Shared abstraction: multi-dimensional geo-temporal

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

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

Shared abstraction: multi-dimensional geo-temporal

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

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

Shared abstraction: multi-dimensional geo-temporal

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

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

Shared abstraction: multi-dimensional geo-temporal

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

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

State

Shared abstraction: multi-dimensional geo-temporal

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

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

State per key

Shared abstraction: multi-dimensional geo-temporal

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

Shared abstraction: multi-dimensional geo-temporal

- 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

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

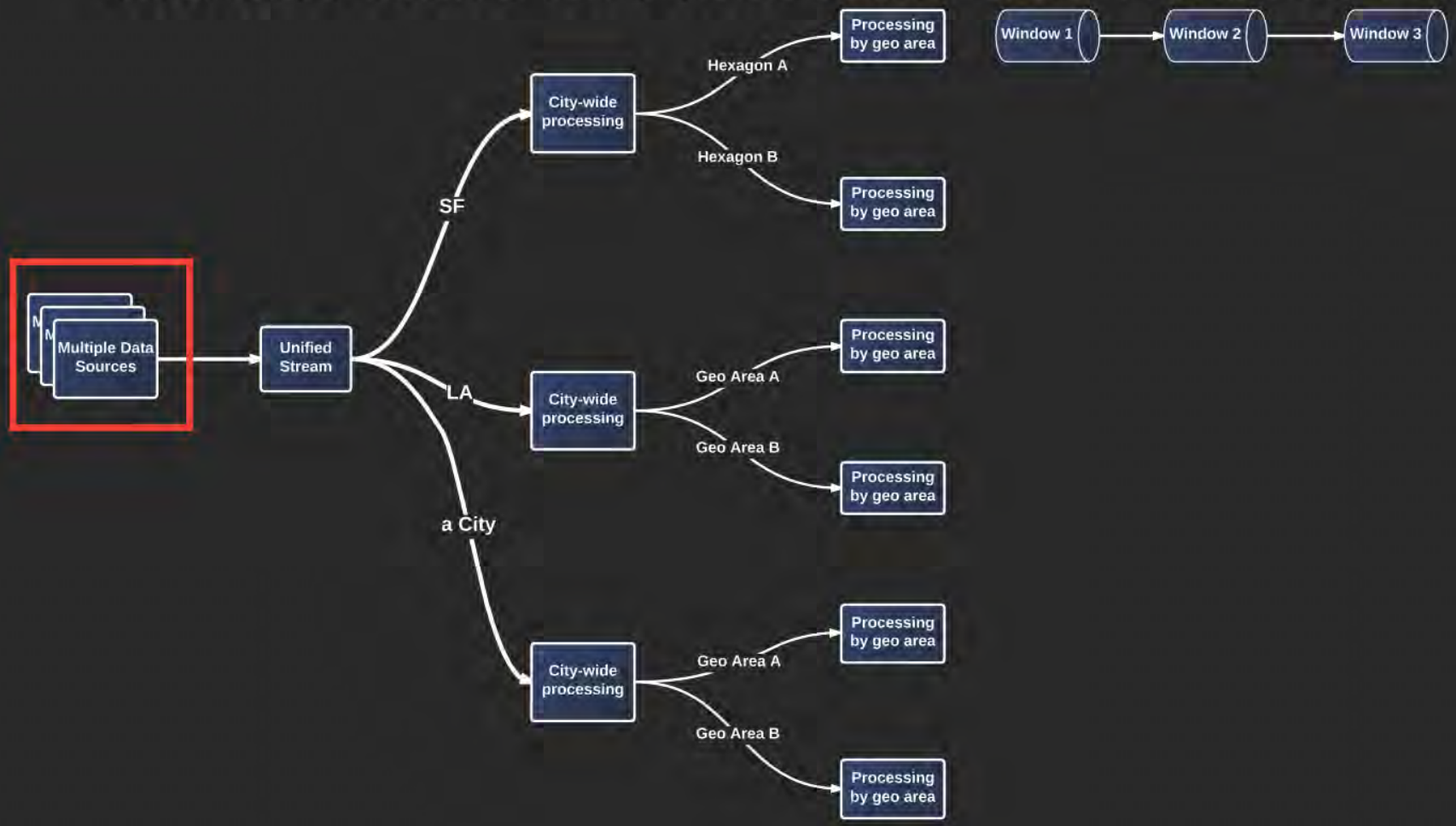
Shared abstraction: multi-dimensional geo-temporal

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

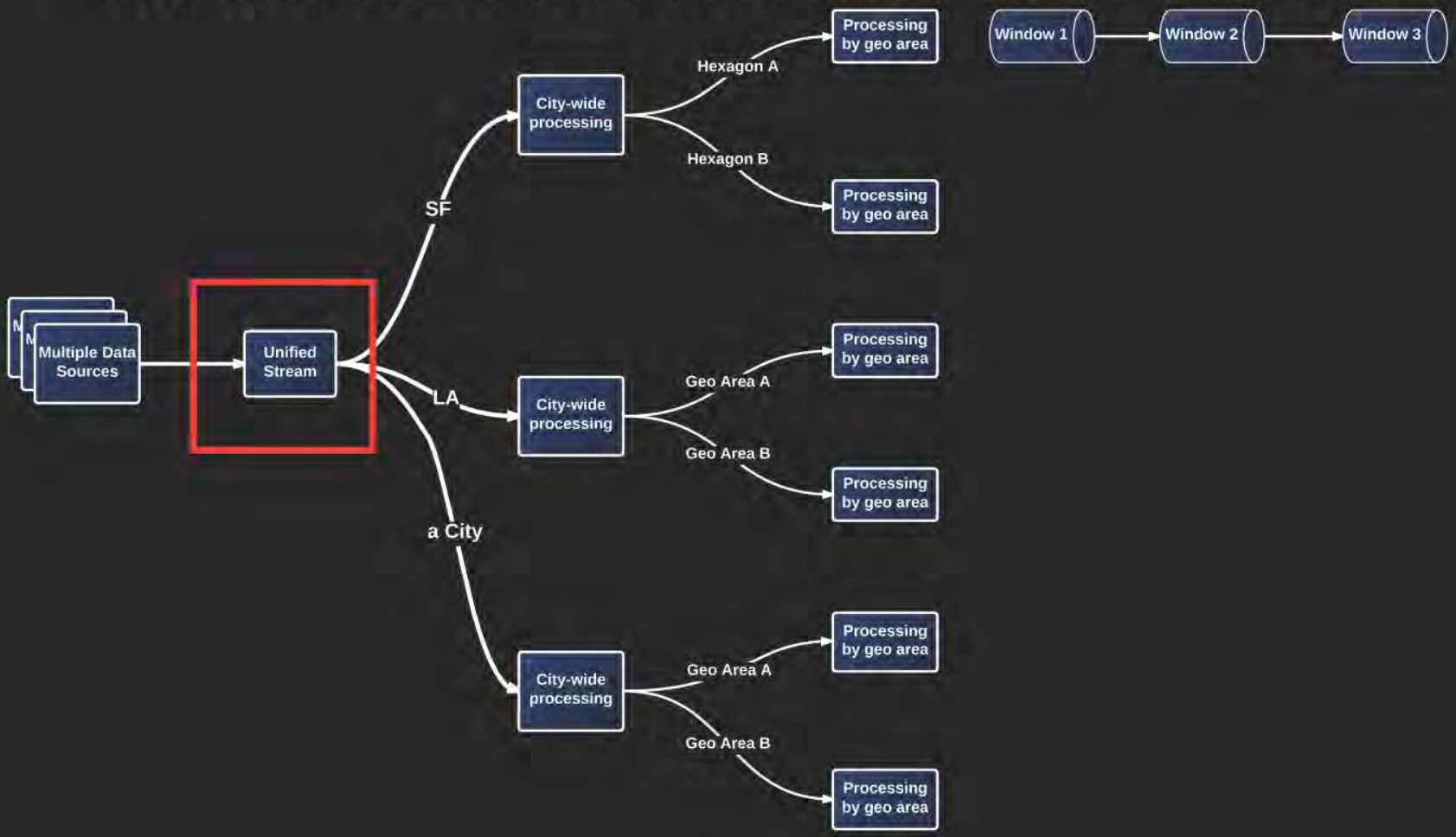
Apache Flink

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

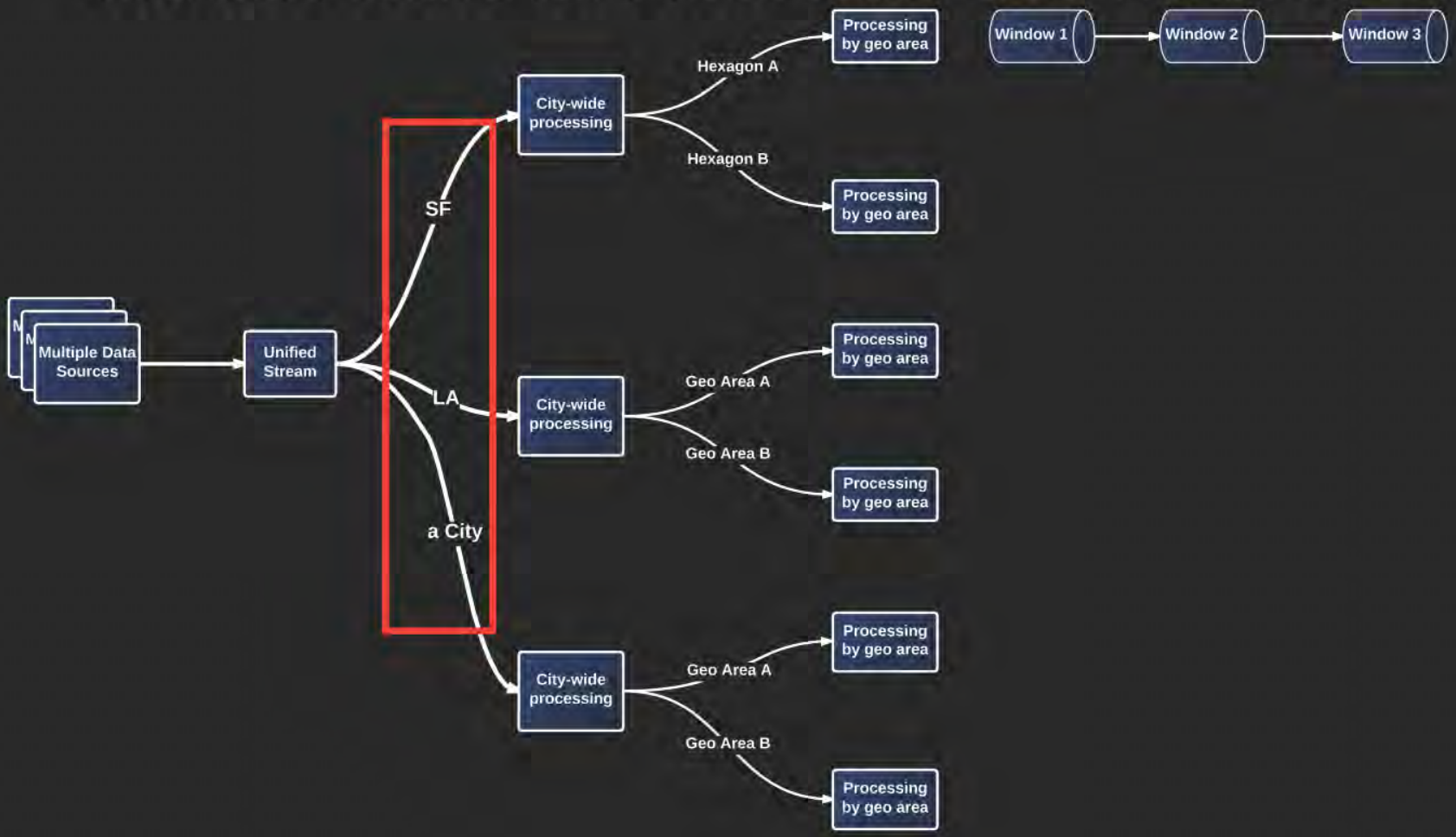
Mental Picture for Processing Geo-temporal Data



