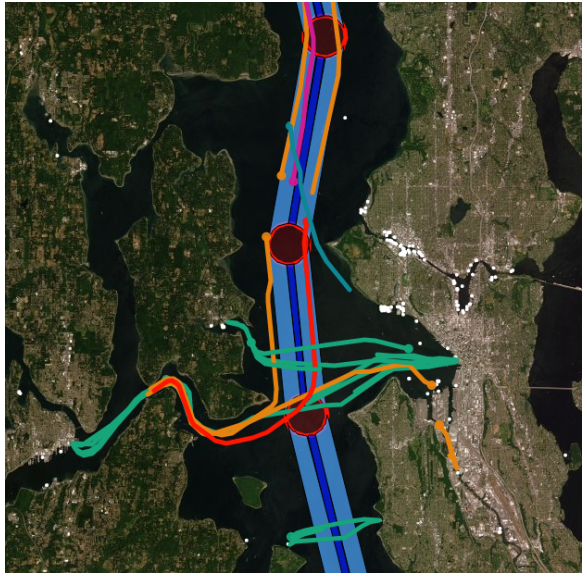


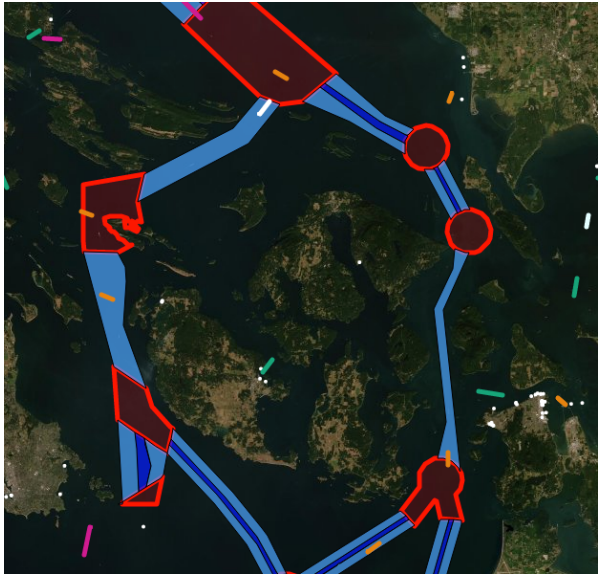
RedisTimeSeries

Using Redis for Geospatial Applications

Rock Pereira

CUGOS Fall Fling

Oct 06, 2019



RedisAI

1 Intro: Data Analysis Vs App Dev

Data Analysis	App Development
Data used is old	Data used is fresh / recent
Larger size	Smaller size
Stored on disk	Stored in memory
Model building / development	Model prediction / serving

1 Intro: SQL Vs NOSQL

SQL	NOSQL
Stored as records in tables	Stored as Key-Value, Documents, or Graphs
Complex SQL queries	Can query, but no joins
ACID Transactions	Eventual consistency
eg. PostgreSQL, MySQL	eg. Redis, MongoDB, Neo4J

2 Redis: Introduction

Redis = Remote Dictionary Service

Redis is a Data Structure server:

Strings of binary safe bytes
Lists
Hashes
Sets eg. “students” ::: {“Alice”, “Bob”, “Carla”}
Sorted Sets eg. “student:ages” ::: {(“Bob”, 19), (“Carla”, 22),(“Alice”, 25)}

Add (“Dan”, 20): The sorted set stays sorted by the score.
“student:ages” ::: {(“Bob”, 19), (“Dan”, 20), (“Carla”, 22),(“Alice”, 25)}

2 Redis: Geospatial

Locations, (lon, lat) are stored as sorted sets.

The score in the sorted set cannot be a tuple.

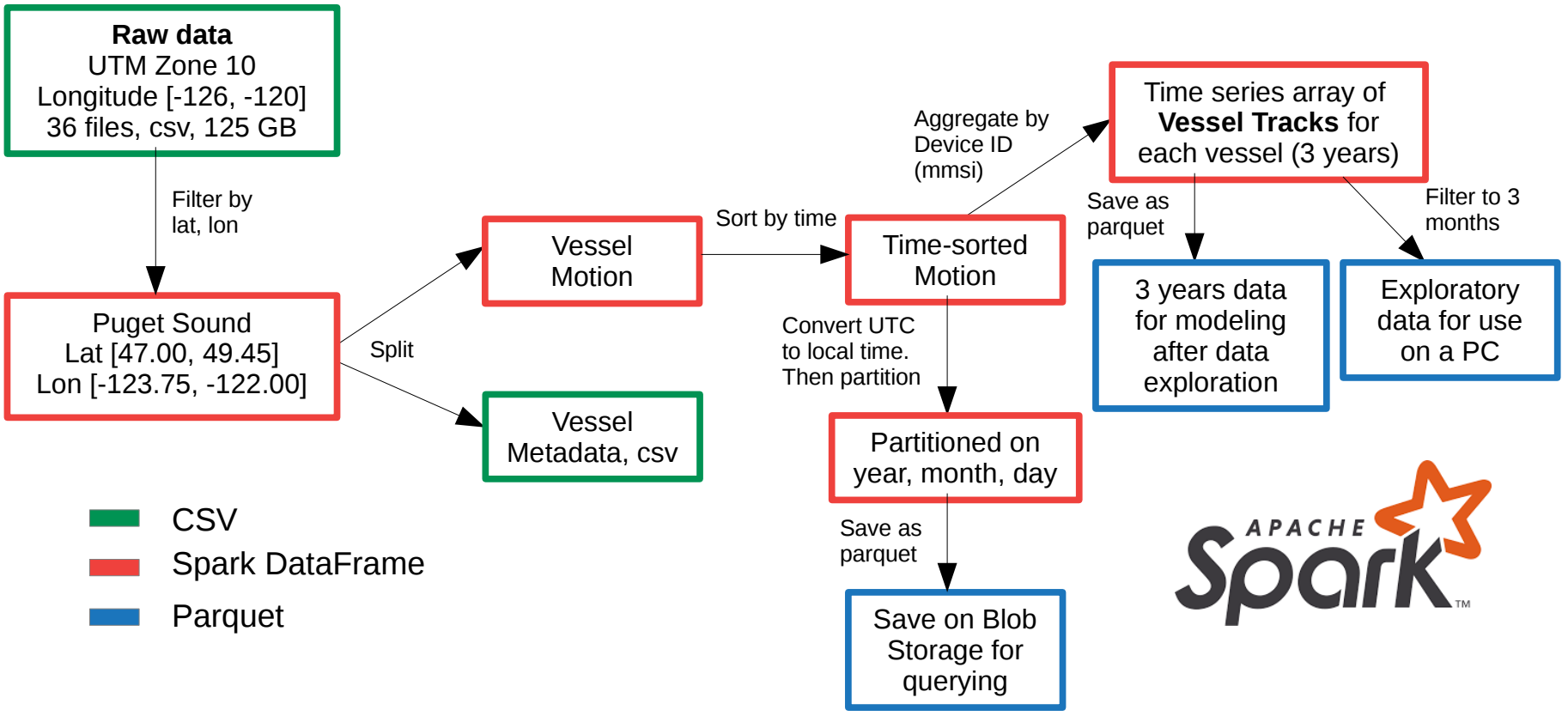
So the (lon, lat) tuple is hashed to a single value

