Accumulo iterators and HBase filters, it is possible to ***filter*** and ***map*** over the key-values pairs scanned.

This will let us apply fine-grained spatial filtering, filter by secondary predicates, and implement projections.

Pushing down filters

Let's consider a query for **tankers** which are inside a bounding box for a given time period.

GeoMesa's Z3 index is designed to provide a set of **key ranges** to scan which will cover the spatio-temporal range.

Additional information such as the **vessel type** is part of the value.

Using server-side programming, we can teach Accumulo and HBase how to understand the records and filter out undesirable records.

This **reduces network traffic** and **distributes** the work.

Projection

To handle projections in a query, Accumulo Iterators and HBase Filters can change the returned key-value pairs.

Changing the key is a bad idea.

Changing the value allows for GeoMesa to return a subset of the columns that a user is requesting.

GeoMesa Server-Side Filters

- Z2/Z3 filter
  - Scan ranges are not decomposed enough to be very accurate - fast bit-wise comparisons on the row key to filter out-of-bounds data
- CQL/Transform filter
  - If a predicate is not handled by the scan ranges or Z filters,
    then slower GeoTools CQL filters are applied to the serialized SimpleFeature in the row value
  - Relational projections (transforms) applied to reduce the amount of data sent back
- Other specialized filters
  - Age-off for expiring rows based on a SimpleFeature attribute
  - Attribute-key-value for populating a partial SimpleFeature with an attribute value from the row
  - Visibility filter for merging columns into a SimpleFeature when using attribute-level visibilities

Server-Side Optimizations

Aggregations

- Generating heatmaps
- Descriptive Stats
- Arrow format

Aggregations

Using Accumulo Iterators and HBase coprocessors, it is possible to aggregate multiple key-value pairs into one response. Effectively, this lets one implement ***map*** and ***reduce*** algorithms.

These aggregations include computing ***heatmaps***, ***stats***, and ***custom data formats***.

The ability to aggregate data can be composed with filtering and projections.

GeoMesa Aggregation Abstractions

Aggregation logic is implemented in a shared module, based on a lifecycle of

1. **Initialization**
2. **observing** some number of features
3. **aggregating** a result.

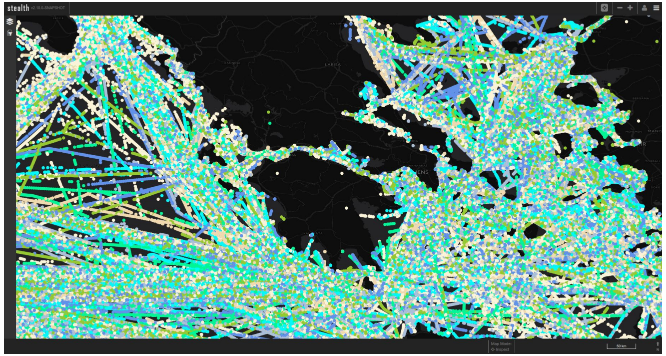
This paradigm is easily adapted to the specific implementations required by Accumulo and HBase.

Notably, all the algorithms we describe work in a single pass over the data.

Visualization Example: Heatmaps

Without powerful visualization options, big data is big nonsense.

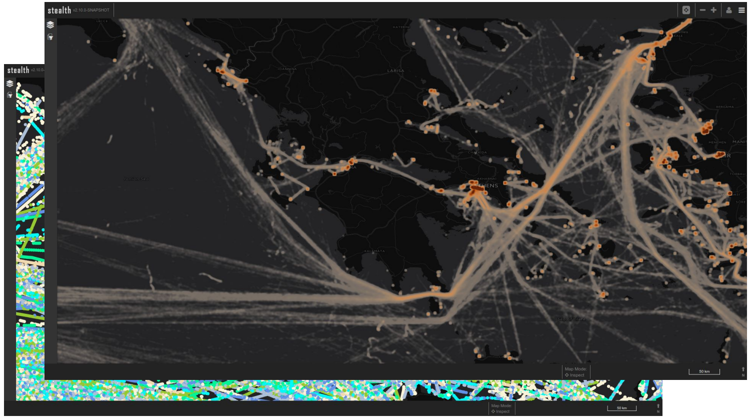
Consider this view of shipping in the Mediterranean sea



Visualization Example: Heatmaps

Without powerful visualization options, big data is big nonsense.

Consider this view of shipping in the Mediterranean sea



Generating Heatmaps

Heatmaps are implemented in DensityScan.

For the scan, we set up a 2D grid array representing the pixels to be displayed. On the region/tablet servers, each feature increments the count of any cells intersecting its geometry. The resulting grid is returned as a serialized array of 64-bit integers, minimizing the data transfer back to the client.

The client process merges the grids from each scan range, then normalizes the data to produce an image.

Since less data is transmitted, **heatmaps are generally faster**.

Statistical Queries

We support a flexible stats API that includes **counts**, **min/max** values, **enumerations**, **top-k** (StreamSummary), **frequency** (CountMinSketch), **histograms** and **descriptive statistics**. We use well-known streaming algorithms backed by data structures that can be serialized and merged together.

Statistical queries are implemented in StatsScan.

On the region/tablet servers, we set up the data structure and then add each feature as we scan. The client receives the serialized stats, merges them together, and displays them as either JSON or a Stat instance that can be accessed programmatically.

Arrow Format

Apache Arrow is a columnar, in-memory data format that GeoMesa supports as an output type. In particular, it can be used to drive complex in-browser visualizations. Arrow scans are implemented in ArrowScan.