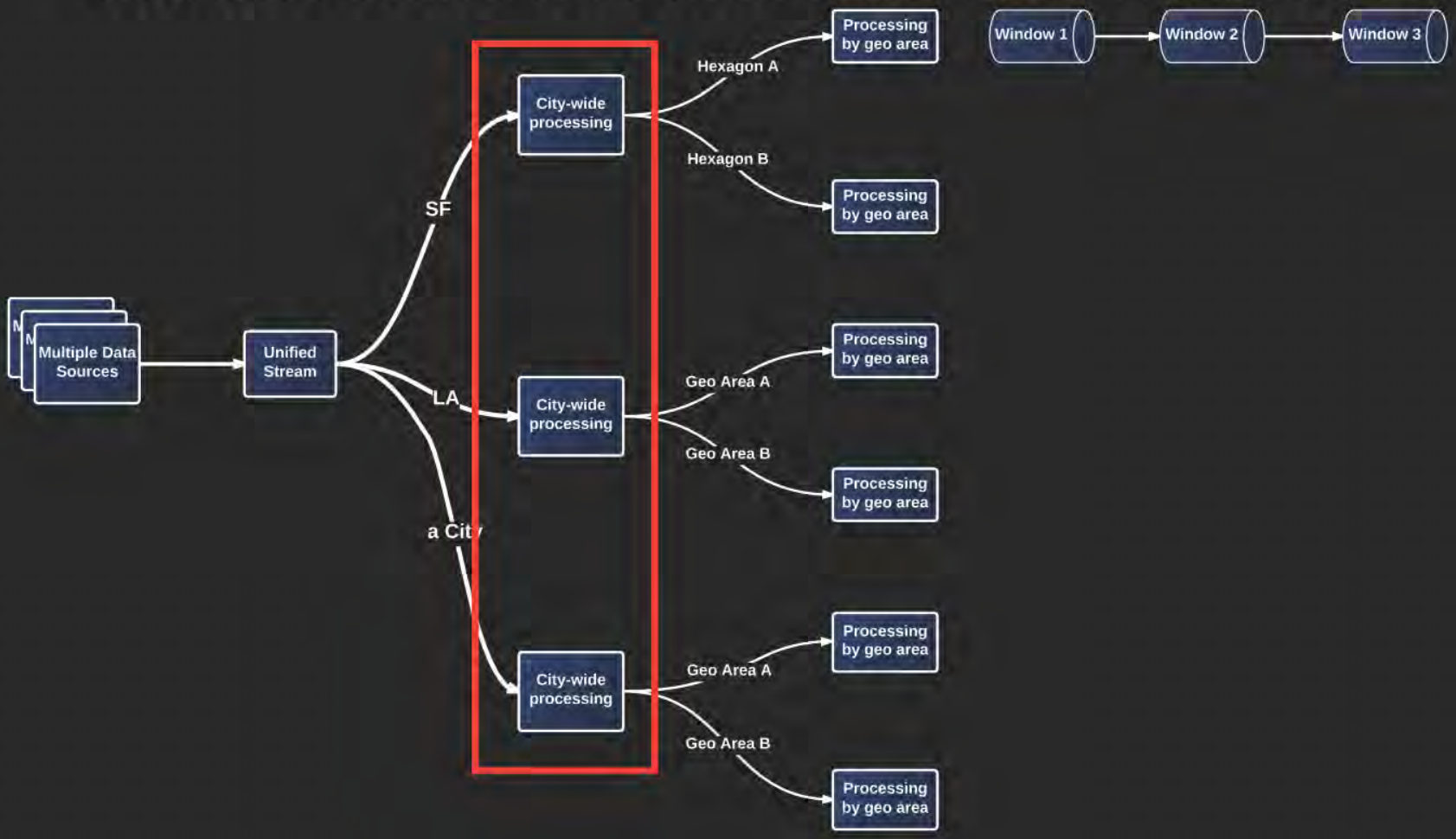
Mental Picture for Processing Geo-temporal Data



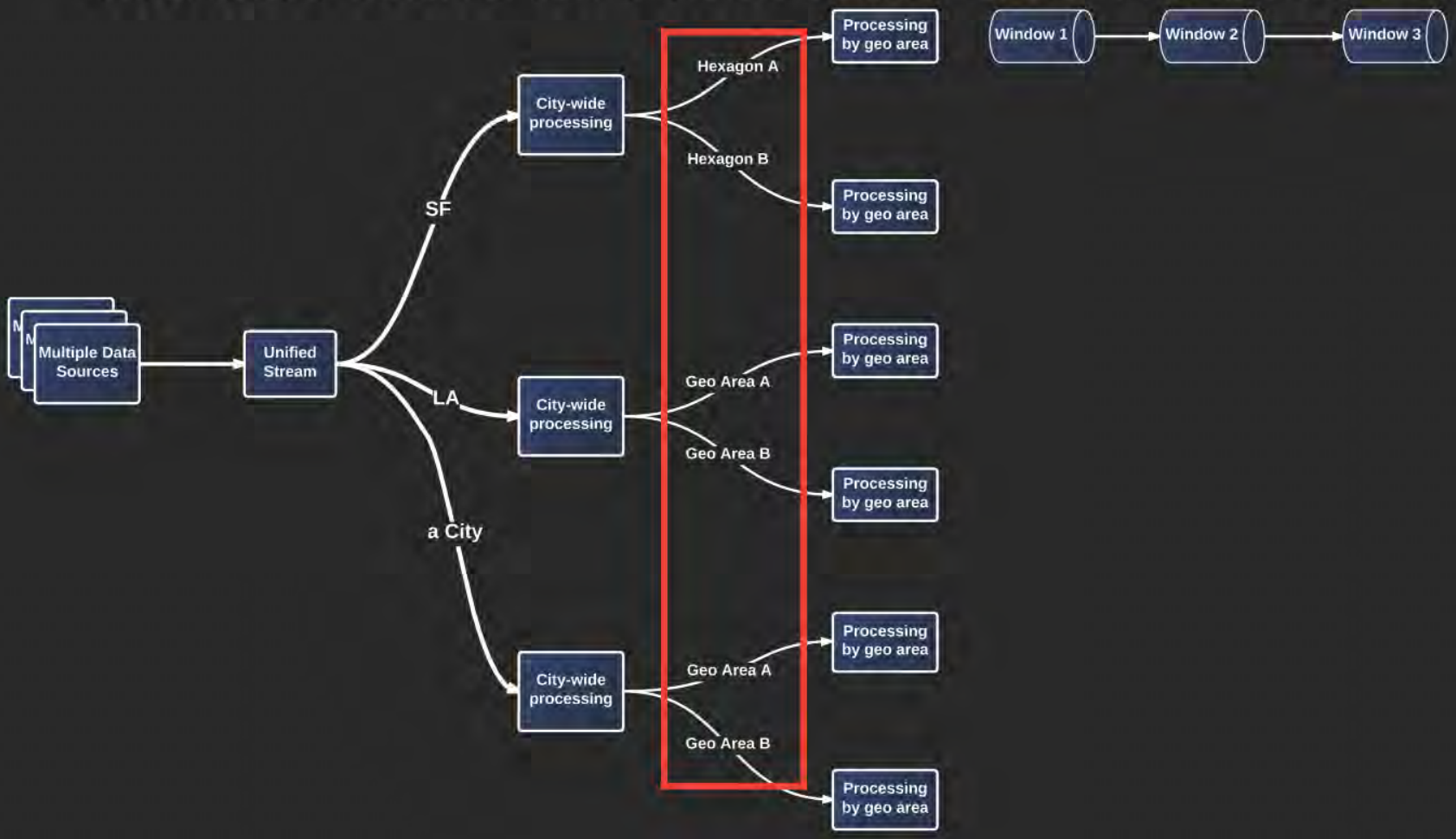
Mental Picture for Processing Geo-temporal Data



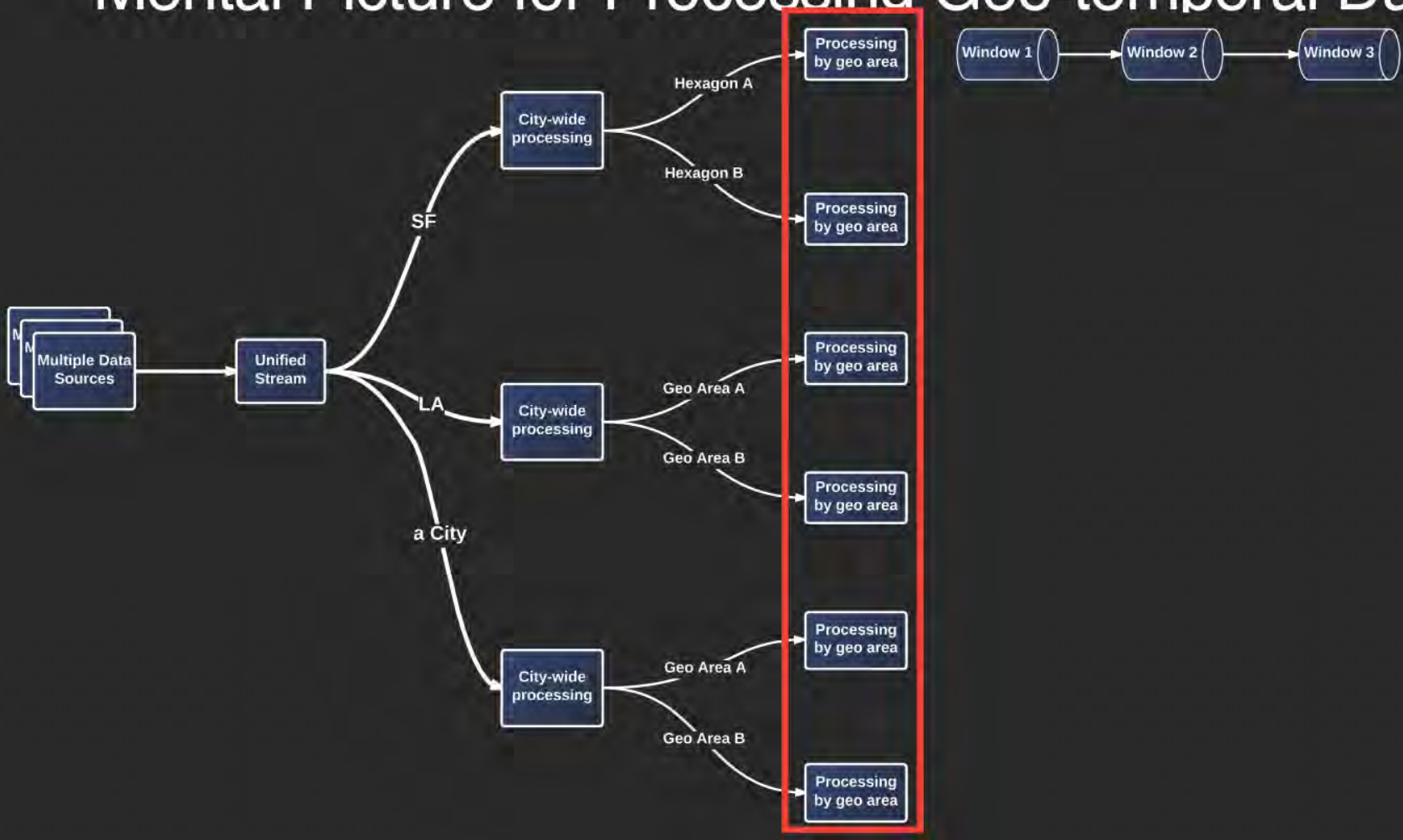
Mental Picture for Processing Geo-temporal Data



