Stream Processing & Analytics with Flink

Danny Yuan, Engineer @ Uber

@g9yuayon

Four Kinds of Analytics

- On demand aggregation and pattern detection
- Clustering
- Forecasting
- Pattern detection on geo-temporal data

Two Ingredients

Geo/Spatial

Time

Real-time aggregation and pattern matching

Complex Event Processing

Examples

How many cars enter and exit a user defined area in past 5 minutes

CEP with full historical context

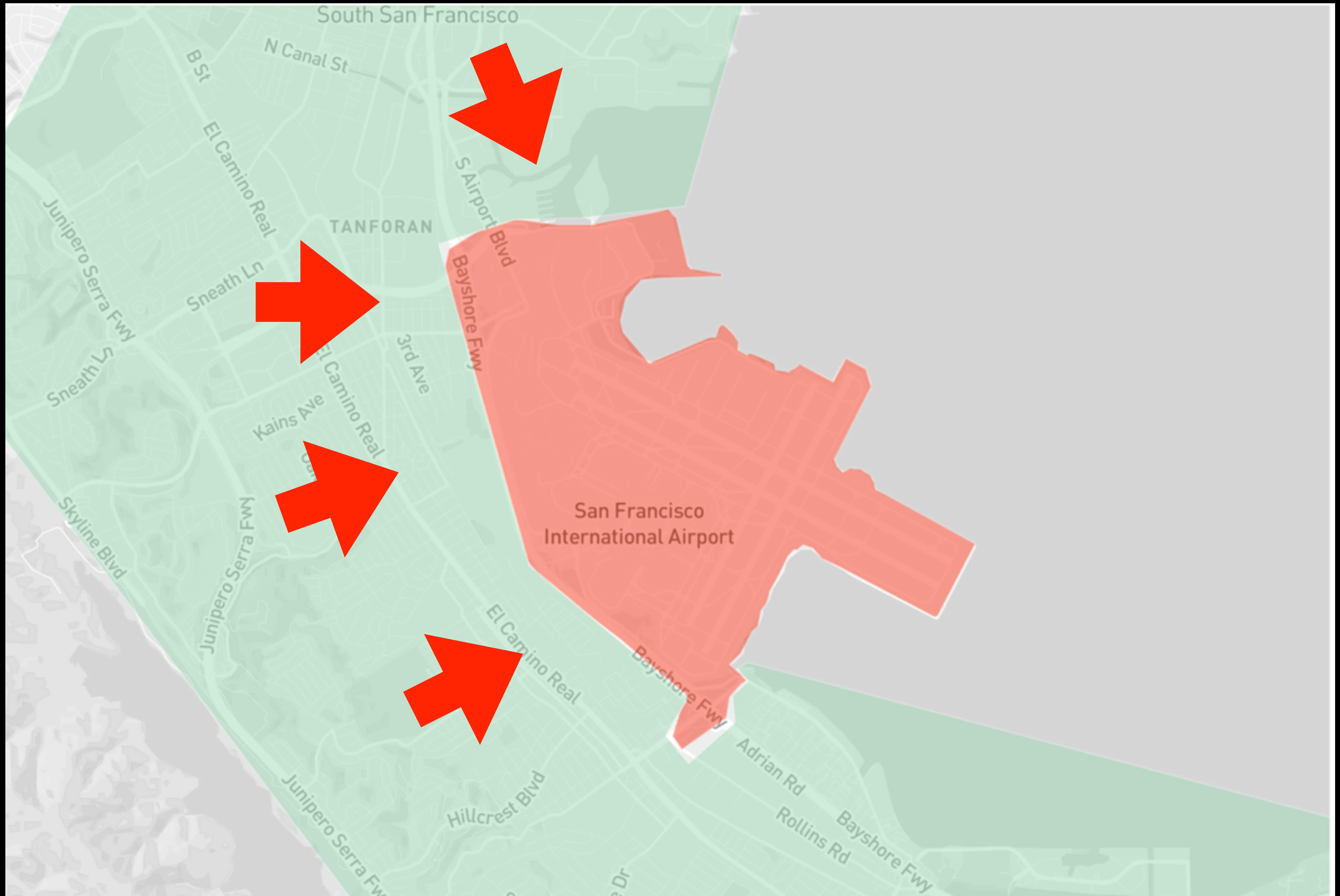
Notify me if a partner completed her 100th trip in a given area just now?

Patterns in the future

How many first-time riders will be dropped off in a given area in the next 5 minutes?

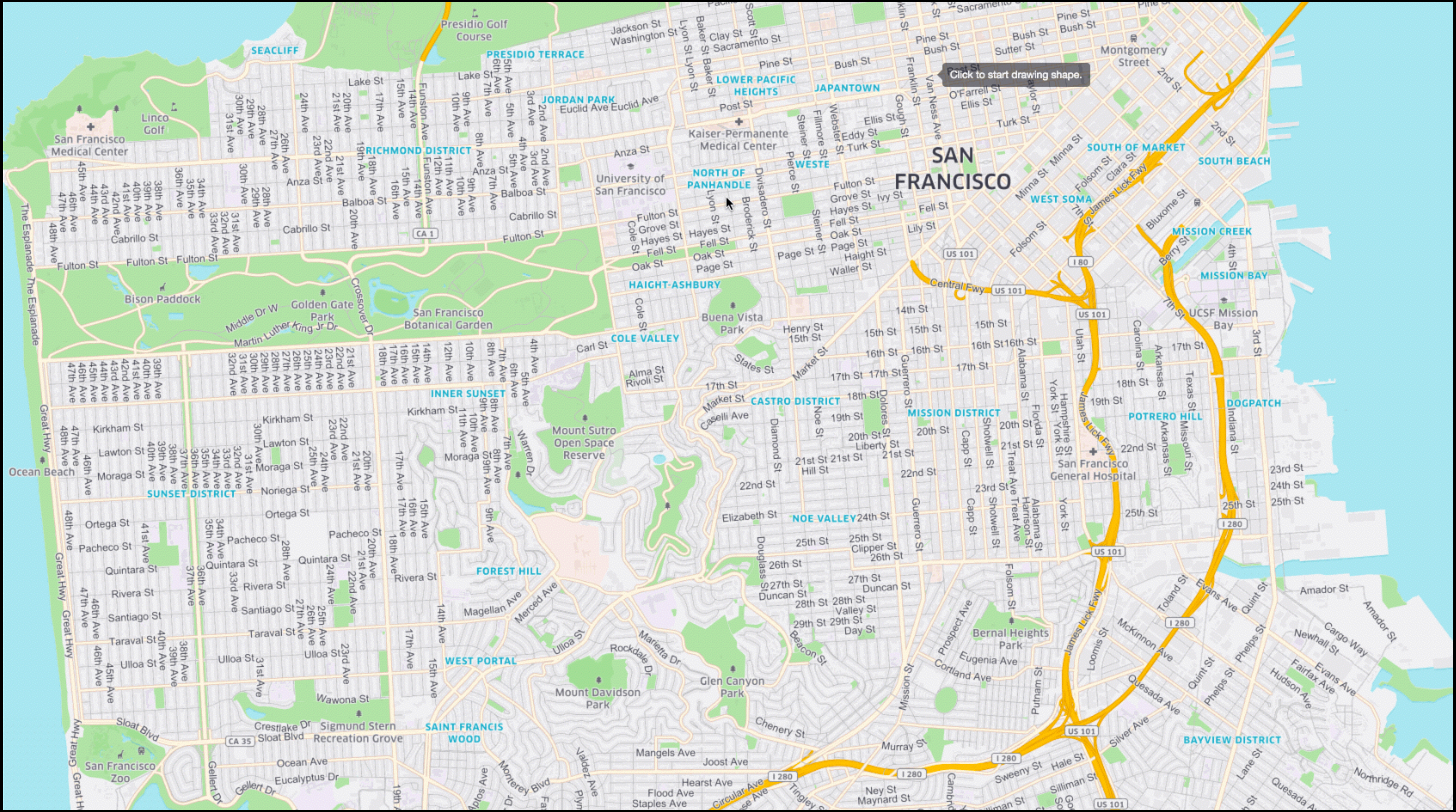
Patterns in the future

How many first-time riders will be dropped off in a given area in the next 5 minutes?



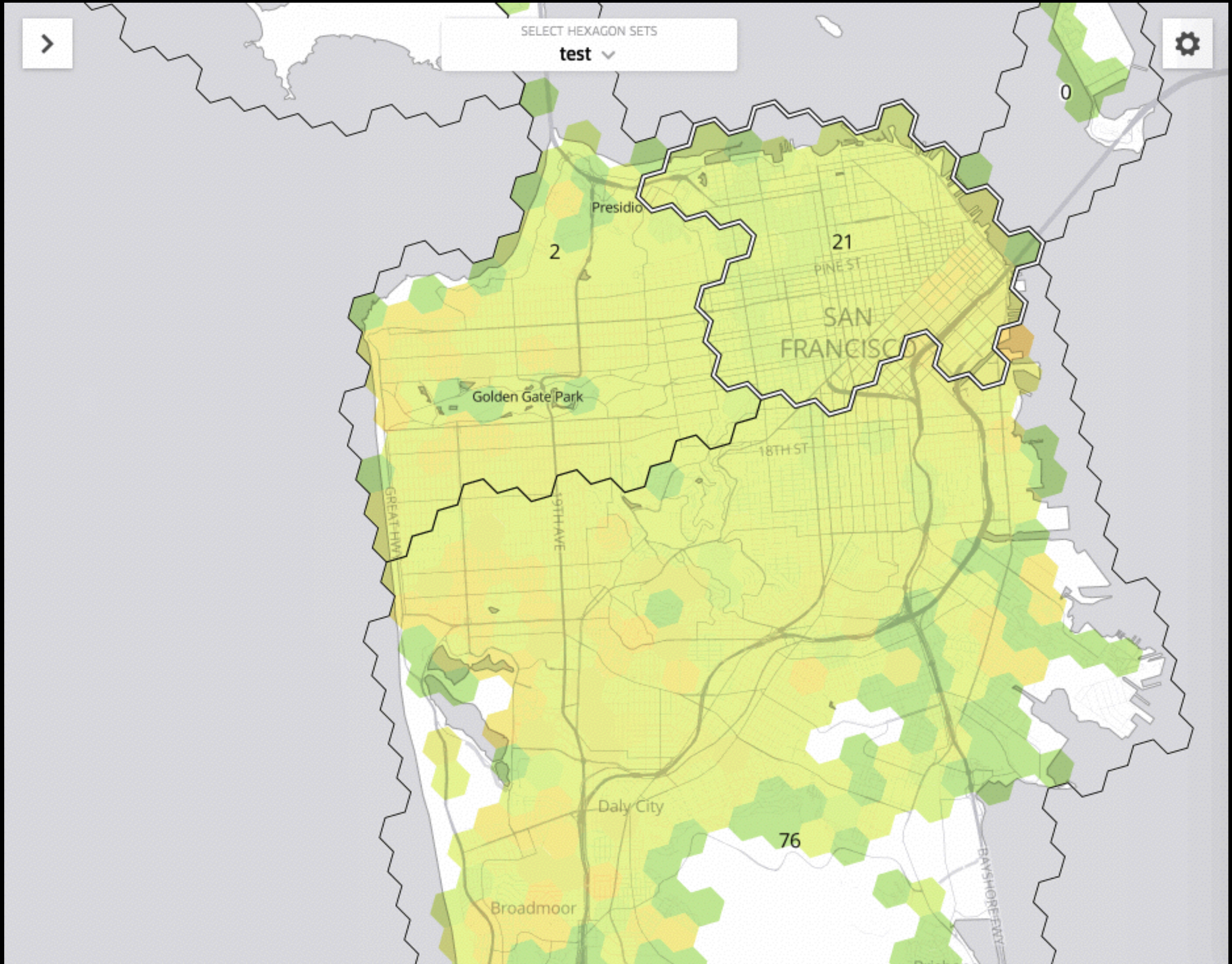
Geo: user flexibility is important

Geo: user flexibility is important

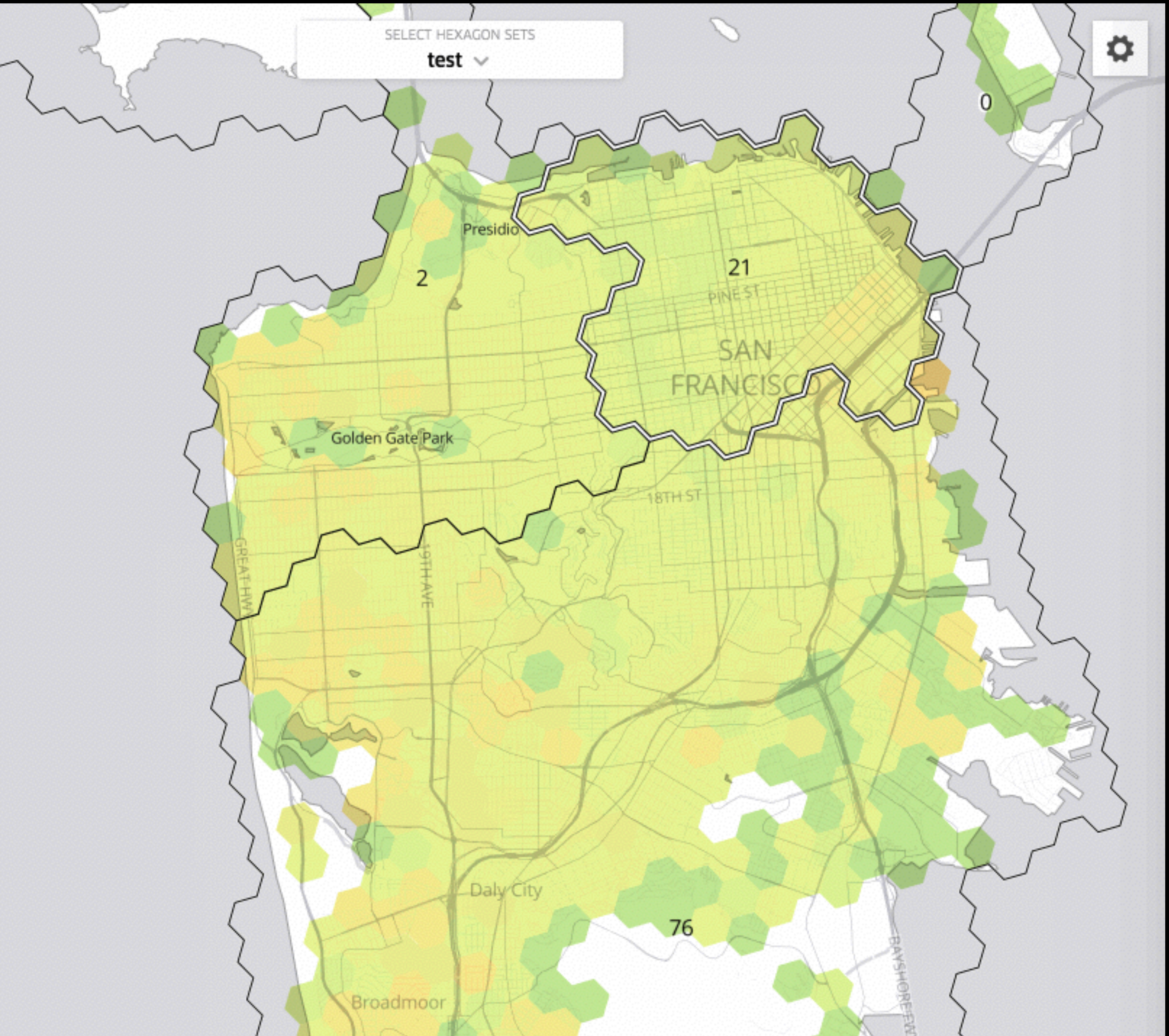


It needs to be scalable

It needs to be scalable



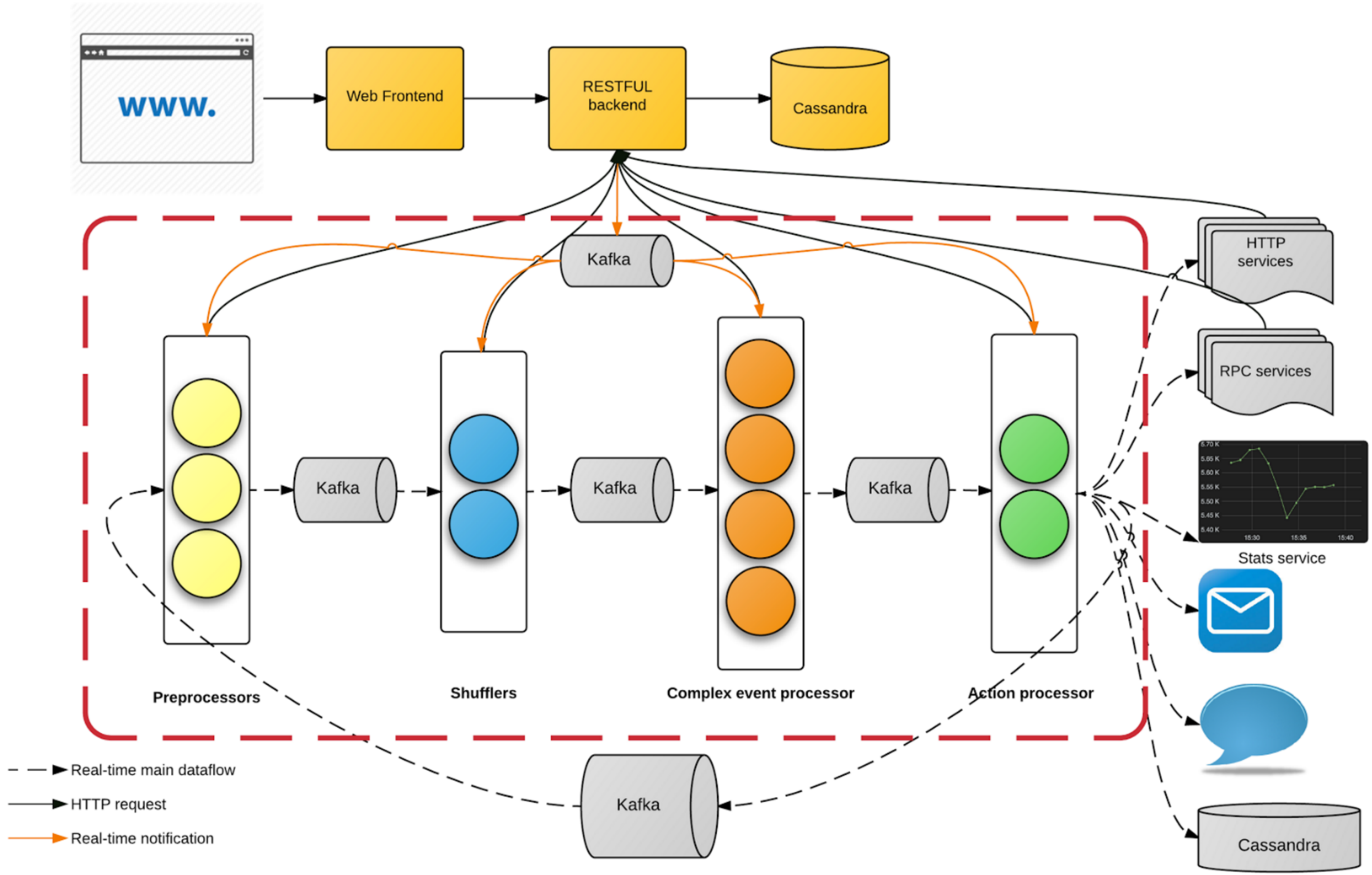
It needs to be scalable



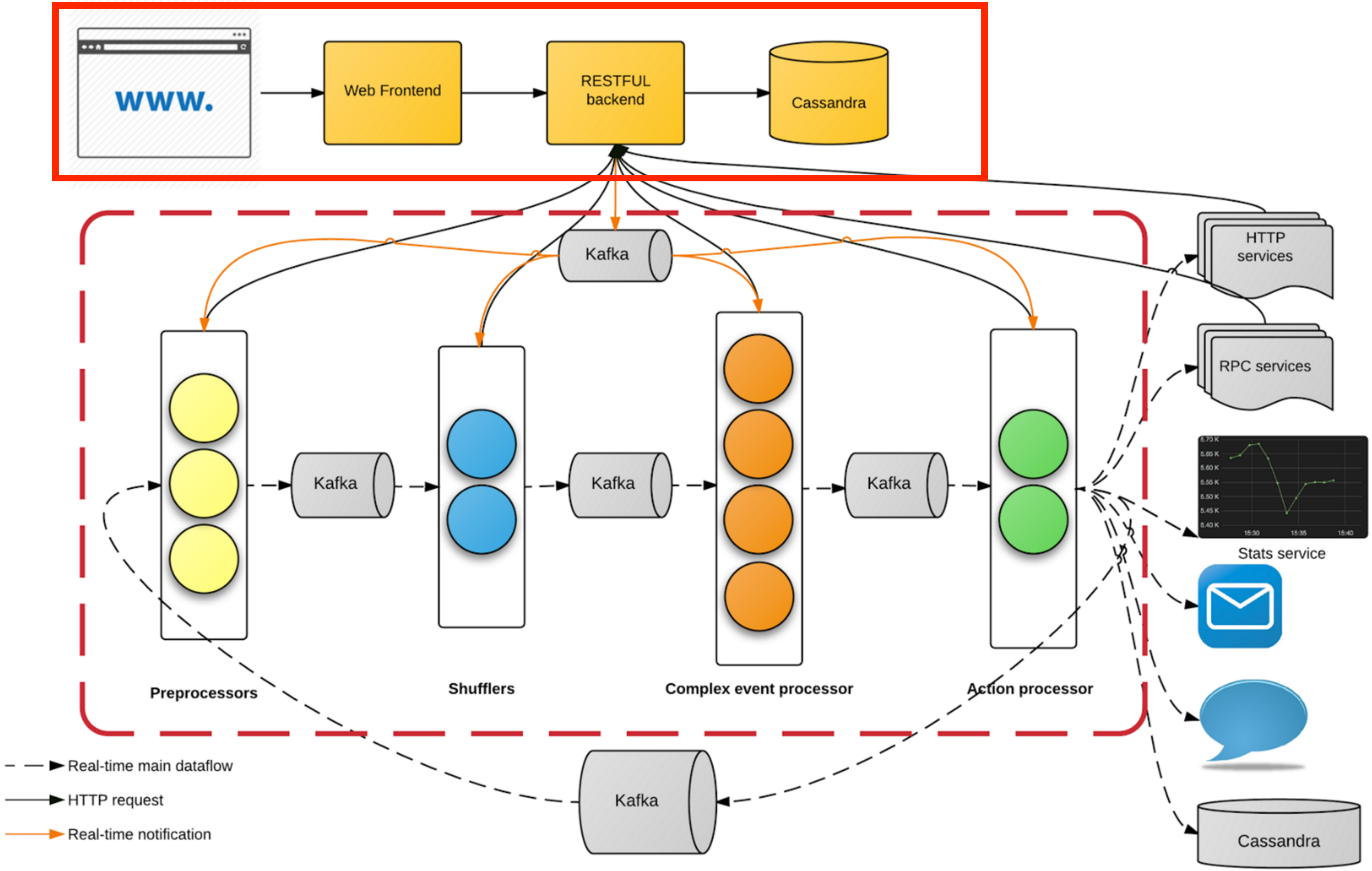
- Every hexagon
- Every driver/rider

CEP Pipeline Built on Samza

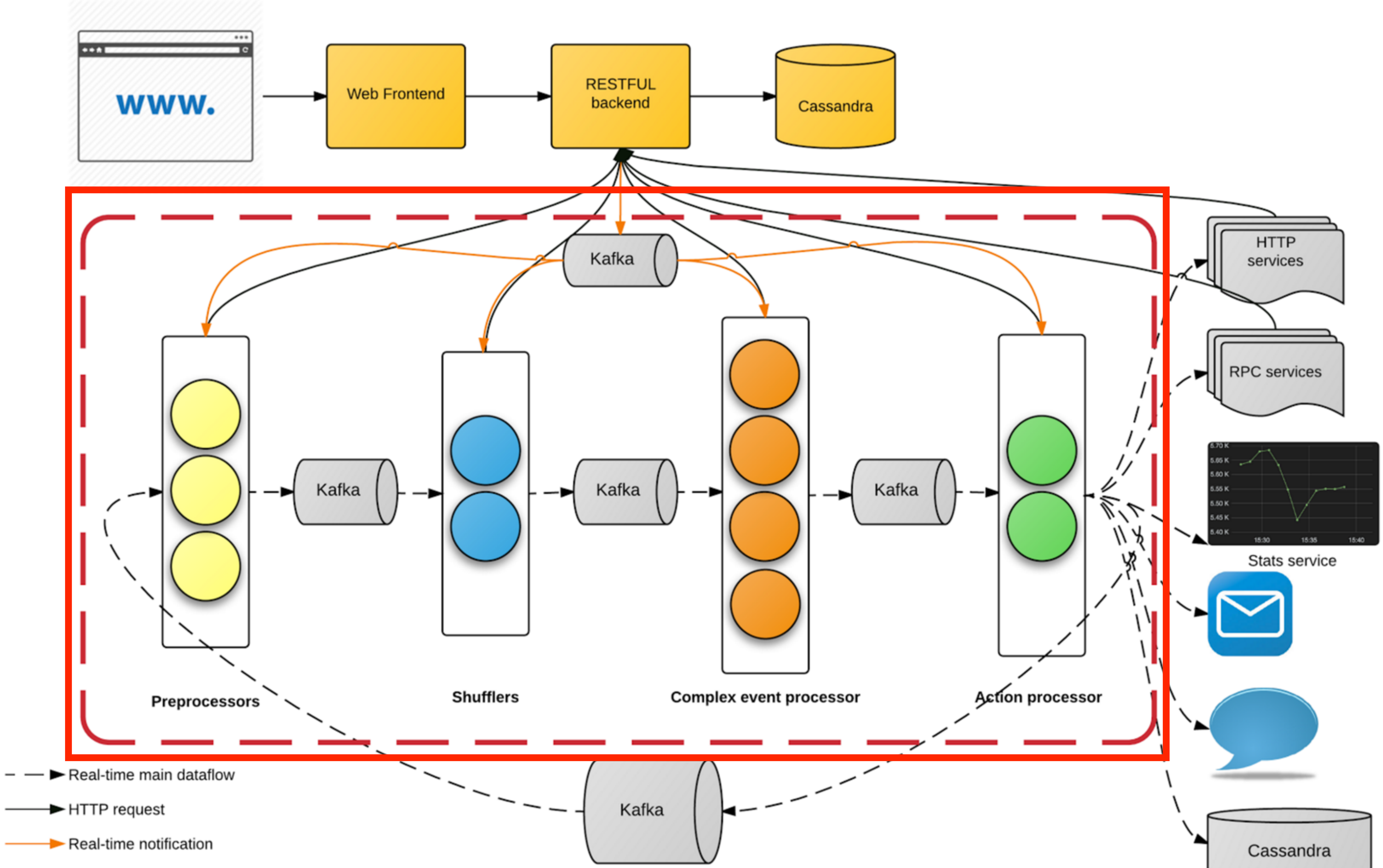
- No hard-coded CEP rules
- Applying CEP rules per individual entity: topic, driver, rider, cohorts, and etc
- Flexible checkpointing and statement management