With Arrow, the data returned from the region/tablet servers is similar in size to a normal query. However, the processing required to generate Arrow files can be distributed across the cluster instead of being done in the client.

As we scan, each feature is added to an in-memory Arrow vector. When we hit the configured batch size, the current vector is serialized into the Arrow IPC format and sent back to the client. All the client needs to do is to create a header and then concatenate the batches into a single response.

Optimizing Big Data Formats for Vector Data

- File formats overview
- Spatial extensions to file formats

Specialized Big data file formats

Parquet

AVRO™

Apache orc™

ARROW

Benefits of big data file formats

- Columnar layouts
- Dictionary encoding
- Efficient compression
- Structured
- Optimized filtering on read
- Language interoperability

Benefits of big data file formats

- Columnar layouts
- Dictionary encoding
- Efficient compression
- Structured
- Optimized filtering on read
- Language interoperability

One problem!

- No spatial types!

Row vs Columnar Layouts

- Row layout
  - All the data for a single **record** is contiguous
  - Easier to write and stream

- Columnar layout
  - All the data for a single **column** is contiguous
  - Can be compressed much more efficiently
  - Requires much less I/O for filtering and projections

Row vs Columnar Layouts

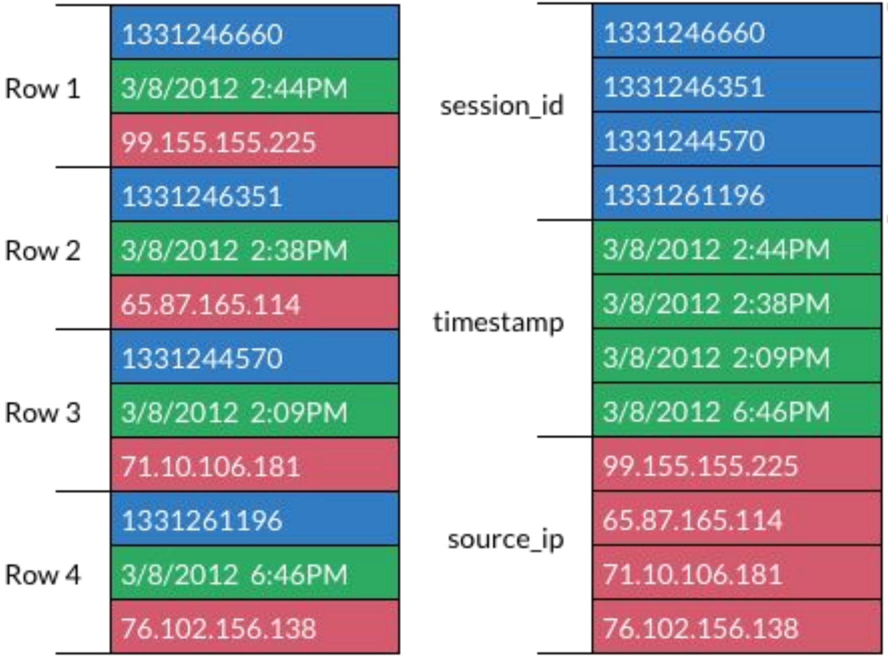
	session_id	timestamp	source_ip
Row 1	1331246660	3/8/2012 2:44PM	99.155.155.225
Row 2	1331246351	3/8/2012 2:38PM	65.87.165.114
Row 3	1331244570	3/8/2012 2:09PM	71.10.106.181
Row 4	1331261196	3/8/2012 6:46PM	76.102.156.138

Source: Apache Arrow

Row vs Columnar Layouts

CCRi DATA TO KNOWLEDGE

	session_id	timestamp	source_ip
Row 1	1331246660	3/8/2012 2:44PM	99.155.155.225
Row 2	1331246351	3/8/2012 2:38PM	65.87.165.114
Row 3	1331244570	3/8/2012 2:09PM	71.10.106.181
Row 4	1331261196	3/8/2012 6:46PM	76.102.156.138



Source: Apache Arrow

Apache Avro

- Row-based layout
- Schemas
  - Embedded (file format) or centralized (message format)
  - Supports versioning and evolution
- Optimal for streaming data (i.e. Apache Kafka), as each message is self-contained

Apache Parquet

- Column-based layout
- Optimized for Hadoop/Spark
- Schema is embedded in the file
- Per-column compression
- Push-down predicates during read
- Column chunking allows skipping I/O

Apache Orc

- Column-based layout
- Optimized for Hadoop/Hive
- Optimized for streaming reads
- Per-column compression
- File-level indices
- Push-down predicates during read
- Column stripes provide parallelism

Apache Arrow

- Column-based layout
- Optimized for in-memory use
- IPC file format
- Dictionary encoding
- Zero-copy reads

Spatial File Formats in GeoMesa

- No native spatial types
- Geometries are built up with lists of primitive columns
- Similar to GeoJSON, can be read without special type awareness

Spatial File Formats in GeoMesa

- Points
  - Stored as two columns of type Double, one for X and one for Y
  - Arrow - stored as tuples (FixedSizeList)
- Allows for push-down filtering against each dimension

Spatial File Formats in GeoMesa

- Points
  - Stored as two columns of type Double, one for X and one for Y
  - Arrow - stored as tuples (FixedSizeList)
- Allows for push-down filtering against each dimension
- LineStrings, MultiPoints
  - Stored as two columns of type List[Double]