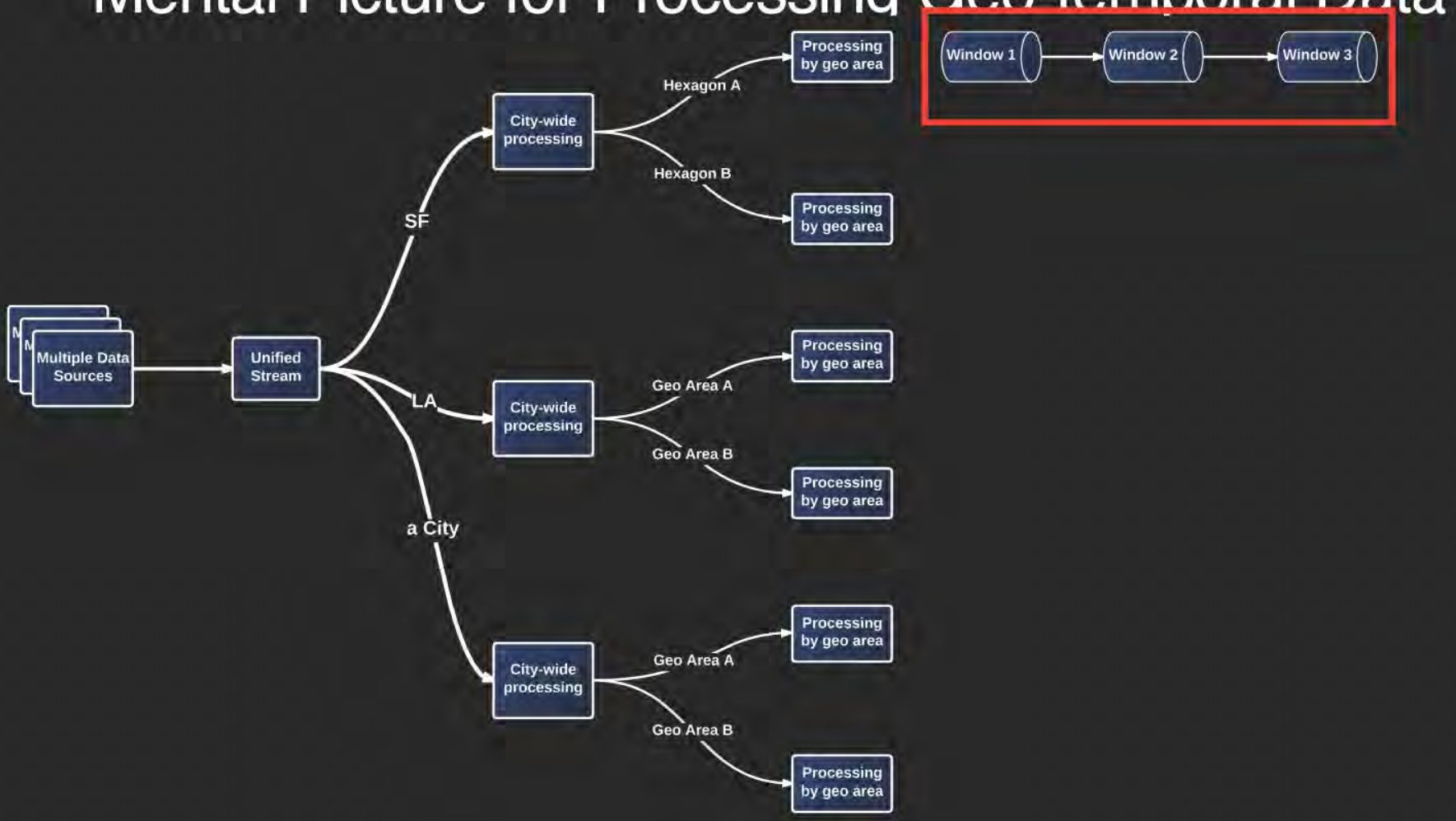
Mental Picture for Processing Geo-temporal Data



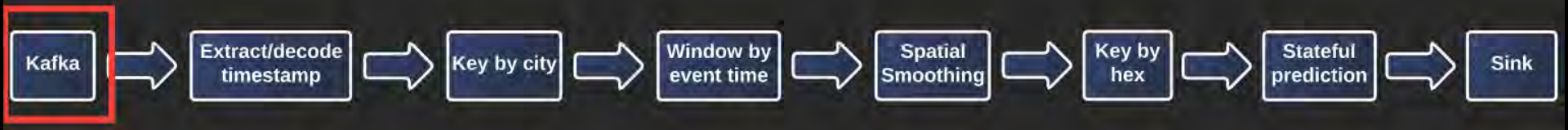
Mental Picture for Processing Geo-temporal Data



Mental Picture for Processing Geo-temporal Data



A Simple Example: simple prediction



Sources

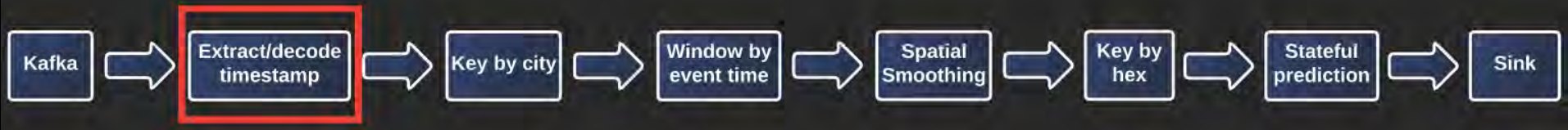
.fromKafka()

.config(config)

.cluster(aCluster)

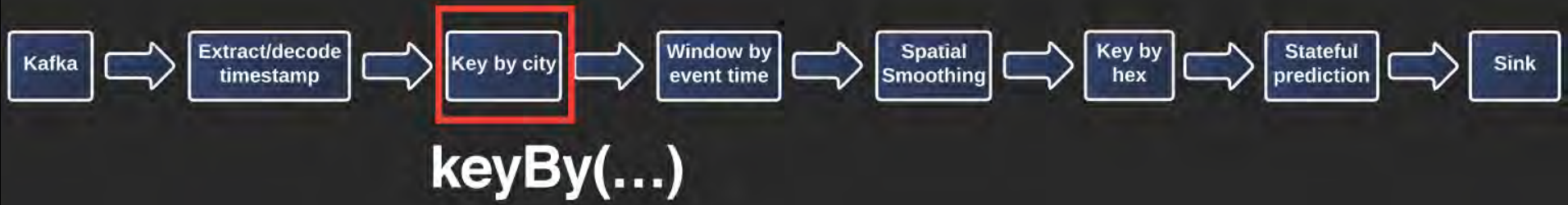
.topics(topicList)

A Simple Example



assignTimestampsAndWatermarks

A Simple Example



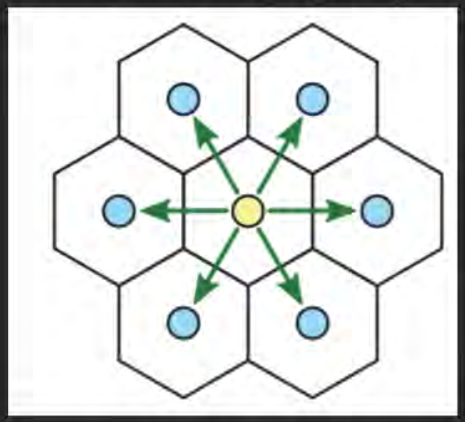
keyBy(…)

A Simple Example

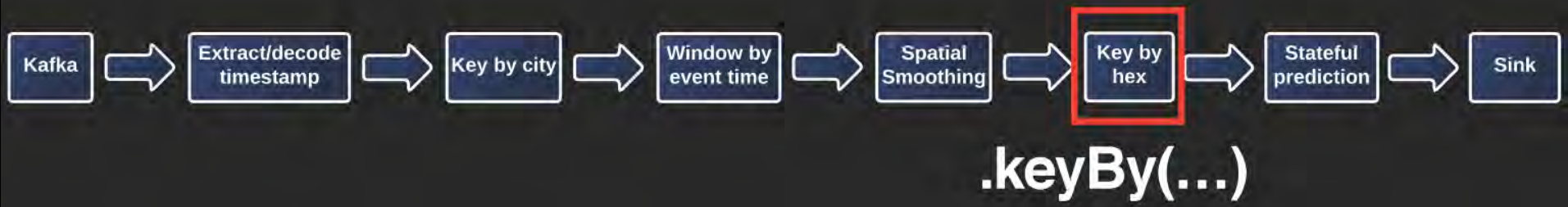
.timeWindow(…)

A Simple Example

.flatMap(…)

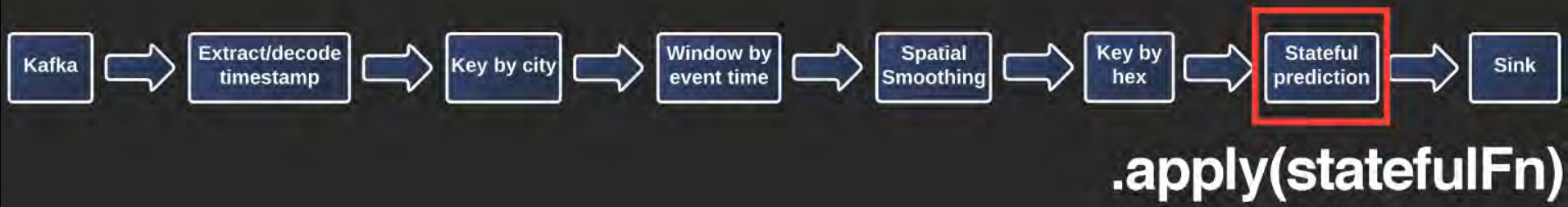


A Simple Example



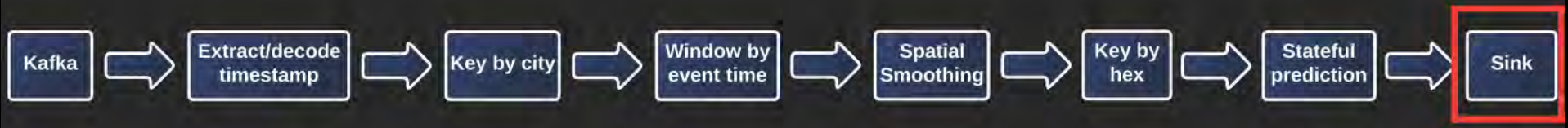
.keyBy(…)

A Simple Example



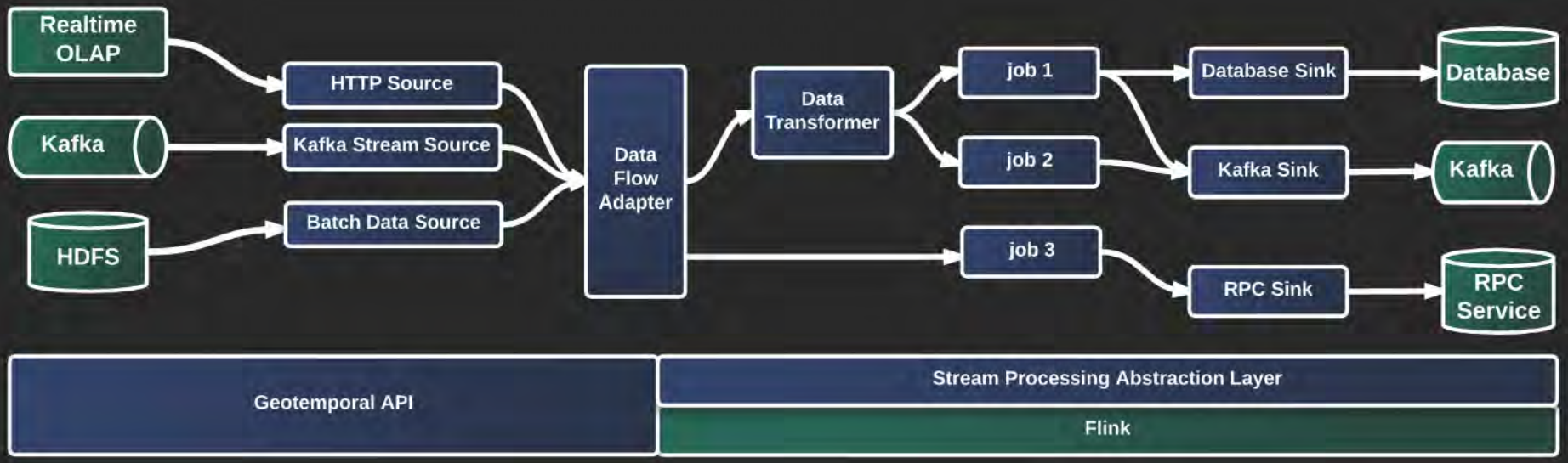
.apply(statefulFn)