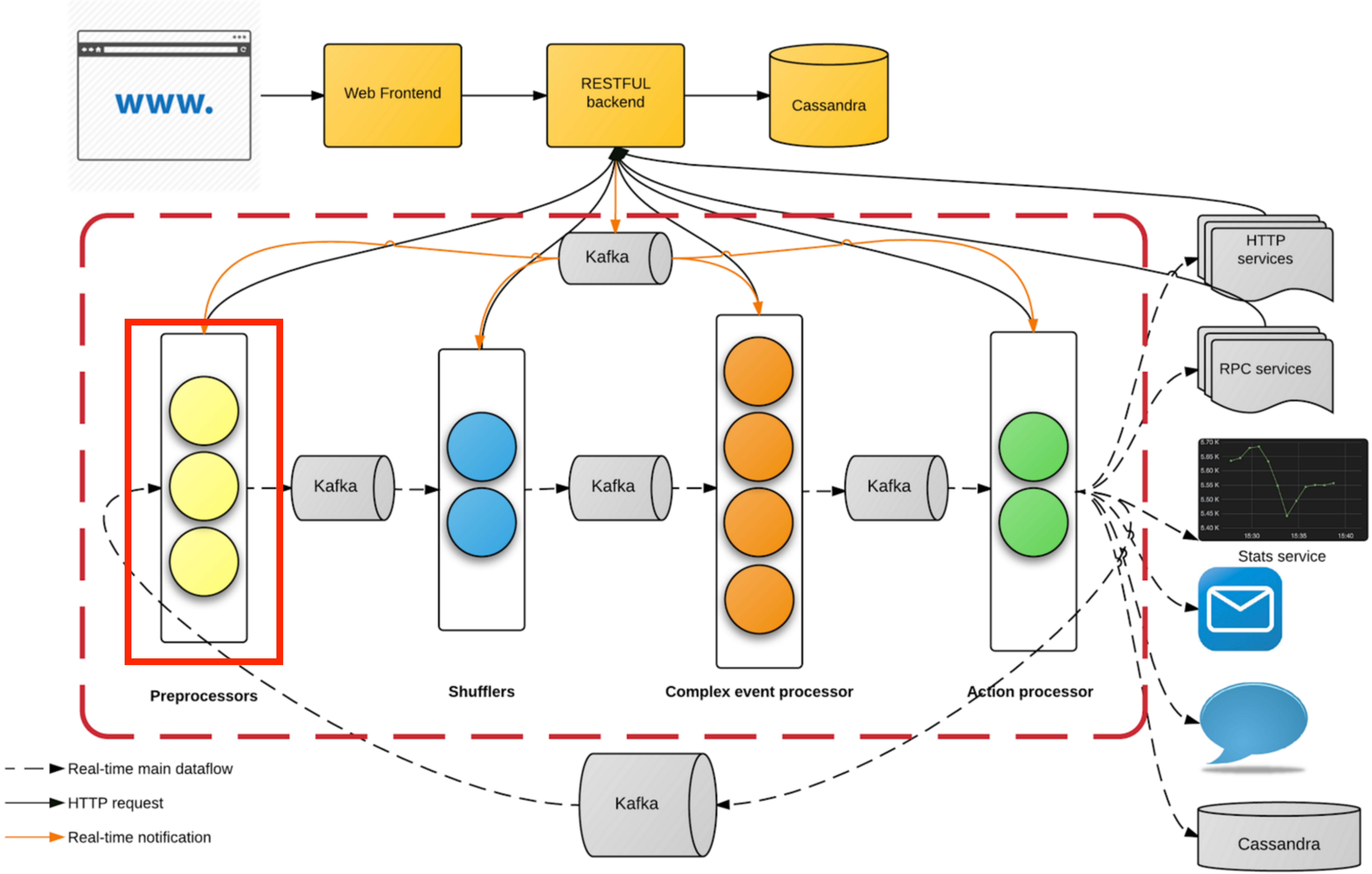
www.
Web Frontend
RESTFUL backend
Cassandra
Kafka
HTTP services
RPC services
Kafka
Kafka
Kafka
Stats service
Preprocessors
Shufflers
Complex event processor
Action processor
Kafka
Cassandra
Real-time main dataflow
HTTP request
Real-time notification

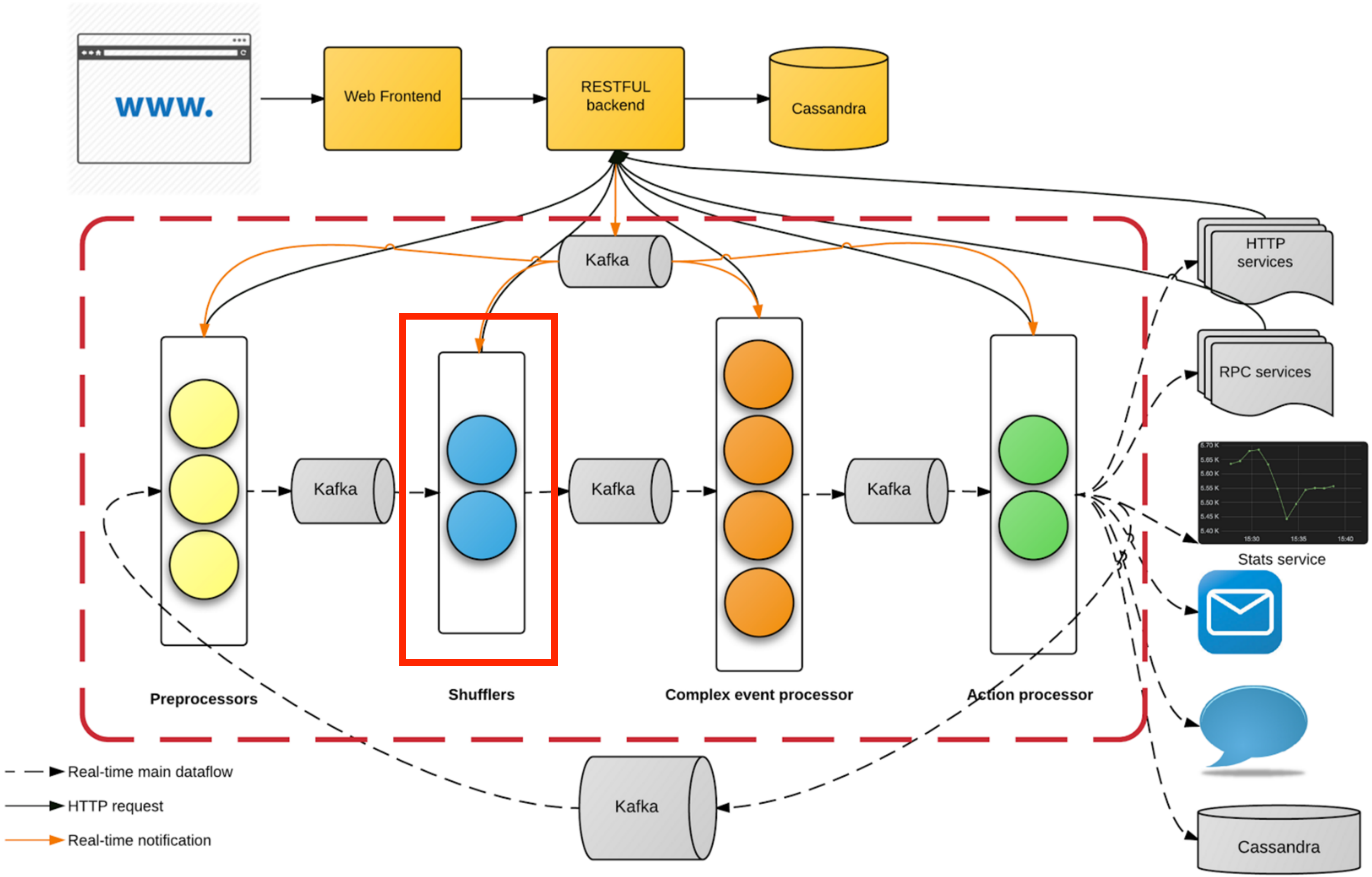


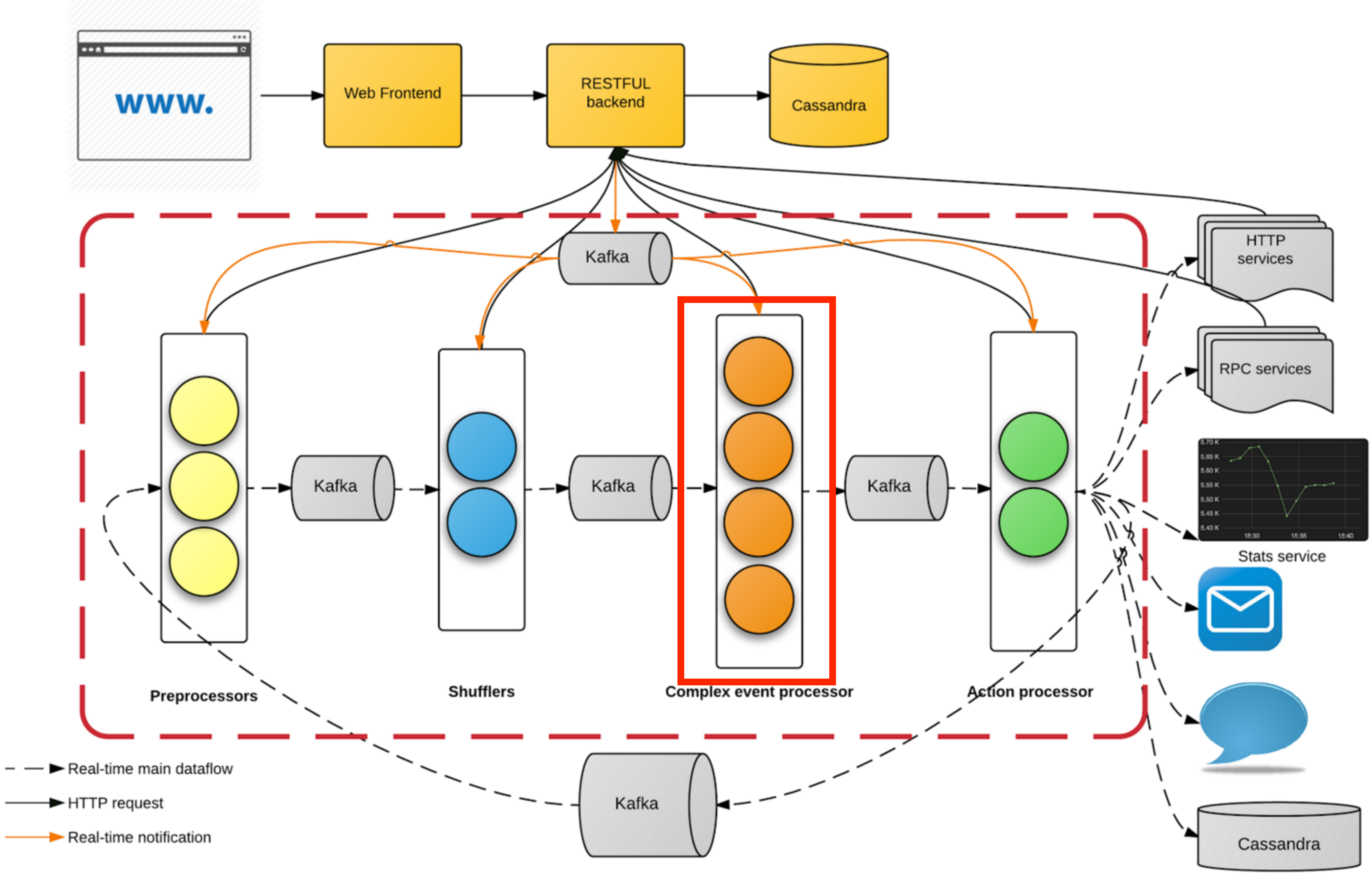
www.
Web Frontend
RESTFUL backend
Cassandra
Kafka
HTTP services
RPC services
Kafka
Kafka
Kafka
Stats service
Preprocessors
Shufflers
Complex event processor
Action processor
Real-time main dataflow
HTTP request
Real-time notification
Kafka
Cassandra

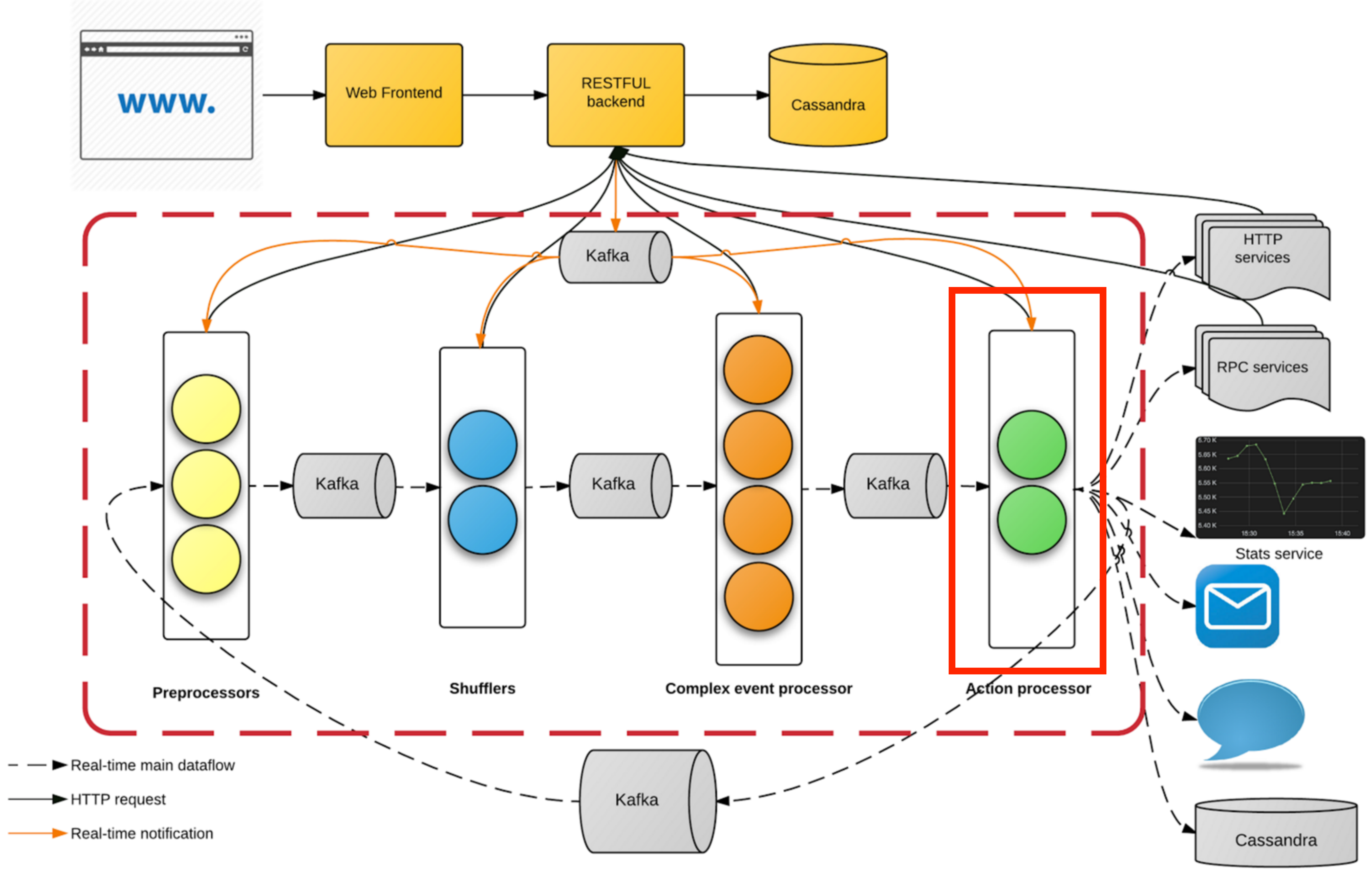


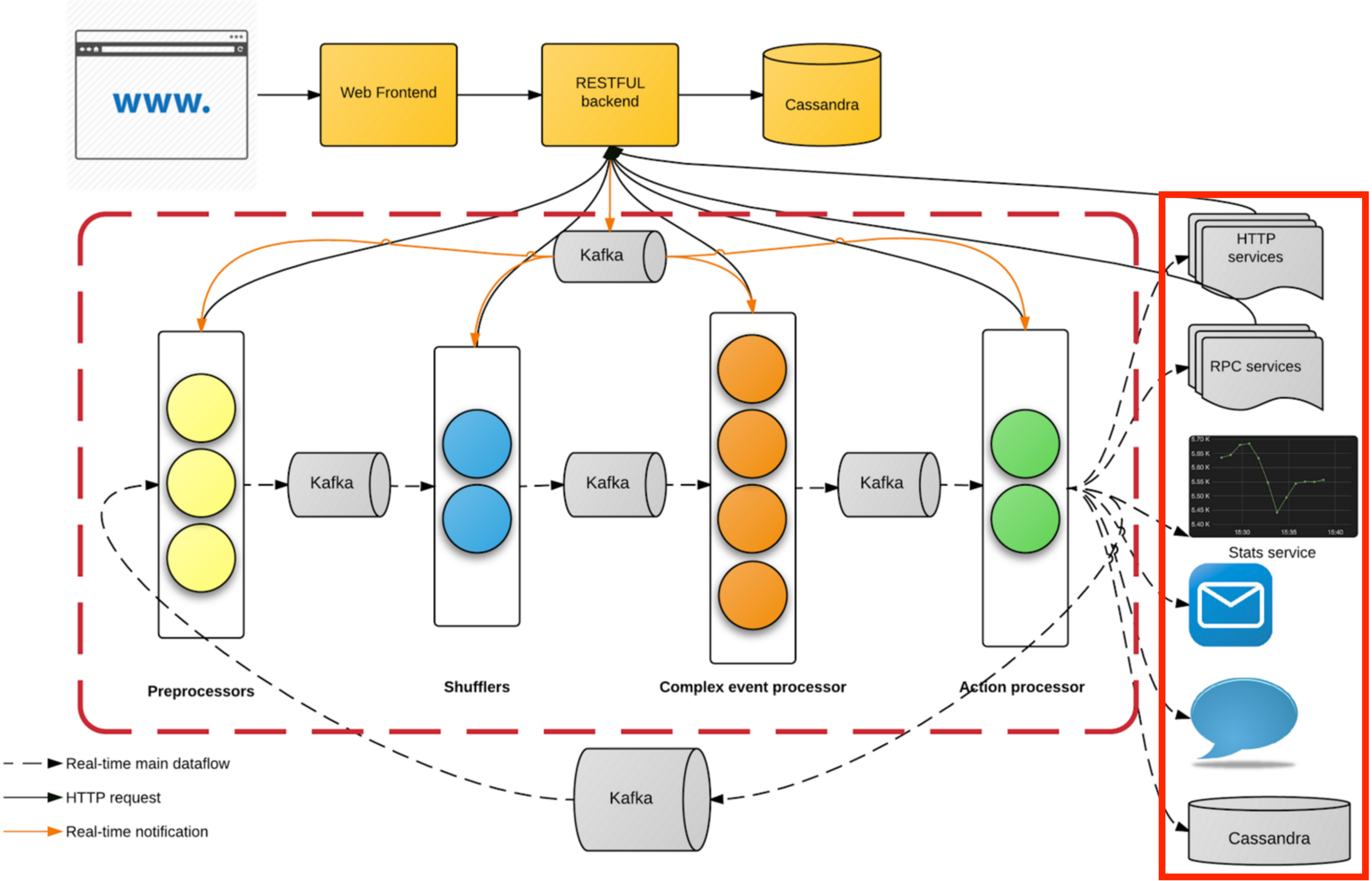
www.
Web Frontend
RESTFUL backend
Cassandra
Kafka
Kafka
Kafka
Kafka
Preprocessors
Shufflers
Complex event processor
Action processor
HTTP services
RPC services
Stats service
Kafka
Cassandra
Real-time main dataflow
HTTP request
Real-time notification


www.
Web Frontend
RESTFUL backend
Cassandra
Kafka
HTTP services
RPC services
Kafka
Kafka
Kafka
Stats service
Preprocessors
Shufflers
Complex event processor
Action processor
Real-time main dataflow
HTTP request
Real-time notification
Kafka
Cassandra


www.
Web Frontend
RESTFUL backend
Cassandra
Kafka
HTTP services
RPC services
Kafka
Kafka
Kafka
Stats service
Preprocessors
Shufflers
Complex event processor
Action processor
Kafka
Cassandra
Real-time main dataflow
HTTP request
Real-time notification


www.
Web Frontend
RESTFUL backend
Cassandra
Kafka
HTTP services
RPC services
Kafka
Kafka
Kafka
Stats service
Preprocessors
Shufflers
Complex event processor
Action processor
Kafka
Cassandra
Real-time main dataflow
HTTP request
Real-time notification


www.
Web Frontend
RESTFUL backend
Cassandra
Kafka
HTTP services
RPC services
Kafka
Kafka
Kafka
Stats service
Preprocessors
Shufflers
Complex event processor
Action processor
Real-time main dataflow
HTTP request
Real-time notification
Kafka
Cassandra

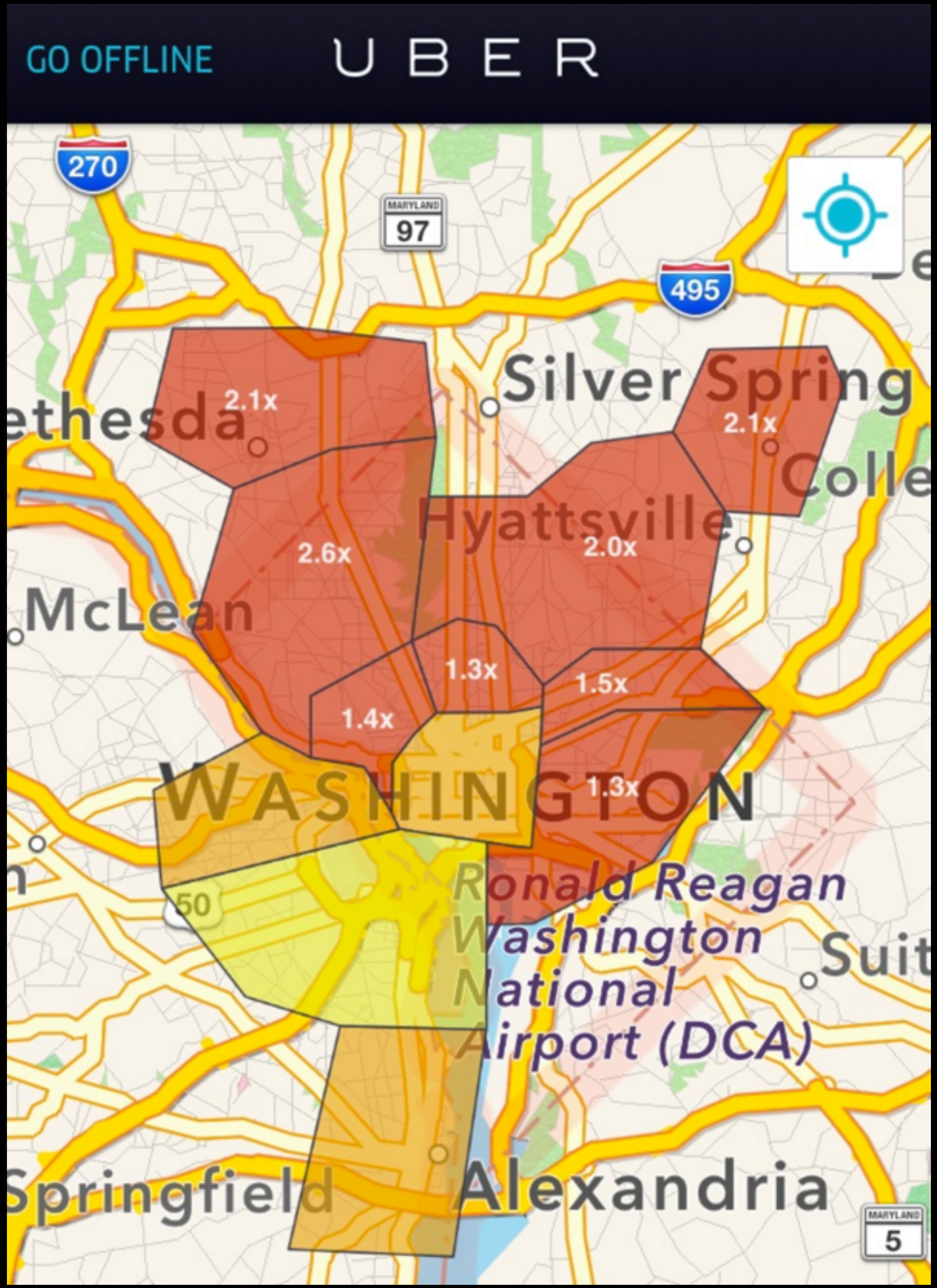


www.
Web Frontend
RESTFUL backend
Cassandra
Kafka
Kafka
Kafka
Kafka
Kafka
Preprocessors
Shufflers
Complex event processor
Action processor
HTTP services
RPC services
Stats service
Cassandra
Real-time main dataflow
HTTP request
Real-time notification

We need to evolve our architecture for other analytics

Clustering

Manually Created Cluster

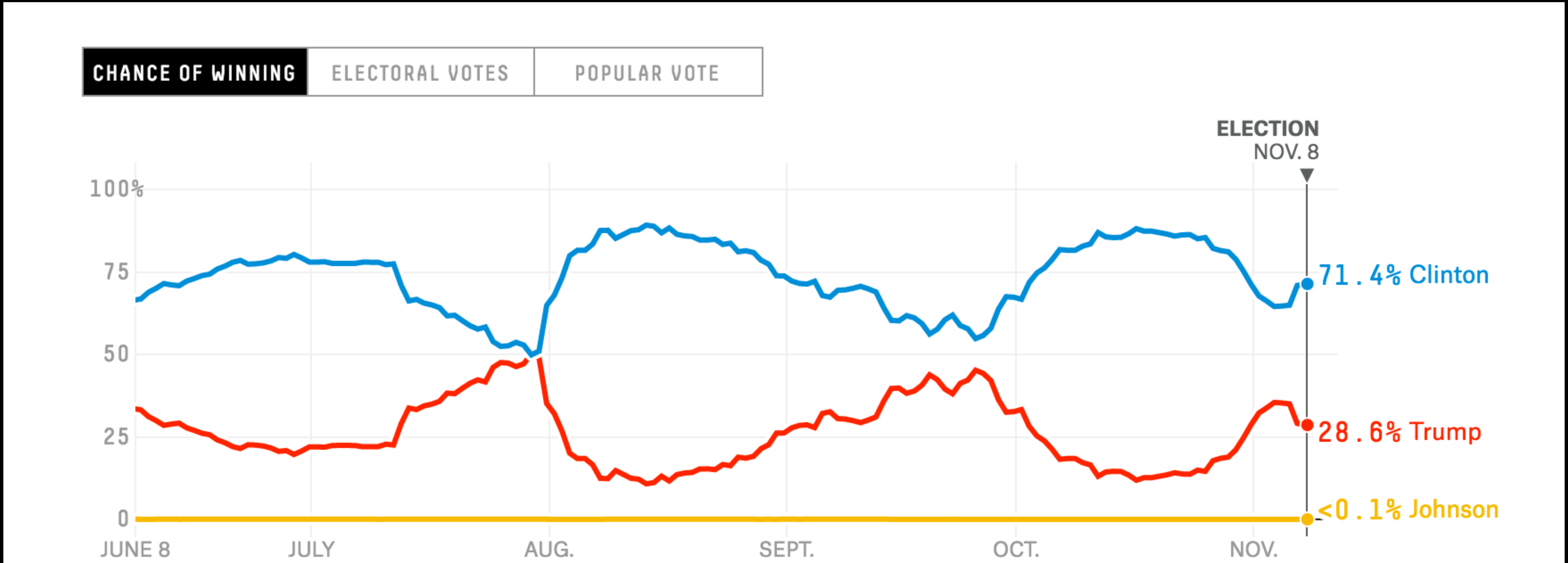


Call for algorithmically created clusters

- Clustering based on key performance metrics

Call for algorithmically created cluster

- Clustering based on key performance metrics
- Continuously measure the clusters



Call for algorithmically created clusters

- Clustering based on key performance metrics
- Continuously measure the clusters
- Different clustering for different business needs

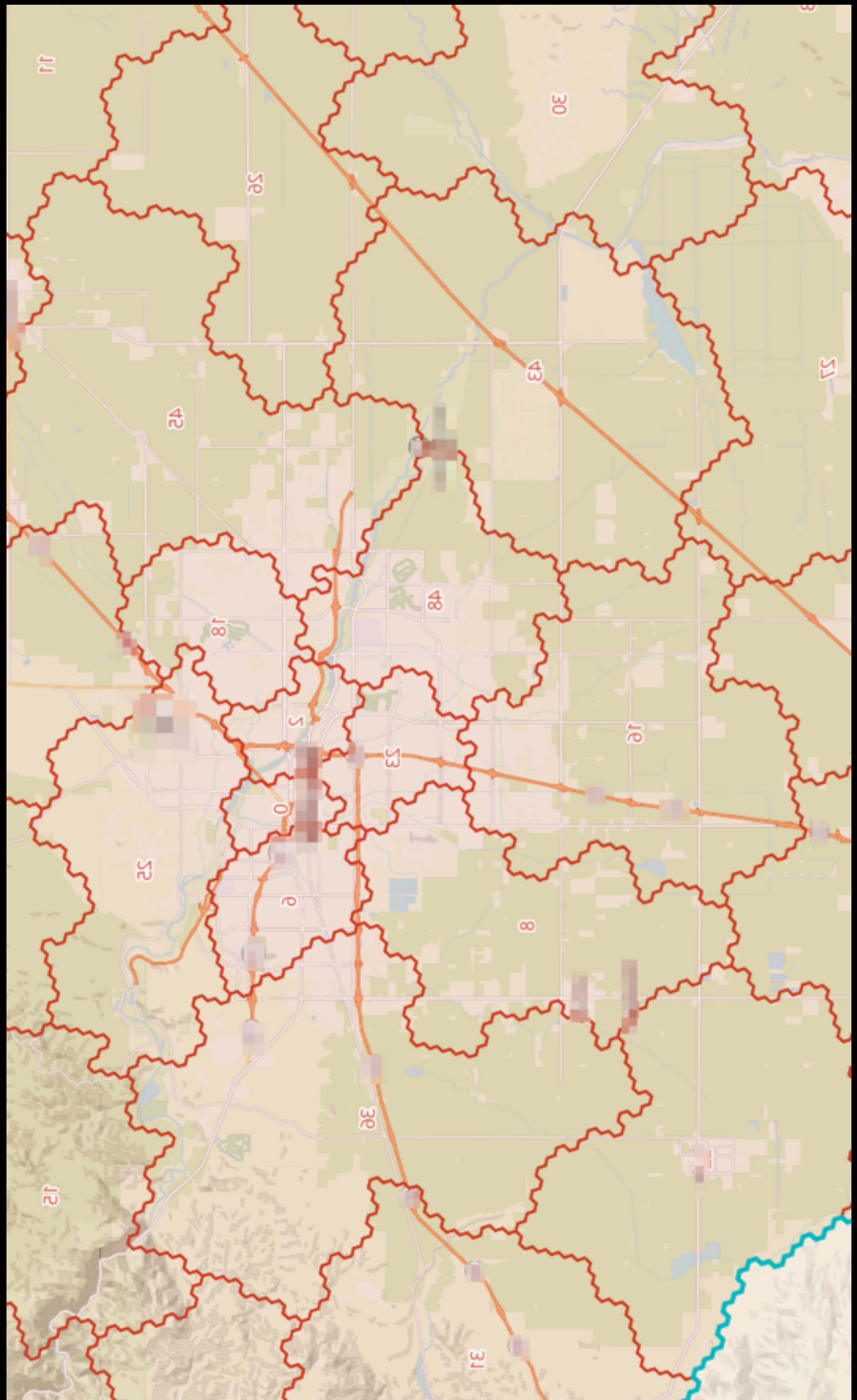
Call for algorithmically created clusters