Spatial File Formats in GeoMesa

- Points
  - Stored as two columns of type Double, one for X and one for Y
  - Arrow - stored as tuples (FixedSizeList)
- Allows for push-down filtering against each dimension
- LineStrings, MultiPoints
  - Stored as two columns of type List[Double]
- MultiLineStrings, Polygons
  - Stored as two double precision List[List[Double]] columns
- MultiPolygons
  - Stored as two double precision List[List[List[Double]]] columns

Spatial File Formats in GeoMesa

- Points
  - Stored as two columns of type Double, one for X and one for Y
  - Arrow - stored as tuples (FixedSizeList)
- Allows for push-down filtering against each dimension
- LineStrings, MultiPoints
  - Stored as two columns of type List[Double]
- MultiLineStrings, Polygons
  - Stored as two double precision List[List[Double]] columns
- MultiPolygons
  - Stored as two double precision List[List[List[Double]]] columns
- Avro - row based - WKB/TWKB/WKT

Reading and Writing Spatial Formats

- Parquet
  - geomesa-fs-storage-parquet - SimpleFeatureParquetWriter, FilteringReader
- Orc
  - geomesa-fs-storage-orc - OrcFileSystemReader/Writer
- Arrow
  - geomesa-arrow-jts - PointVector, LineStringVector, etc
- Avro
  - geomesa-feature-avro - AvroFeatureSerializer, AvroDataFileReader/Writer

Reading and Writing Spatial Formats

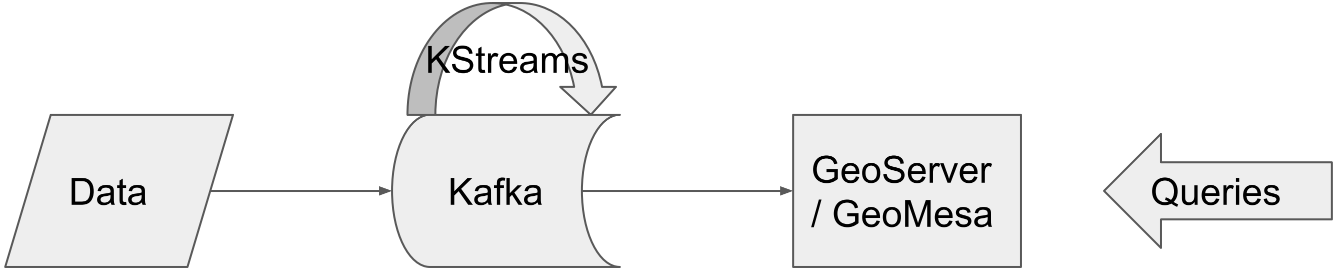
- Parquet/Orc
  - GeoMesa file system data store
  - GeoMesa CLI export/ingest
- Arrow
  - WFS/WPS requests through GeoServer
  - GeoMesa CLI export
- Avro
  - WFS/WPS requests through GeoServer
  - GeoMesa CLI export/ingest
- Standard format tools

Spatial File Format Use Cases

- Streaming Data
- Spark analytics
- ETL and tiered storage
- Data visualization

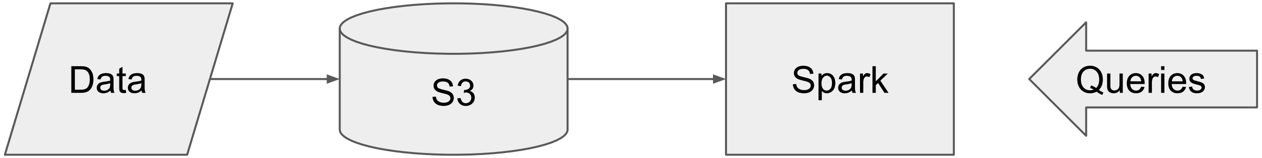
Streaming Data - Apache Avro

- Each message is a single record (row based)
- Apache Kafka/Streams for data exchange
- Confluent schema registry is used for managing schemas
  - Small header per message uniquely identifies schema
  - Schema evolution for adding/removing fields
- GeoMesa Kafka data store for in-memory indexing



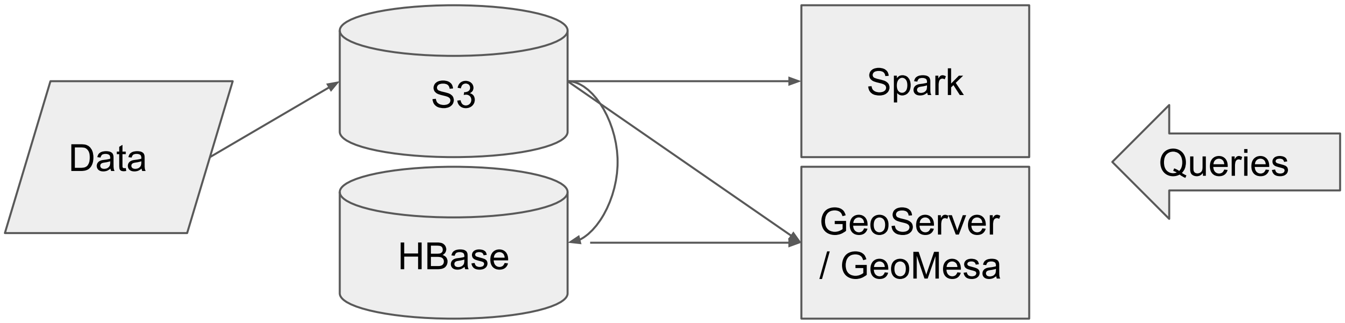
Spark Analytics - Apache Parquet and Orc