A Simple Example

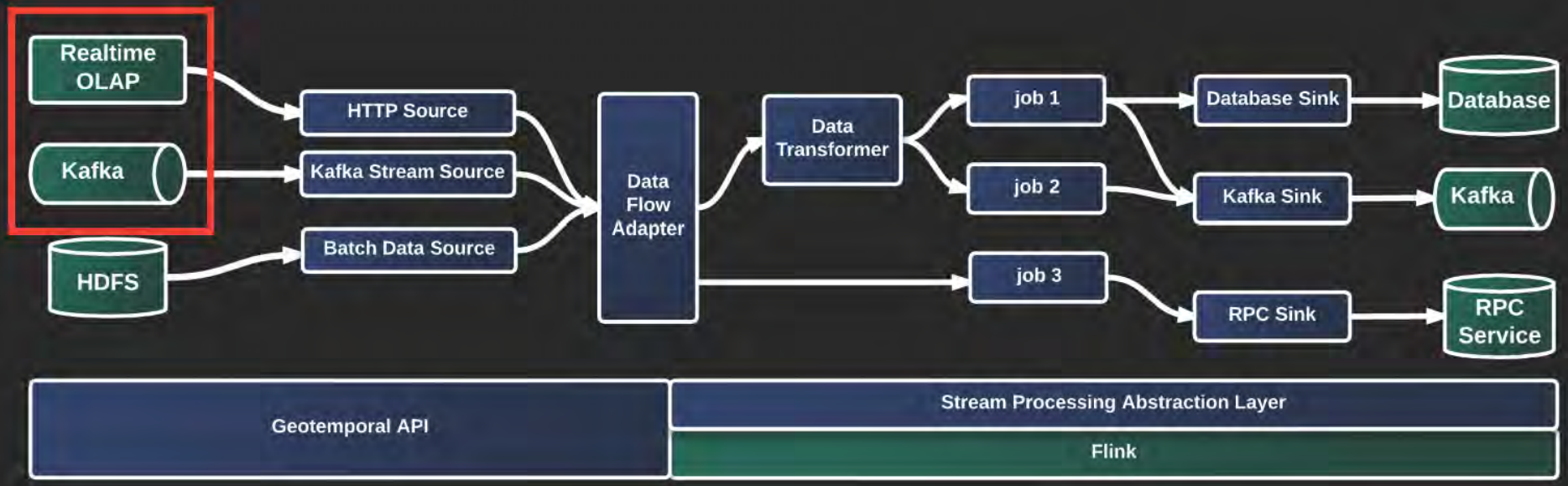


.addSink(.