- Clustering based on key performance metrics
- Continuously measure the clusters
- Different clustering for different business needs
- Create clusters in minutes for all cities

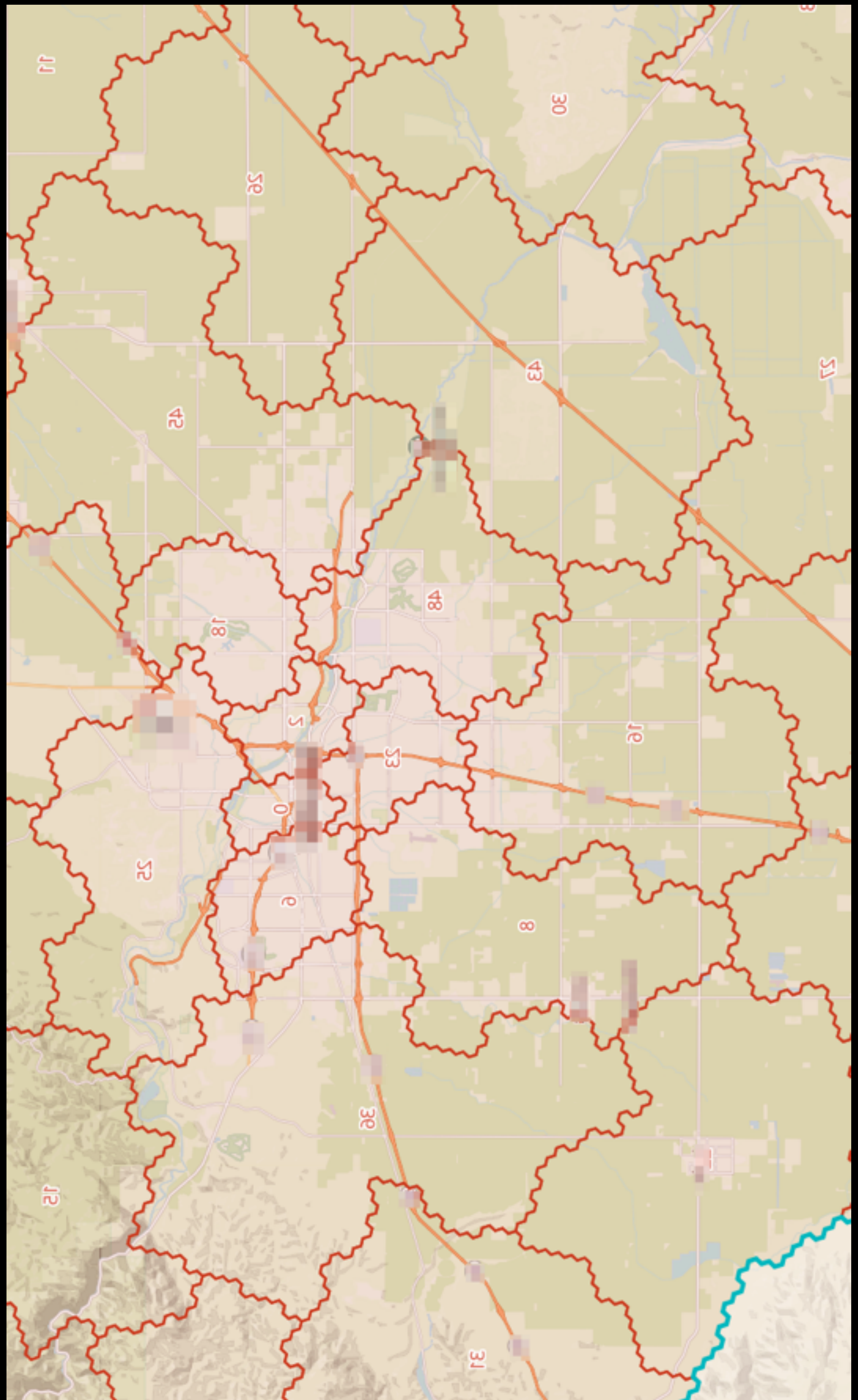
Call for algorithmically created clusters

- Clustering based on key performance metrics
- Continuously measure the clusters
- Different clustering for different business needs
- Create clusters in minutes for all cities
- Foundation for other stream analytics

Home-grown Clustering Service

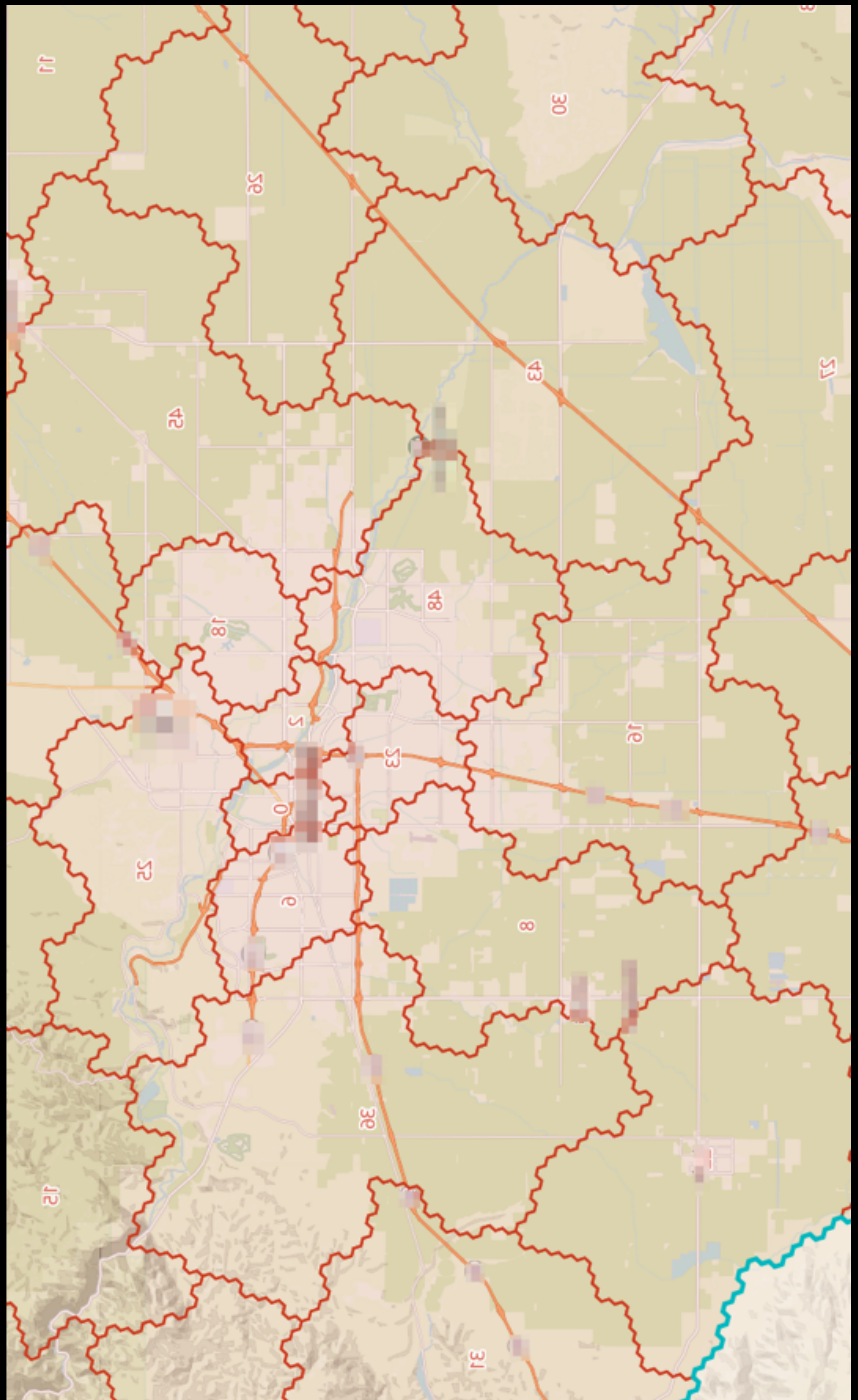


Home-grown Clustering Service



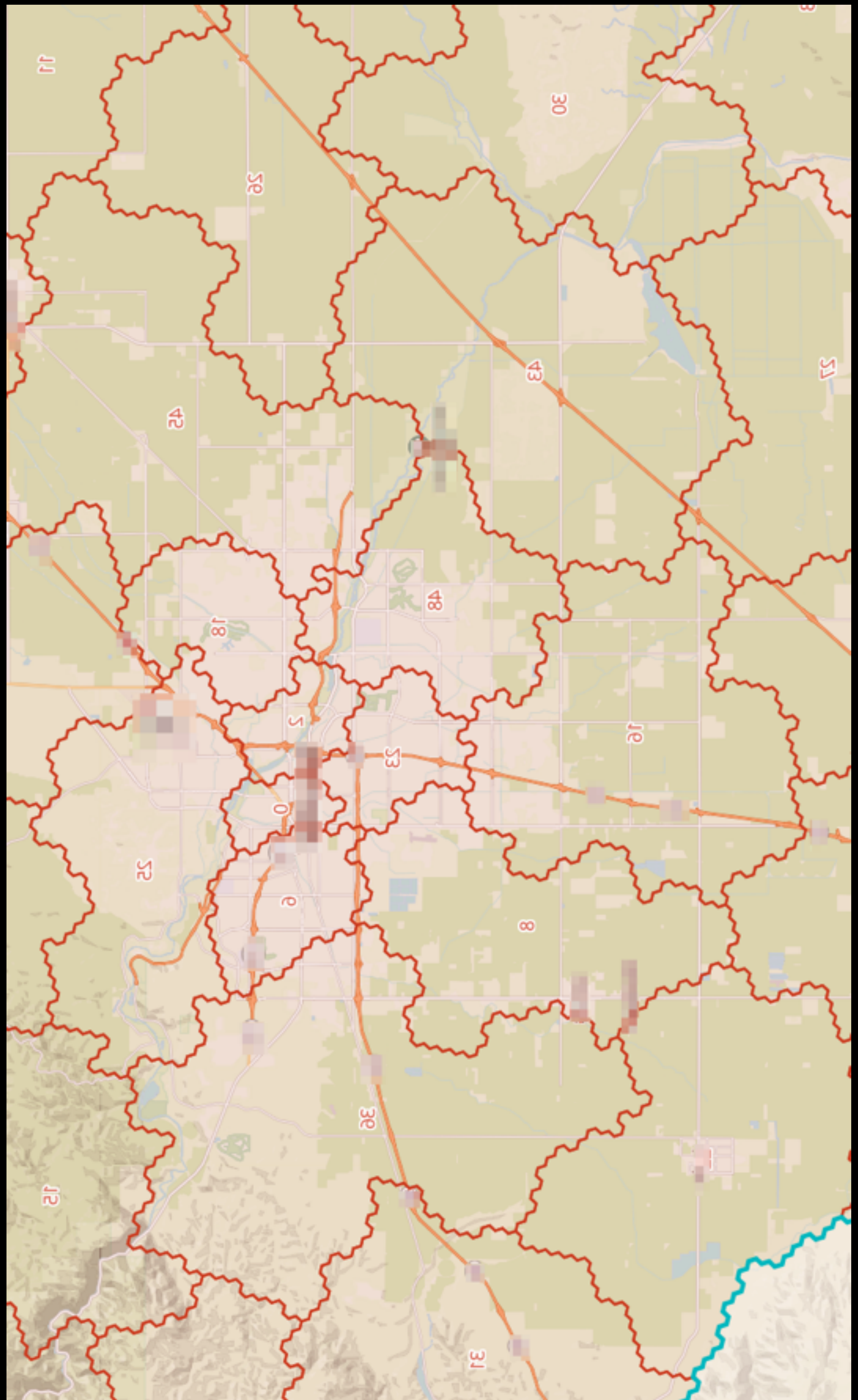
- All cities under 3 minutes

Home-grown Clustering Service



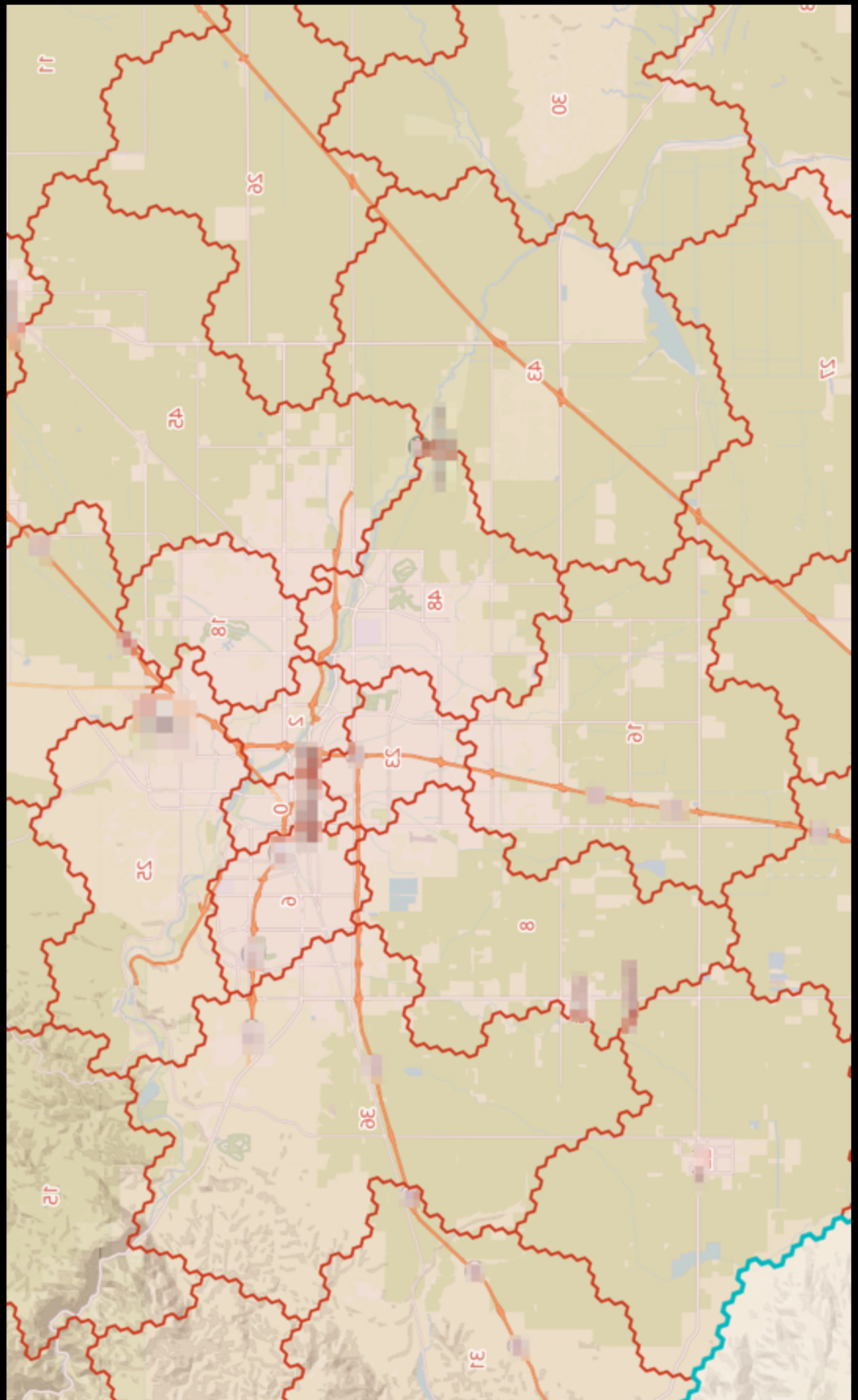
- All cities under 3 minutes
- Pluggable algorithms and measurements

Home-grown Clustering Service



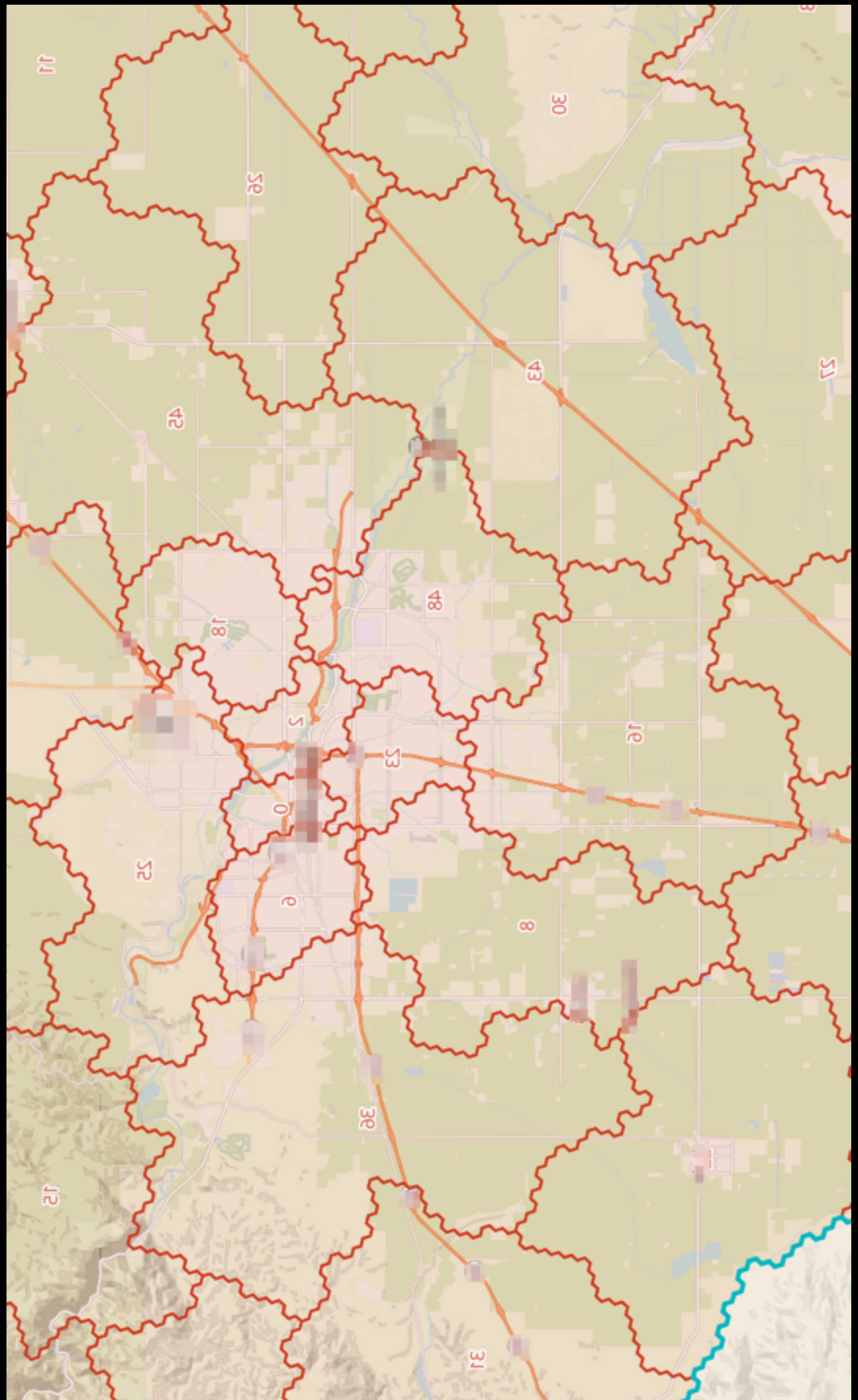
- All cities under 3 minutes
- Easily pluggable algorithms and measurements
- Historical geo-temporal data for clustering

Home-grown Clustering Service



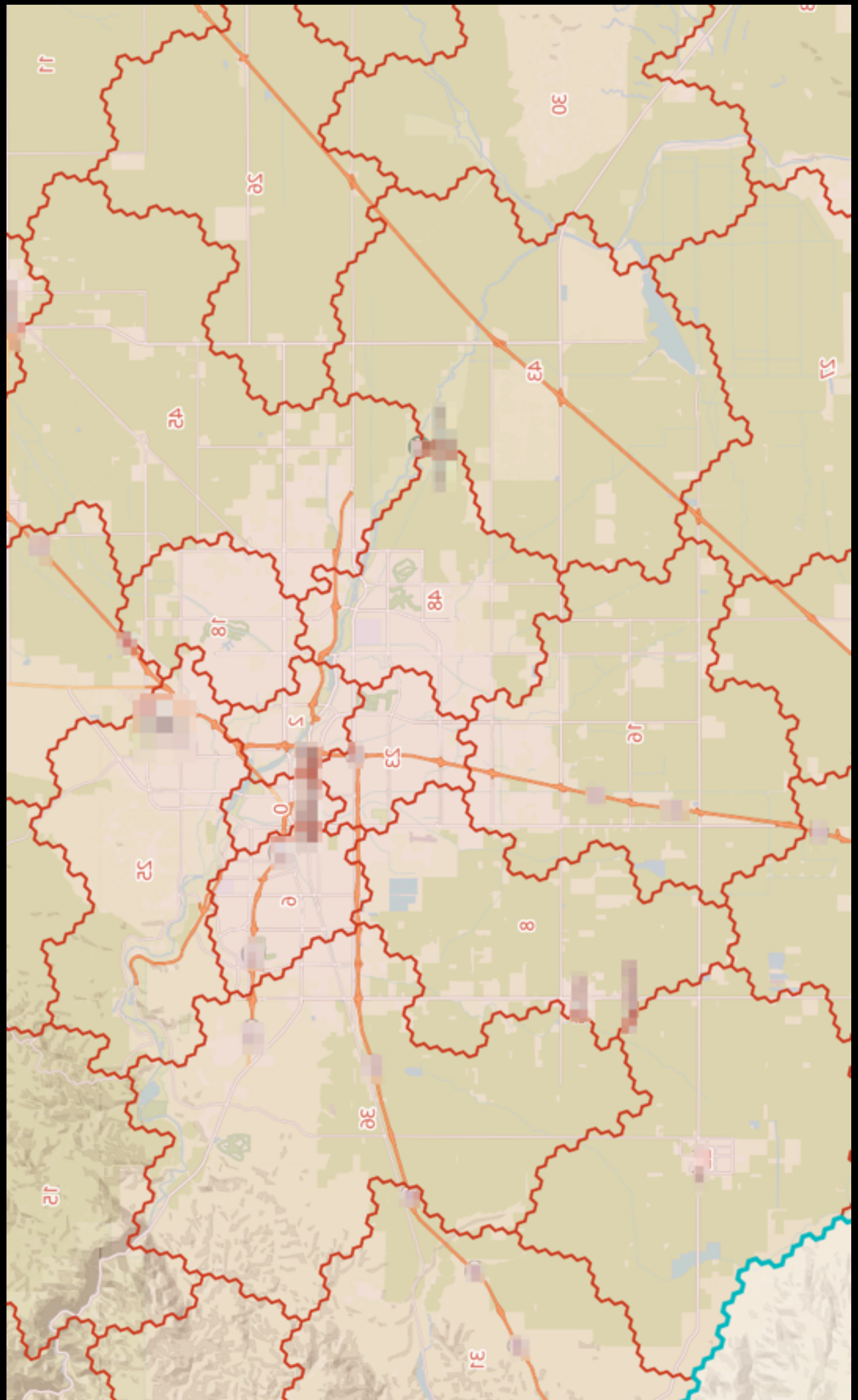
- All cities under 3 minutes
- Easily pluggable algorithms and measurements
- Historical geo-temporal data for clustering
- Real-time geo-temporal data for measurement

Home-grown Clustering Service



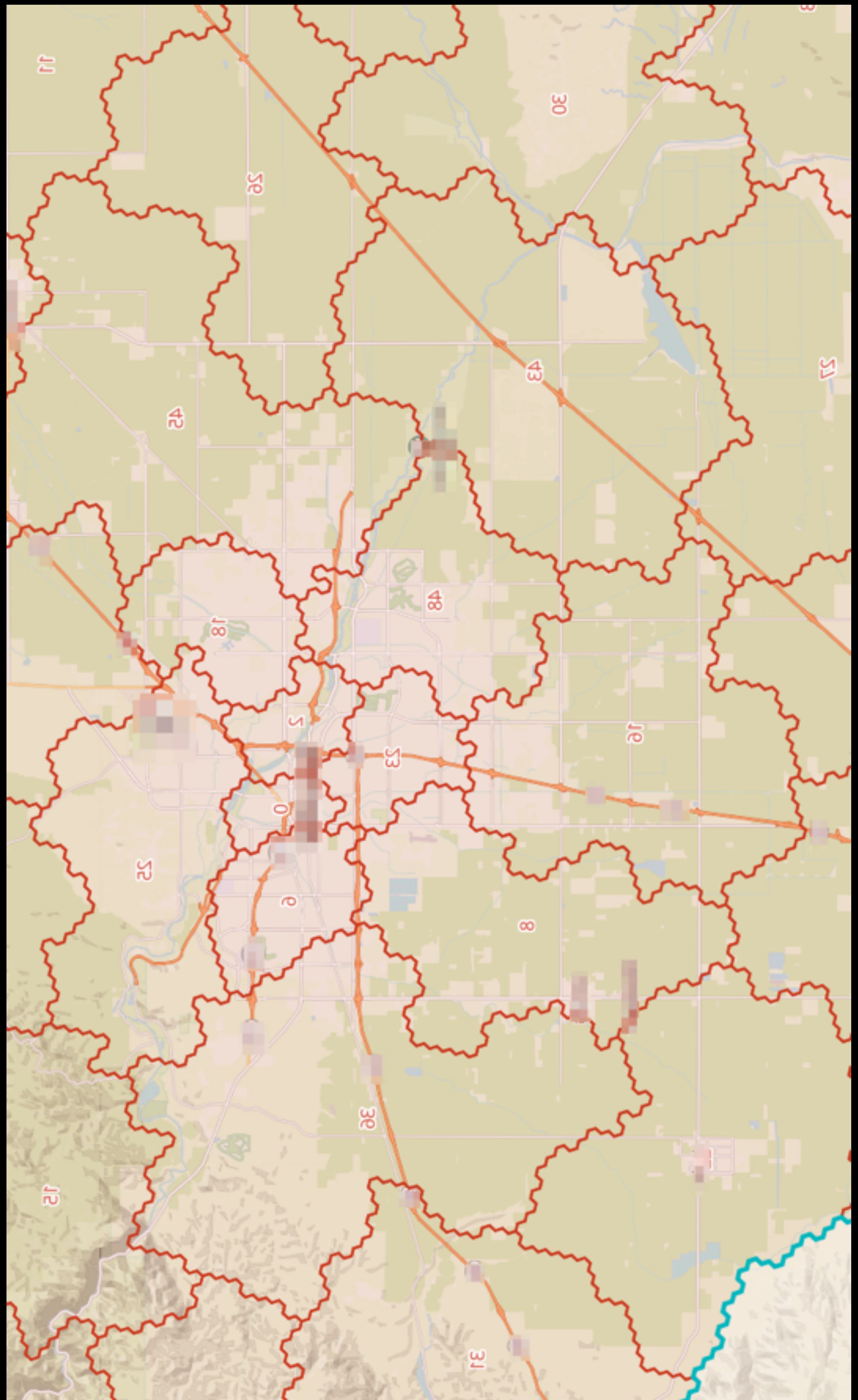
- All cities under 3 minutes
- Easily pluggable algorithms and measurements
- Historical geo-temporal data for clustering
- Real-time geo-temporal data for measurement
- Shared optimizations

Home-grown Clustering Service



- All cities under 3 minutes
- Easily pluggable algorithms and measurements
- Historical geo-temporal data for clustering
- Real-time geo-temporal data for measurement
- Shared optimizations. To put things in perspective:
    - 70,000 hexagons in SF
    - Naive distance function requires at least 70,000 x 70,000 = 4.9 billion pairs!