- GeoMesa Spark integration adds spatial UDFs/UDTs
  - st_contains, st_point, etc
- Native input formats provide high throughput
- Relational projections take advantage of columnar layouts
- Predicates are pushed down into the file reads



Tiered Storage and ETL - Apache Parquet and Orc

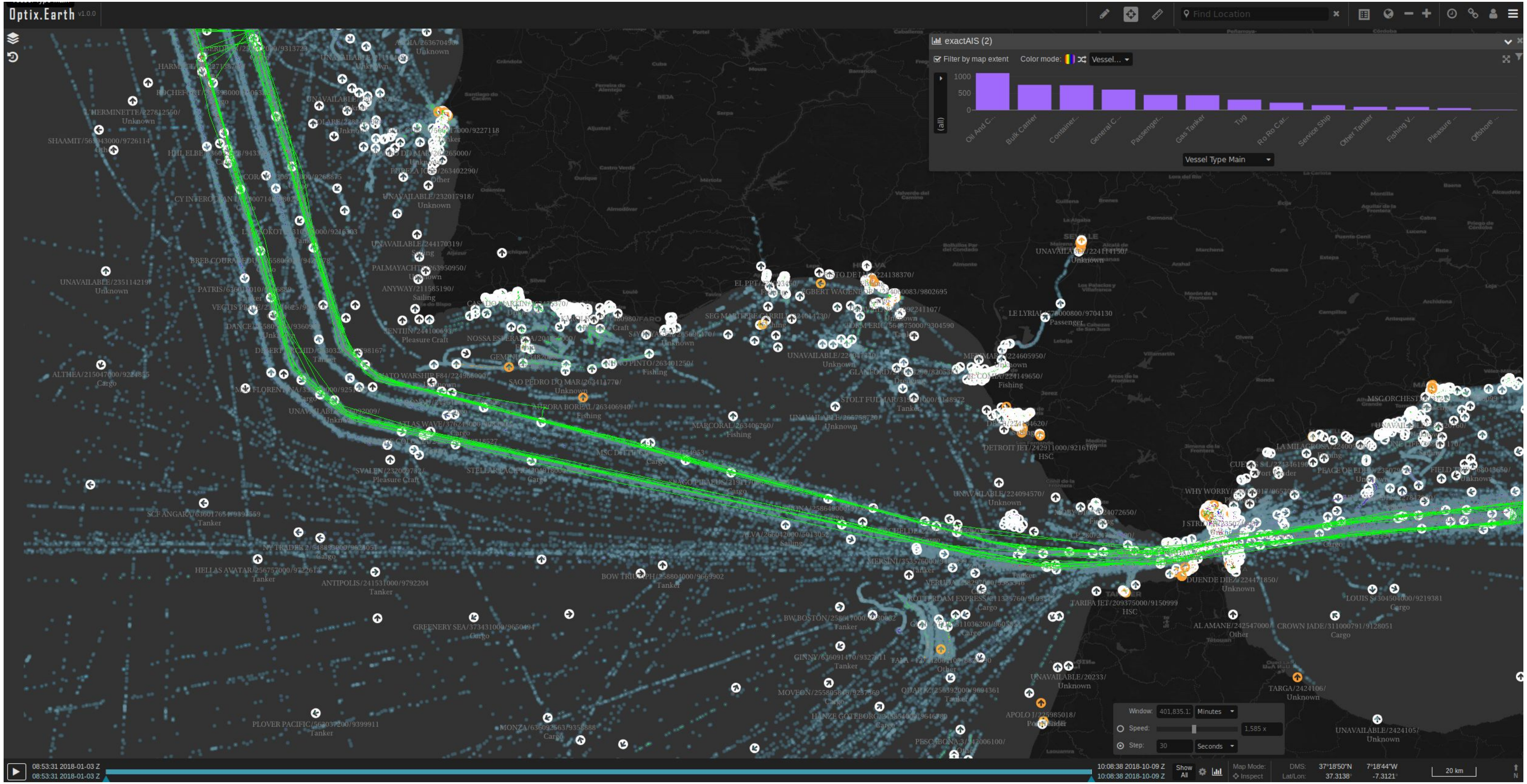
- Data is pre-processed into S3 using the GeoMesa converter library to create Parquet or Orc
- Processed files are ingested directly from S3 into HBase
- Processed files are accessed with the GeoMesa file system data store for large-scale analytics
- Data age-off is used to keep your HBase cluster small
- Merged view data store shows combined HBase + S3



Data Visualization - Apache Arrow

- Query Arrow IPC data through WFS/WPS
  - Distributed aggregation used where possible
- Arrow-js wraps the raw bytes and exposes the underlying data
- Can efficiently filter, sort, count, etc to display maps, histograms, timelapses

Data Visualization in browser with Apache Arrow



Big-Data LocationTech Projects

- GeoMesa
- GeoTrellis
- GeoWave
- RasterFrames

GeoWave

What is GeoWave?