“cities” ::: {(“Olympia”, “a8b65e..”), (“Tacoma”, “1bc3e4..”), (“Seattle”, “3df24a..”)}

2 Redis: Use Cases

Caching: To manage fresh data
Message broker: Redis has Pub Sub
Real-time analytics
Session management: eg Shopping carts
Gaming: Leaderboards & User activity
Time-series logs
Job and Queue Management
Ride-Hailing: Matching customers with nearby taxis

Data Prep – Spark Workflow



Slide Type Slide

```
import datetime as dt
import itertools
import os
from calendar import monthrange
from random import randint

# from google.cloud import bigquery
import geopandas as gpd
import ml2rt  # for loading and serializing models
import numpy as np
import pandas as pd
import pyarrow.parquet as pq
import redis
from geopy.distance import distance
from ipyleaflet import Map, basemaps, basemap_to_tiles, Polyline, Polygon, Circle, Rectangle, LayerGroup, Layer
from palettable.lightbartlein.sequential import Blues10_4  # For Shipping Lanes
from palettable.cartocolors.qualitative import Bold_10, Pastel_10  # For ships & tracks
from redisearch import Client, TextField, NumericField, Query

# All three Redis modules have a constructor named "Client".
# To avoid name clashes, import the modules just before you need them
# from redistimeseries.client import Client
# from redisai import Client, DType, Tensor, Backend, Device

CODE_DIR = "/media/rock/x/data/ais_20180921/"
DATA_DIR = CODE_DIR + "output/"
os.chdir(CODE_DIR)

## os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = "/media/rock/x/data/ais_20180921/ais-bigquery-sa.json"
## client = bigquery.Client()
## dataset_id = 'by_ymd_3m'  # for BigQuery
# https://googleapis.dev/python/bigquery/latest/index.html

r = redis.Redis()
# r = redis.Redis(host='localhost', port=6379, db=0, password=None, socket_timeout=None)
# https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Timestamp.html

esri_satellite_layer = basemap_to_tiles(bm=basemaps.Esri.WorldImagery)
GOOGLE_MAPS_API_KEY = [redacted]
# jupyter nbextension enable --py --sys-prefix ipyleaflet  # For Jupyter Notebook
# jupyter labextension install @jupyter-widgets/jupyterlab-manager jupyter-leaflet  # for JupyterLab
```

Slide Type Slide

BigQuery

BigQuery can read partitioned parquet datasets that were produced in Spark

Slide Type Fragment

```
# INCOMPLETE
# https://cloud.google.com/bigquery/docs/loading-data-cloud-storage-parquet
dataset_ref = client.dataset(dataset_id)
job_config = bigquery.LoadJobConfig()
job_config.source_format = bigquery.SourceFormat.PARQUET
uri = "gs://ais-rmp/output/by_ymd_3m_pqt/"

load_job = client.load_table_from_uri(uri, dataset_ref.table("by_ymd_3m"), job_config=job_config)
```

Slide Type Slide

PyArrow, Pandas

PyArrow can read partitioned parquet datasets on the desktop

The dataset is a directory named "by_ymd_3m", with nested subdirectories for year, month, and day.

Slide Type Slide

```
def vessels_one_minute(y, m, d, H, M, S):
    """
    Returns a dataframe of vessels in Puget Sound at the specified time
    """
    DIR = DATA_DIR + "by_ymd_3m_pqt" + "/year=" + str(y) + "/month=" + str(m) + "/day=" + str(d)
    df = pq.read_table(DIR).to_pandas()  # Create a pandas dataframe from the pyarrow table
    vessels_minute = df.loc[(df['localtime'] > pd.Timestamp(year=y, month=m, day=d, hour=H, minute=M-1, second=
                            (df['localtime'] <= pd.Timestamp(year=y, month=m, day=d, hour=H, minute=M, second=S
    return vessels_minute


def vessels_one_hour(y, m, d, H, M, S):
    """
    Returns a dataframe of vessels in Puget Sound at the specified time
    This dataframe is used in the RedisTimeSeries section
    """
    DIR = DATA_DIR + "by_ymd_3m_pqt" + "/year=" + str(y) + "/month=" + str(m) + "/day=" + str(d)
    df = pq.read_table(DIR).to_pandas()
    vessels_hour = df.loc[(df['localtime'] > pd.Timestamp(year=y, month=m, day=d, hour=H-1, minute=M, second=S)
                            (df['localtime'] <= pd.Timestamp(year=y, month=m, day=d, hour=H, minute=M, second=S
    return vessels_hour
```

Slide Type: Slide

```
def create_geokey(keyname, df):
    """
    Creates a Geospatial key from a DataFrame, df.
    Returns nothing
    """
    lonlatnames = [[df['lon'].iloc[row], df['lat'].iloc[row], str(df['mmsi'].iloc[row])] for row in range(len(d
    concat_list = list(itertools.chain(*lonlatnames))
    r.geoadd(keyname, *concat_list)
    return


def get_vessel_metadata(mmsi):
    return vessels.loc[vessels.mmsi == mmsi]
```

Slide Type: Fragment

```
# Select a random time point in 3 months / 3 years. Pick 1 minute / 1 hour of data before that time point.
y = 2017
m = randint(10, 12)
d = randint(1, monthrange(y,m)[1])
H = randint(1, 23)
M = randint(0, 59)
S = randint(0, 59)
```

Slide Type: Fragment

```
y,m,d,H,M,S
# vessels_one_hour(2017, 11, 3, 17, 58, 35)
```

```
(2017, 12, 4, 5, 25, 15)
```

Slide Type Slide

Read Metadata

Attributes for each vessel

Slide Type Fragment

```
# RUN: Read the metadata file which has "vessel_group" column added
col_names = ['mmsi', 'vessel_code', 'vessel_name', 'imo', 'call_sign', 'l', 'w', 'draft', 'cargo', 'vessel_grou
types = ['int32', 'category', 'object', 'object', 'object', 'float64', 'float64', 'float64', 'category', 'categ
col_dtypes = dict(zip(col_names, types))

metadata_file = DATA_DIR + 'metadata_vg_3m.csv'  # !!! CHANGE for 3 months <==> 3 years
# metadata_file = DATA_DIR + 'metadata_vg_3y.csv'  # !!! CHANGE for 3 months <==> 3 years

vessels = pd.read_csv(metadata_file, header=0, names=col_names, dtype=col_dtypes)
```

Slide Type Slide

```
vgroup2color = {'pleasure': Bold_10.hex_colors[7],
                'publicService': "red",
                'passenger': Bold_10.hex_colors[1],
                'cargo': Bold_10.hex_colors[8],
                'tanker': Bold_10.hex_colors[3],
                'fishing': Blues10_4.hex_colors[0],
                'tugTow': Bold_10.hex_colors[6],
                'military': "olive",
                'research': "violet",
                'unknown': "white"
               }

vgroups = list(vgroup2color.keys())
```

Slide Type Slide

Read Motion data from Parquet

Slide Type Fragment

```
vm = vessels_one_minute(y, m, d, H, M, S)
# vm = vessels_one_minute(2017, 11, 3, 17, 58, 35)
vm_moving = vm.loc[vm['sog'] >= 4]
vm_stationary = vm.loc[vm['sog'] < 4]
```