Home-grown Clustering Service



- All cities under 3 minutes
- Easily pluggable algorithms and measurements
- Historical geo-temporal data for clustering
- Real-time geo-temporal data for measurement
- Shared optimizations
  - Incremental updates
  - Compact data representation
  - Memoization
  - Avoid anything more complex than O(nlog(n))

Forecasting

- Every decision is based on forecasting

Forecasting

- Forecasting based on both historical data and stream input

Forecasting

- Forecasting based on both historical data and stream input

Forecasting

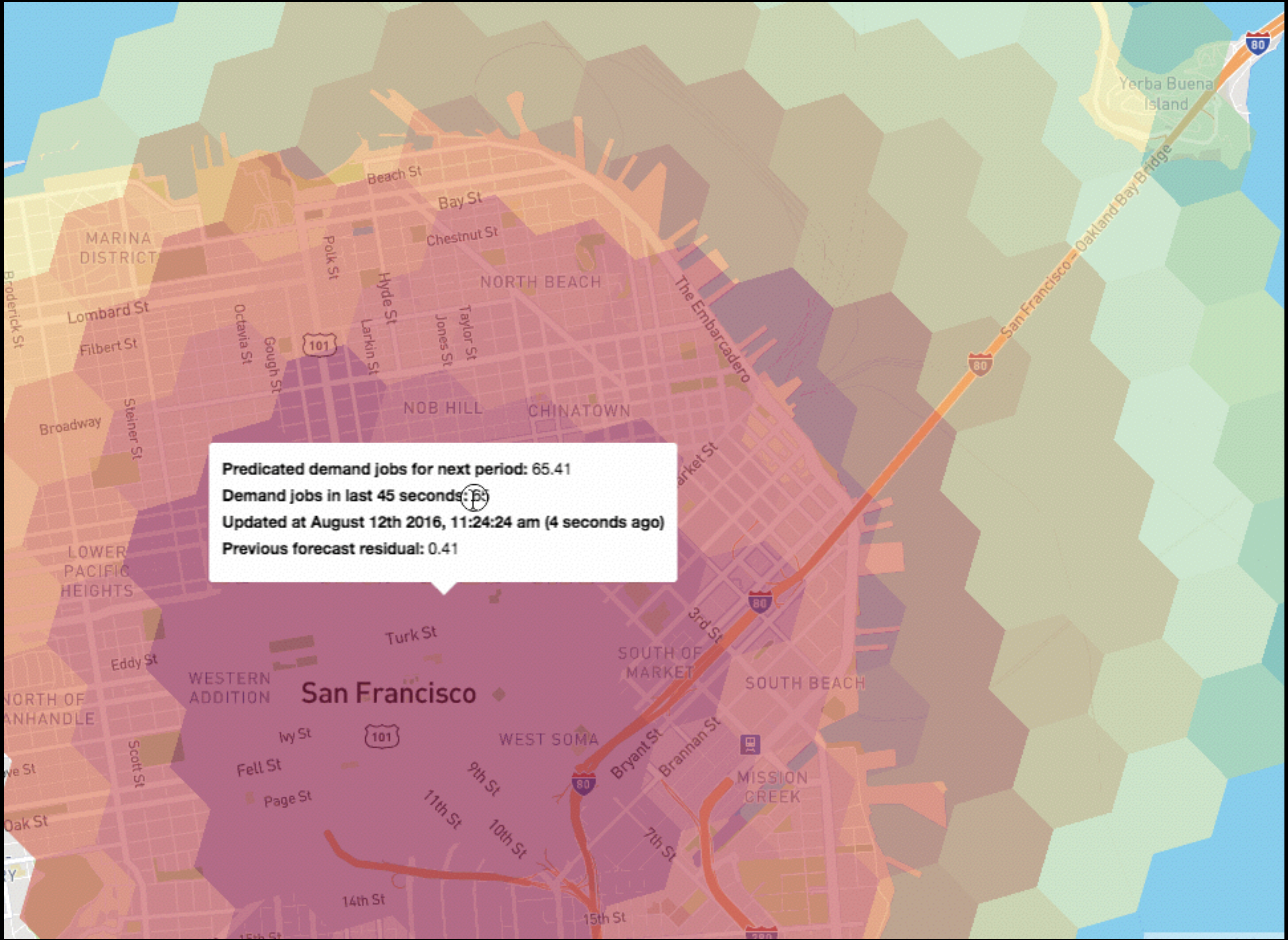
- Forecasting based on both historical data and stream input

Forecasting

- Spatially granular forecasting - down to every hexagon

Forecasting

- Spatially granular forecasting - down to every hexagon

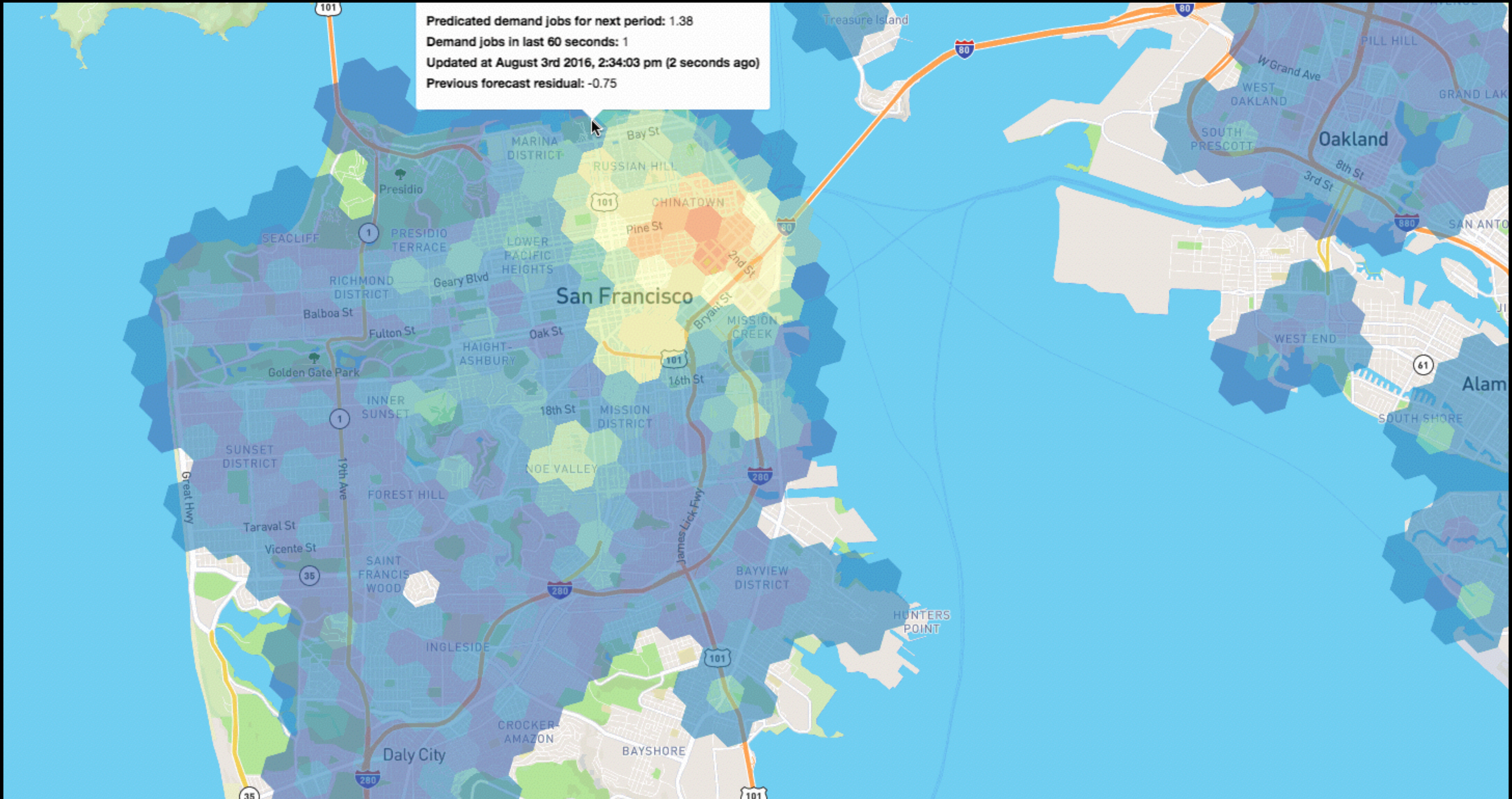


Forecasting

- Temporally granular forecasting - down to every minute

Forecasting

- Temporally granular forecasting - down to every minute



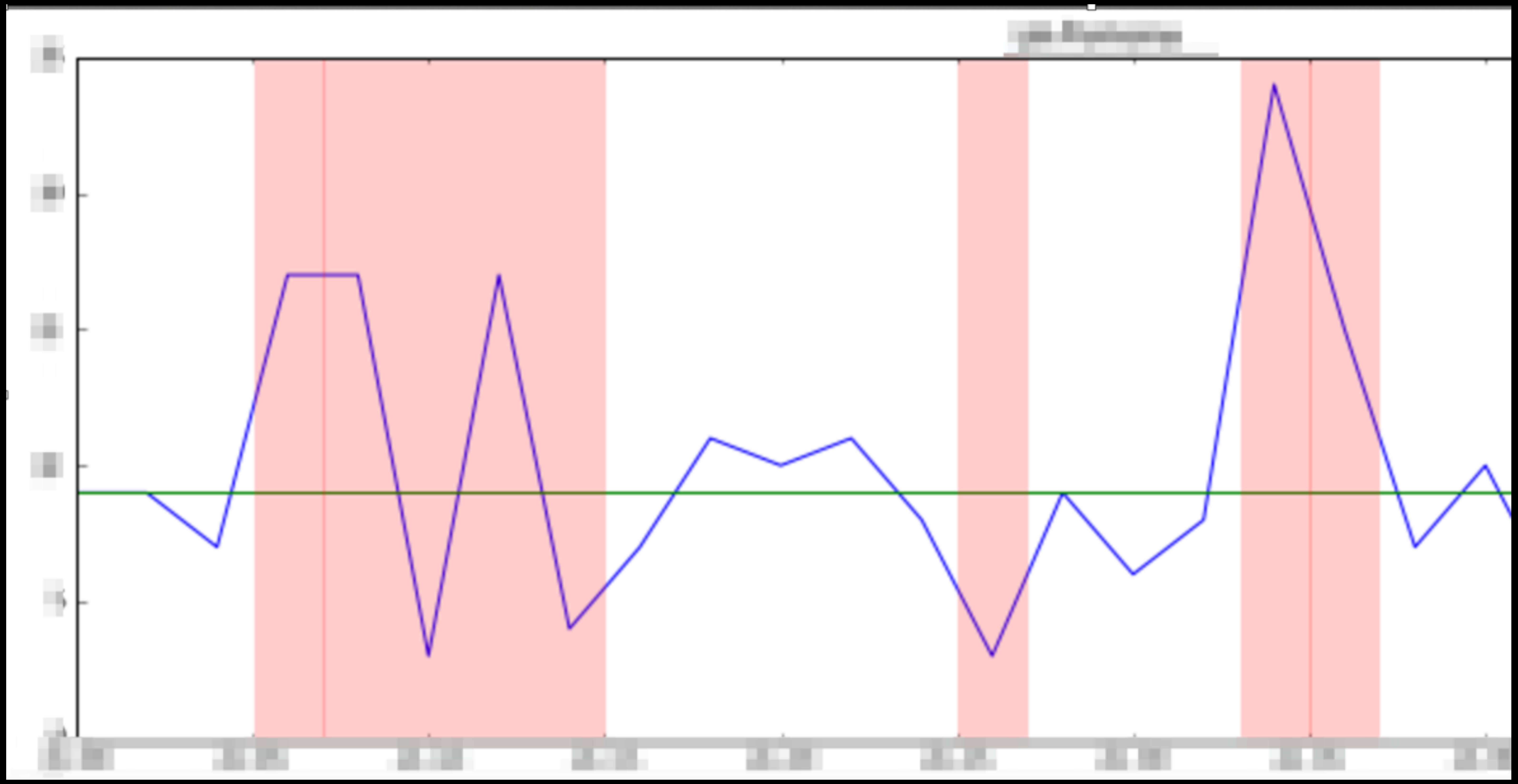
Pattern Detection

- Similarity of different metrics across geolocation and time
- Metric outliers across geolocations and time
- Frequent occurrences of certain patterns
- Clustered behavior
- Anomalies

Common Requirements in Pattern Detection

- Not just traditional time series analysis
- Incorporating insights on marketplace data
- Required both historical data and real-time input
- Spatially granular patterns - down to every hexagon
- Temporally granular patterns - down to every minute

Example: Anomaly Detection



- Simple time series analysis
- For a single geo area
- Can be noisy

A More Realistic Anomaly Detection

Example: Anomaly Detection

Example: Anomaly Detection

What’s the right architecture to support the analytics use cases?

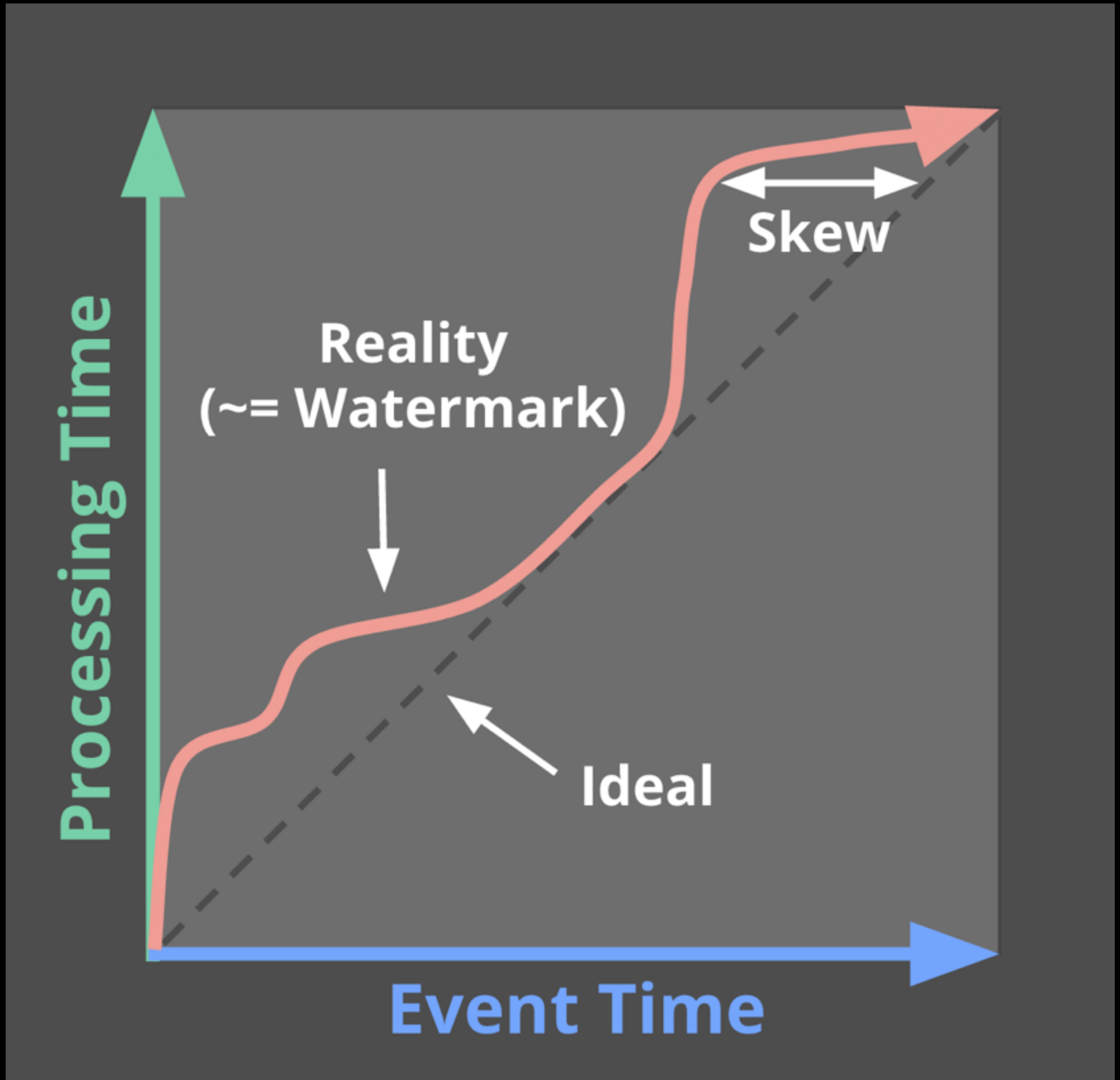
Shared abstraction: multi-dimensional geo-temporal data

Shared abstraction: multi-dimensional geo-temporal data

- Time series by event time

Shared abstraction: multi-dimensional geo-temporal data

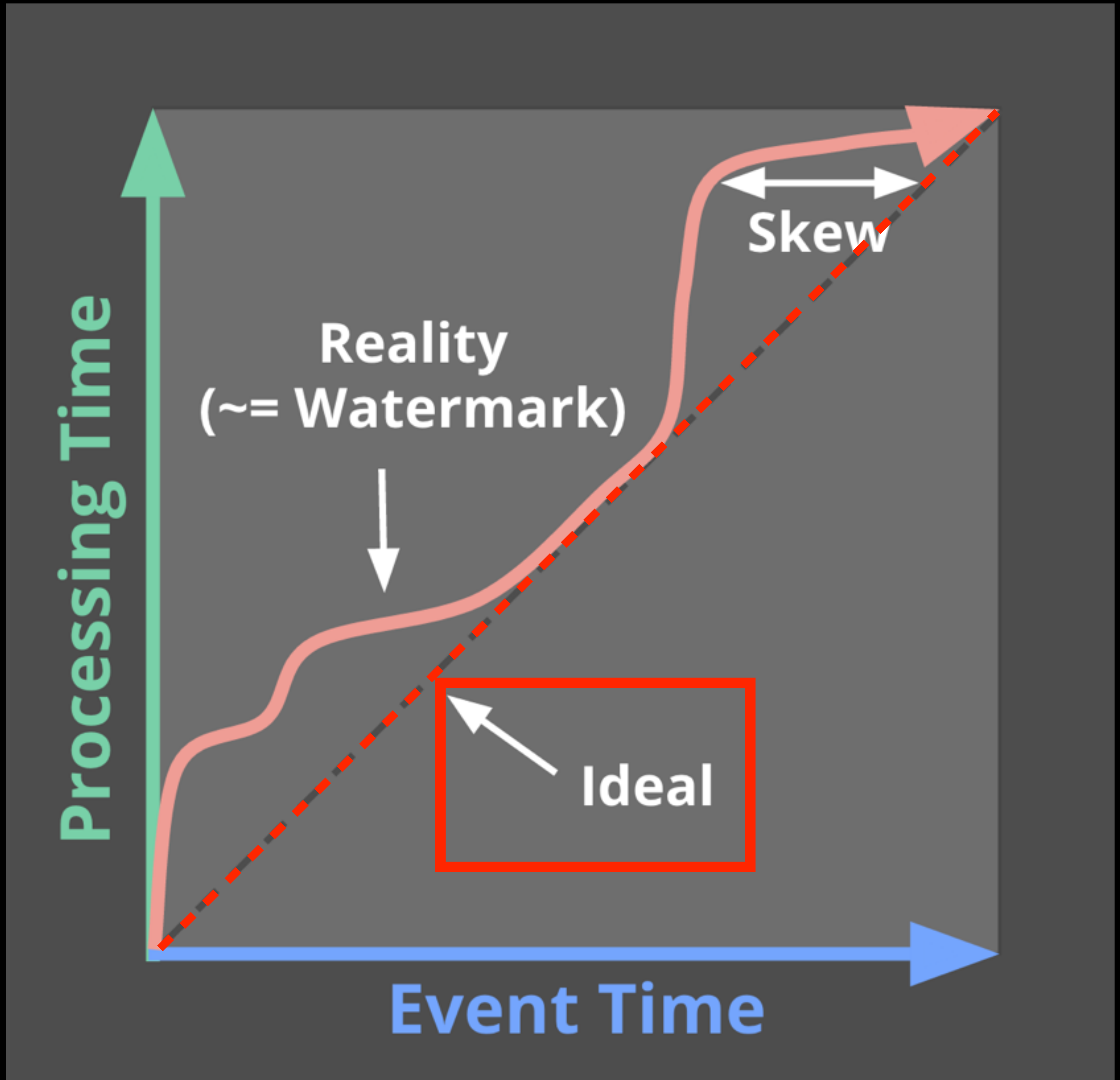
- Time series by event time



https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101

Shared abstraction: multi-dimensional geo-temporal data

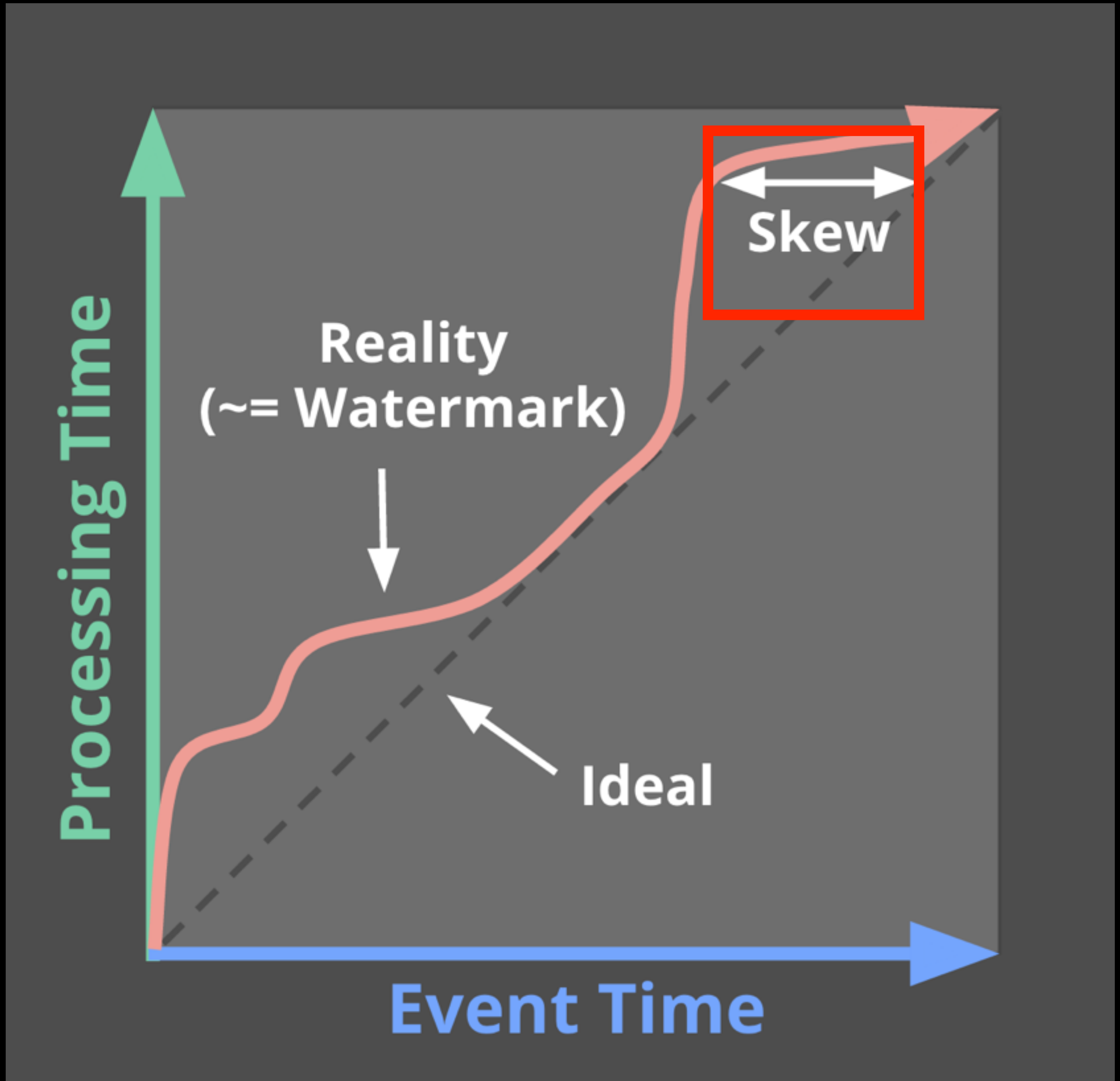
- Time series by event time



https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101

Shared abstraction: multi-dimensional geo-temporal data

- Time series by event time



https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101

Shared abstraction: multi-dimensional geo-temporal data

- Time series by event time
- Flexible windowing - tumbling, sliding, conditionally triggered

Shared abstraction: multi-dimensional geo-temporal data

- Time series by event time
- Flexible windowing - tumbling, sliding, conditionally triggered
  - e.g. event-based triggers

Shared abstraction: multi-dimensional geo-temporal data

- Time series by event time
- Flexible windowing - tumbling, sliding, conditionally triggered
  - e.g. event-based triggers
  - e.g., triggers of computation results

Shared abstraction: multi-dimensional geo-temporal data

- Time series by event time
- Flexible windowing - tumbling, sliding, conditionally triggered
- Stateful processing

Shared abstraction: multi-dimensional geo-temporal data

- Time series by event time
- Flexible windowing - tumbling, sliding, conditionally triggered
- Stateful processing. E.g.,

Shared abstraction: multi-dimensional geo-temporal data

- Time series by event time
- Flexible windowing - tumbling, sliding, conditionally triggered
- Stateful processing. E.g.,

$$\boxed{Y(t)} = \alpha X(t) + (1 - \alpha) Y(t - 1)$$

Shared abstraction: multi-dimensional geo-temporal data

- Time series by event time
- Flexible windowing - tumbling, sliding, conditionally triggered
- Stateful processing. E.g.,

$$Y(t) = \alpha \boxed{X(t)} + (1-\alpha)Y(t-1)$$

Shared abstraction: multi-dimensional geo-temporal data

- Time series by event time
- Flexible windowing - tumbling, sliding, conditionally triggered
- Stateful processing. E.g.,

$$Y(t) = \alpha X(t) + (1-\alpha)\boxed{Y(t-1)}$$

Shared abstraction: multi-dimensional geo-temporal data

- Time series by event time
- Flexible windowing - tumbling, sliding, conditionally triggered
- Stateful processing. E.g.,

$$Y(t) = \alpha X(t) + (1 - \alpha)\underbrace{\boxed{Y(t-1)}}_{\text{State}}$$

Shared abstraction: multi-dimensional geo-temporal data

- Time series by event time
- Flexible windowing - tumbling, sliding, conditionally triggered
- Stateful processing. E.g.,

$$Y(t) = \alpha X(t) + (1-\alpha)\boxed{Y(t-1)}$$

State per key

Shared abstraction: multi-dimensional geo-temporal data

- Time series by event time
- Flexible windowing - tumbling, sliding, conditionally triggered
- Stateful processing
- Unified stream

Shared abstraction: multi-dimensional geo-temporal data

- Time series by event time
- Flexible windowing - tumbling, sliding, conditionally triggered
- Stateful processing
- Unified stream
    - Real-time streams: unbounded streams

Shared abstraction: multi-dimensional geo-temporal data

- Time series by event time
- Flexible windowing - tumbling, sliding, conditionally triggered
- Stateful processing
- Unified stream
  - Real-time streams: unbounded streams
  - Batch: bounded streams