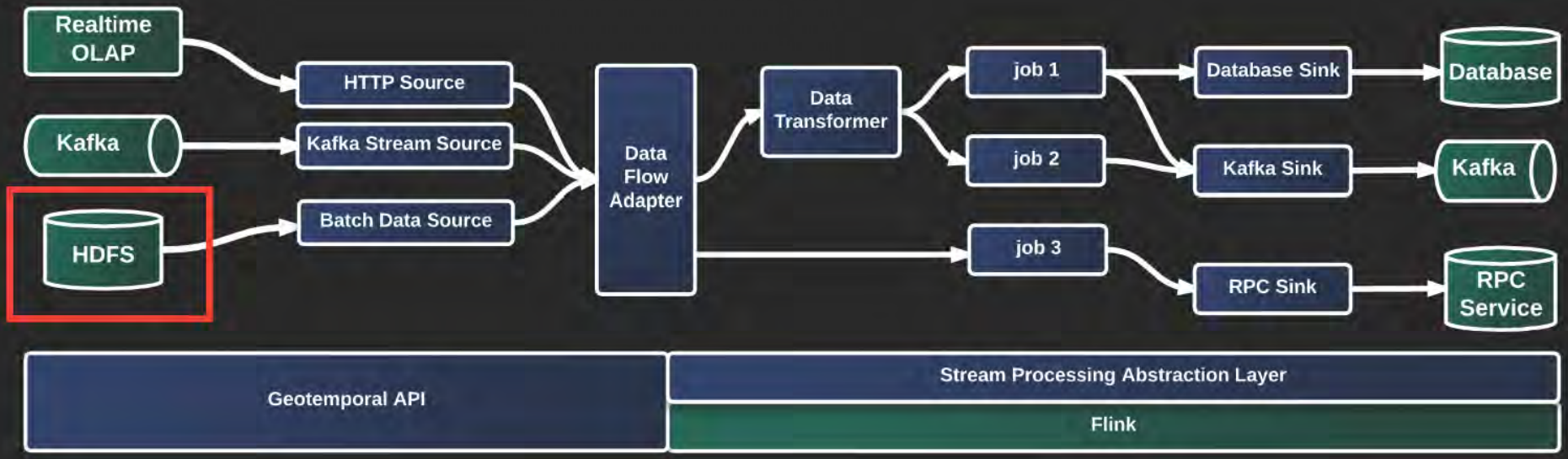
High Level Data Flow



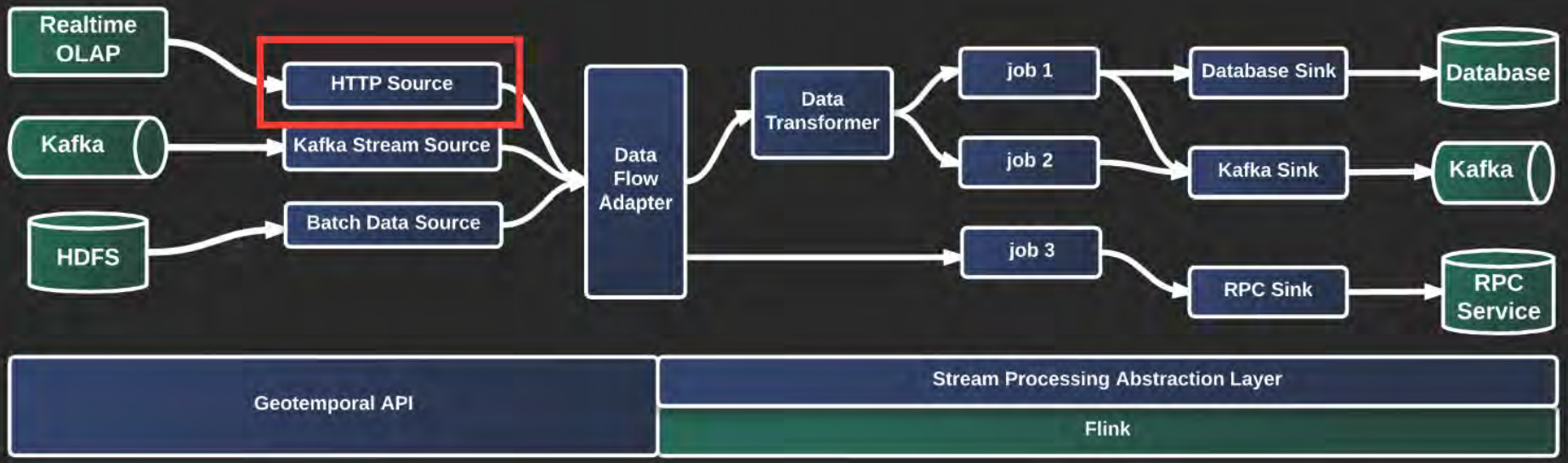
High Level Data Flow



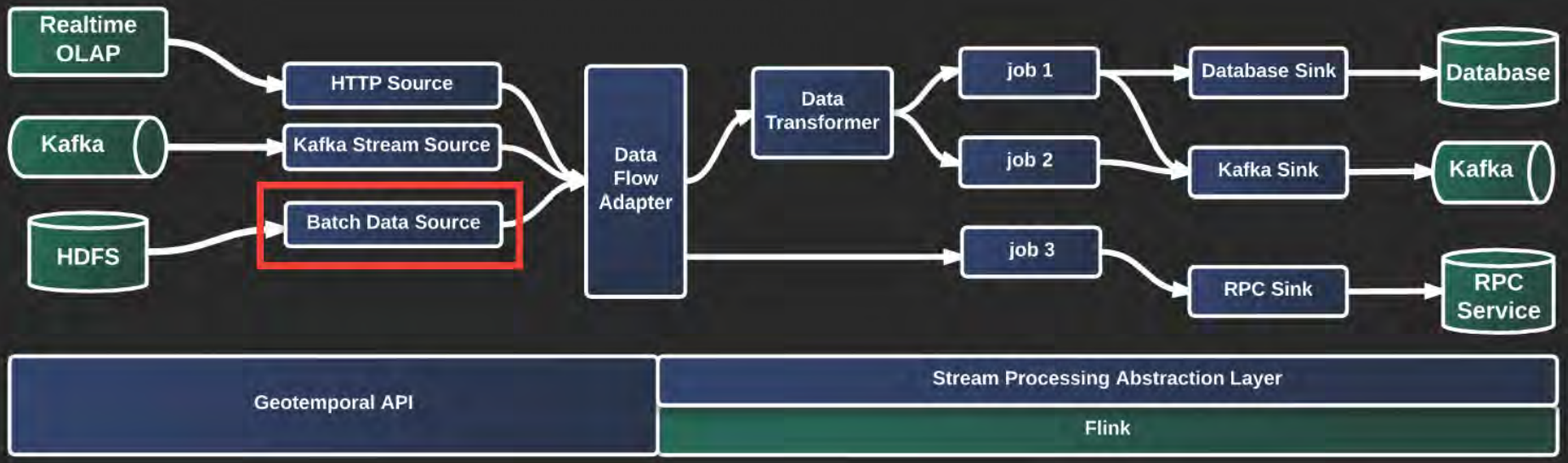
High Level Data Flow



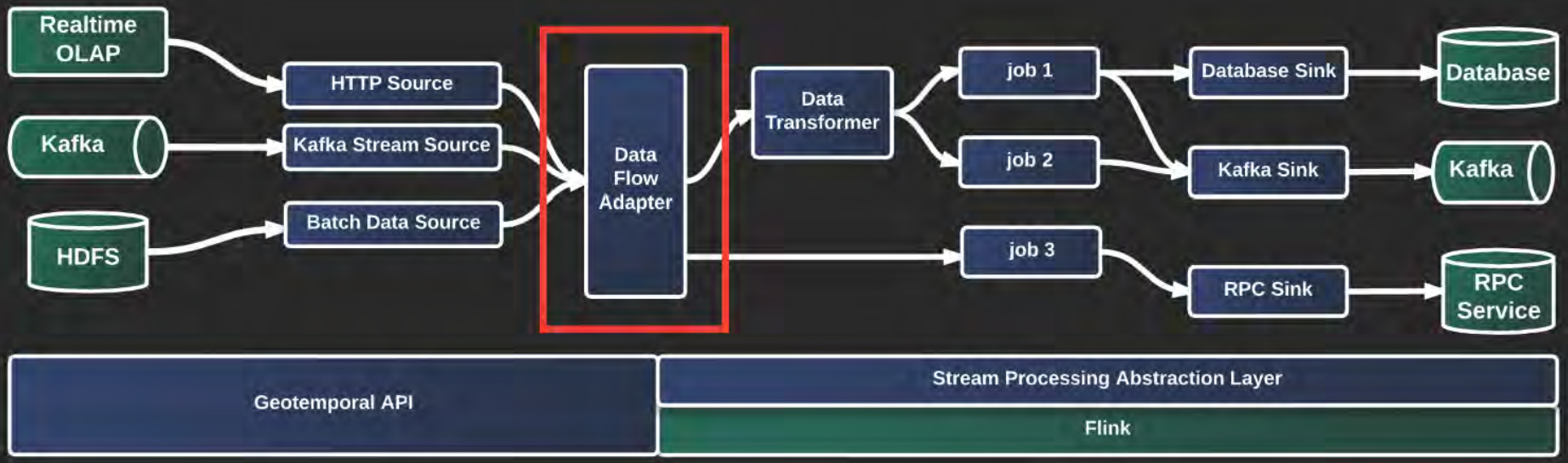
High Level Data Flow



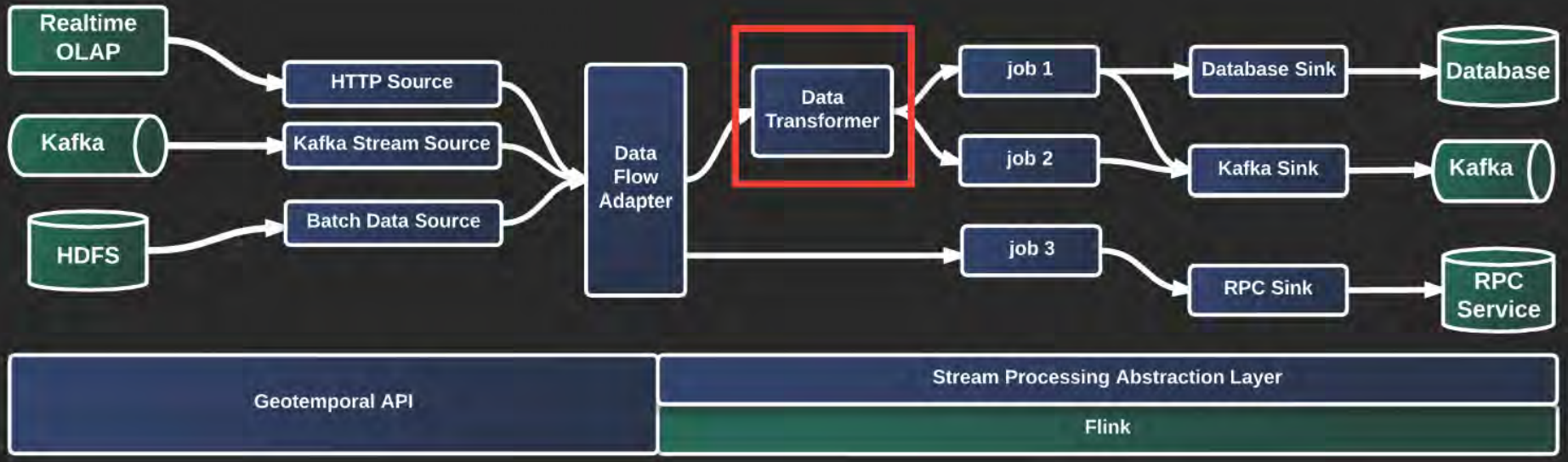
High Level Data Flow



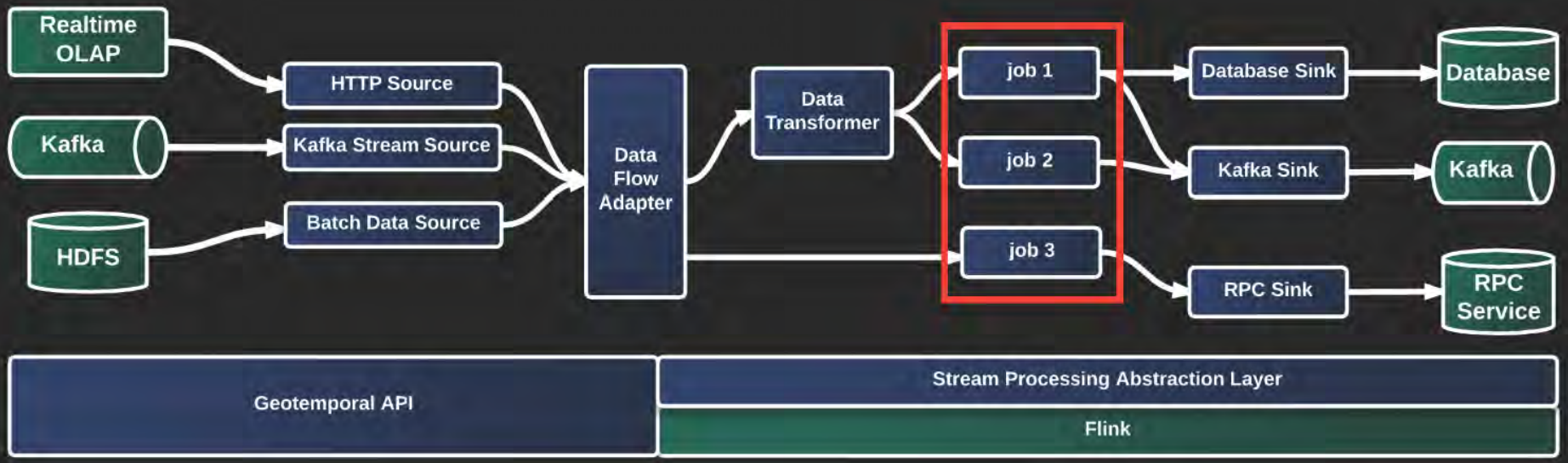
High Level Data Flow



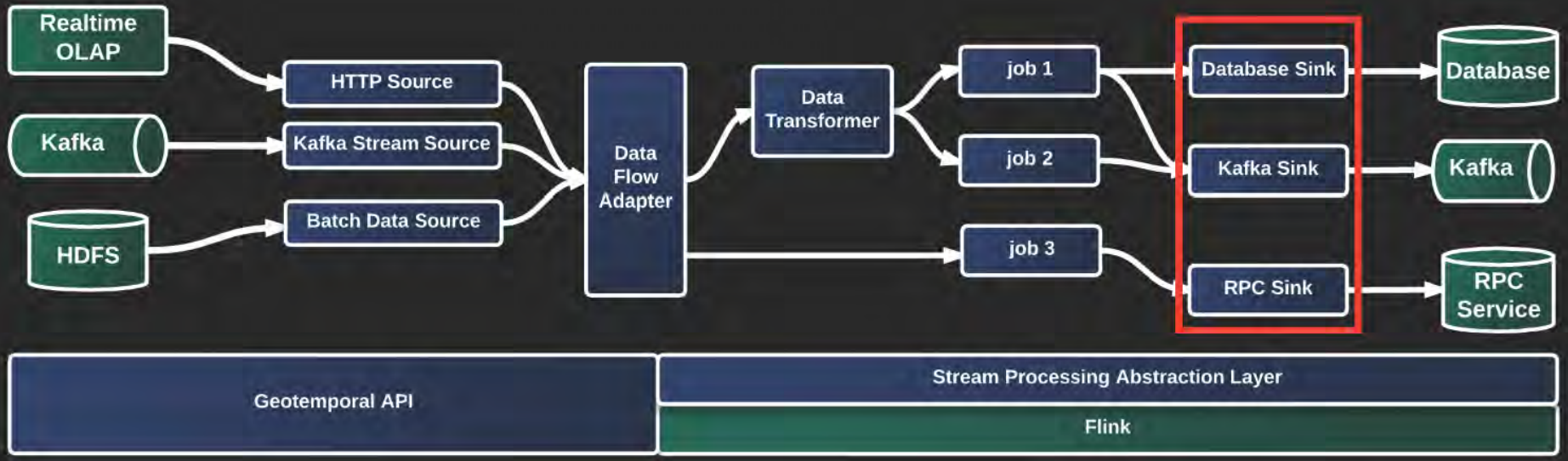
High Level Data Flow



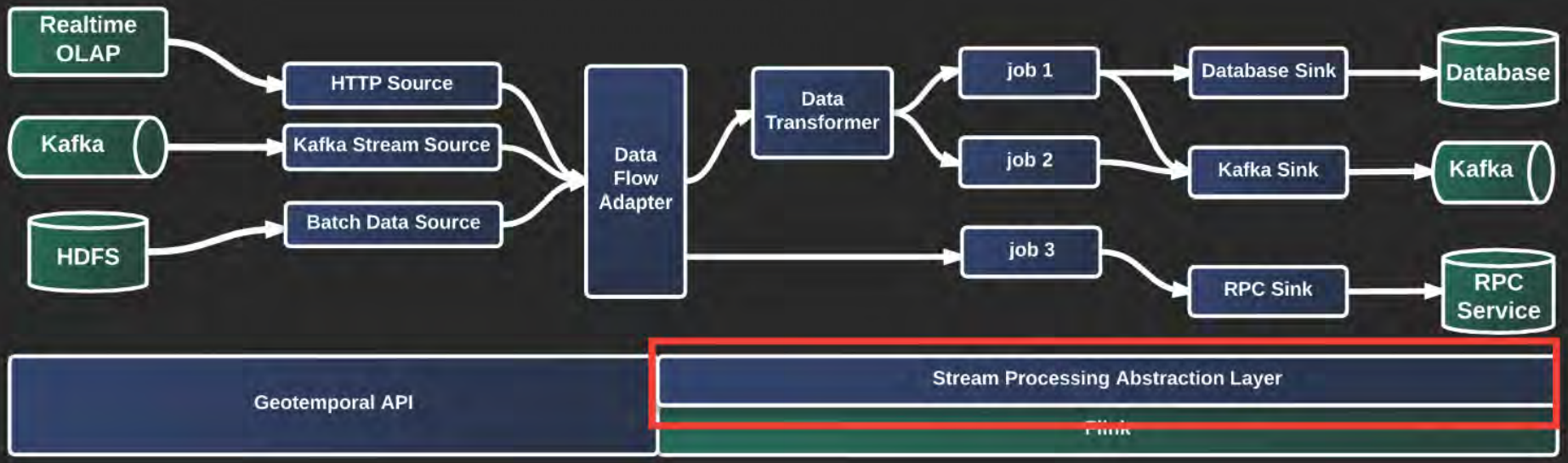
High Level Data Flow



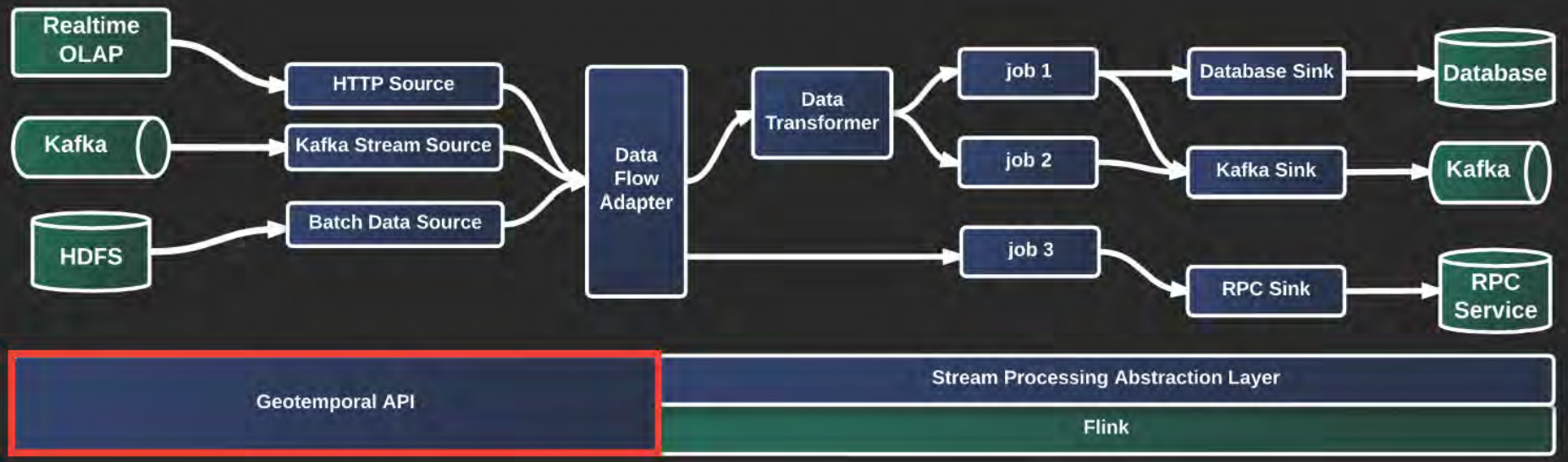