Slide Type Slide

Create Geospatial Keys in Redis

Slide Type Fragment

```
# DO NOT RUN if Keys already exist
create_geokey("moving", vm_moving)
create_geokey("stationary", vm_stationary)
create_geokey("all", vm)
```

Slide Type Slide

Get the MMSIs for the moving vessels from the Sorted Set

Slide Type Fragment

```
1 moving_mmsis = [mmsi.decode("utf-8") for mmsi in r.zrange("moving", 0, -1)]
2 moving_mmsis[:10]
```

```
['367153930',
 '367529420',
 '366773070',
 '366710820',
 '367474960',
 '367705050',
 '366772760',
 '367548420',
 '366673090',
 '356334000']
```

Slide Type Slide

Geospatial Functions in Redis

GEOPOS, GEODIST, GEORADIUSBYMEMBER, GEOADD, GEOHASH, GEORADIUS

Slide Type Fragment

```
1 r.geopos("moving", moving_mmsis[10])[0]
```

```
(-123.33863228559494, 48.38502986698124)
```

Slide Type Fragment

```
1 r.geodist("moving", moving_mmsis[10], moving_mmsis[11], unit="mi")
```

```
0.0403
```

Slide Type Fragment

```
1 r.georadiusbymember("moving", moving_mmsis[10], 3, unit="mi", withdist=True)
```

```
[[b'316013431', 0.0], [b'229569000', 0.0403]]
```

Slide Type Fragment

```
1 r.georadiusbymember("all", moving_mmsis[10], 3, unit="mi", withdist=True)
```

```
[[b'316003657', 2.9871],
 [b'316027782', 2.964],
 [b'533130187', 2.9665],
 [b'316003567', 2.9318],
 [b'316013431', 0.0],
 [b'229569000', 0.0403]]
```

Slide Type Slide

Map Vessel Locations

Slide Type Fragment

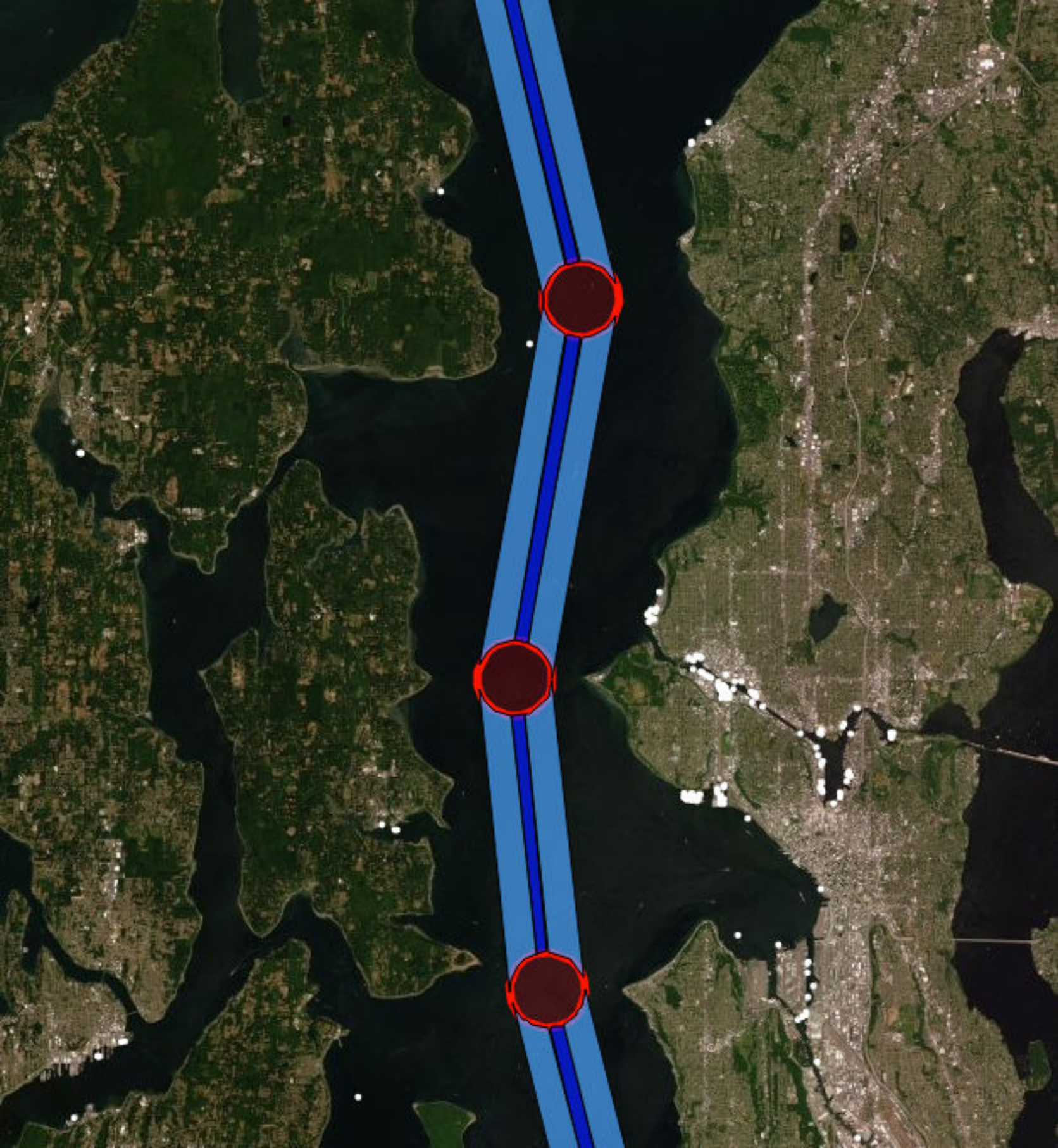
Read the shapefile for Puget Sound shipping lanes with GeoPandas