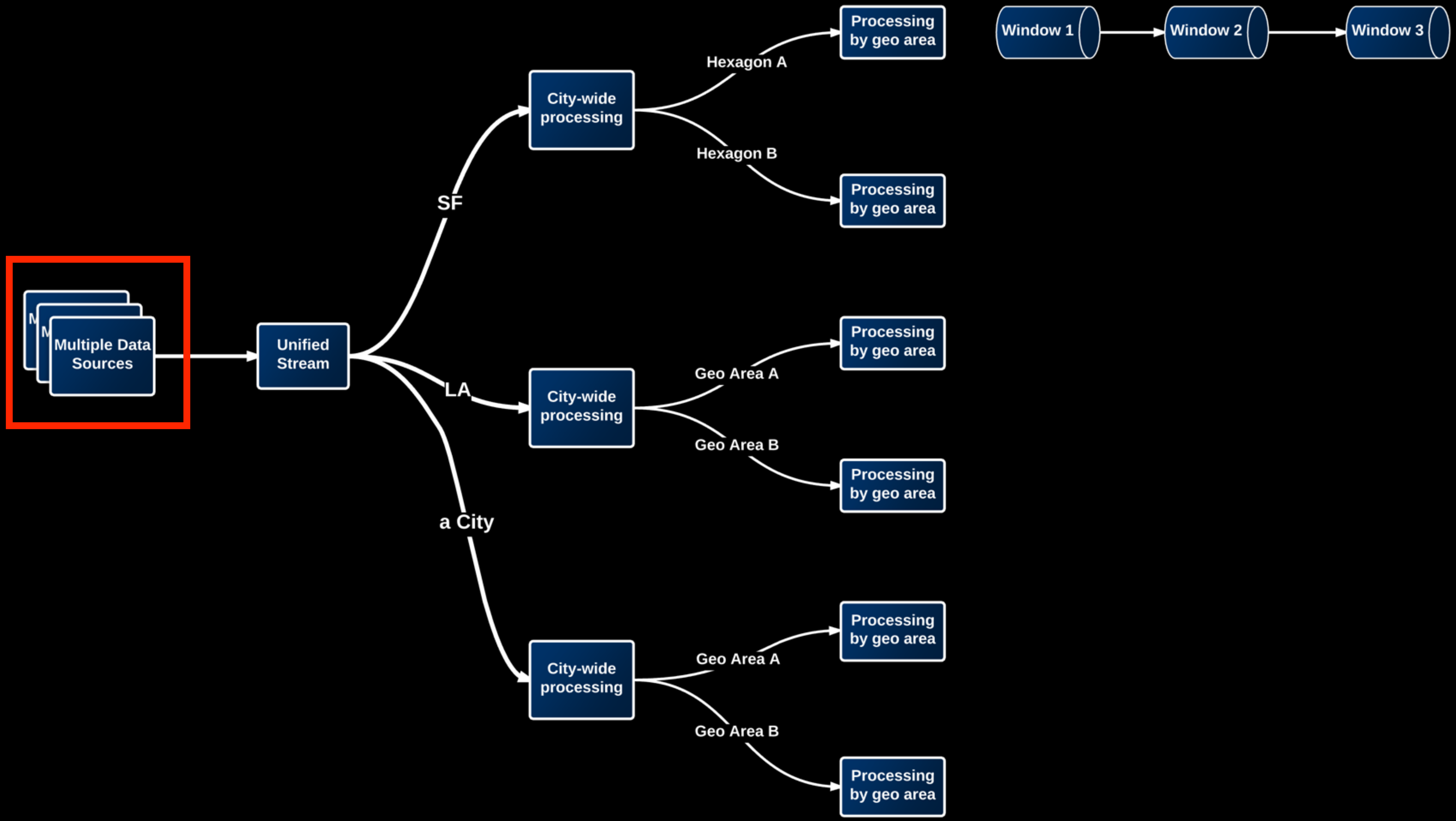
Shared abstraction: multi-dimensional geo-temporal data

- Time series by event time
- Flexible windowing - tumbling, sliding, conditionally triggered
- Stateful processing
- Unified stream
    - Real-time streams: unbounded streams
    - Batch: bounded streams
    - s/lambda/kappa

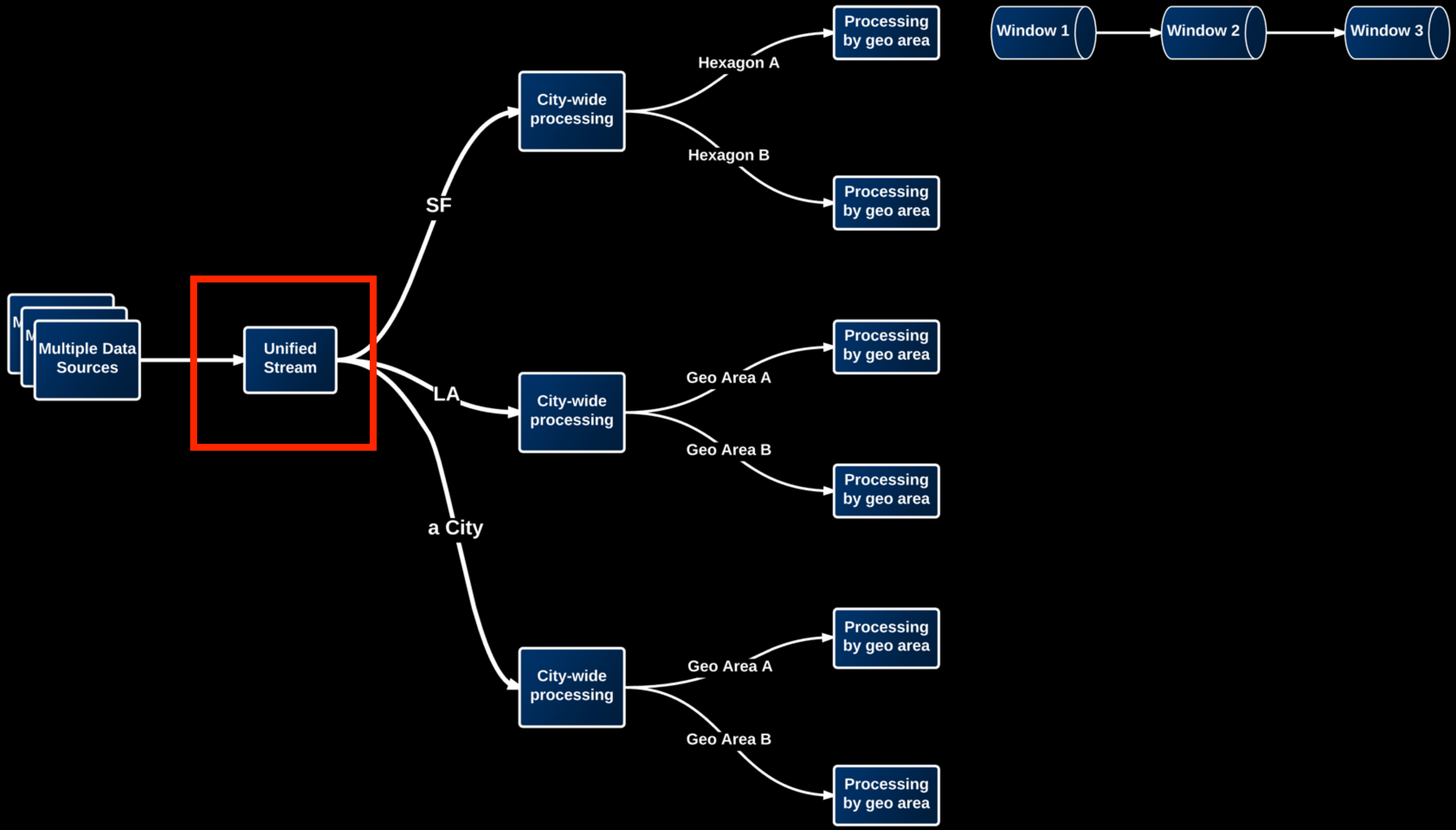
Apache Flink

- Ordering by event time
- Flexible windowing with watermark and triggers
- Exactly-once semantics
- Built-in state management and checkpointing
- Nice data flow APIs

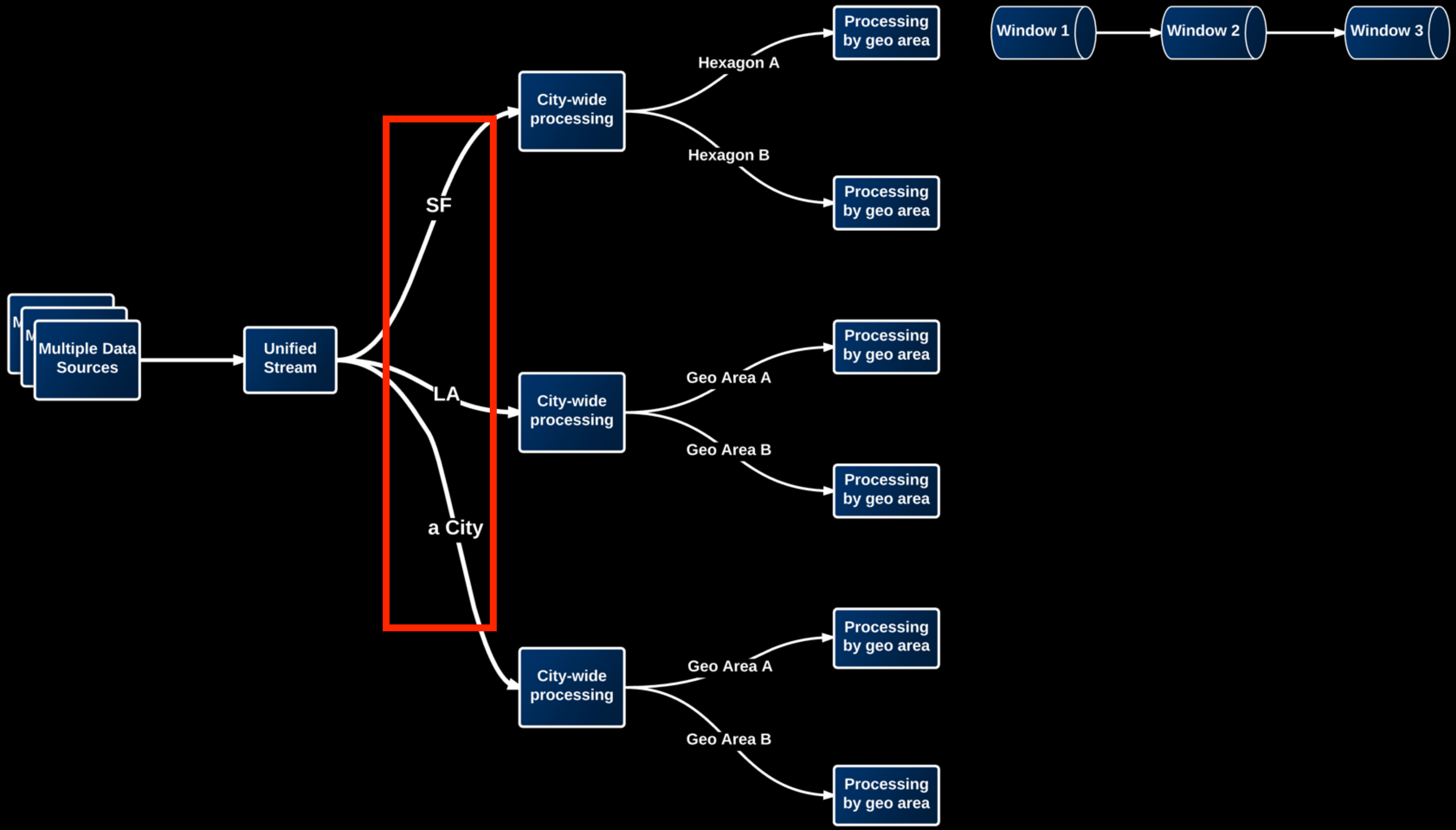
Mental Picture for Processing Geo-temporal Data



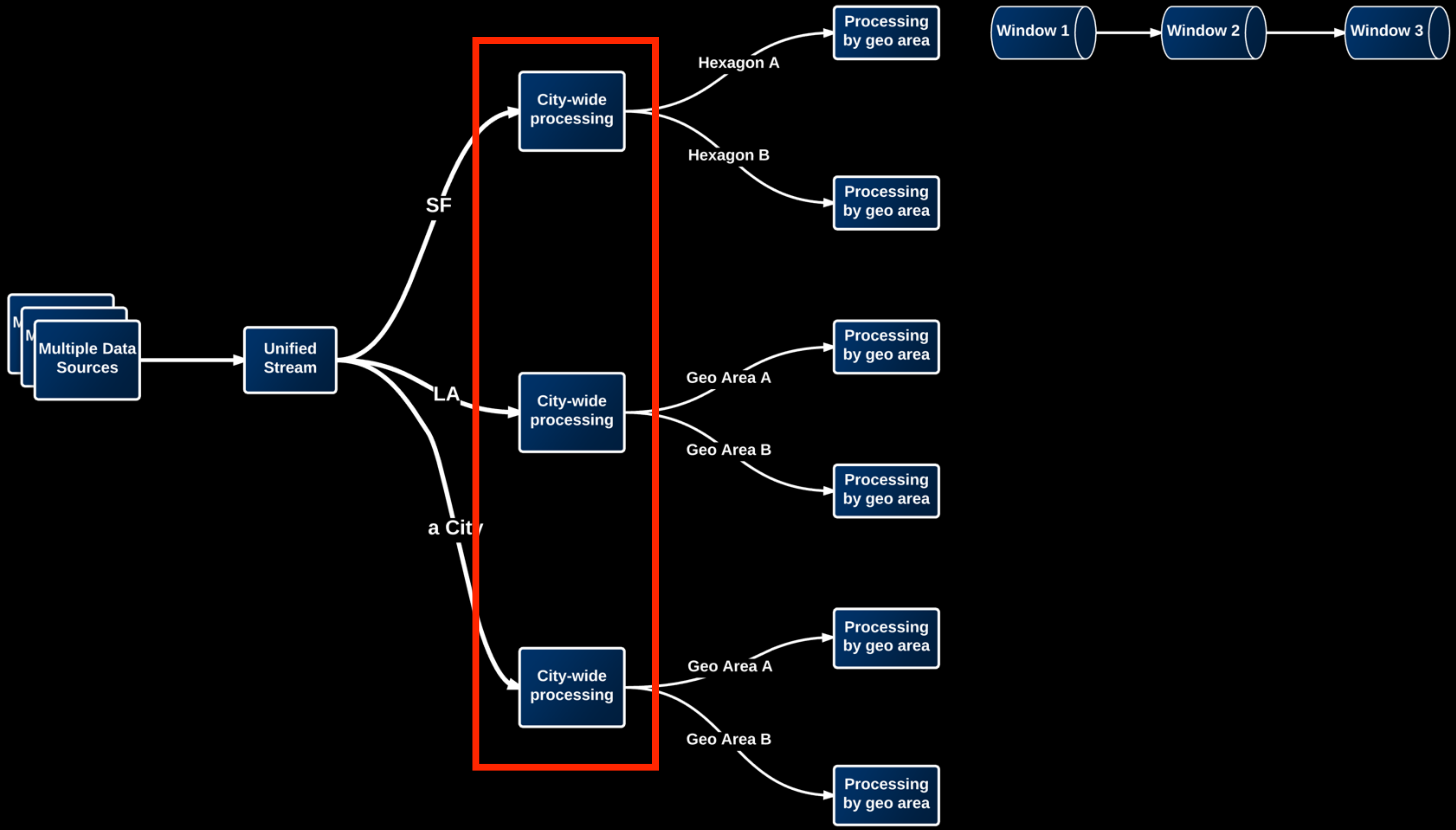
Mental Picture for Processing Geo-temporal Data



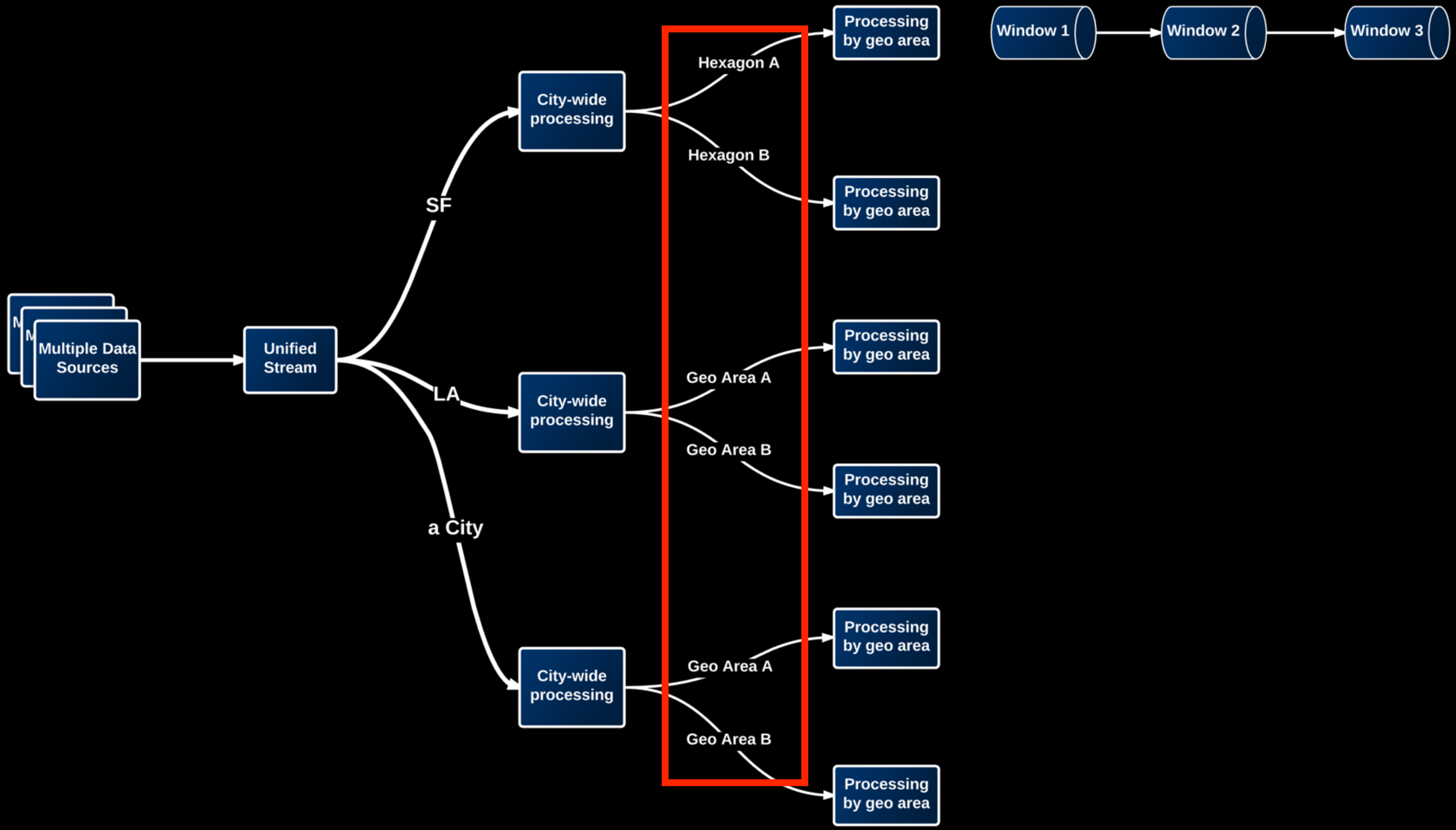
Mental Picture for Processing Geo-temporal Data



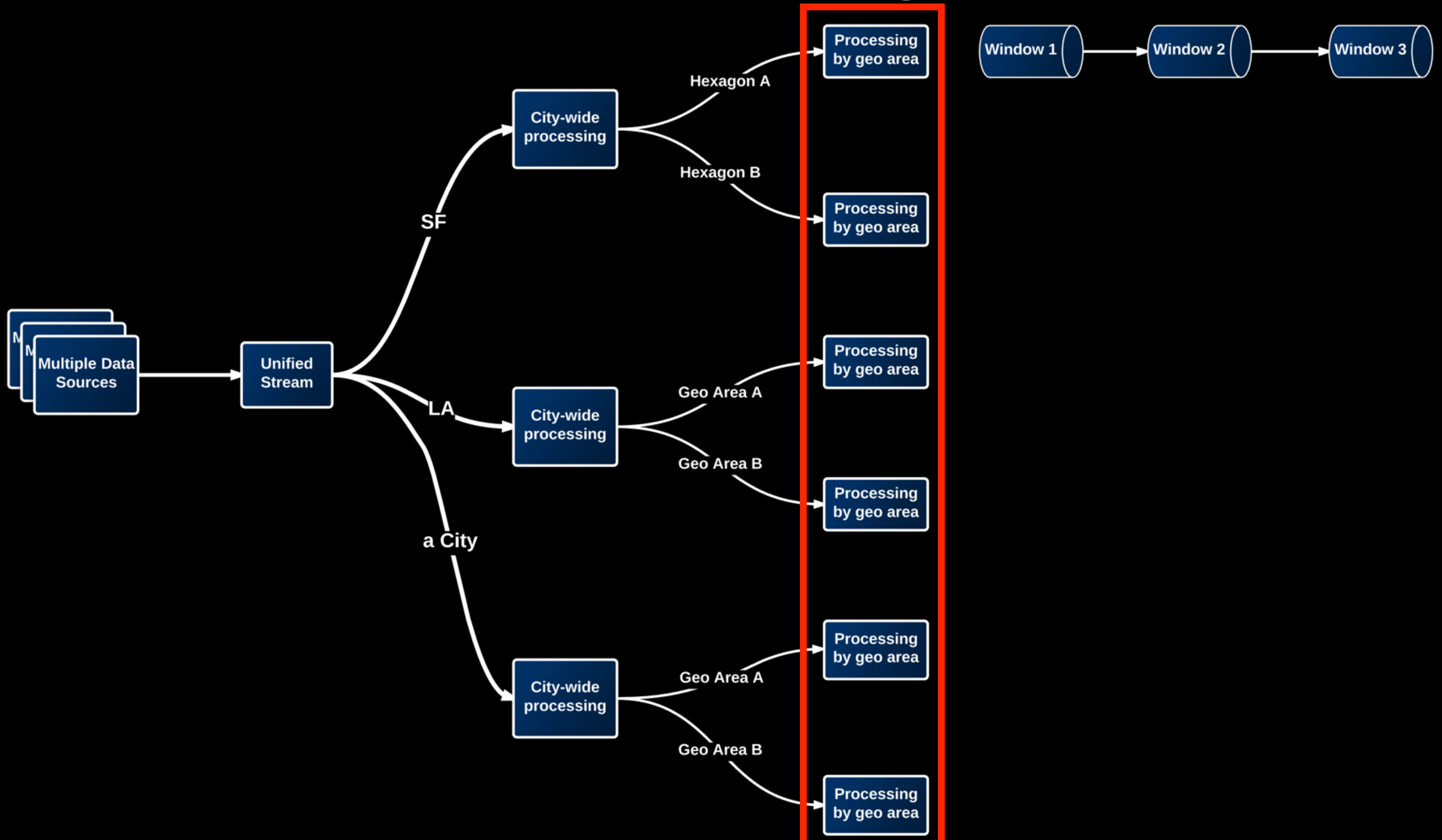
Mental Picture for Processing Geo-temporal Data



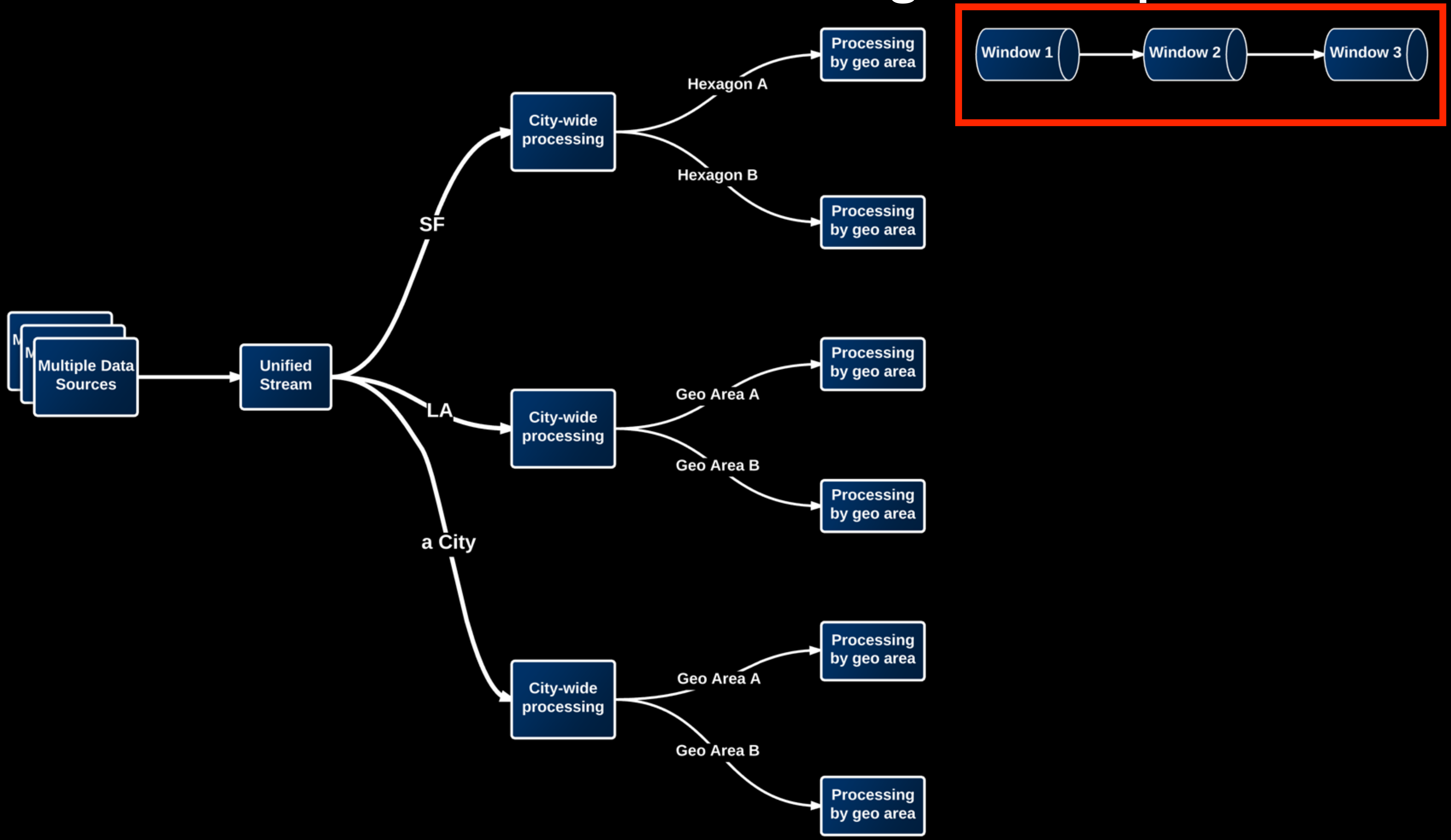
Mental Picture for Processing Geo-temporal Data



Mental Picture for Processing Geo-temporal Data



Mental Picture for Processing Geo-temporal Data



A Simple Example: simple prediction

Sources

```
.fromKafka()
.config(config)
.cluster(aCluster)
.topics(topicList)
```

A Simple Example

`assignTimestampsAndWatermarks`

A Simple Example

keyBy(...)