High Level Data Flow



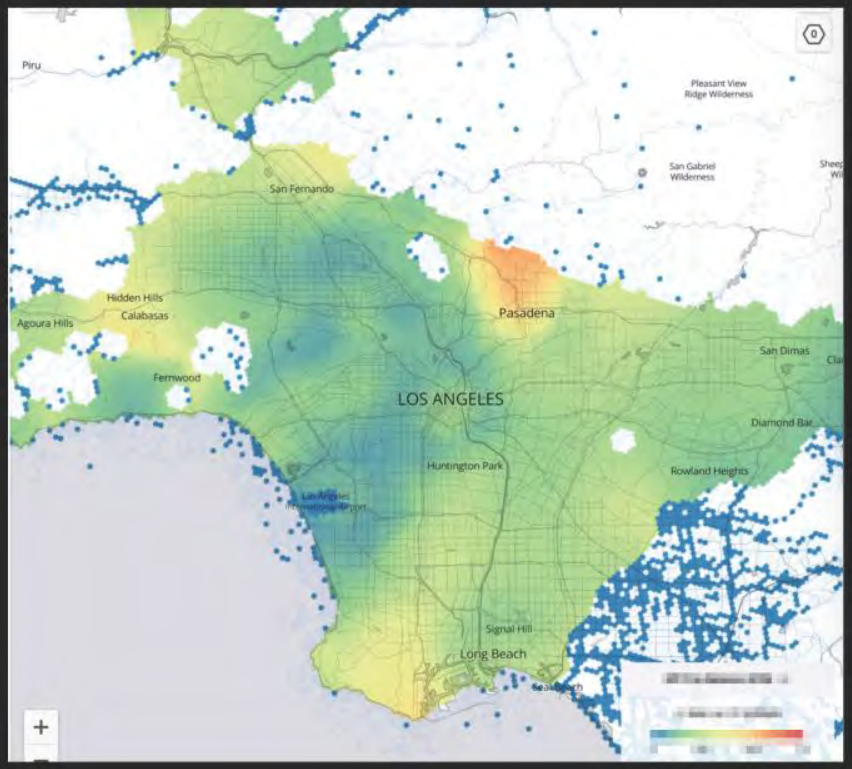
High Level Data Flow



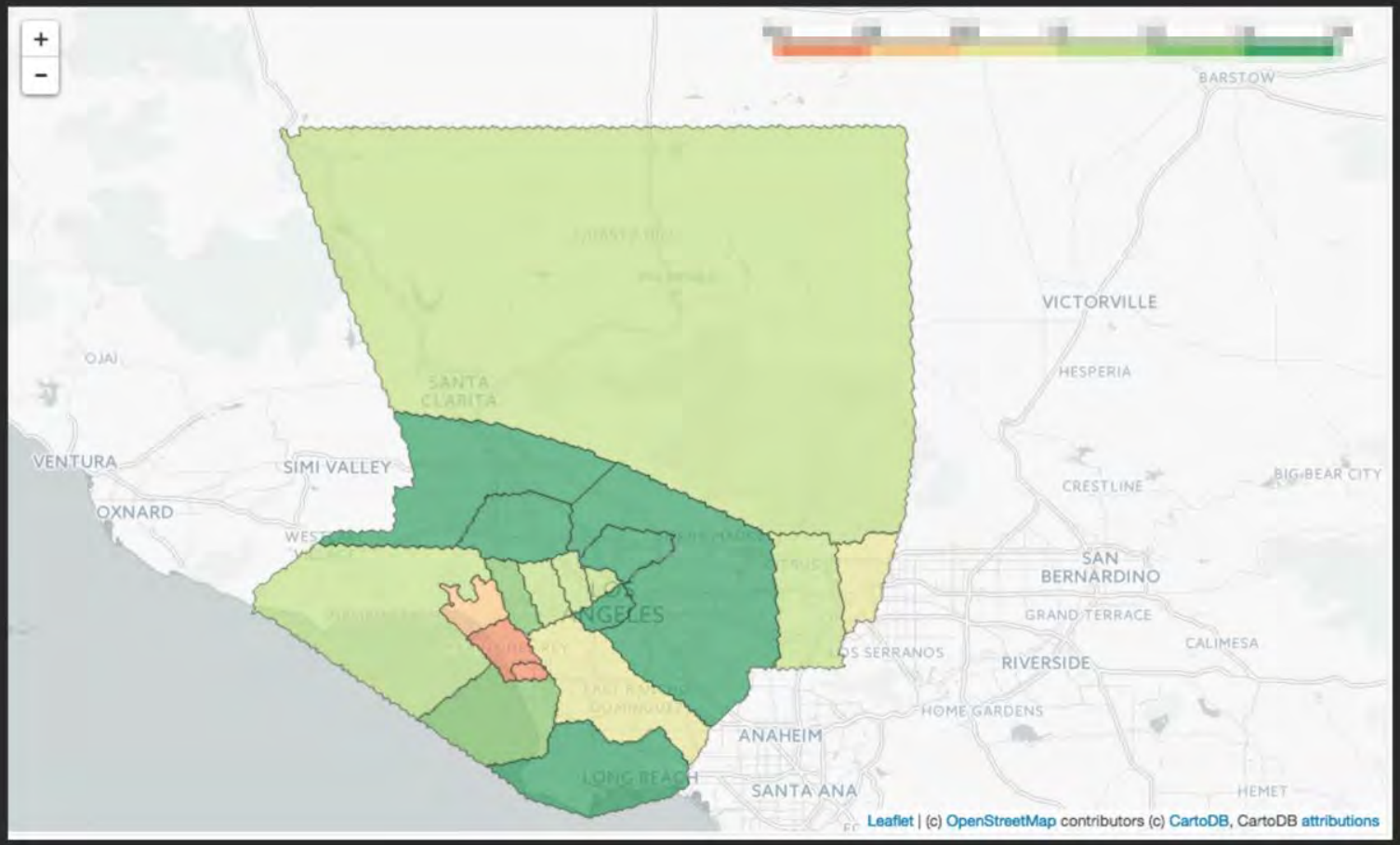
High Level Data Flow



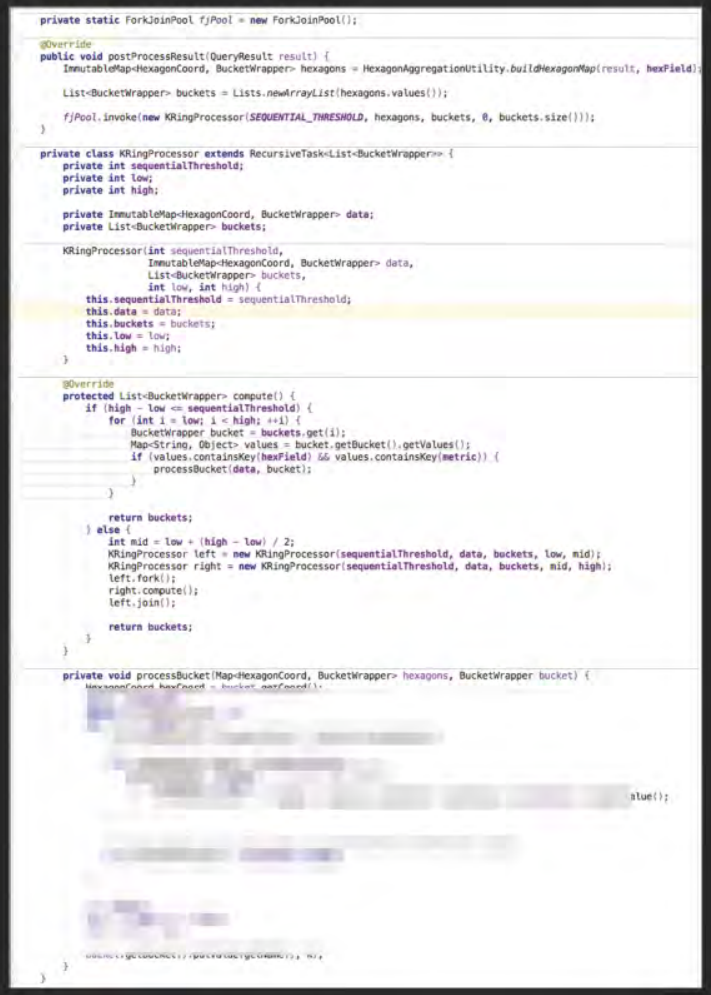
Geotemporal API for efficiency



Geotemporal API for efficiency

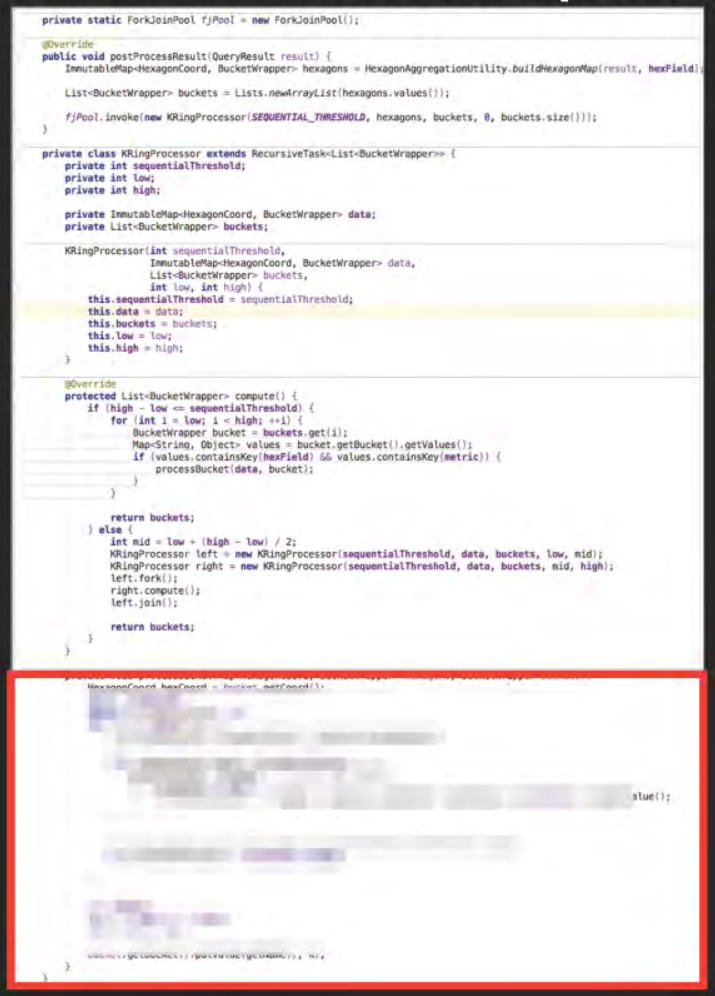


Geotemporal API for productivity



```
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();
        [redacted]
                                                                                    alue();
        [redacted]
    }
}
```