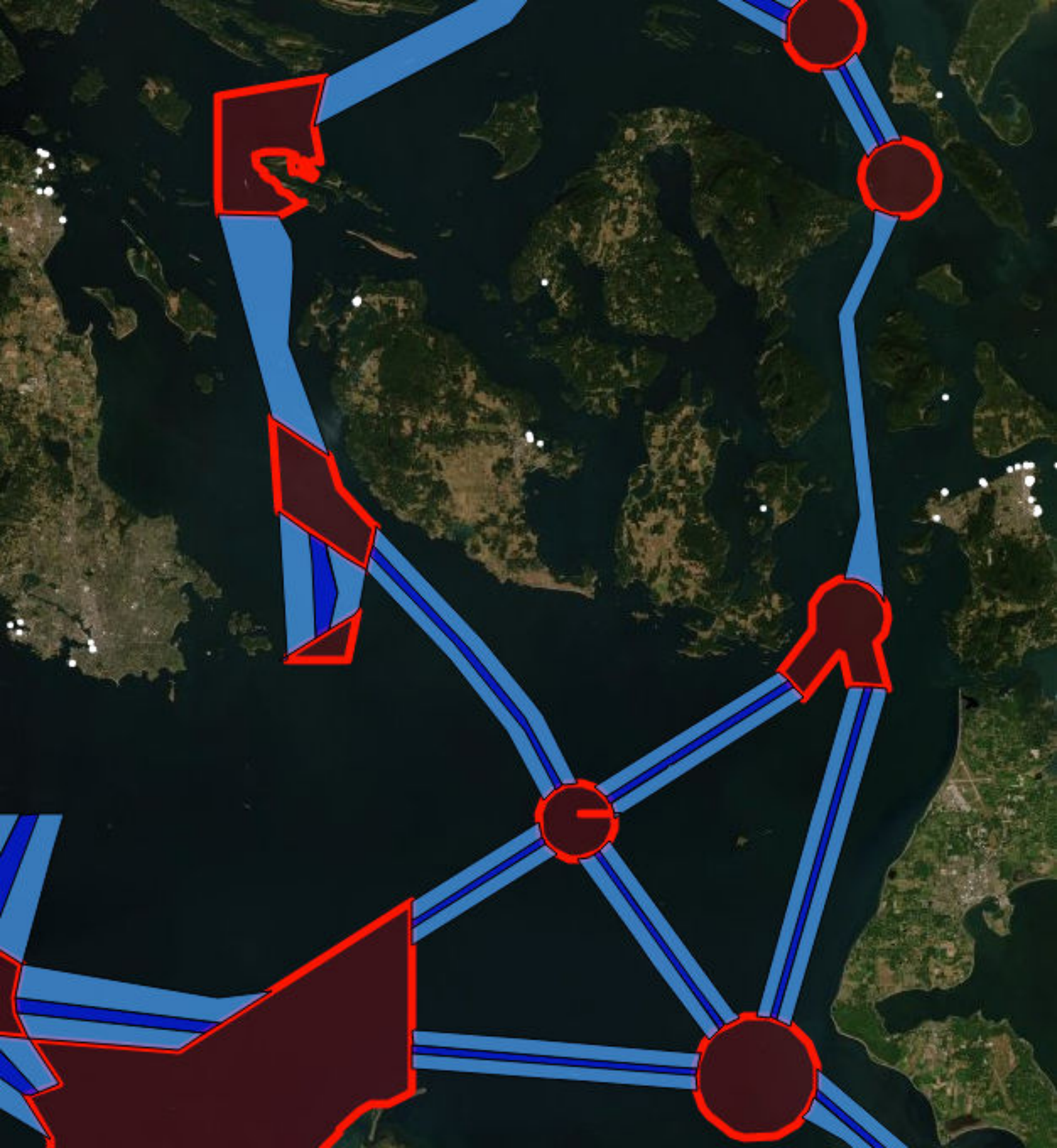
Slide Type Fragment

```
sss = gpd.read_file('/media/rock/x/data/ais_20180921/shapefiles/shiplanes_puget/shiplanes_puget.shp')
# The US Shiplanes shapefile has been cut to (lat BETWEEN 47 AND 49.45) AND (lon BETWEEN -123.75 AND -122)
```

Slide Type Slide

```
def map_shipping_lanes(mmsi, zoom=11):
    """
    Creates a leaflet map, with shipping lanes and stationary vessels (white circles)
    """
    vessel_location = r.geopos("moving", mmsi)[0]
    m = Map(center=(vessel_location[1], vessel_location[0]), zoom=zoom)
    m.layout.width = '65%'
    m.layout.height = '1000px'
    m.add_layer(esri_satellite_layer)
    m.add_control(LayersControl())

    # Precautionary Areas
    precAreas_gs = sss[sss.OBJL == '96'].geometry
    num_precAreas = len(precAreas_gs)
    precAreas = [None] * num_precAreas
    for i in range(num_precAreas):
        precAreas[i] = ((((precAreas_gs.iloc[i]).exterior.coords)[k][1], ((precAreas_gs.iloc[i]).exterior.coord

    # Traffic Separation Schemes
    sepSchemes_gs = sss[sss.OBJL == '150'].geometry
    num_sepSchemes = len(sepSchemes_gs)
    sepSchemes = [None] * num_sepSchemes
    for i in range(num_sepSchemes):
        sepSchemes[i] = ((((sepSchemes_gs.iloc[i]).exterior.coords)[k][1], ((sepSchemes_gs.iloc[i]).exterior.co

    # Shipping Lanes
    shiplanes_gs = sss[(sss.OBJL == '148') | (sss.OBJL == '152')].geometry
    num_shiplanes = len(shiplanes_gs)
    shiplanes = [None] * num_shiplanes
    for i in range(num_shiplanes):
        shiplanes[i] = ((((shiplanes_gs.iloc[i]).exterior.coords)[k][1], ((shiplanes_gs.iloc[i]).exterior.coord

    colors = Blues10_4.hex_colors
    precAreas_polygons_list = [Polygon(locations=precAreas[i], color='red', weight=5, fill_color='red') for i i
    sepSchemes_polygons_list = [Polygon(locations=sepSchemes[i], color='black', weight=1, fill_color='blue', fi
    shiplanes_polygons_list = [Polygon(locations=shiplanes[i], color='black', weight=1, fill_color=colors[2], f

    precAreas_layer_group = LayerGroup(layers=tuple(precAreas_polygons_list))
    sepSchemes_layer_group = LayerGroup(layers=tuple(sepSchemes_polygons_list))
    shiplanes_layer_group = LayerGroup(layers=tuple(shiplanes_polygons_list))
    m.add_layer(precAreas_layer_group)
    m.add_layer(sepSchemes_layer_group)
    m.add_layer(shiplanes_layer_group)
```