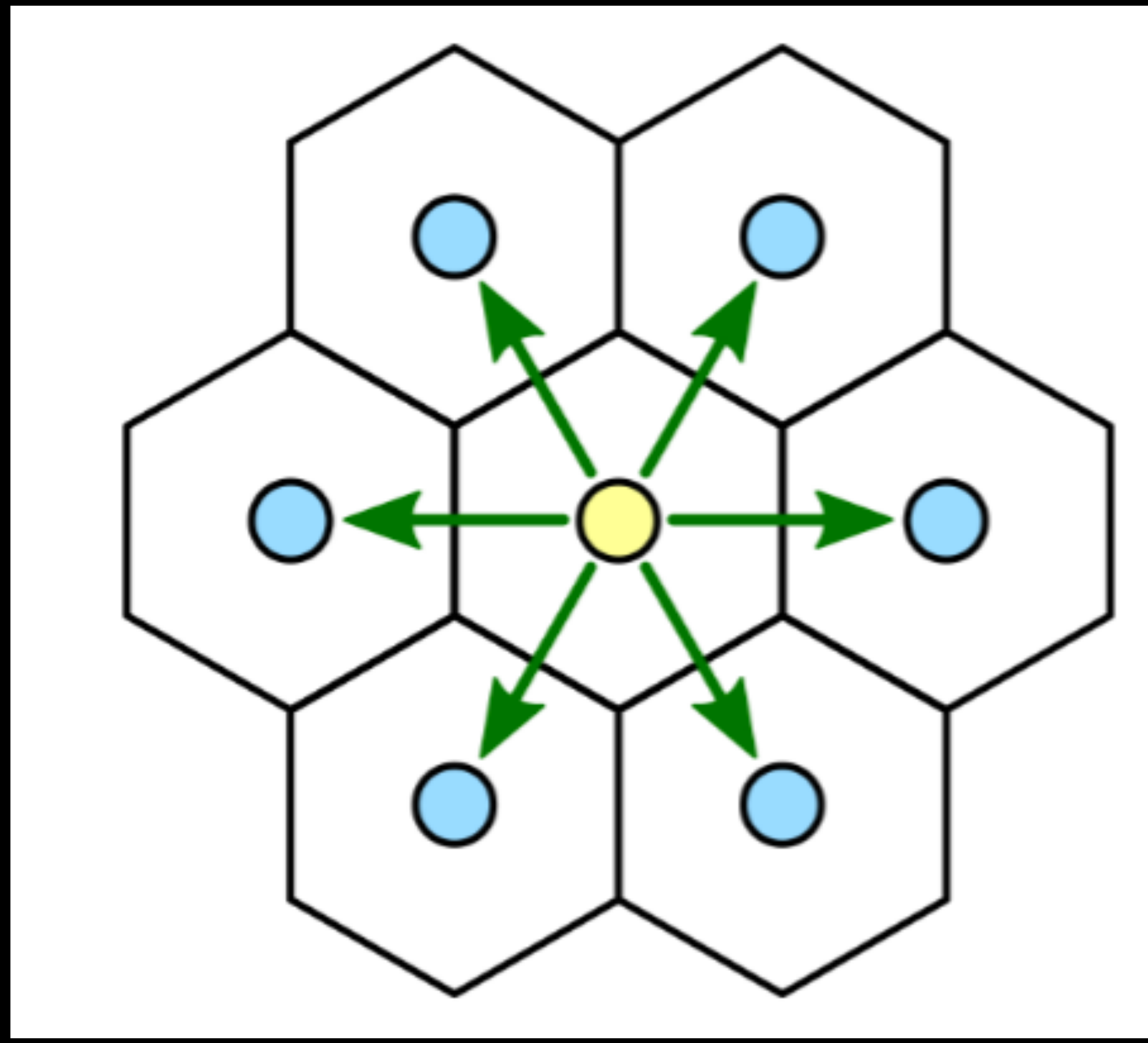
A Simple Example

`.timeWindow(…)`

A Simple Example

Kafka → Extract/decode timestamp → Key by city → Window by event time → Spatial Smoothing → Key by hex → Stateful prediction → Sink

.flatMap(…)