An open source framework that leverages the scalability of **key-value stores** for effective storage, retrieval, and analysis of massive **geospatial datasets**

At its core, GeoWave handles spatial and spatiotemporal indexing within distributed key-value stores with natural integrations for various popular frameworks

GeoWave bridges the gap between popular geospatial platforms *and* distributed processing frameworks

GeoTrellis

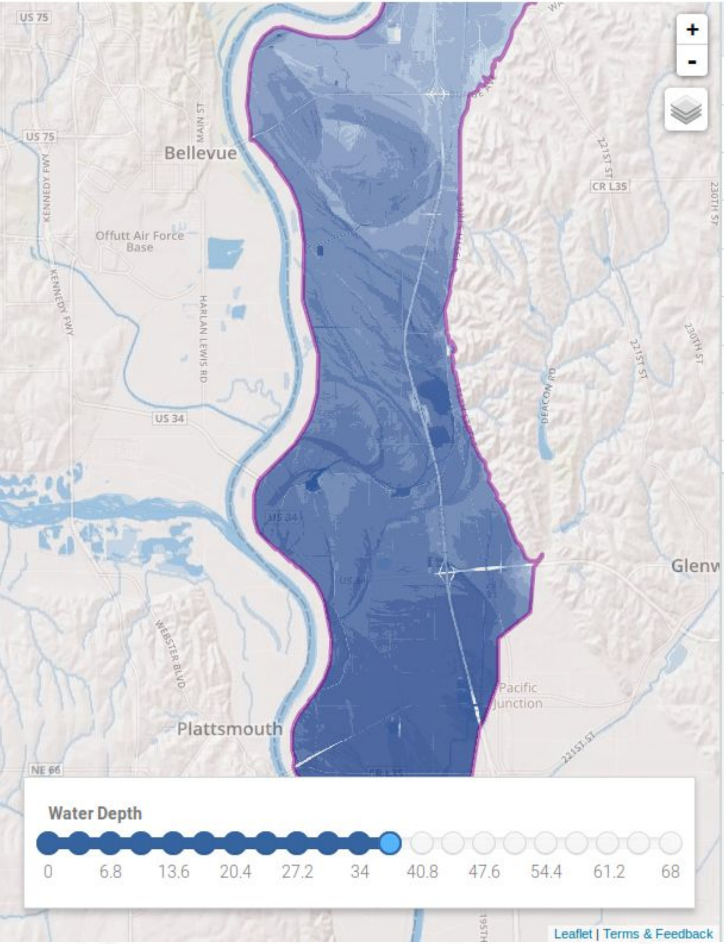
GeoTrellis

- a Scala library for geospatial data types and operations.
- enables Spark with geospatial capabilities
- Build tile servers for on-the-fly transformations of COGs
- storage and query raster from HDFS, Accumulo, Cassandra and S3