Slide Type Slide

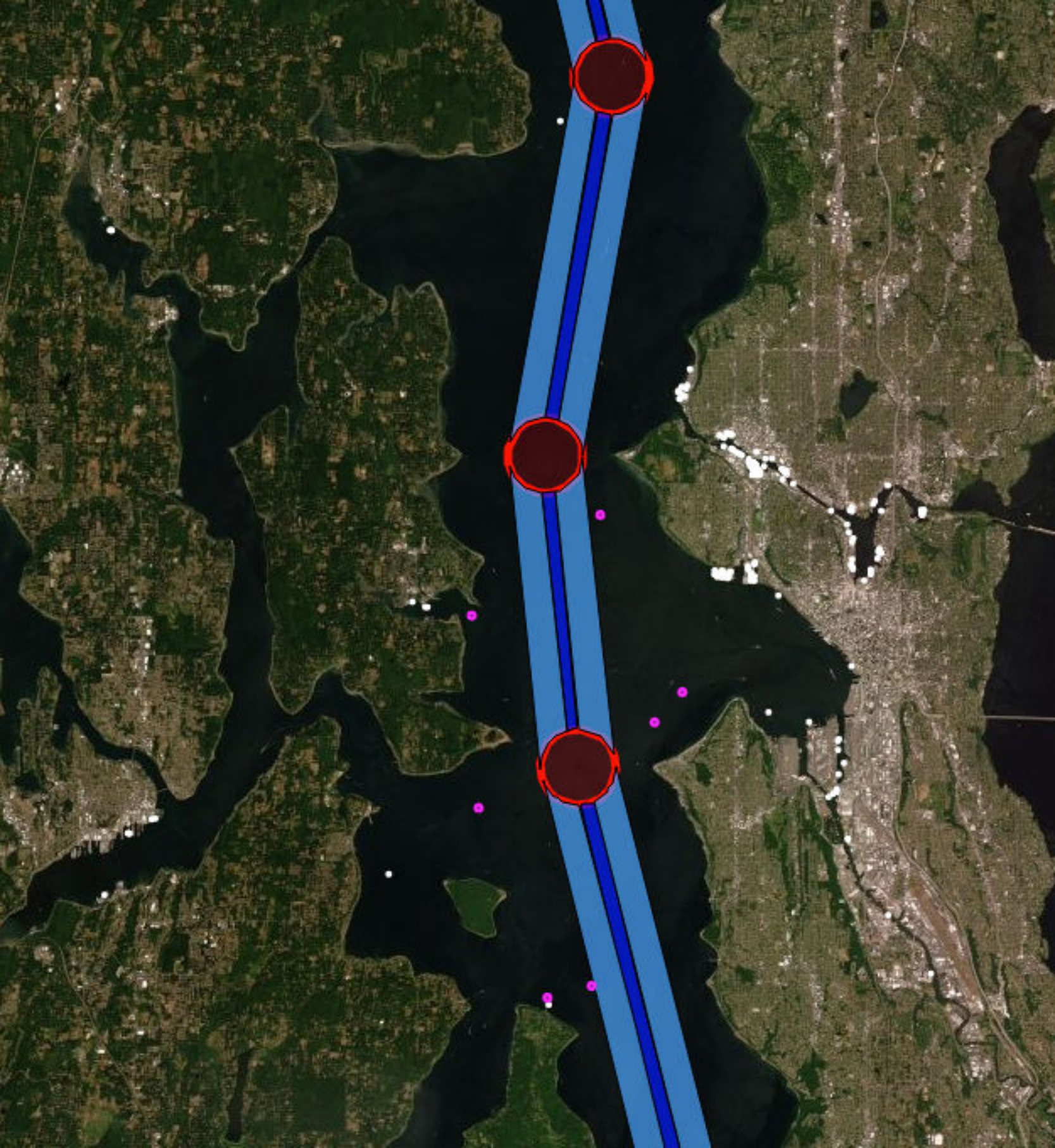
Use GEOPOS to get the latitude and longitude of moving vessels

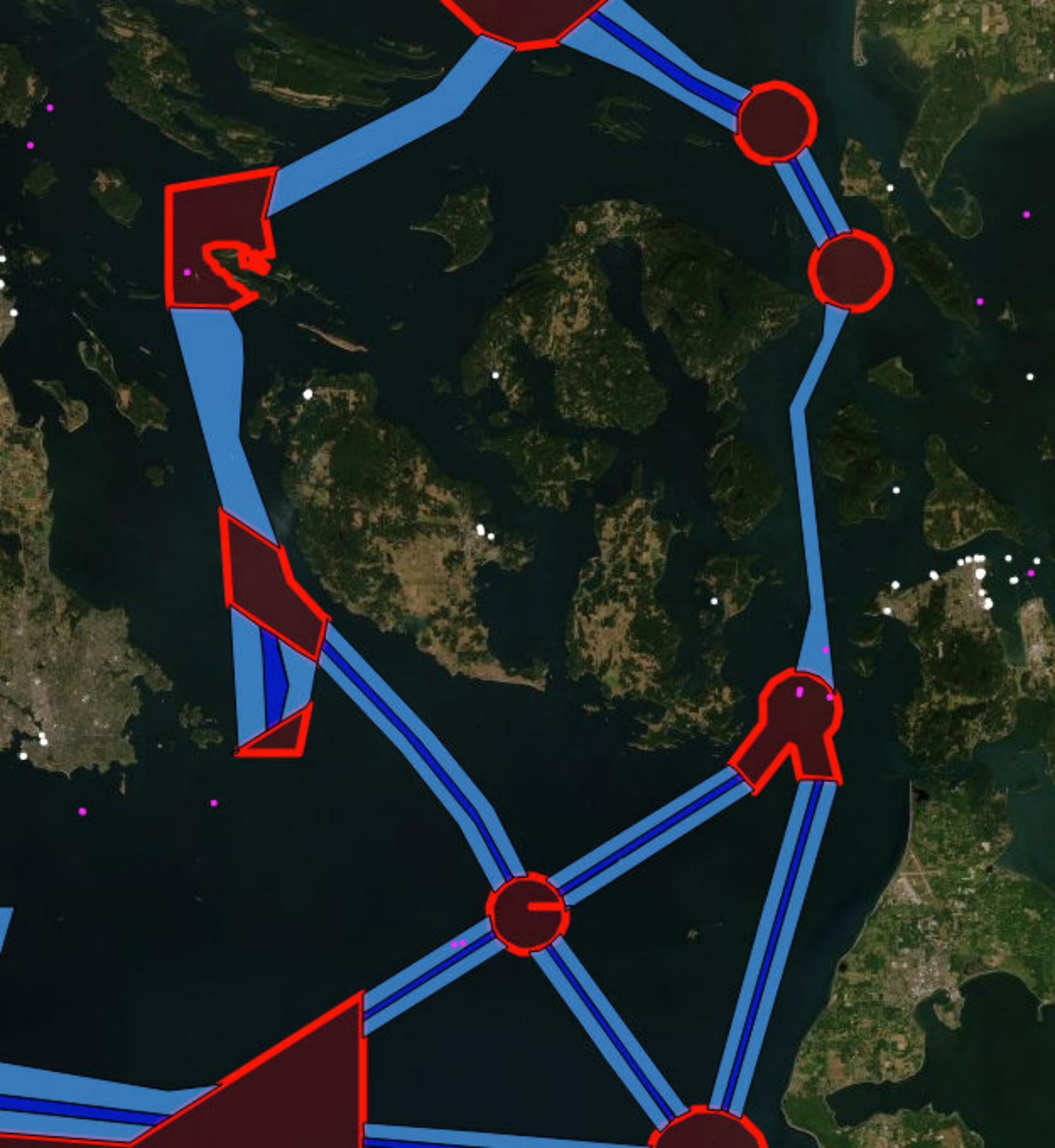
Slide Type Fragment

```
def map_moving_vessels(mmsi, zoom=11):
    """
    Adds moving vessels (speed > 4 knots) as magenta circles
    """
    m = map_shipping_lanes(mmsi, zoom=zoom)
    # Add moving vessels
    moving_latlons = r.geopos("moving", *moving_mmsis)
    moving_vessels = [Circle(location=(moving_latlons[i][1], moving_latlons[i][0]), radius=120, color='magenta'
    moving_layer_group = LayerGroup(layers=tuple(moving_vessels))
    m.add_layer(moving_layer_group)

    return m
```

Slide Type Slide

```
map_moving_vessels(moving_mmsis[10], zoom=11)
```





Slide Type Slide ▼

Redis Modules

RediSearch, RedisTimeSeries, RedisAI

Slide Type Fragment ▼

Loading Modules:

1. From python:

os.system("redis-server --loadmodule ./redisearch.so")