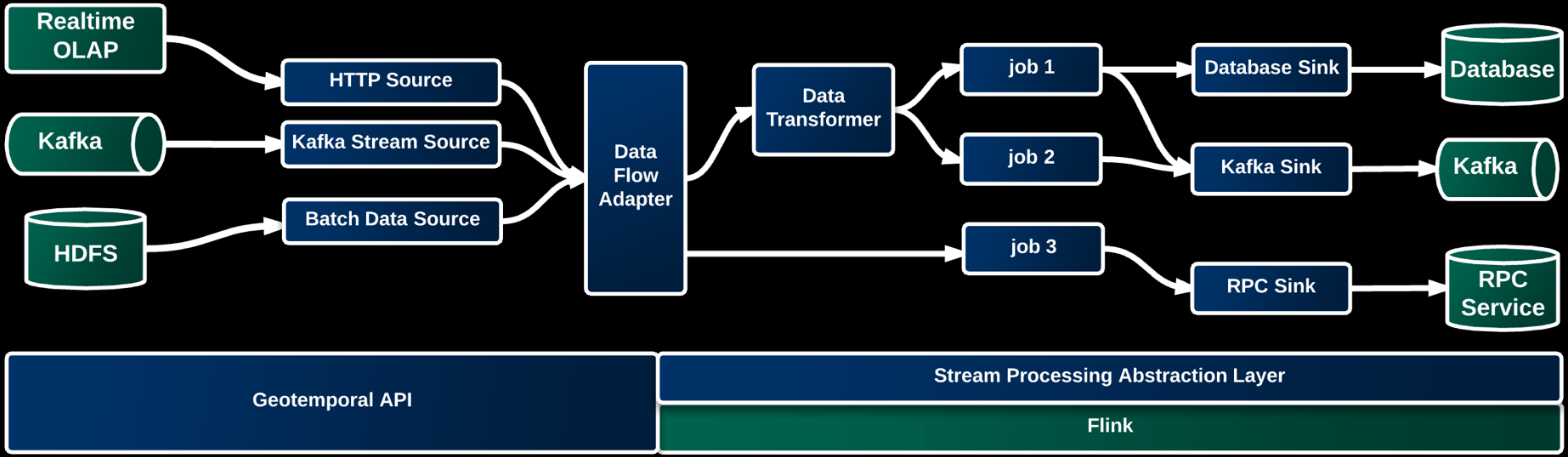
A Simple Example

A Simple Example

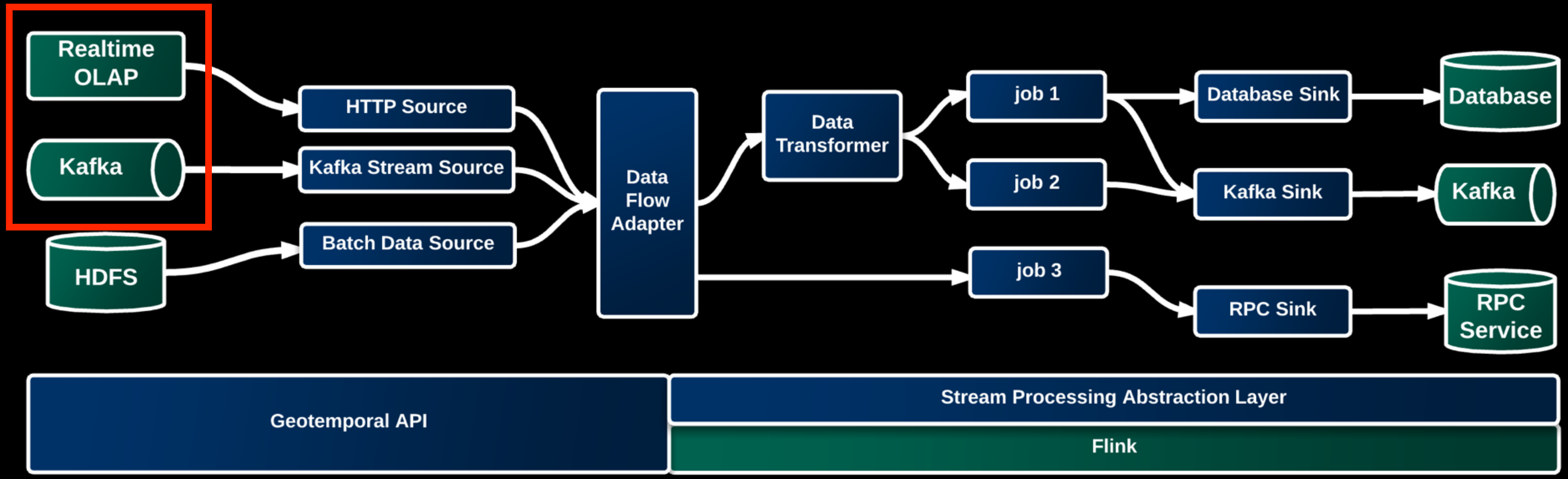
A Simple Example

`.addSink(…)`

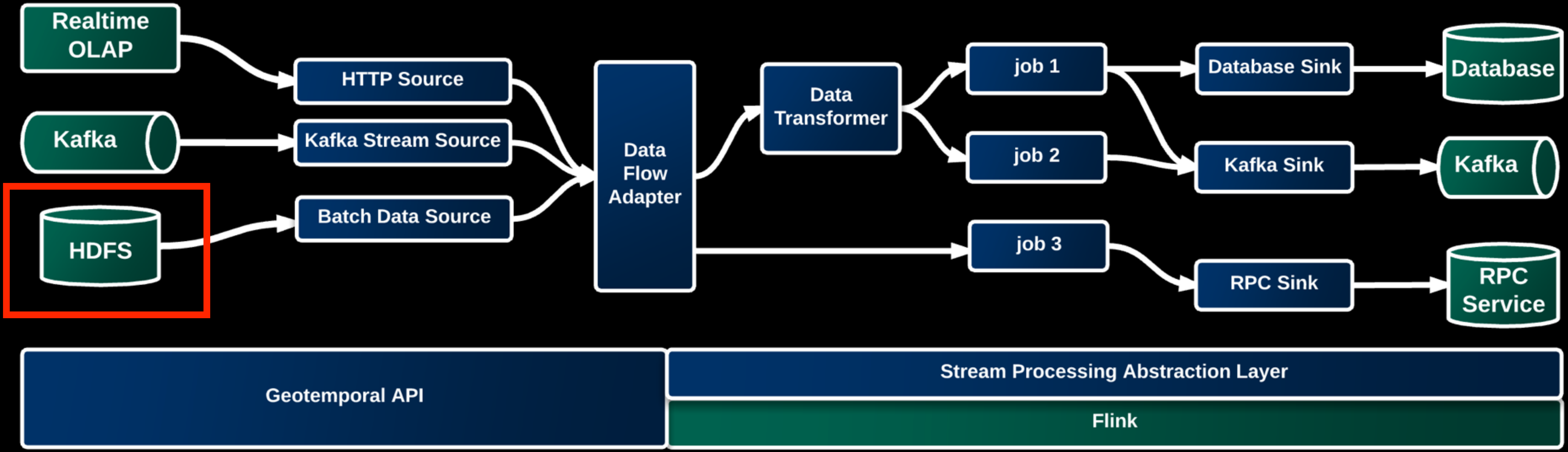
High Level Data Flow



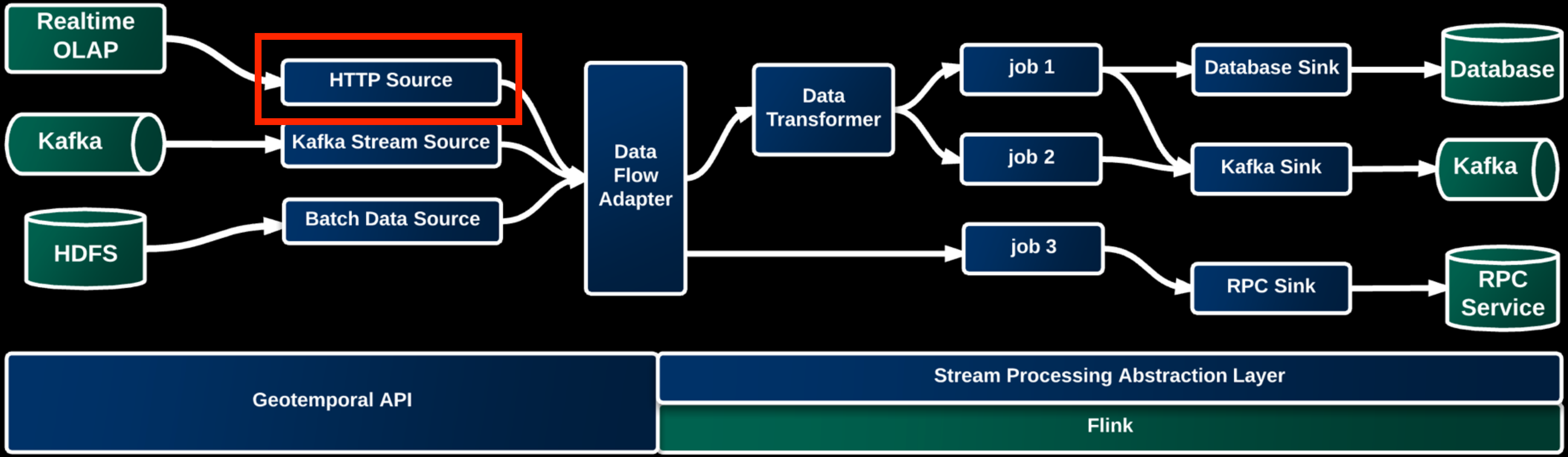
High Level Data Flow



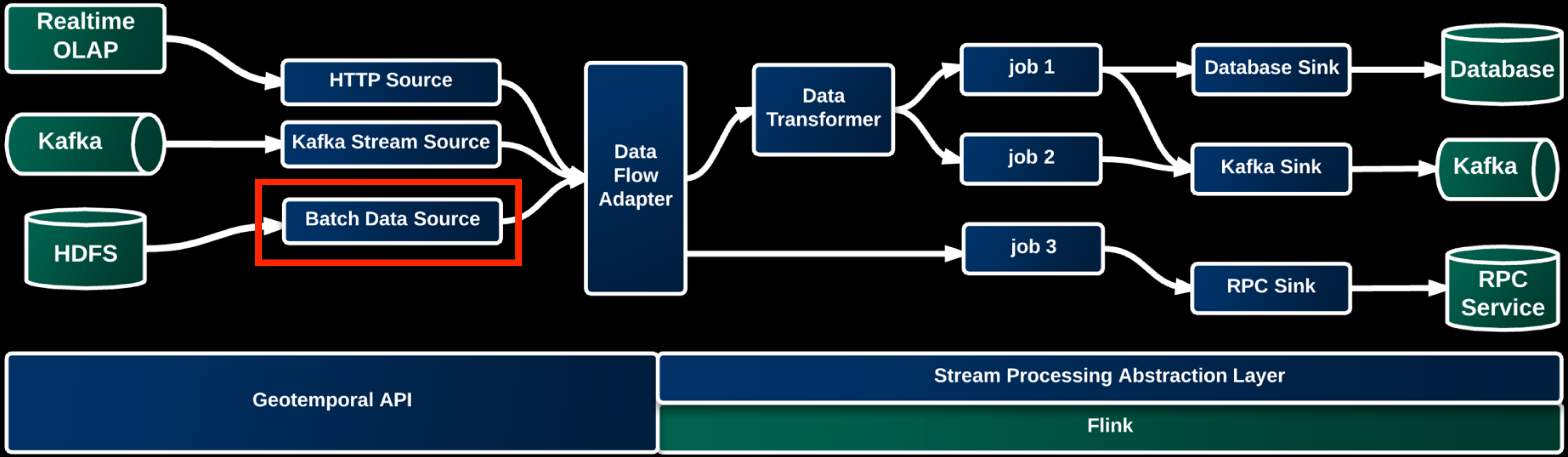
High Level Data Flow



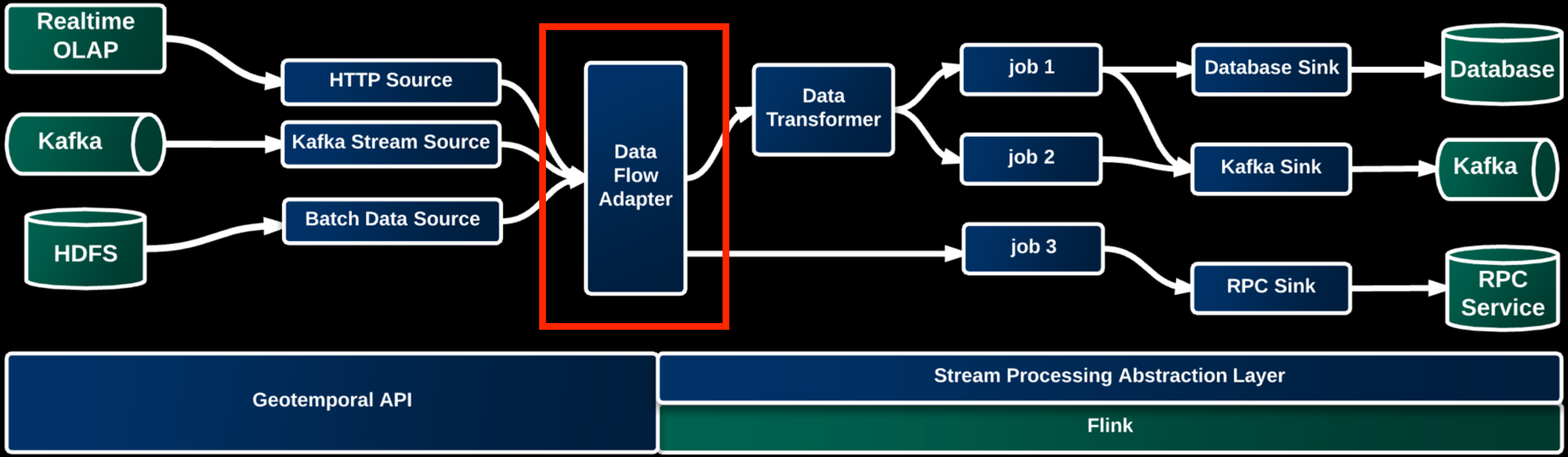
High Level Data Flow



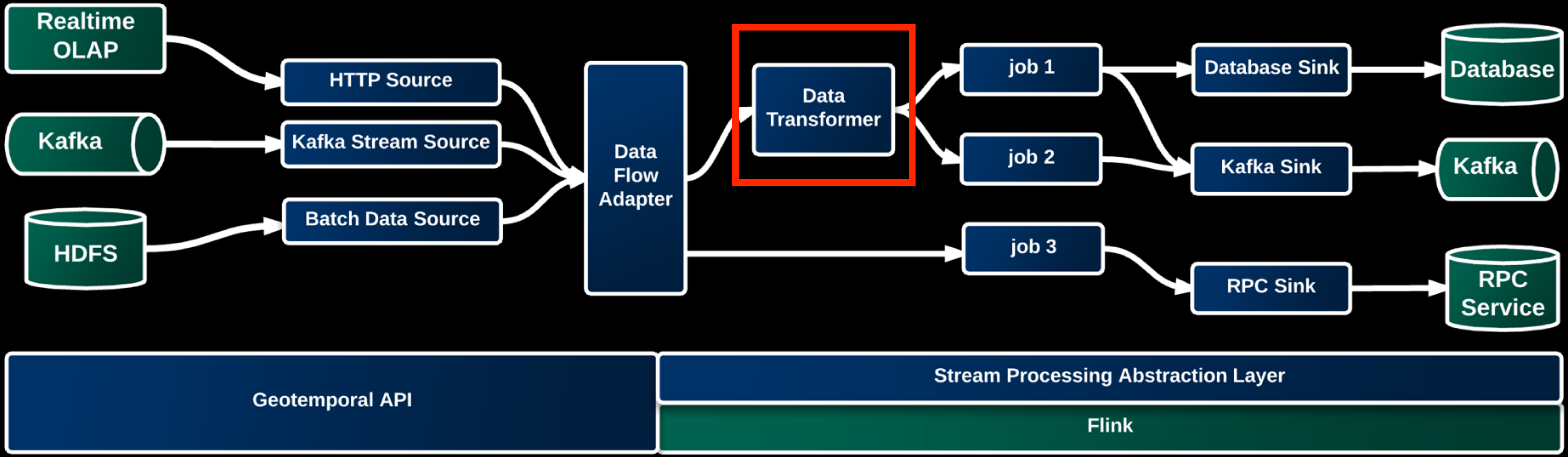
High Level Data Flow



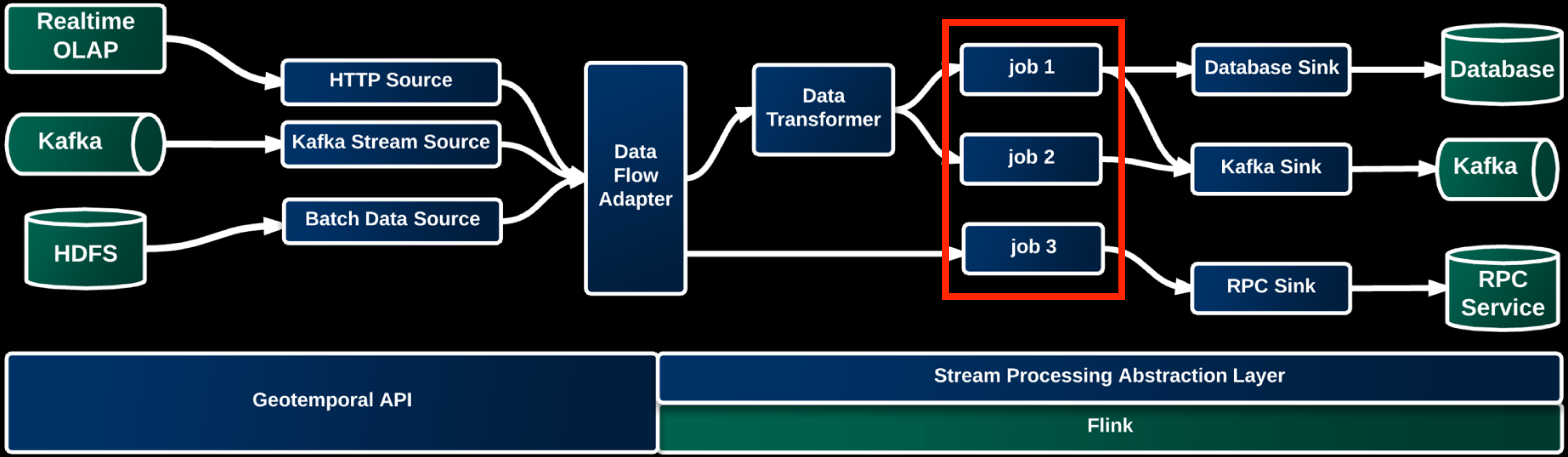
High Level Data Flow



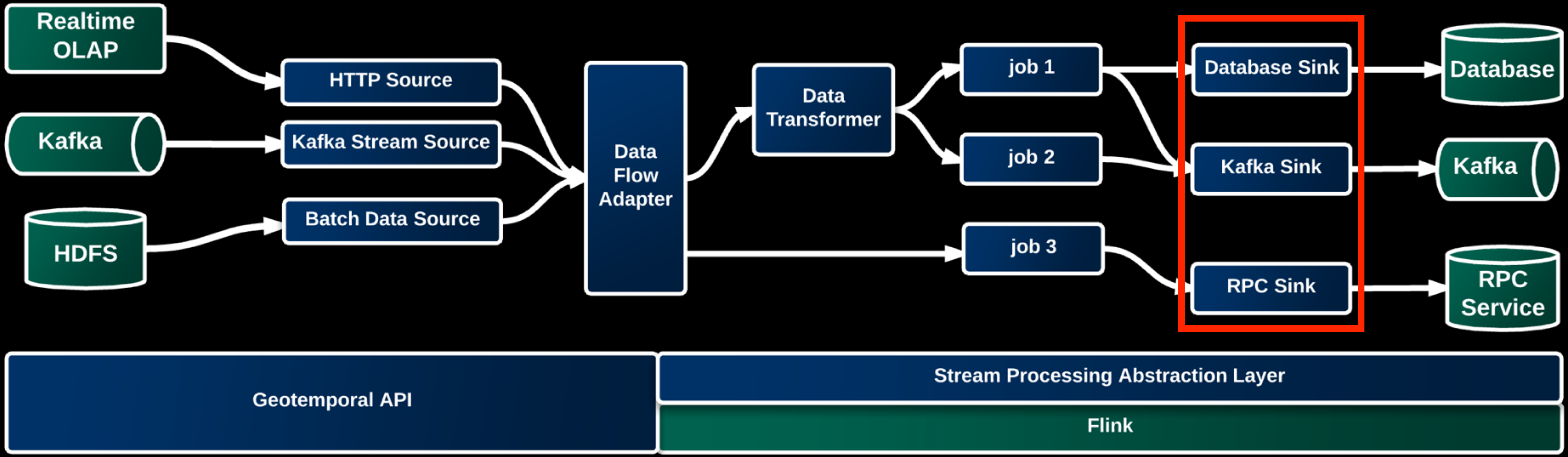
High Level Data Flow



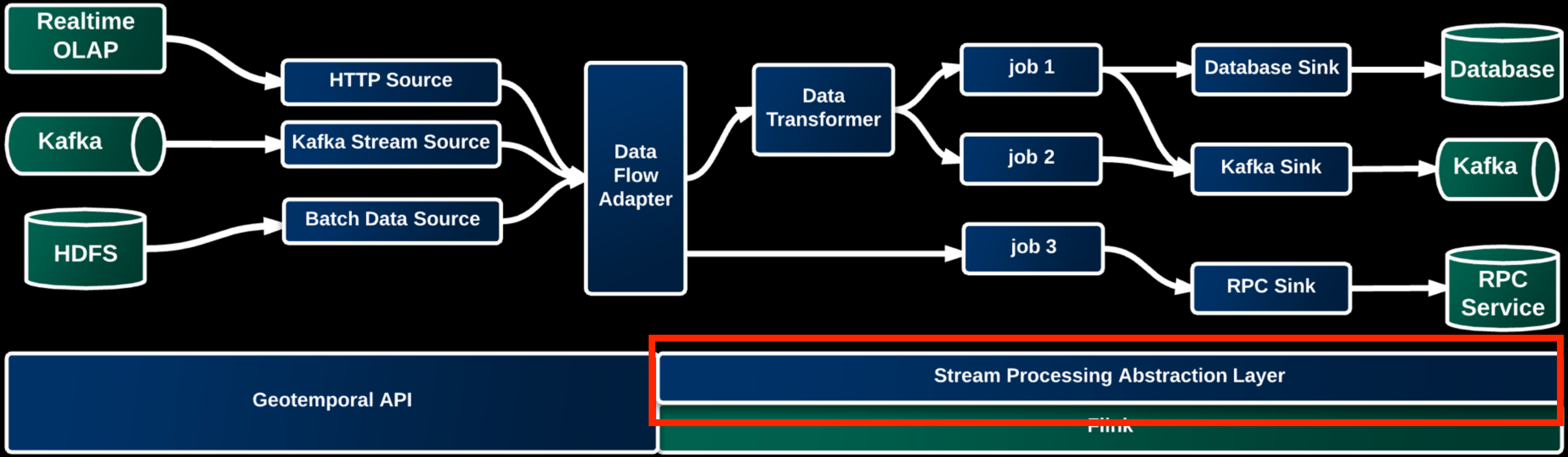
High Level Data Flow



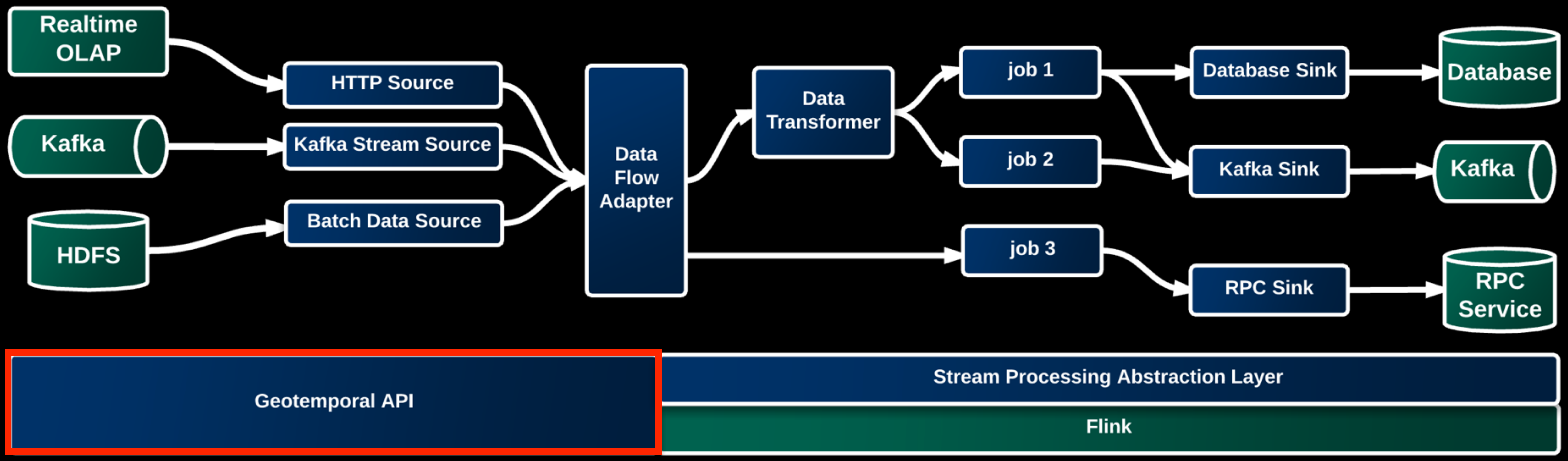
High Level Data Flow



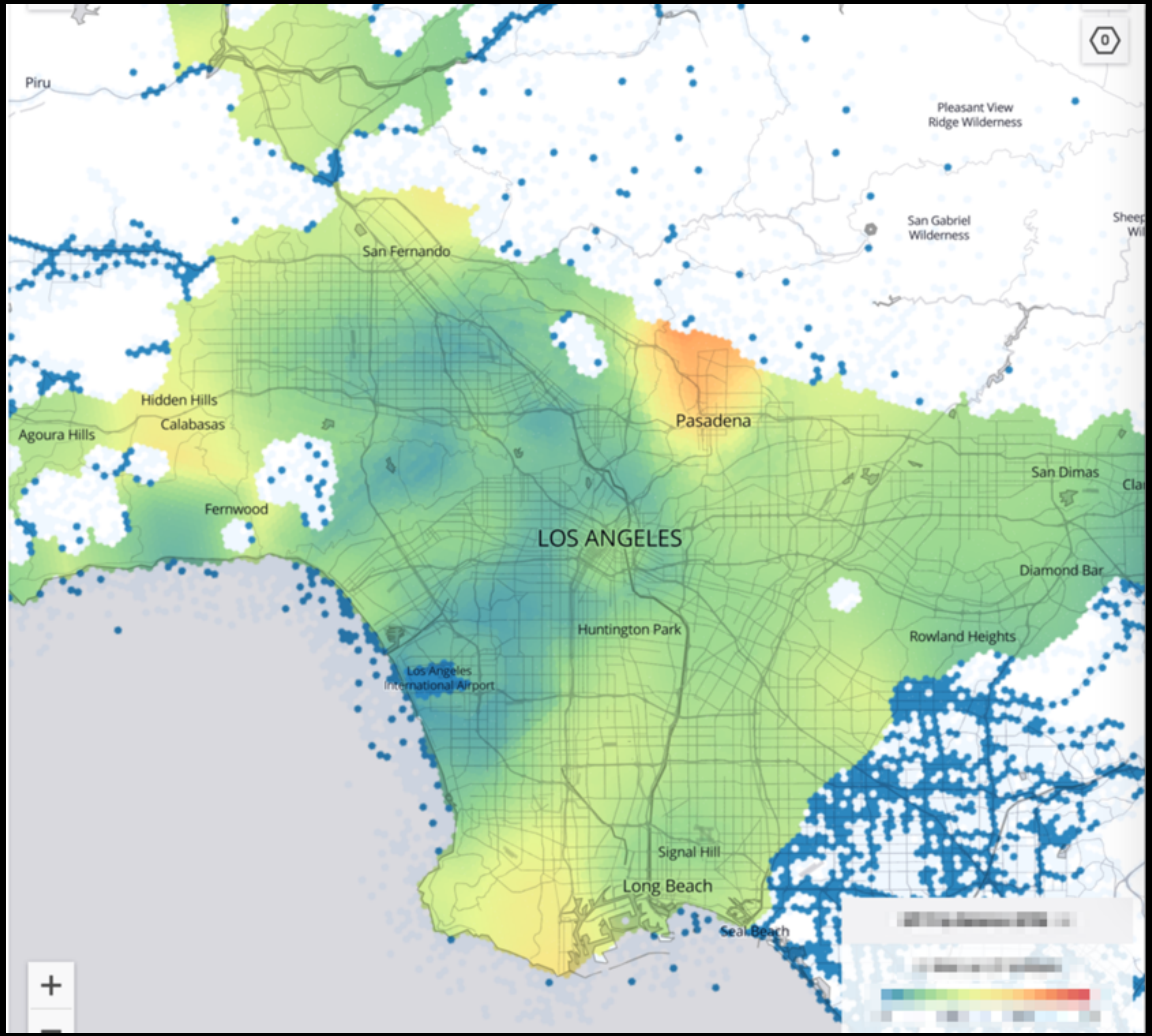
High Level Data Flow

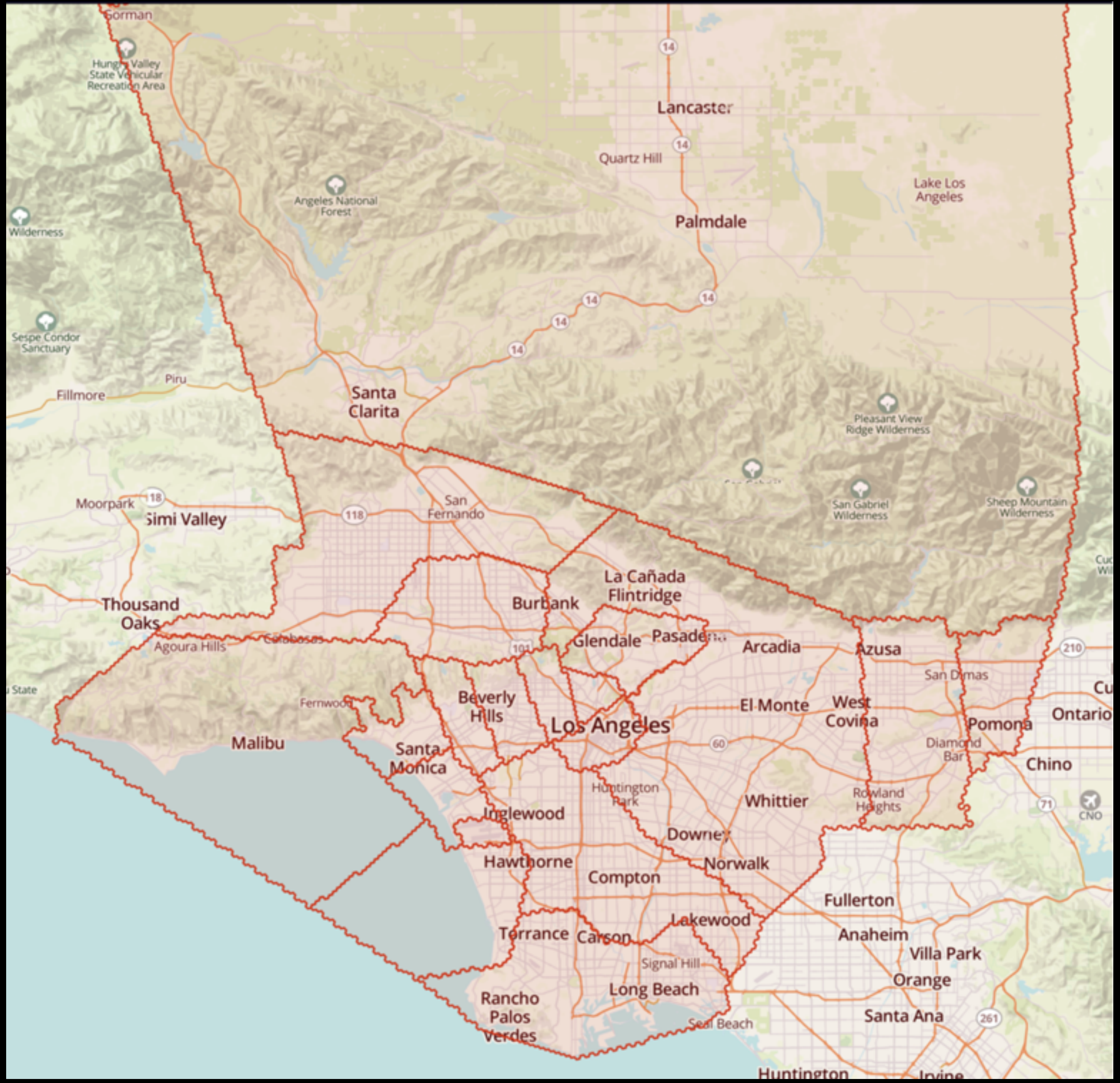


High Level Data Flow

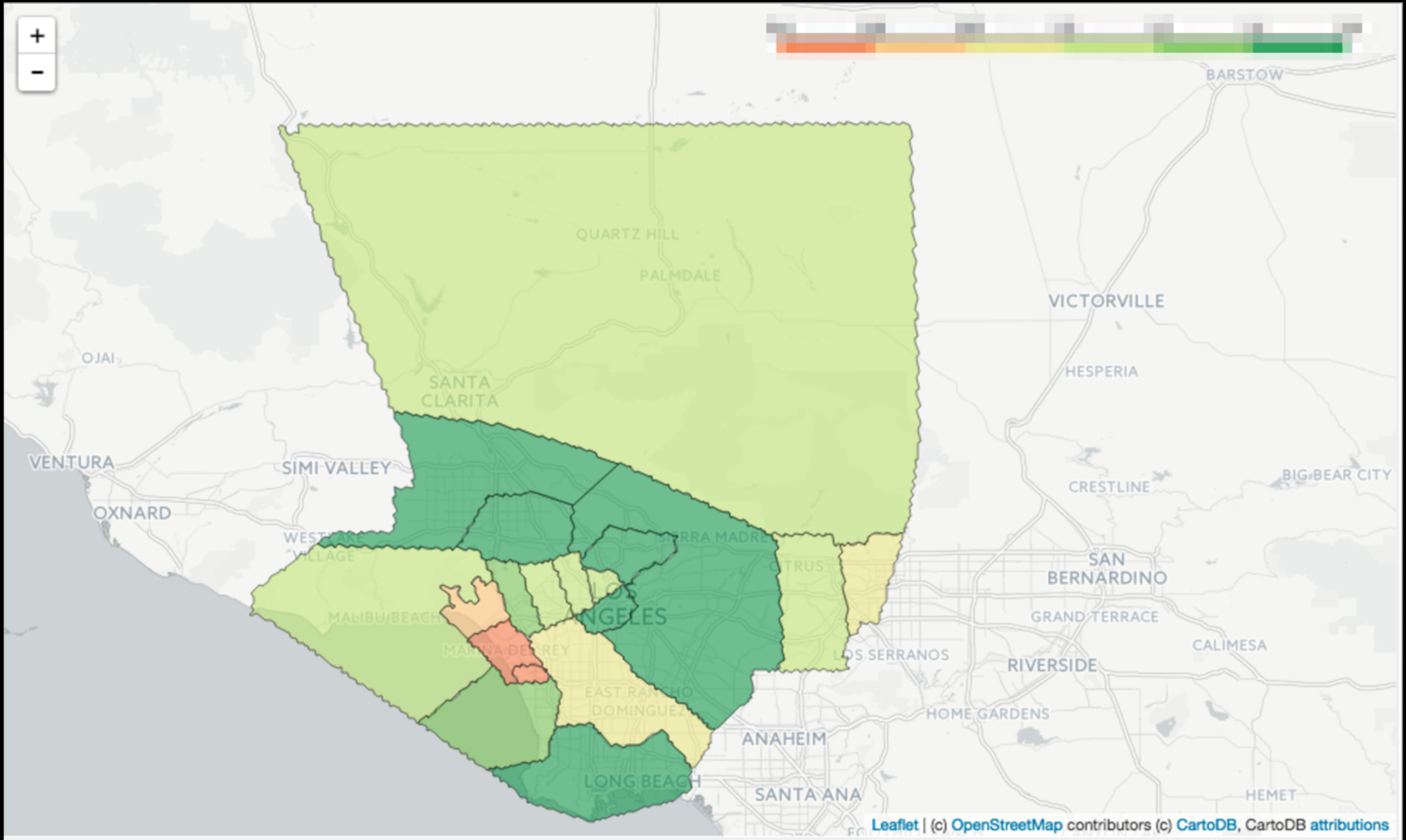


Geotemporal API for efficiency





Geotemporal API for efficiency



Geotemporal API for productivity

```java
private static ForkJoinPool fjPool = new ForkJoinPool();

@Override
public void postProcessResult(QueryResult result) {
    ImmutableMap<HexagonCoord, BucketWrapper> hexagons = HexagonAggregationUtility.buildHexagonMap(result, hexField);

    List<BucketWrapper> buckets = Lists.newArrayList(hexagons.values());

    fjPool.invoke(new KRingProcessor(SEQUENTIAL_THRESHOLD, hexagons, buckets, 0, buckets.size()));
}

private class KRingProcessor extends RecursiveTask<List<BucketWrapper>> {
    private int sequentialThreshold;
    private int low;
    private int high;

    private ImmutableMap<HexagonCoord, BucketWrapper> data;
    private List<BucketWrapper> buckets;

    KRingProcessor(int sequentialThreshold,
                   ImmutableMap<HexagonCoord, BucketWrapper> data,
                   List<BucketWrapper> buckets,
                   int low, int high) {
        this.sequentialThreshold = sequentialThreshold;
        this.data = data;
        this.buckets = buckets;
        this.low = low;
        this.high = high;
    }

    @Override
    protected List<BucketWrapper> compute() {
        if (high - low <= sequentialThreshold) {
            for (int i = low; i < high; ++i) {
                BucketWrapper bucket = buckets.get(i);
                Map<String, Object> values = bucket.getBucket().getValues();
                if (values.containsKey(hexField) && values.containsKey(metric)) {
                    processBucket(data, bucket);
                }
            }

            return buckets;
        } else {
            int mid = low + (high - low) / 2;
            KRingProcessor left = new KRingProcessor(sequentialThreshold, data, buckets, low, mid);
            KRingProcessor right = new KRingProcessor(sequentialThreshold, data, buckets, mid, high);
            left.fork();
            right.compute();
            left.join();

            return buckets;
        }
    }

    private void processBucket(Map<HexagonCoord, BucketWrapper> hexagons, BucketWrapper bucket) {
        [redacted]
    }
}
```

Geotemporal API for productivity

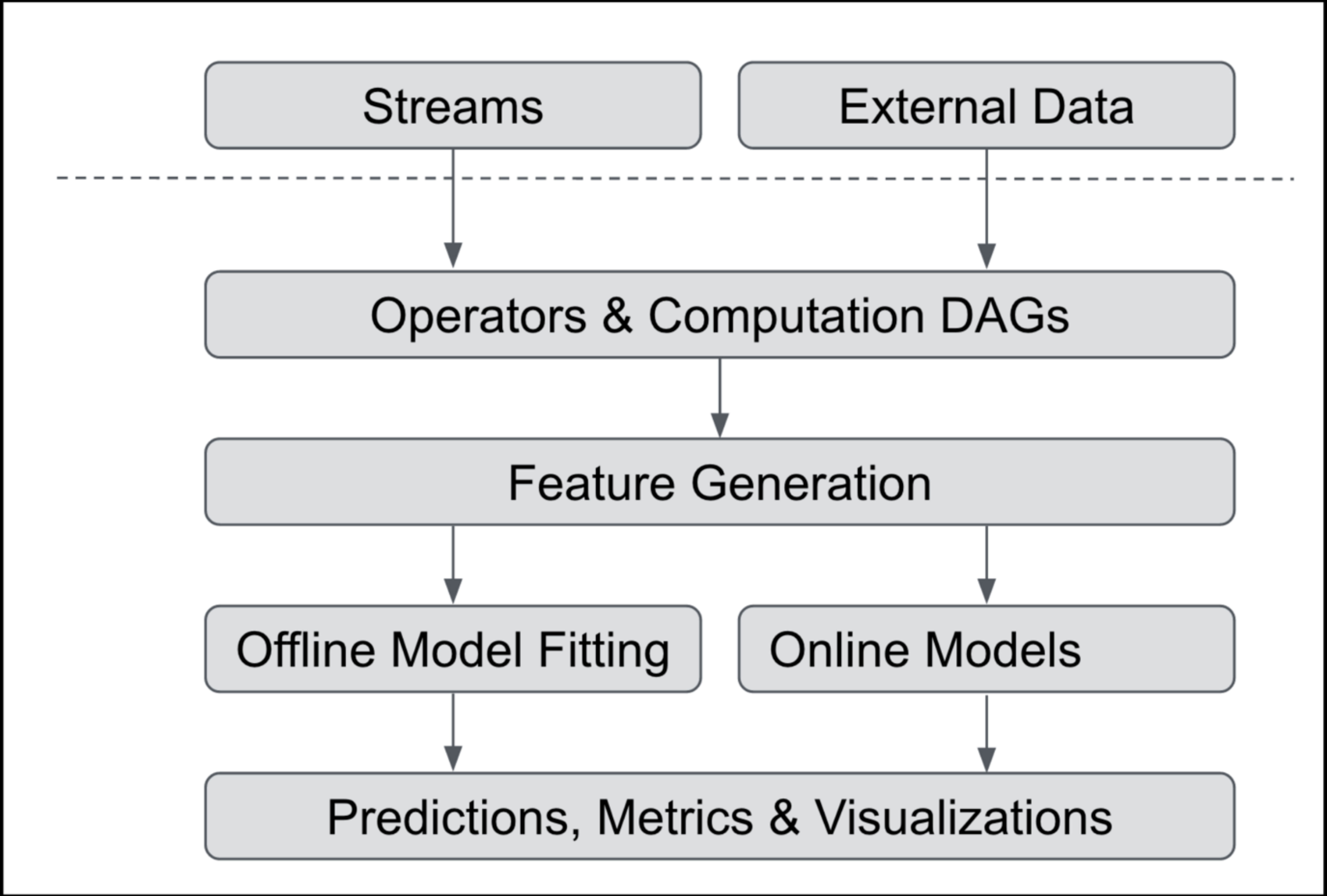
```java
private static ForkJoinPool fjPool = new ForkJoinPool();

@Override
public void postProcessResult(QueryResult result) {
    ImmutableMap<HexagonCoord, BucketWrapper> hexagons = HexagonAggregationUtility.buildHexagonMap(result, hexField);

    List<BucketWrapper> buckets = Lists.newArrayList(hexagons.values());

    fjPool.invoke(new KRingProcessor(SEQUENTIAL_THRESHOLD, hexagons, buckets, 0, buckets.size()));
}

private class KRingProcessor extends RecursiveTask<List<BucketWrapper>> {
    private int sequentialThreshold;
    private int low;
    private int high;

    private ImmutableMap<HexagonCoord, BucketWrapper> data;
    private List<BucketWrapper> buckets;

    KRingProcessor(int sequentialThreshold,
                   ImmutableMap<HexagonCoord, BucketWrapper> data,
                   List<BucketWrapper> buckets,
                   int low, int high) {
        this.sequentialThreshold = sequentialThreshold;
        this.data = data;
        this.buckets = buckets;
        this.low = low;
        this.high = high;
    }

    @Override
    protected List<BucketWrapper> compute() {
        if (high - low <= sequentialThreshold) {
            for (int i = low; i < high; ++i) {
                BucketWrapper bucket = buckets.get(i);
                Map<String, Object> values = bucket.getBucket().getValues();
                if (values.containsKey(hexField) && values.containsKey(metric)) {
                    processBucket(data, bucket);
                }
            }

            return buckets;
        } else {
            int mid = low + (high - low) / 2;
            KRingProcessor left = new KRingProcessor(sequentialThreshold, data, buckets, low, mid);
            KRingProcessor right = new KRingProcessor(sequentialThreshold, data, buckets, mid, high);
            left.fork();
            right.compute();
            left.join();

            return buckets;
        }
    }
```

Geotemporal API for productivity

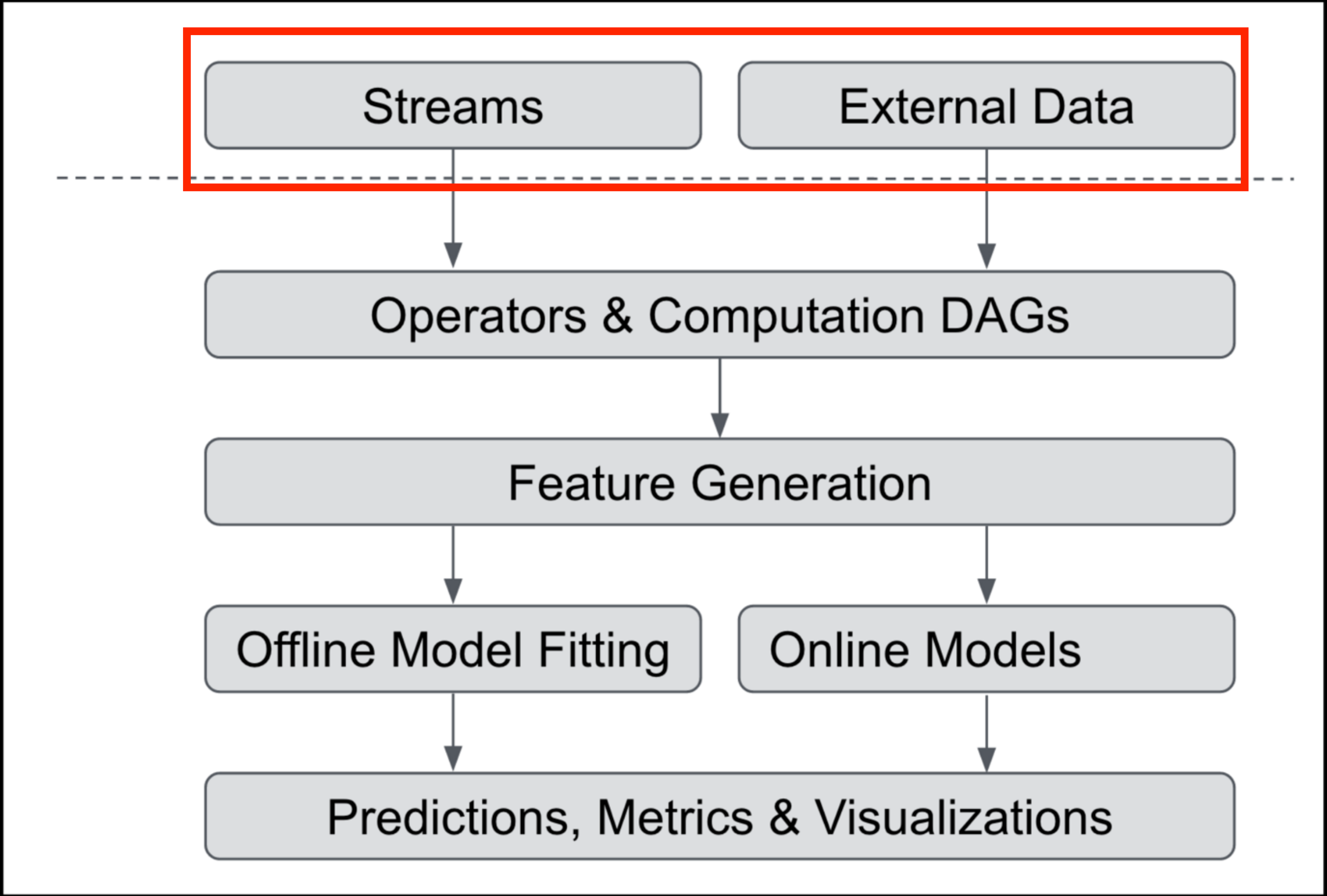
```java
private static ForkJoinPool fjPool = new ForkJoinPool();

@Override
public void postProcessResult(QueryResult result) {
    ImmutableMap<HexagonCoord, BucketWrapper> hexagons = HexagonAggregationUtility.buildHexagonMap(result, hexField);

    List<BucketWrapper> buckets = Lists.newArrayList(hexagons.values());

    fjPool.invoke(new KRingProcessor(SEQUENTIAL_THRESHOLD, hexagons, buckets, 0, buckets.size()));
}

private class KRingProcessor extends RecursiveTask<List<BucketWrapper>> {
    private int sequentialThreshold;
    private int low;
    private int high;

    private ImmutableMap<HexagonCoord, BucketWrapper> data;
    private List<BucketWrapper> buckets;

    KRingProcessor(int sequentialThreshold,
                   ImmutableMap<HexagonCoord, BucketWrapper> data,
                   List<BucketWrapper> buckets,
                   int low, int high) {
        this.sequentialThreshold = sequentialThreshold;
        this.data = data;
        this.buckets = buckets;
        this.low = low;
        this.high = high;
    }

    @Override
    protected List<BucketWrapper> compute() {
        if (high - low <= sequentialThreshold) {
            for (int i = low; i < high; ++i) {
                BucketWrapper bucket = buckets.get(i);
                Map<String, Object> values = bucket.getBucket().getValues();
                if (values.containsKey(hexField) && values.containsKey(metric)) {
                    processBucket(data, bucket);
                }
            }

            return buckets;
        } else {
            int mid = low + (high - low) / 2;
            KRingProcessor left = new KRingProcessor(sequentialThreshold, data, buckets, low, mid);
            KRingProcessor right = new KRingProcessor(sequentialThreshold, data, buckets, mid, high);
            left.fork();
            right.compute();
            left.join();

            return buckets;
        }
    }

    private void processBucket(Map<HexagonCoord, BucketWrapper> hexagons, BucketWrapper bucket) {
        HexagonCoord hexCoord = bucket.getCoord();
        ...
    }
}
```

```java
return hexagonContext.mapGeoArea(
    (context, area) -> {
        double incrementalValue = 0;
        ...
    }
```