Geotemporal API for productivity



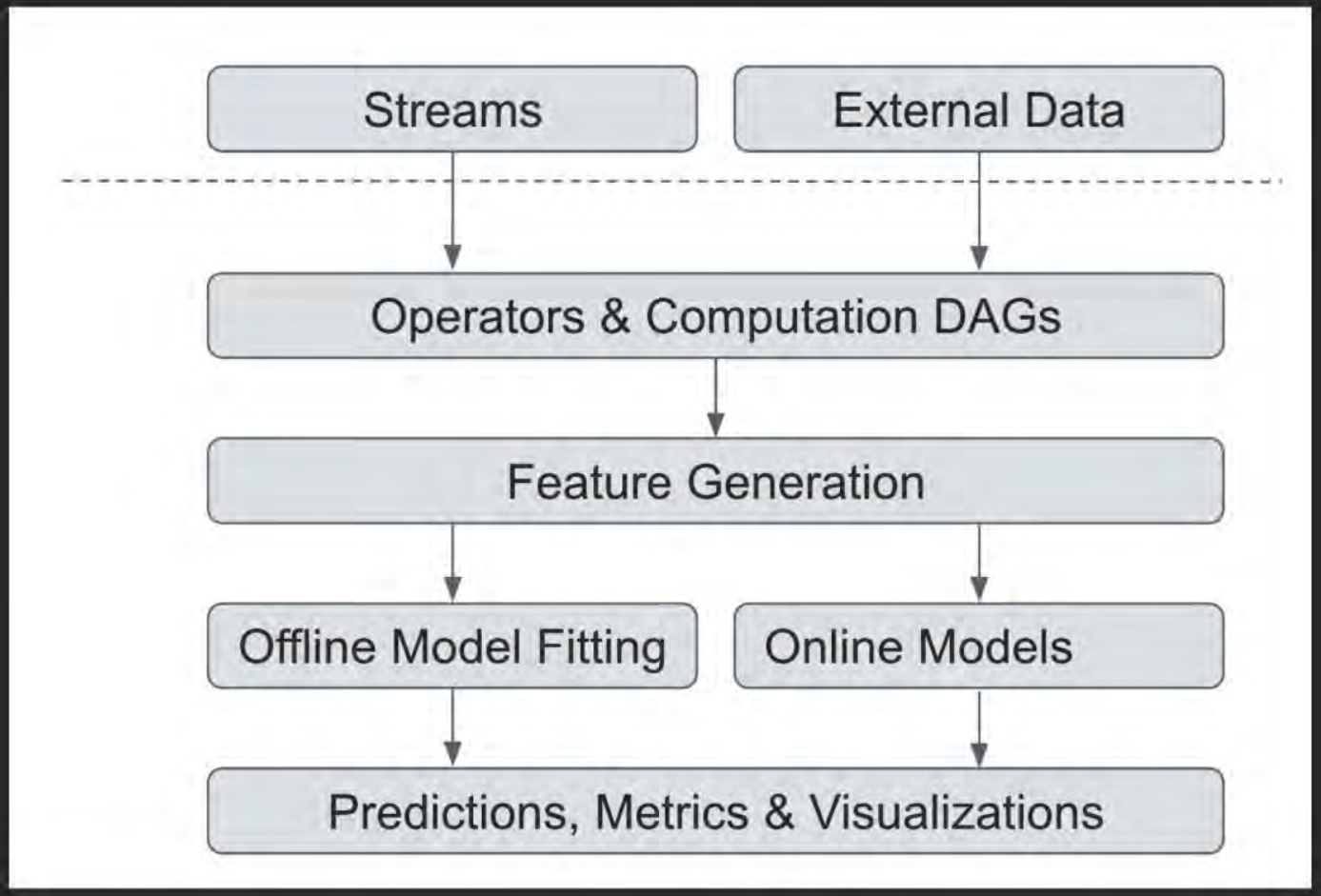
```
private static ForkJoinPool fjPool = new ForkJoinPool();

@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;
        }
    }

    [redacted]
}
```

Geotemporal API for productivity

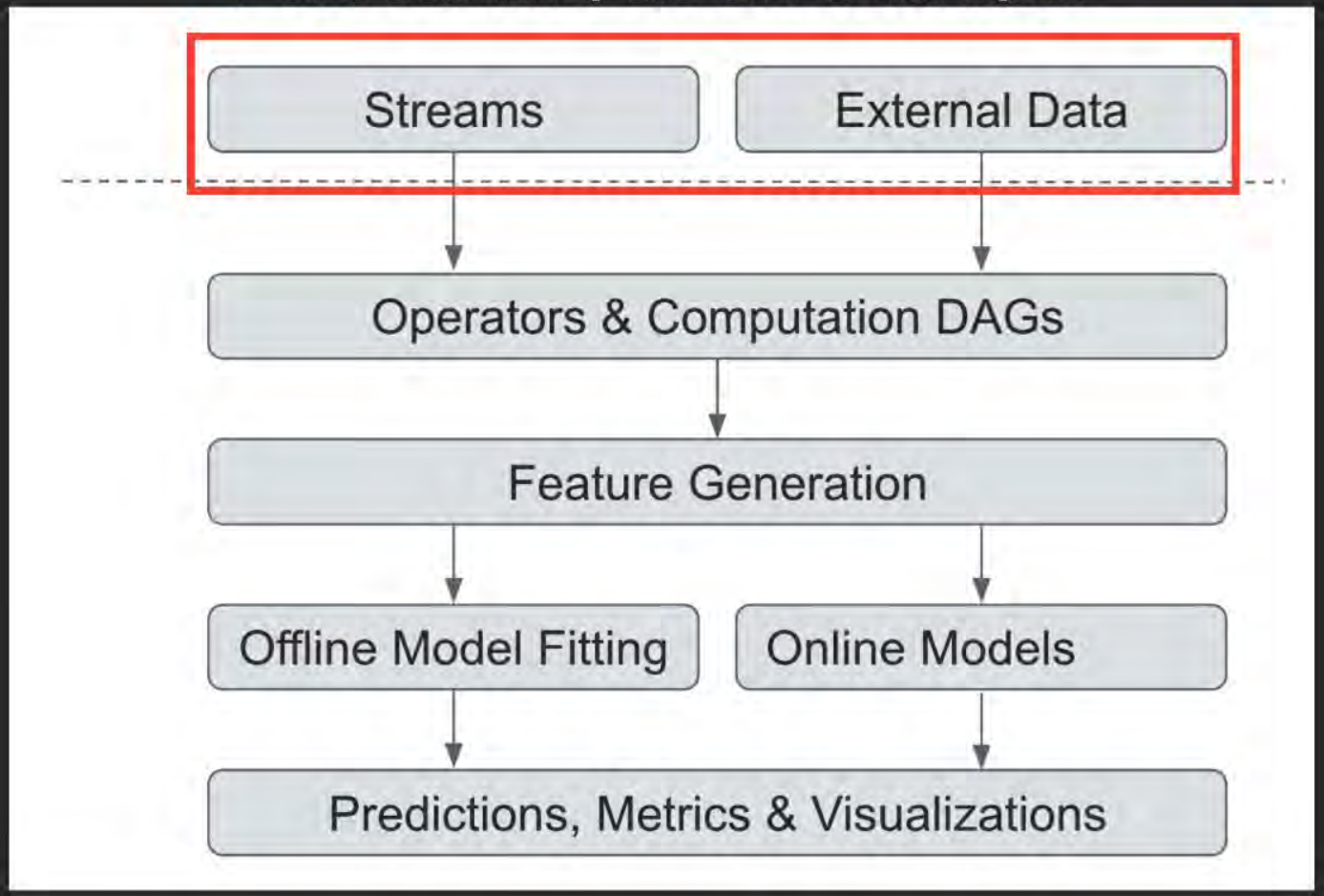
```
private static ForkJoinPool fjPool = new ForkJoinPool();

@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();
        [redacted]
        alue();
        [redacted]
    }
}
```