2. From the configuration file /etc/redis/redis.conf

loadmodule /path/to/redisearch.so

3. From the CLI (redis-cli)

127.0.0.6379> MODULE load /path/to/redisearch.so

https://oss.redislabs.com/redisearch/Commands.html#ftsearch

https://oss.redislabs.com/redisearch/Query_Syntax.html

Slide Type Slide

RediSearch

Slide Type Fragment

```
# from redisearch import Client, TextField, NumericField, Query

# Creating a client with a index name, metadataIndex
meta = Client('metadataIndex')

# Creating the index definition and schema
meta.create_index([TextField('vessel_group'), NumericField('length'), NumericField('width'), NumericField('draf
```

```
b'OK'
```

Slide Type Slide

Add Documents to the Search Index from Metadata

Slide Type Fragment

```
# DO NOT RUN if the Index & Keys have been created
for row in range(len(vessels.index)):
    meta.add_document(doc_id="meta_"+str(vessels['mmsi'].iloc[row]), replace=True,
                      vessel_group=vessels['vessel_group'].iloc[row] if str(vessels['vessel_group'].iloc[row])
                      length=vessels['l'].iloc[row] if not np.isnan(vessels['l'].iloc[row]) else -999,
                      width=vessels['w'].iloc[row] if not np.isnan(vessels['w'].iloc[row]) else -999,
                      draft=vessels['draft'].iloc[row] if not np.isnan(vessels['draft'].iloc[row]) else -999)
```

Slide Type Slide

Query the Search Index

Slide Type Fragment

```
results = {vg:meta.search(Query(vg).no_content().paging(0,5000)).docs for vg in vgroups}
vg2mmsiList = {vg:[doc.id[5:] for doc in results[vg]] for vg in vgroups}
mmsi2vg = {mmsi:vg for vg in vgroups for mmsi in vg2mmsiList[vg]}
```

Slide Type Fragment

```
{vg:len(vg2mmsiList[vg]) for vg in vgroups}
```

```
{'pleasure': 2302,
 'publicService': 222,
 'passenger': 333,
 'cargo': 1951,
 'tanker': 402,
 'fishing': 604,
 'tugTow': 462,
 'military': 2,
 'research': 37,
 'unknown': 2463}
```

Slide Type: Slide

Color-code moving vessels (passenger, cargo, fishing etc) using RediSearch query results

Slide Type: Fragment

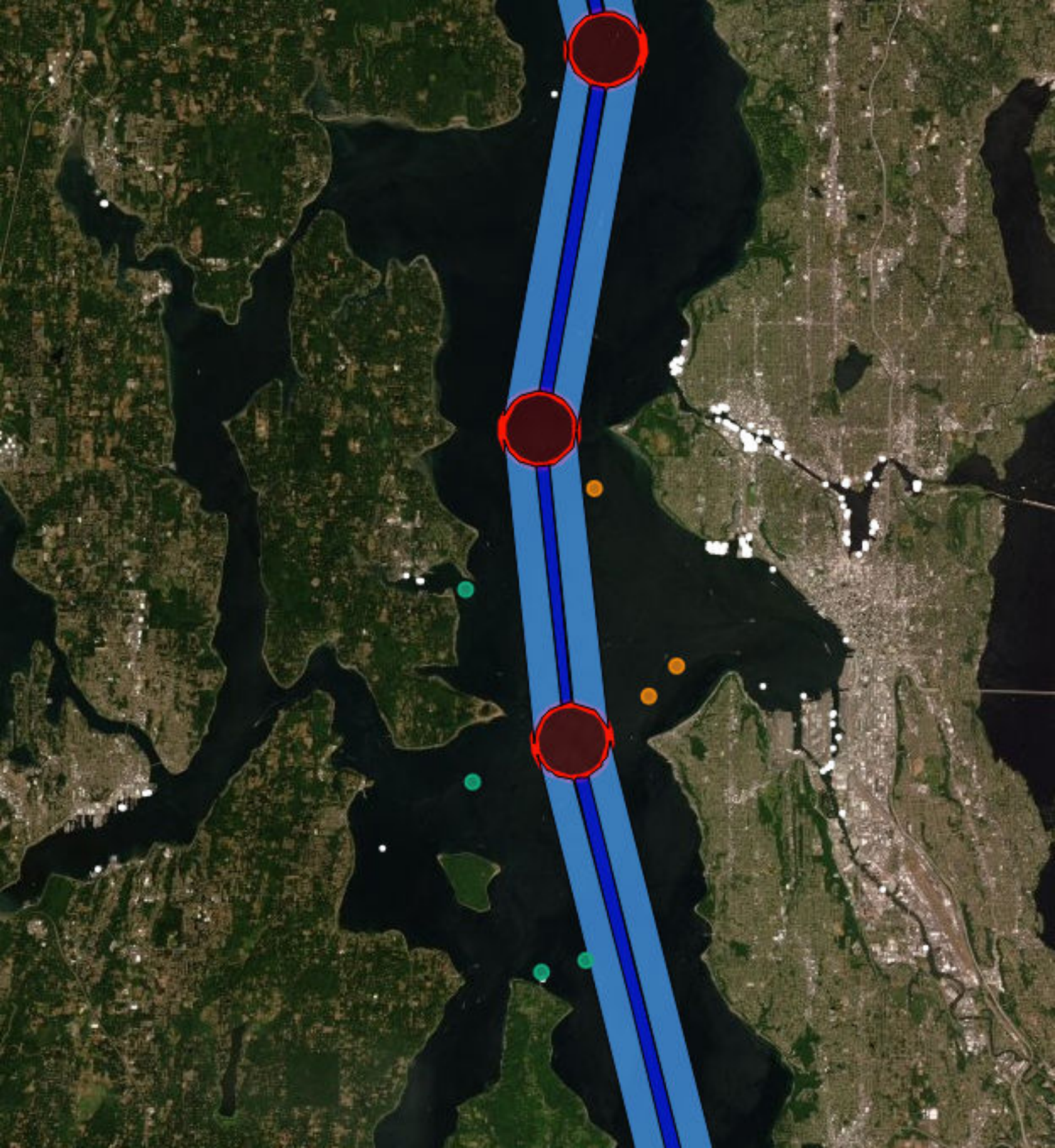
```
moving_mmsis = [mmsi.decode("utf-8") for mmsi in r.zrange("moving", 0, -1)]
moving_mmsi_vgroups = [mmsi2vg.get(mmsi) for mmsi in moving_mmsis]
moving_mmsi_colors = [vgroup2color[vg] for vg in moving_mmsi_vgroups]
```