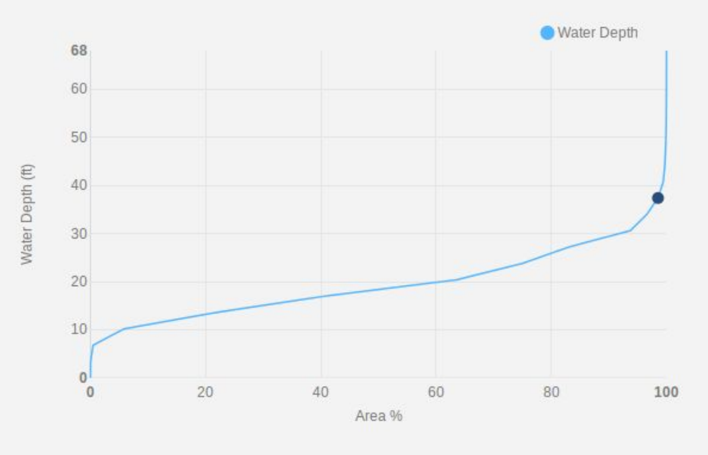
USACE WISDM Depth - Area Analysis

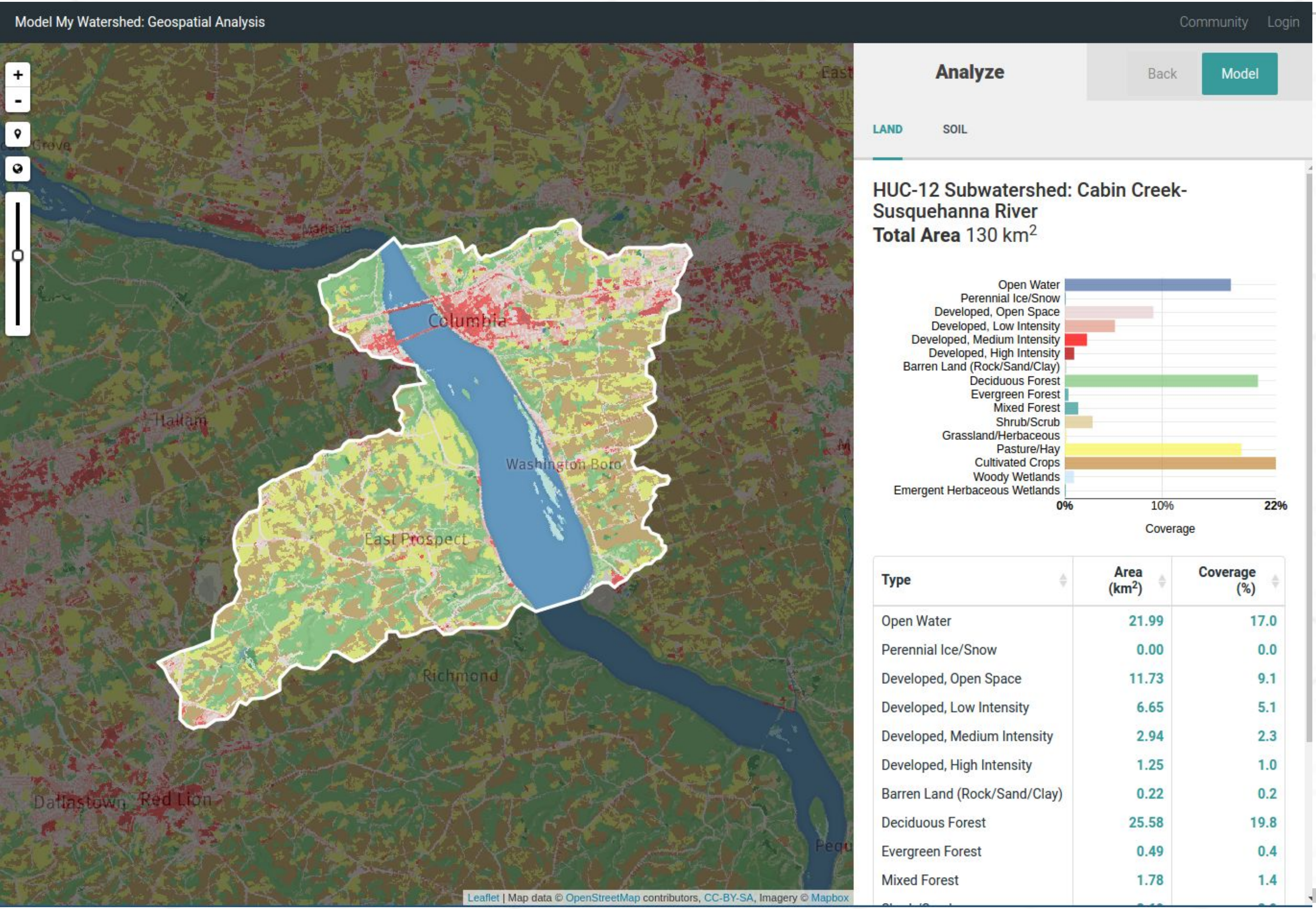
Map | Split | Data

Reset 23,685 acres processed in 7.4 seconds.