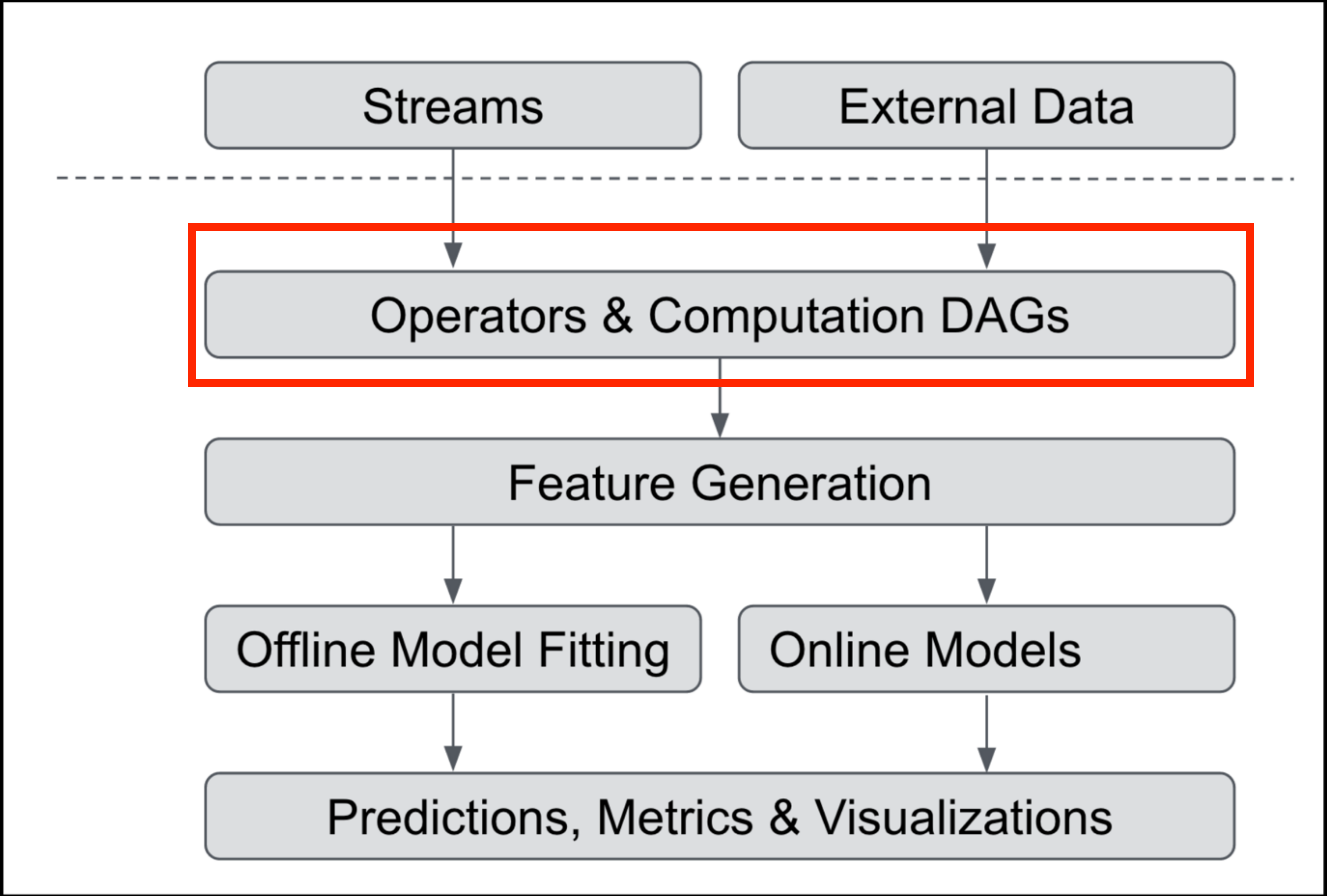
Forecasting as an example



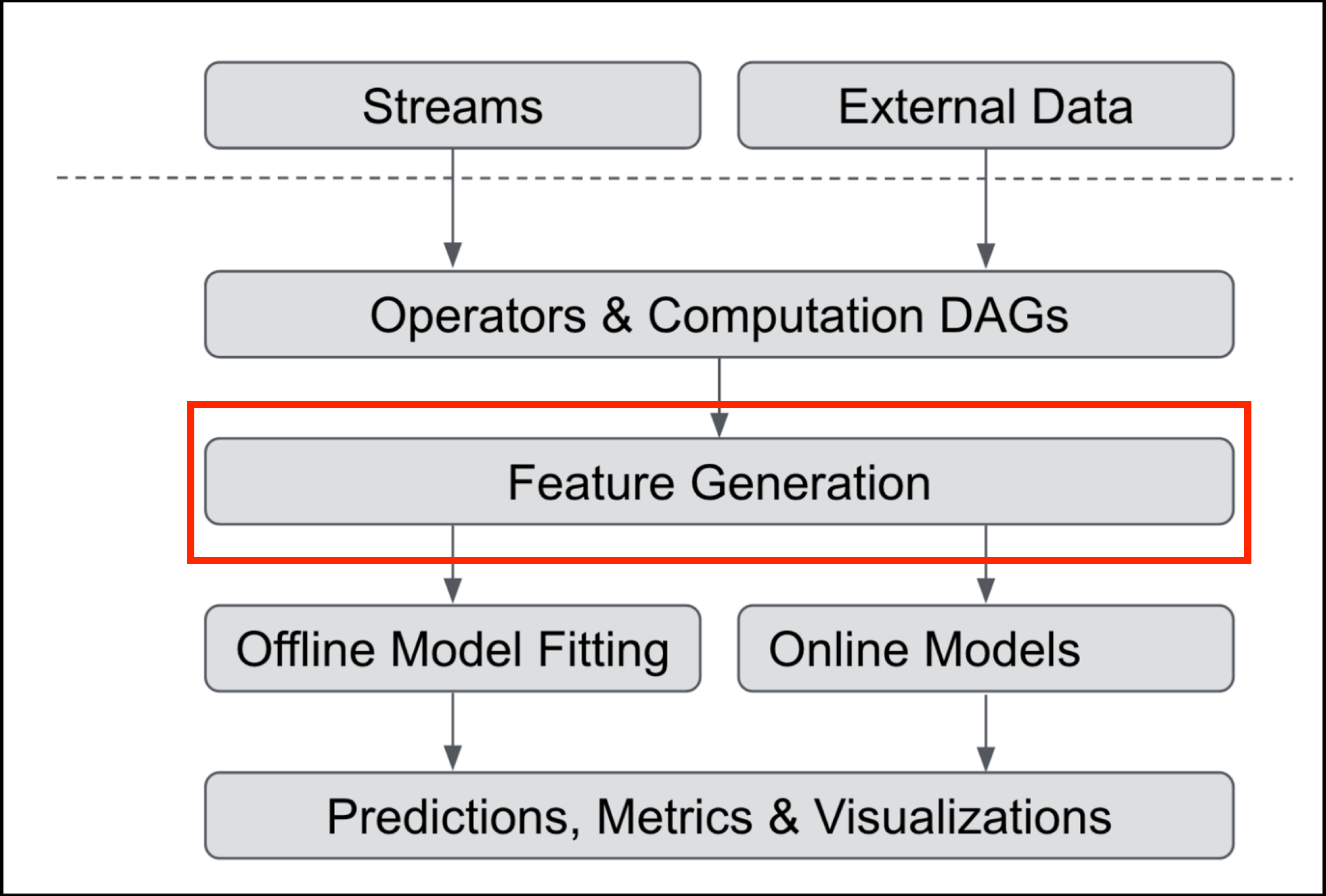
Forecasting as an example



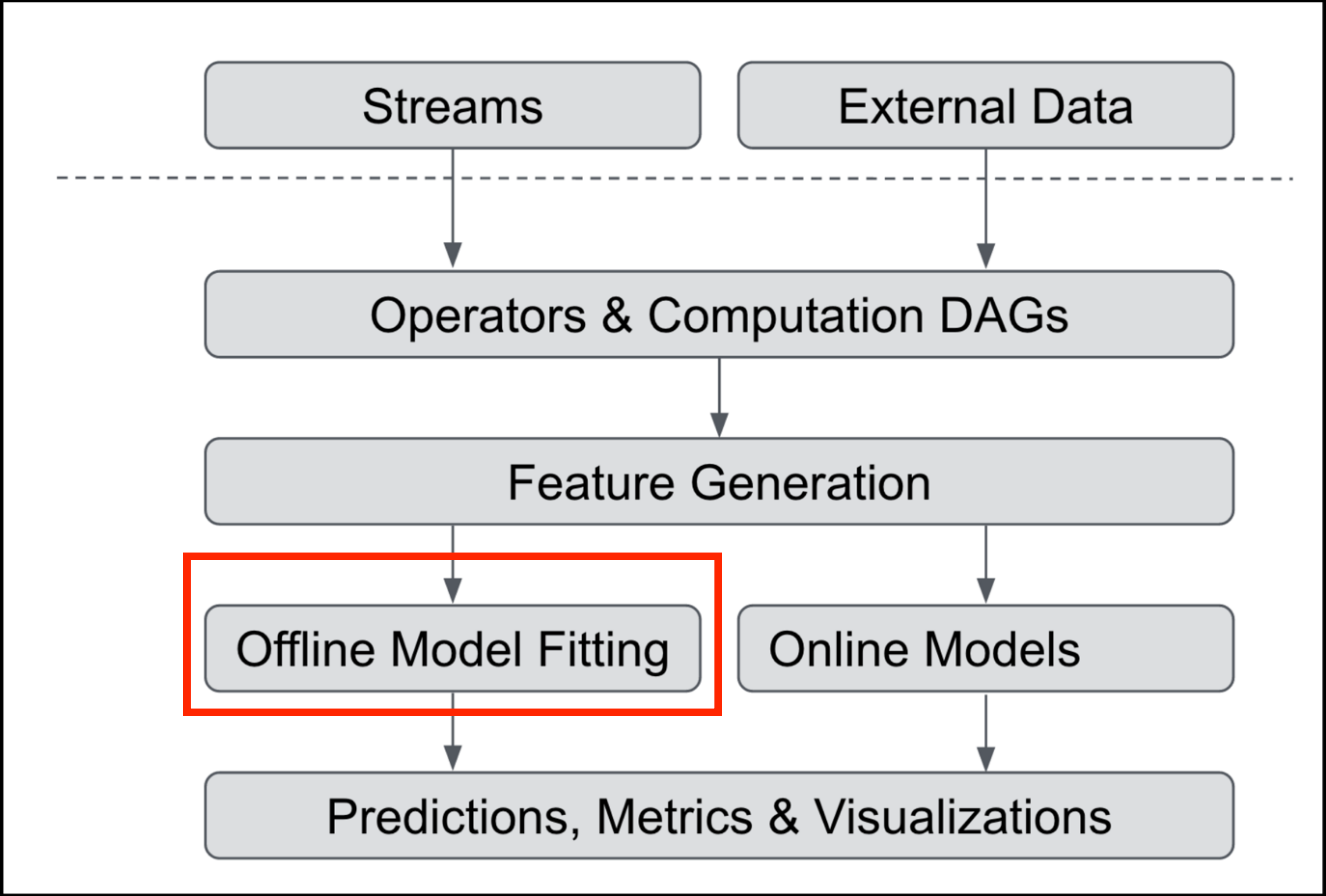
Forecasting as an example



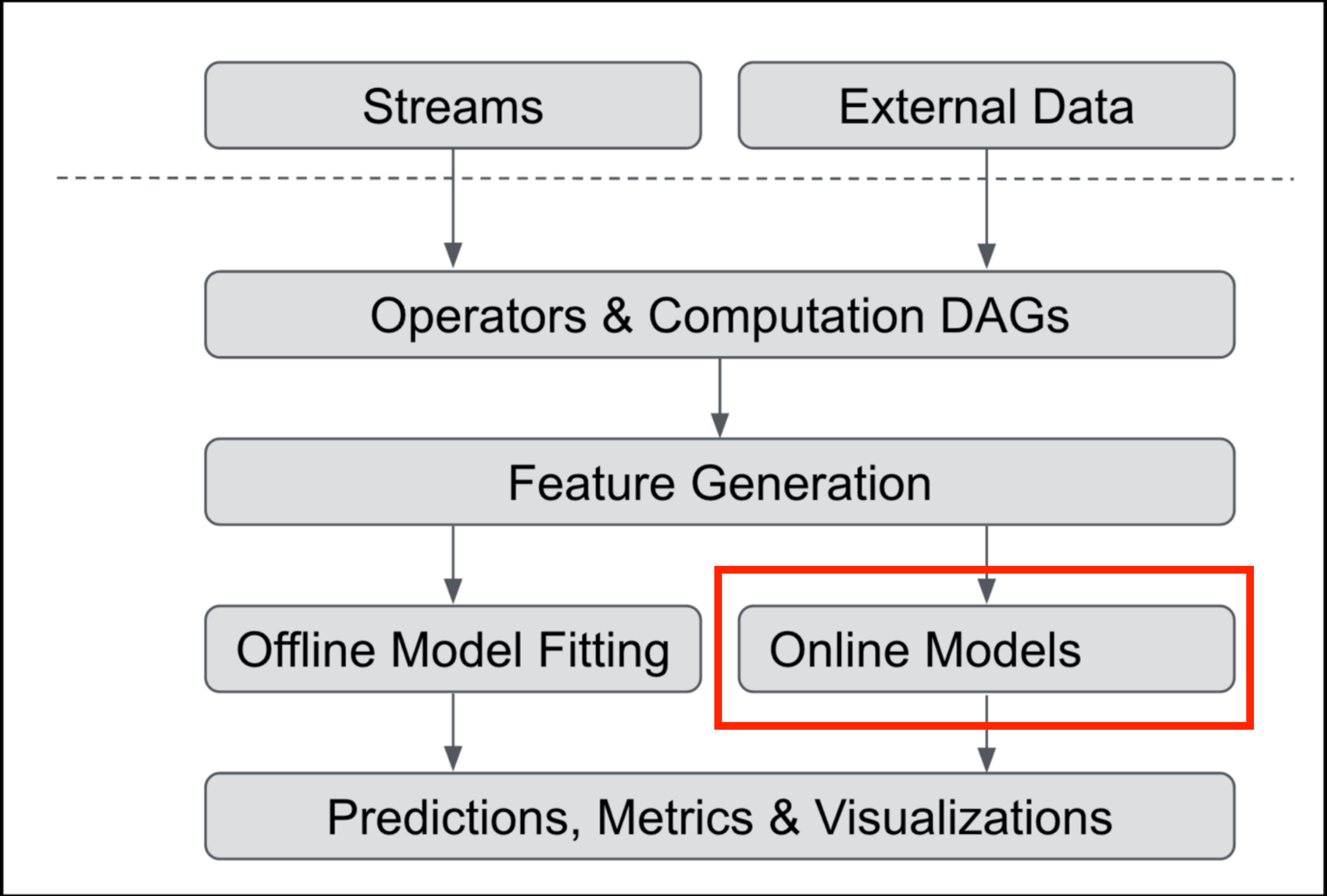
Forecasting as an example



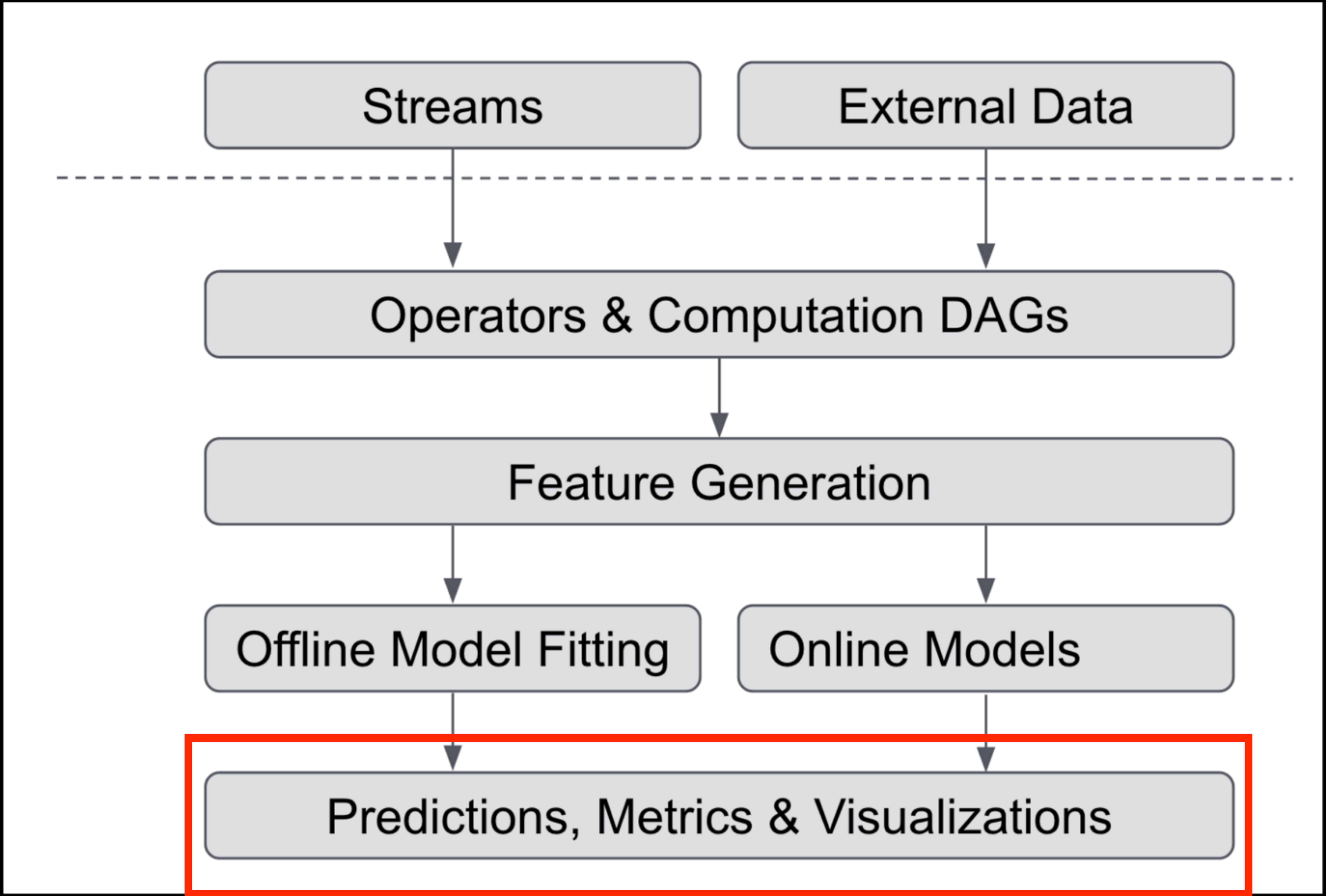
Forecasting as an example



Forecasting as an example

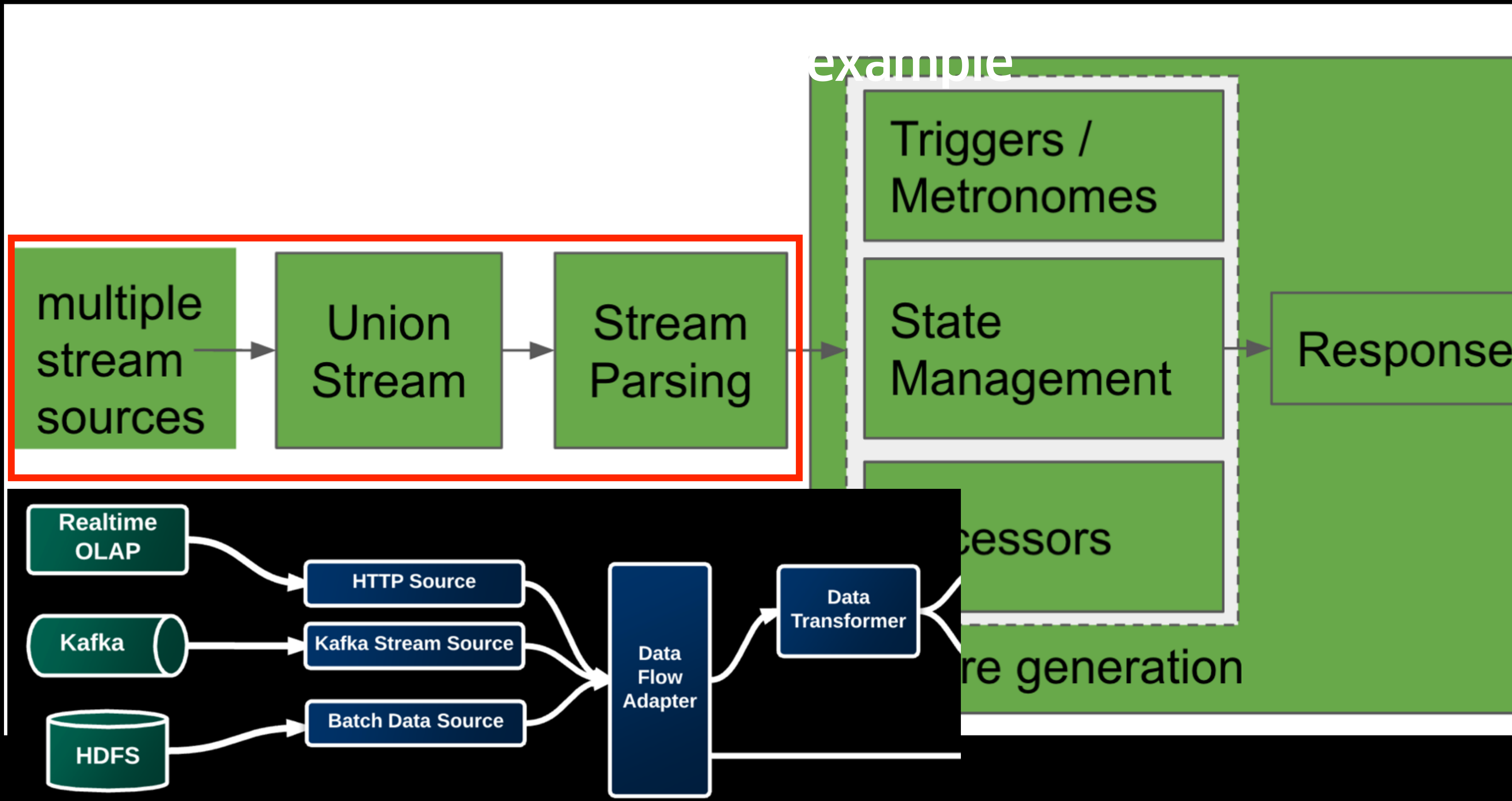


Forecasting as an example



Forecasting as an example

Forecasting as an example


example
multiple stream sources
Union Stream
Stream Parsing
Triggers / Metronomes
State Management
cessors
re generation
Response
Realtime OLAP
Kafka
HDFS
HTTP Source
Kafka Stream Source
Batch Data Source
Data Flow Adapter
Data Transformer

Forecasting as an example

Forecasting as an example

Forecasting as an example

Forecasting as an example

Forecasting as an example

Forecasting as an example

Forecasting as an example

Lessons Learned

- Make sure you have robust infrastructure support
- Scaling up, namely single-node optimization matters
- Ensure exactly-once by proper data modeling
- Use external state store to avoid too much snapshotting
- Standardize monitoring and data validation

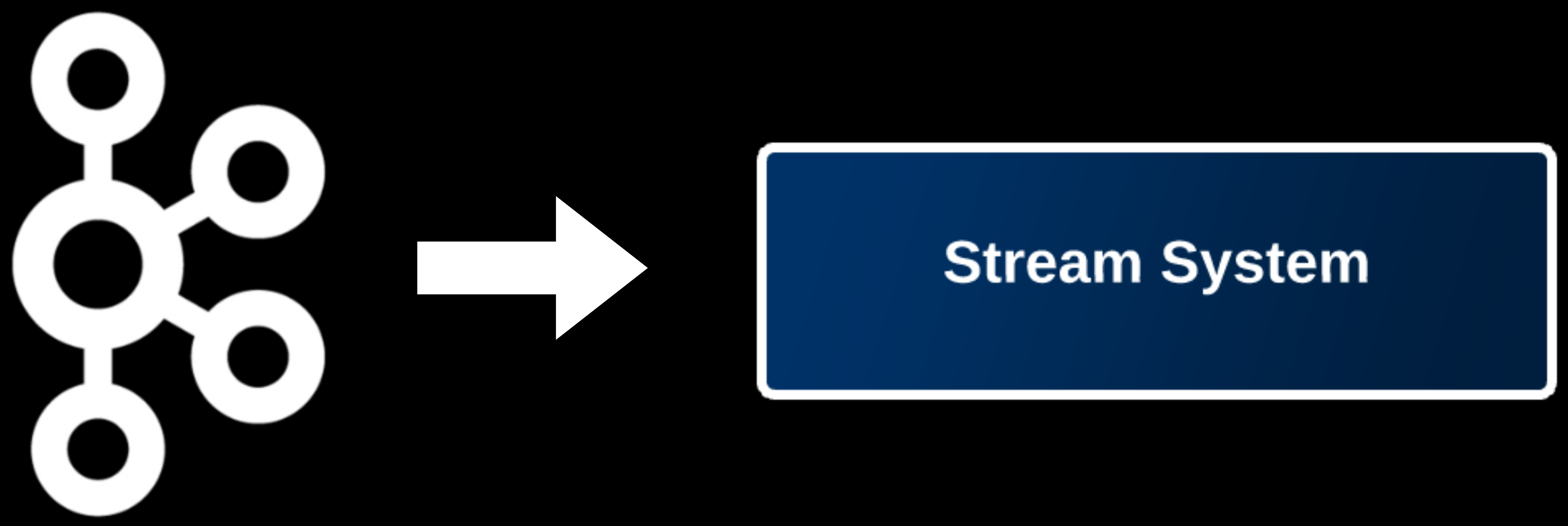
Lessons Learned

- Make sure you have robust infrastructure support

Stream System

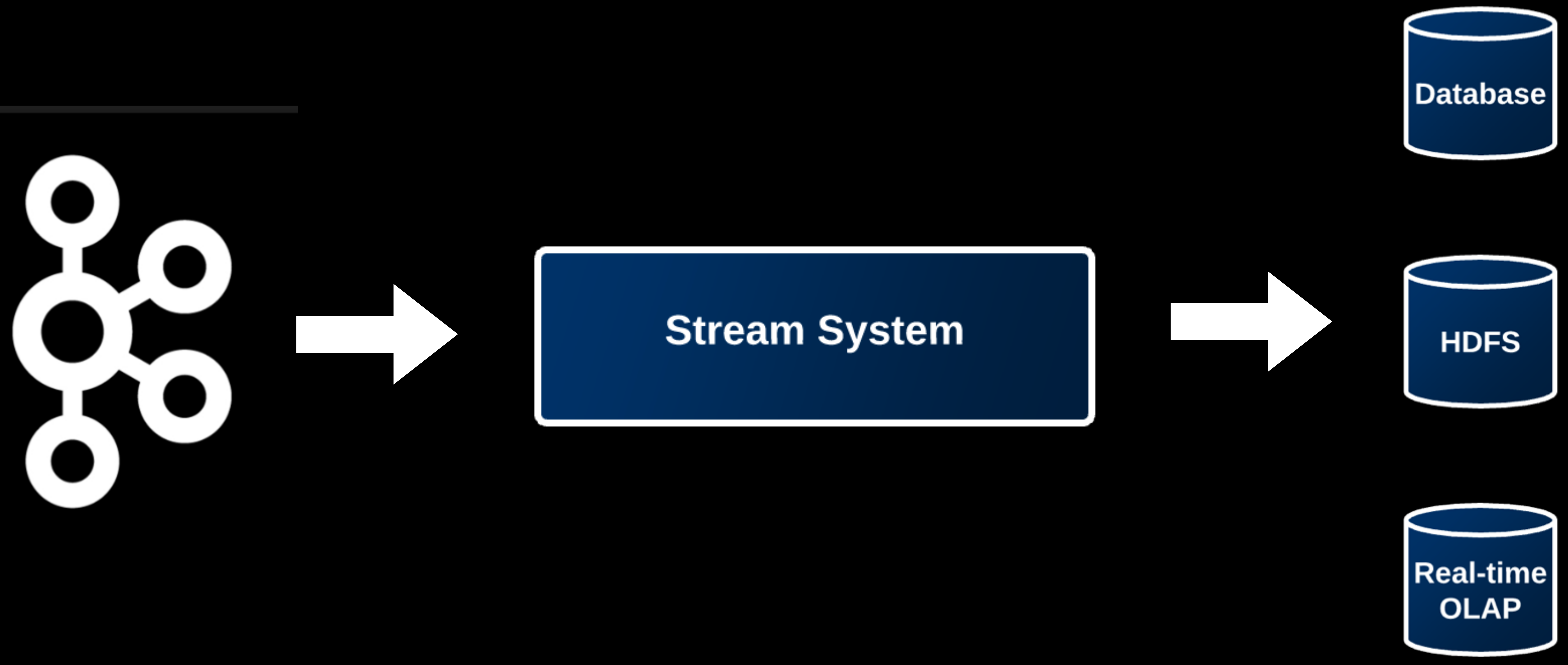
Lessons Learned

- Make sure you have robust infrastructure support



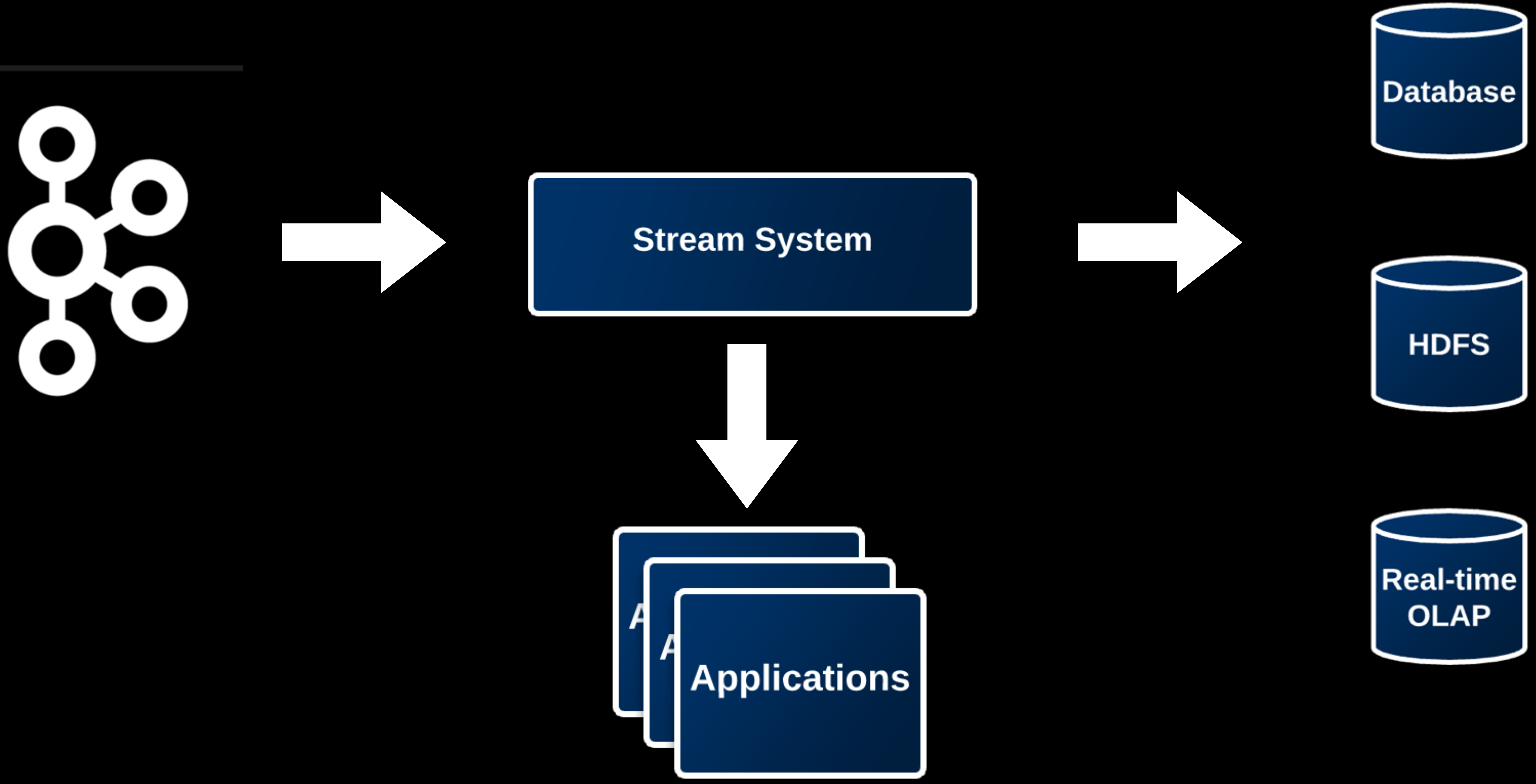
Lessons Learned

- Make sure you have robust infrastructure support



Lessons Learned

- Make sure you have robust infrastructure support



Lessons Learned

- Make sure you have robust infrastructure support
- Scaling up, namely single-node optimization matters

Lessons Learned

- Make sure you have robust infrastructure support
- Scaling up, namely single-node optimization matters
- Ensure exactly-once by proper data modeling

Lessons Learned

- Make sure you have robust infrastructure support
- Scaling up, namely single-node optimization matters
- Ensure exactly-once by proper data modeling
- Use external state store to avoid too much snapshotting
- Standardize monitoring and data validation

Lessons Learned

- Make sure you have robust infrastructure support
- Scaling up, namely single-node optimization matters
- Ensure exactly-once by proper data modeling
- Standardize monitoring and data validation

Choose a Stream Processing Platform

samza

Spark

beam

Thank You