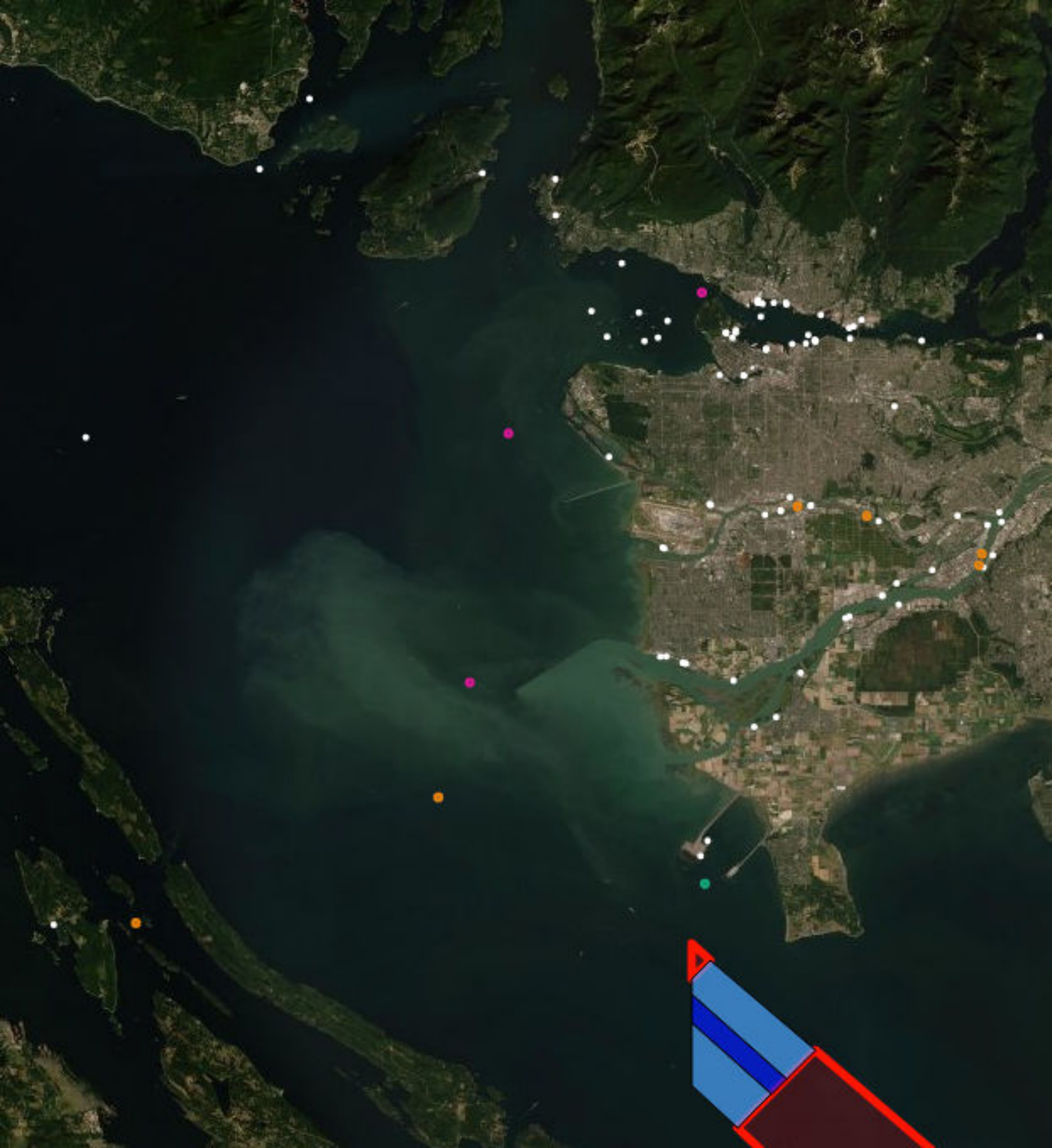
Slide Type: Fragment

```
def map_moving_vessels_by_color(mmsi, zoom=11):
    m = map_shipping_lanes(mmsi, zoom=zoom)
    # Add moving vessels
    moving_latlons = r.geopos("moving", *moving_mmsis)
    moving_vessels = [Circle(location=(moving_latlons[i][1], moving_latlons[i][0]), radius=200,
                             color=moving_mmsi_colors[i], weight=2, fill_color=moving_mmsi_colors[i], fill_opac
    moving_layer_group = LayerGroup(layers=tuple(moving_vessels))
    m.add_layer(moving_layer_group)

    return m
```

Slide Type: Slide

```
map_moving_vessels_by_color(moving_mmsis[10], zoom=11)
```





Slide Type Slide

Redis Time Series

Slide Type Fragment

```
from redistimeseries.client import Client
rts = Client()
```

Slide Type Fragment

```
v1h = vessels_one_hour(y, m, d, H, M, S)
# v1h = vessels_one_hour(2017, 11, 3, 17, 58, 35)
mmsi2latlon = {str(mmsi):(v1h.loc[v1h['mmsi'] == int(mmsi)]['lat'].values, v1h.loc[v1h['mmsi'] == int(mmsi)]['l
```

Slide Type Slide

Create Keys to hold the Time Series data with 1 hour retention

Slide Type Fragment

```
# DO NOT RUN if Keys exist
for mmsi in moving_mmsis:
    rts.create("ts_lat_" + mmsi, retention_msecs=3600000, labels={'Time':'Lat'})
    rts.create("ts_lon_" + mmsi, retention_msecs=3600000, labels={'Time':'Lon'})

# r.keys(pattern=u'ts_lat*')
# rts.get('ts_lat_' + mmsis[10])
```

Slide Type Fragment

```
# DO NOT RUN if Keys exist
for mmsi in moving_mmsis:
    v1h_subset = v1h.loc[v1h['mmsi'] == int(mmsi)]
    lats = v1h_subset['lat'].values
    lons = v1h_subset['lon'].values
    time = v1h_subset['localtime']
    for i in range(len(lats)):
        rts.add("ts_lat_" + mmsi, int(time.iloc[i].timestamp())*1000, lats[i])
        rts.add("ts_lon_" + mmsi, int(time.iloc[i].timestamp())*1000, lons[i])

# time is UNIX timestamp in milliseconds
# YES, time = v1h_subset['localtime'] is a pandas series of type pd.Timestamp. Then int(time.iloc[i].timestamp(
# NO, time = v1h_subset['localtime'].apply(lambda lt: int(lt.timestamp())) seems to be the same as above, but w
# NO, time = v1h_subset['localtime'].apply(lambda lt: int(lt.timestamp())).values. The array has type np.dateti
```