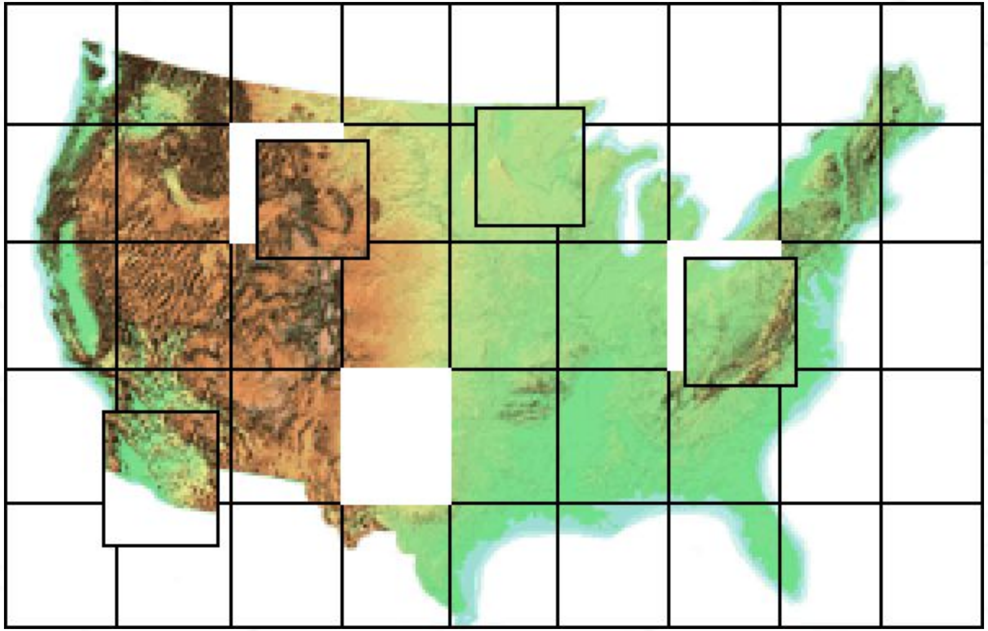


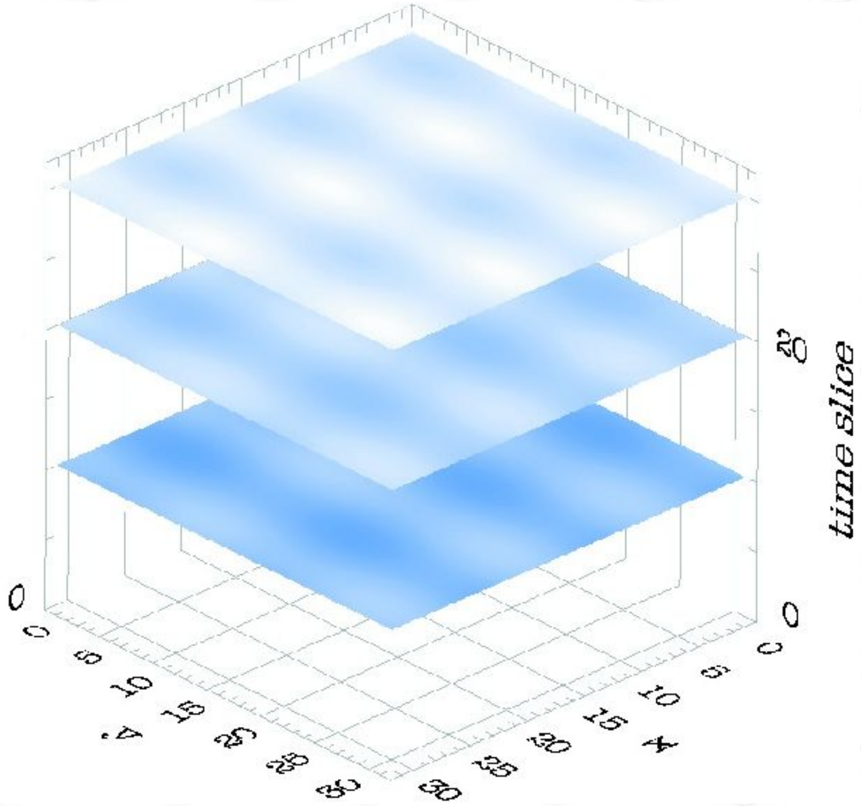
Water Depth	Area %	Area acres	Structures Impacted
0.0 feet	0.00%	0 acres	0
3.4 feet	0.02%	4 acres	0
6.8 feet	0.45%	106 acres	2
10.2 feet	5.80%	1,373 acres	5
13.6 feet	21.76%	5,154 acres	23
17.0 feet	40.72%	9,644 acres	49
20.4 feet	63.58%	15,059 acres	76
23.8 feet	75.00%	17,764 acres	100
27.2 feet	83.05%	19,671 acres	150