Slide Type Slide

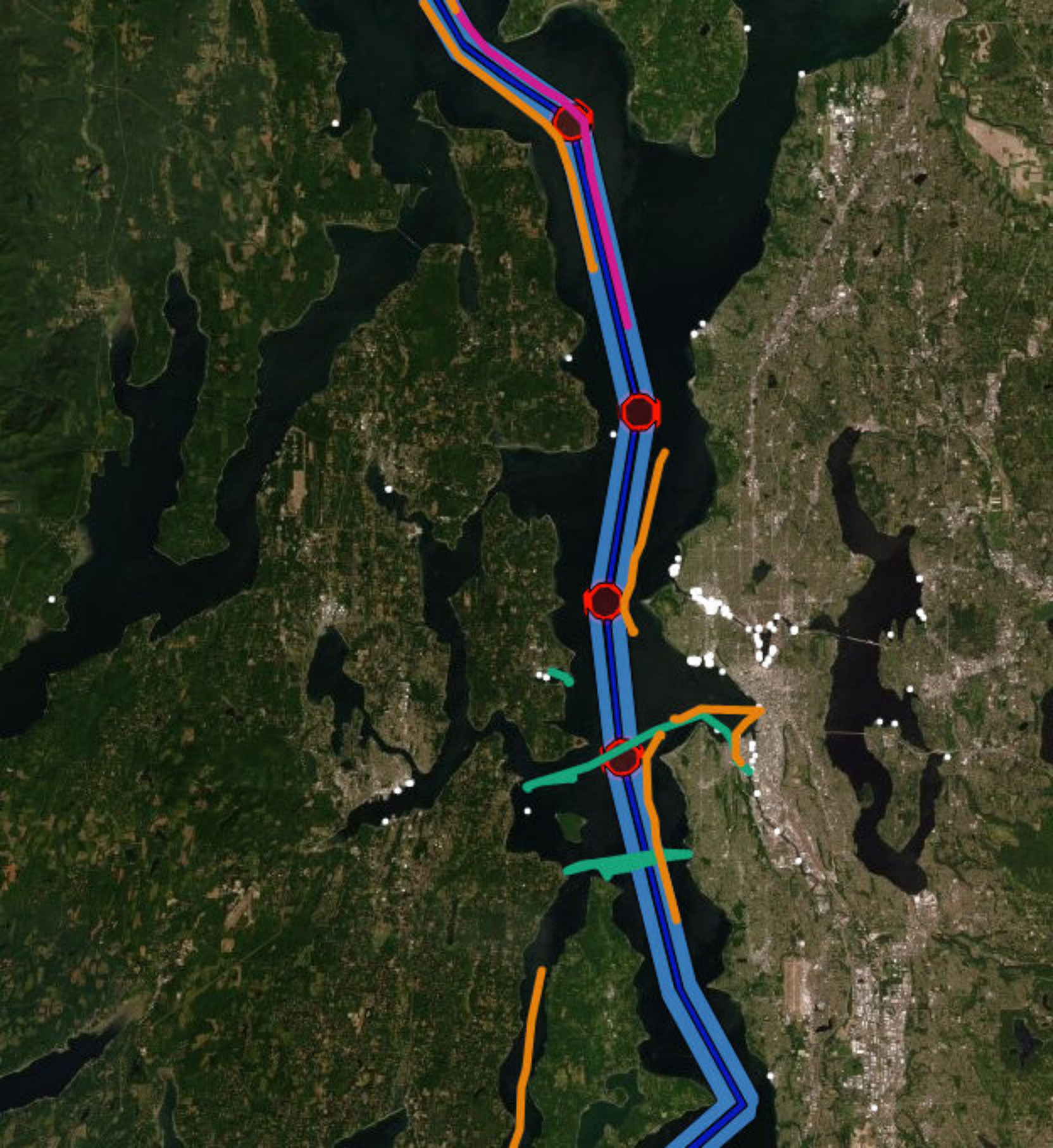
Map Vessel Tracks with RedisTimeSeries

Slide Type Fragment

```
def map_vessel_tracks_redis_ts(mmsi, zoom=11):
    m = map_moving_vessels_by_color(mmsi, zoom=zoom)
    mmsi2lats = {mmsi:[float(tup[1]) for tup in rts.range(key='ts_lat_' + mmsi, from_time='0', to_time='+')] fo
    mmsi2lons = {mmsi:[float(tup[1]) for tup in rts.range(key='ts_lon_' + mmsi, from_time='0', to_time='+')] fo
    vessel_tracks = [Polyline(locations=[[mmsi2lats[moving_mmsis[j]][i], mmsi2lons[moving_mmsis[j]][i]] for i i
                      color=moving_mmsi_colors[j], fill_color=moving_mmsi_colors[j], fill_opacity=0.0) for j in
    tracks_layer_group = LayerGroup(layers=tuple(vessel_tracks))
    m.add_layer(tracks_layer_group)
    return m
```

Slide Type Slide

```
map_vessel_tracks_redis_ts(moving_mmsis[10], zoom=11)
```



Slide Type Slide

RedisAI

Slide Type Fragment

```
from redisai import Client, DType, Tensor, Backend, Device
# import ml2rt
client = Client()
# https://oss.redislabs.com/redisai/  # docs
# https://github.com/RedisAI/redisai-py  # good
# https://github.com/RedisAI/RedisAI
# https://github.com/RedisAI/redisai-examples

# https://pypi.org/project/ml2rt/
# ML utilities: Converting models (sparkml, sklearn, xgboost to ONNX), serializing models to disk, loading it b
```

Slide Type Slide

Load the Backend Runtime

using redisai-py or redis-cli

Slide Type Fragment

```
client.loadbackend('TORCH', '/home/rock/git/RedisAI/install-cpu/backends/redisai_torch/redisai_torch.so')
client.loadbackend('TF', '/home/rock/git/RedisAI/install-cpu/backends/redisai_tensorflow/redisai_tensorflow.so')
client.loadbackend('ONNX', '/home/rock/git/RedisAI/install-cpu/backends/redisai_onnxruntime/redisai_onnxruntime.so')

$ redis-cli AI.CONFIG LOADBACKEND TF /home/rock/git/RedisAI/install-cpu/backends/redisai_tensorflow/redisai_tensorflow.so
$ redis-cli -x AI.MODELSET foo TF CPU INPUTS a b OUTPUTS c < test/test_data/graph.pb
```

Slide Type Slide

(Convert) and Load the Model with ml2rt

Slide Type Fragment

```
model = ml2rt.load_model('/home/rock/git/RedisAI/test/test_data/graph.pb')
client.modelset('m', Backend.tf, Device.cpu, inputs=['input_1', 'input_2'], outputs='output', data=model)
# ?client.modelset
```

Slide Type: Slide

Get the (naive) predicted tracks for the next 5 minutes

Slide Type: Fragment

```
# This does not use RedisAI yet. It estimates the 5-minute position using the geodesic distance in geopy
# from geopy.distance import distance
vm_moving['dest'] = vm_moving.apply(lambda row: distance(miles=row.sog * 5/60 * 1.15078).destination((row.lat,

# lat2,lon2,_ = distance(miles=distMiles).destination((lat1, lon1), bearing)
# 1 knot = 1.15078 miles/hour
# ~/anaconda3/lib/python3.7/site-packages/geopy/distance.py  # bearing is in degrees, not radians
```

...

Slide Type: Slide

```
def map_predicted_track(mmsi, zoom=11):
    m = map_shipping_lanes(mmsi, zoom=11)
    # Add moving vessels
    moving_latlons = r.geopos("moving", *moving_mmsis)
    moving_vessels = [Rectangle(bounds=((moving_latlons[i][1]-0.0010, moving_latlons[i][0]-0.0015), (moving_lat
                                color=moving_mmsi_colors[i], weight=2, fill_color=moving_mmsi_colors[i], fill_op
    moving_layer_group = LayerGroup(layers=tuple(moving_vessels))
    m.add_layer(moving_layer_group)
    predicted_tracks = [Polyline(locations=[[vm_moving[vm_moving['mmsi'] == int(moving_mmsis[j])]['lat'].iloc[0
                                            [vm_moving[vm_moving['mmsi'] == int(moving_mmsis[j])]['dest'].iloc[
                                color=moving_mmsi_colors[j], fill_color=moving_mmsi_colors[j], fill_opacity=0.
    pred_tracks_layer_group = LayerGroup(layers=tuple(predicted_tracks))
    m.add_layer(pred_tracks_layer_group)

    return m
```

Slide Type: Slide

```
map_predicted_track(moving_mmsis[10], zoom=11)
```