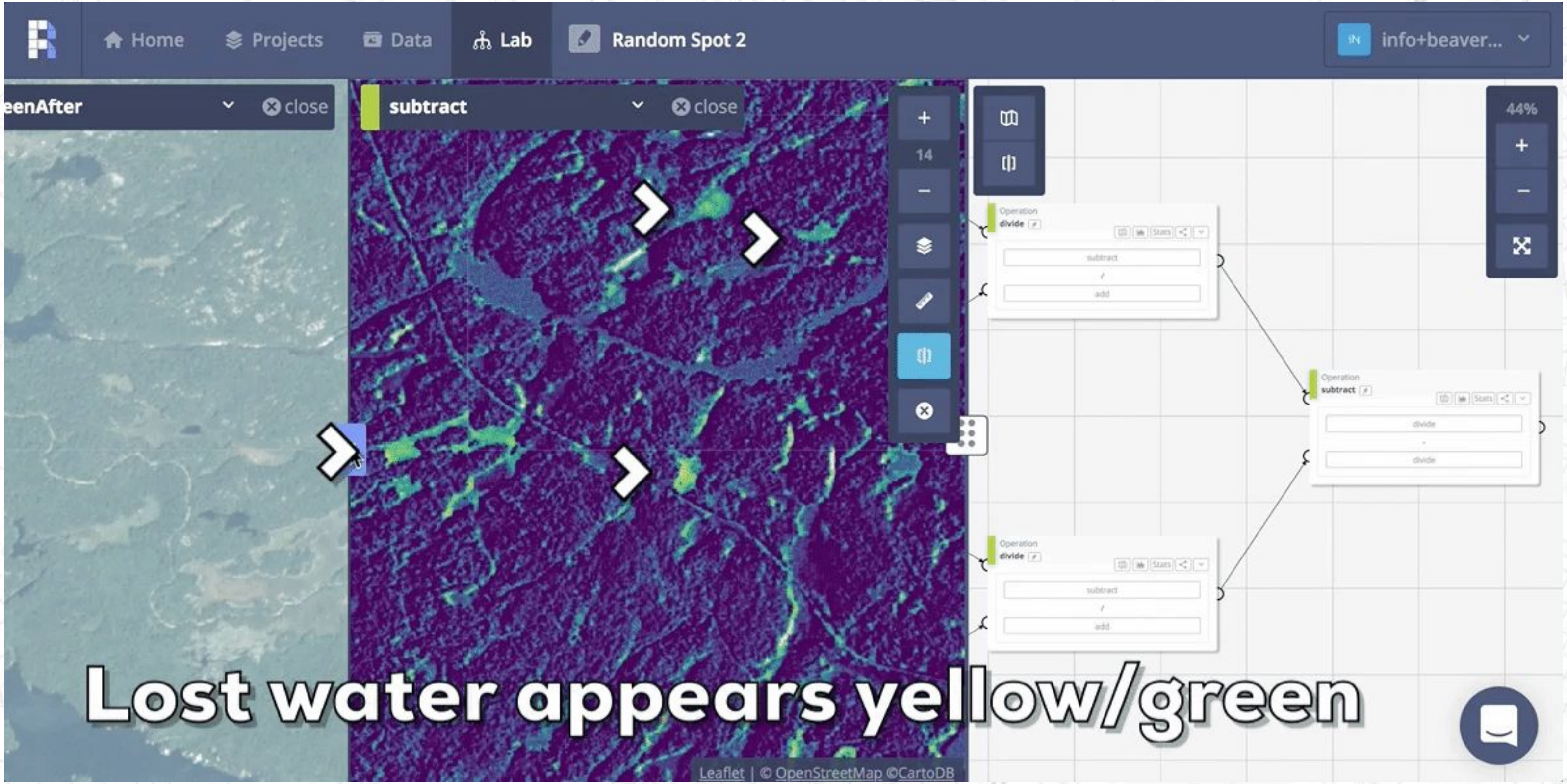


Model My Watershed: Geospatial Analysis
Community
Login
Columbia
Washington Boro
East Prospect
Richmond
Hallam
Marietta
Dallastown
Red Lion
Leaflet | Map data © OpenStreetMap contributors, CC-BY-SA, Imagery © Mapbox
Analyze
Back
Model
LAND
SOIL
HUC-12 Subwatershed: Cabin Creek-Susquehanna River
Total Area 130 km2
Open Water
Perennial Ice/Snow
Developed, Open Space
Developed, Low Intensity
Developed, Medium Intensity
Developed, High Intensity
Barren Land (Rock/Sand/Clay)
Deciduous Forest
Evergreen Forest
Mixed Forest
Shrub/Scrub
Grassland/Herbaceous
Pasture/Hay
Cultivated Crops
Woody Wetlands
Emergent Herbaceous Wetlands
0%
10%
22%
Coverage
Type | Area (km2) | Coverage (%)
Open Water | 21.99 | 17.0
Perennial Ice/Snow | 0.00 | 0.0
Developed, Open Space | 11.73 | 9.1
Developed, Low Intensity | 6.65 | 5.1
Developed, Medium Intensity | 2.94 | 2.3
Developed, High Intensity | 1.25 | 1.0
Barren Land (Rock/Sand/Clay) | 0.22 | 0.2
Deciduous Forest | 25.58 | 19.8
Evergreen Forest | 0.49 | 0.4
Mixed Forest | 1.78 | 1.4





time slice
0
20
0
y
0
5
10
15
20
25
30
x
0
5
10
15
20
25
30

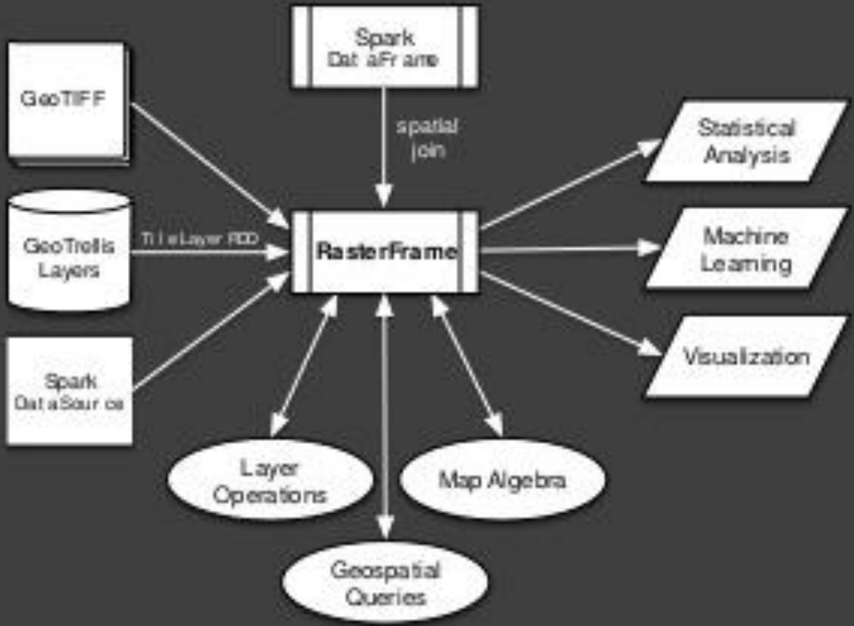


Home
Projects
Data
Lab
Random Spot 2
info+beaver...
eenAfter
close
subtract
close
14
44%
Operation
divide
subtract
add
Operation
subtract
divide
divide
Operation
divide
subtract
add
Lost water appears yellow/green
Leaflet | © OpenStreetMap ©CartoDB

RasterFrames

RasterFrames

- Incubating LocationTech project
- Provides ability to work with global-scale remote sensing imagery in a convenient yet scalable format
- Integrates with multiple data sources and libraries, including Spark ML, GeoTrellis Map Algebra and GeoMesa Spark-JTS
- Python, Scala and SQL APIs



Polyglot API

```sql
SELECT spatial_key,
    rf_localAggMin(red) as red_min,
    rf_localAggMax(red) as red_max,
    rf_localAggMean(red) as red_mean
FROM df
GROUP BY spatial_key
```

```python
df.groupBy(df.spatial_key).agg( \
    localAggMin(df.red).alias('red_min'), \
    localAggMax(df.red).alias('red_max'), \
    localAggMean(df.red).alias('red_mean'))
```

```scala
df.groupBy("spatial_key").agg(
    localAggMin($"red") as "red_min",
    localAggMax($"red") as "red_max",
    localAggMean($"red") as "red_mean")
```

RasterFrame Take-Aways

- DataFrames lower cognitive friction when modeling. Good Ergonomics!
- Rich set of raster processing primitives
- Support for descriptive and predictive analysis
- Via `spark-shell`, Jupyter Notebook, Zeppelin, etc. can interact with data and iterate over solution
- It scales!
- Many more examples at http://rasterframes.io

Thanks!

CCRi
DATA TO KNOWLEDGE

James Hughes

- jhughes@ccri.com
- http://geomesa.org
- http://gitter.im/locationtech/geomesa
- https://github.com/locationtech/geomesa/
- @CCR_inc