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

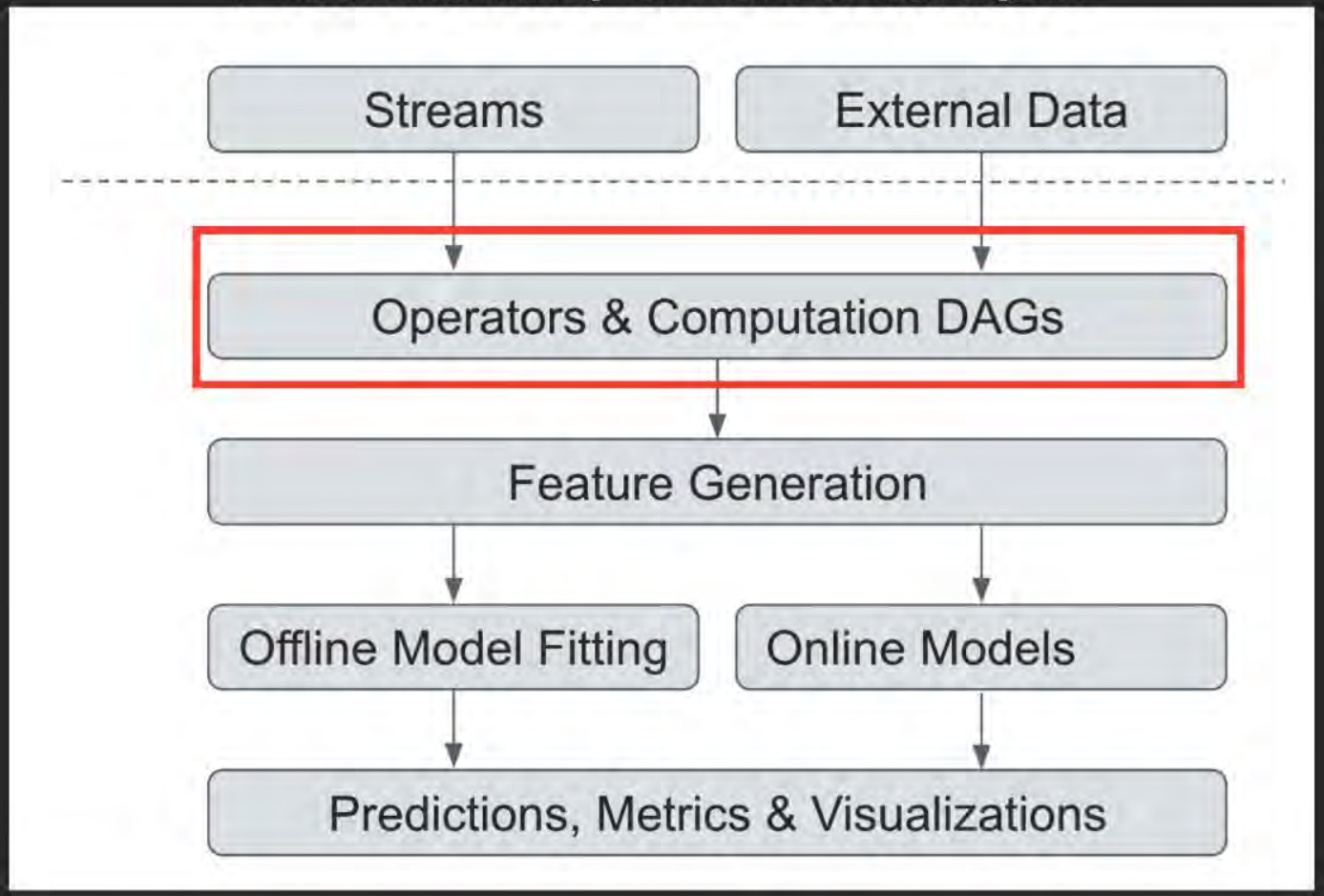
Forecasting as an example



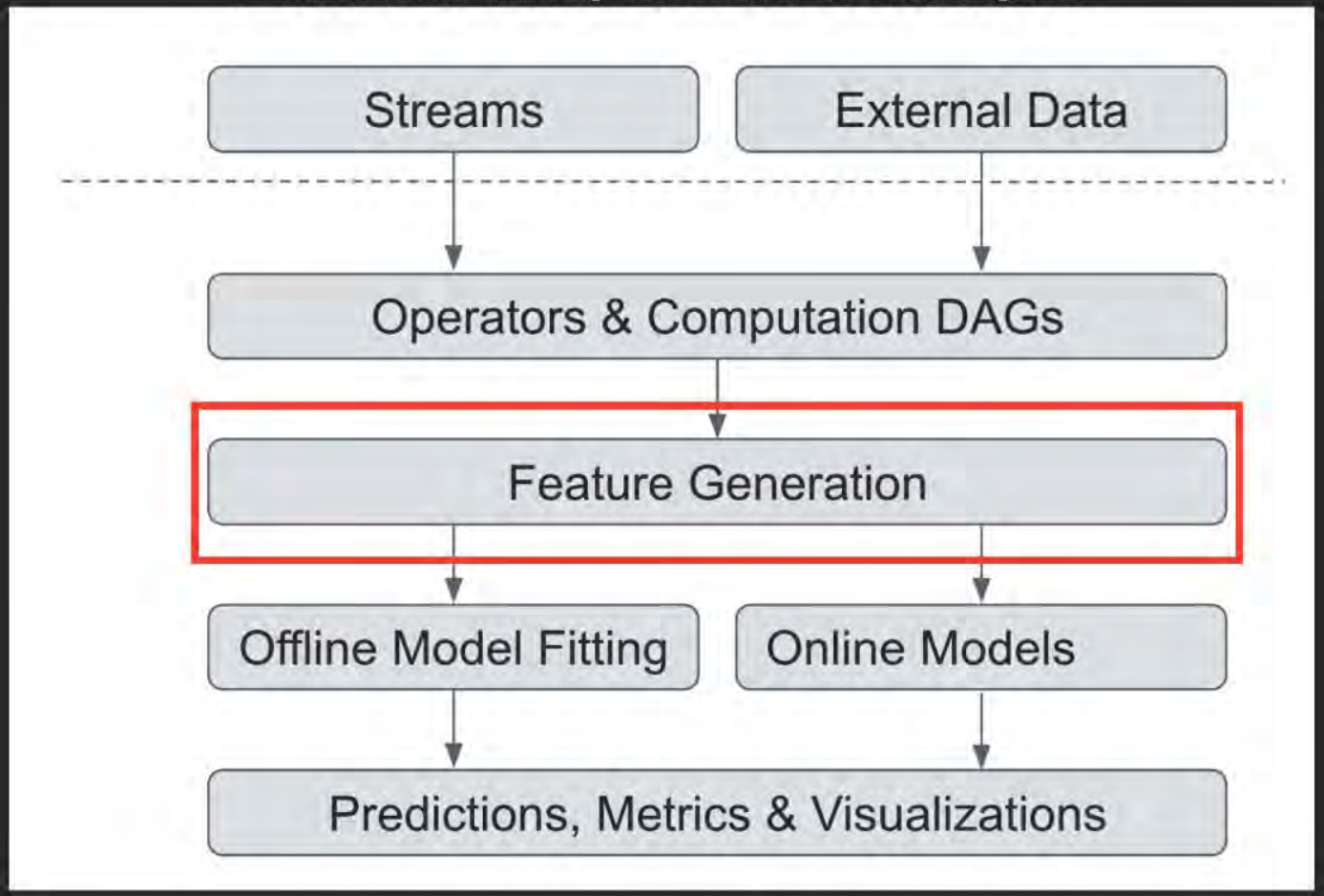
Forecasting as an example



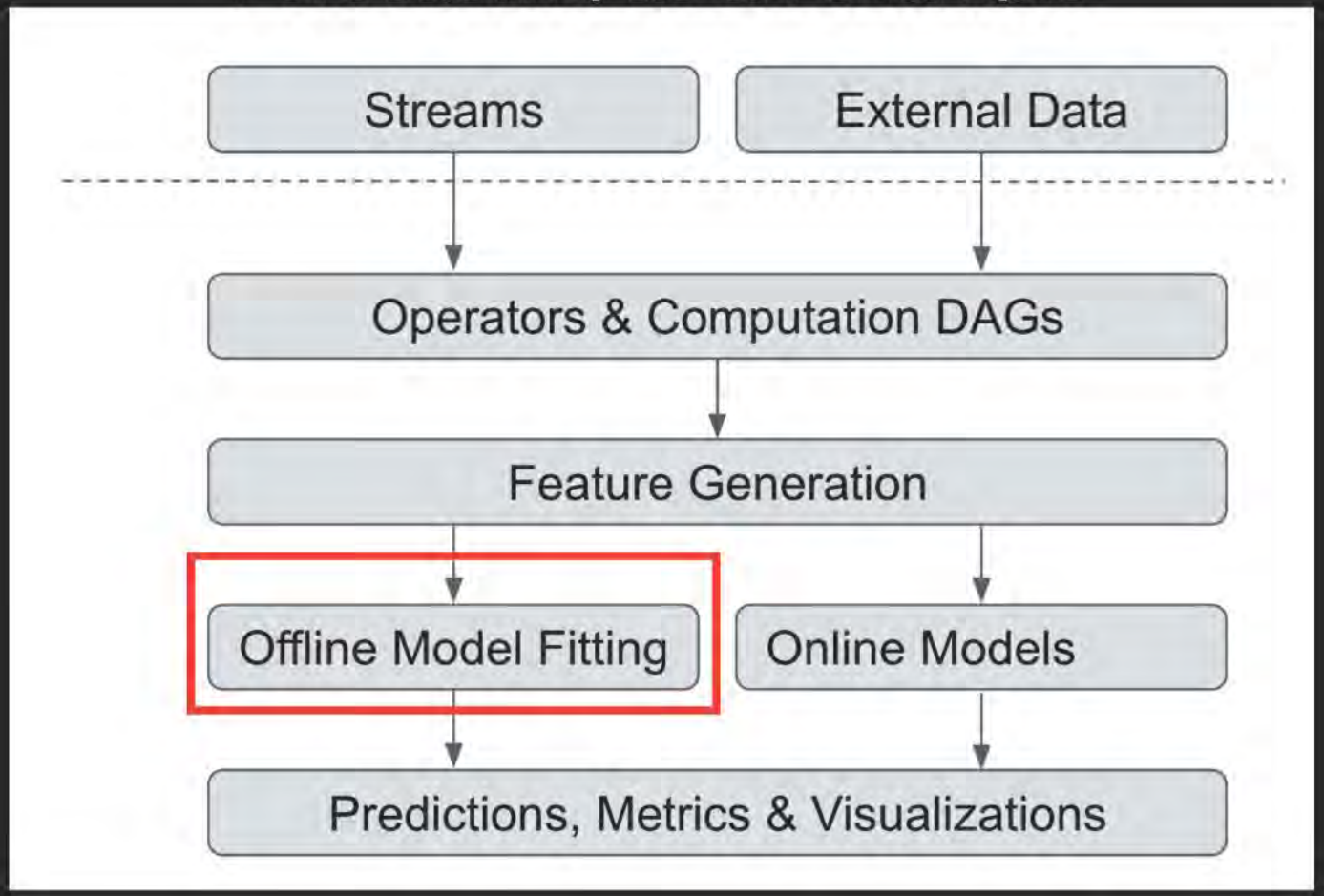
Forecasting as an example



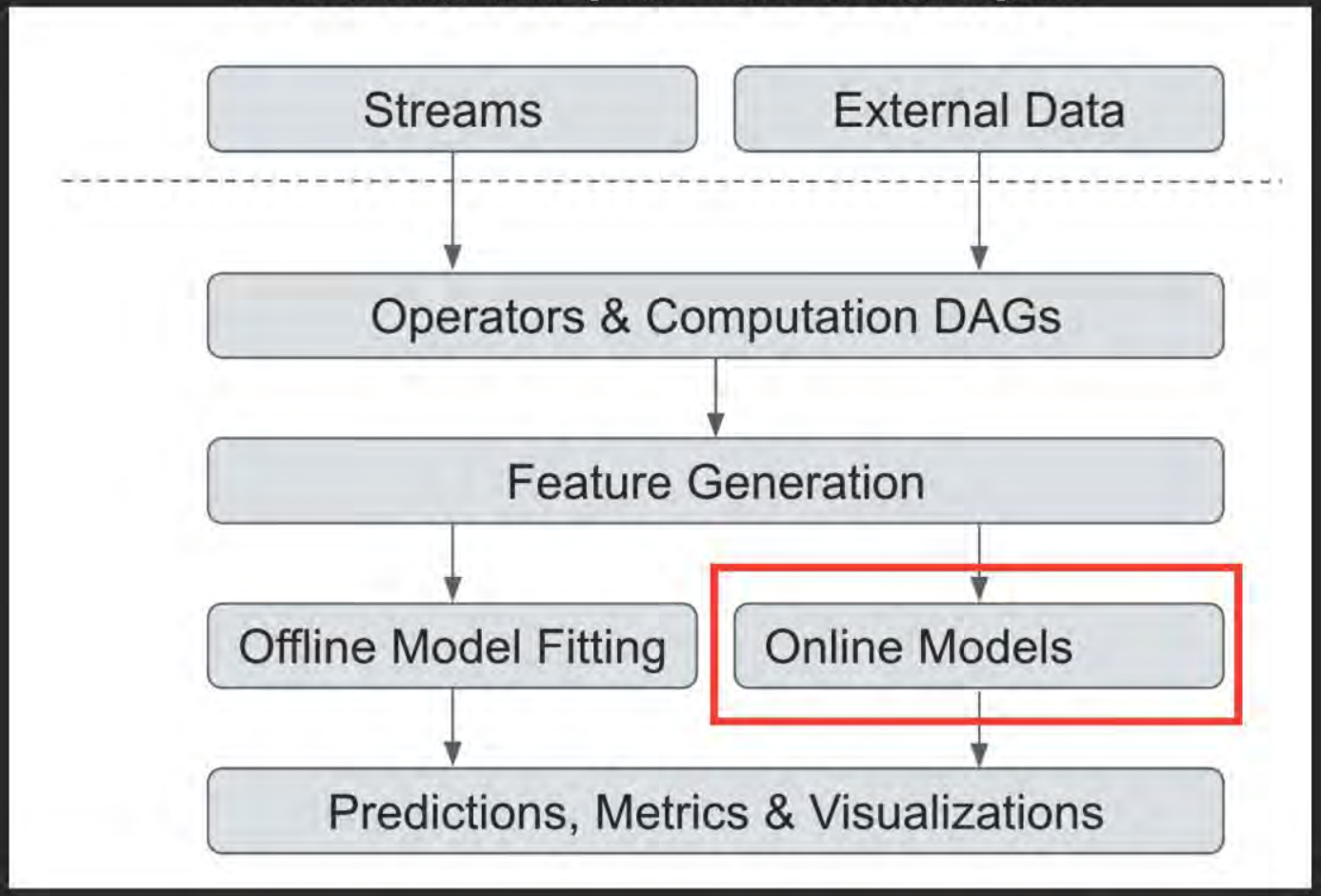
Forecasting as an example



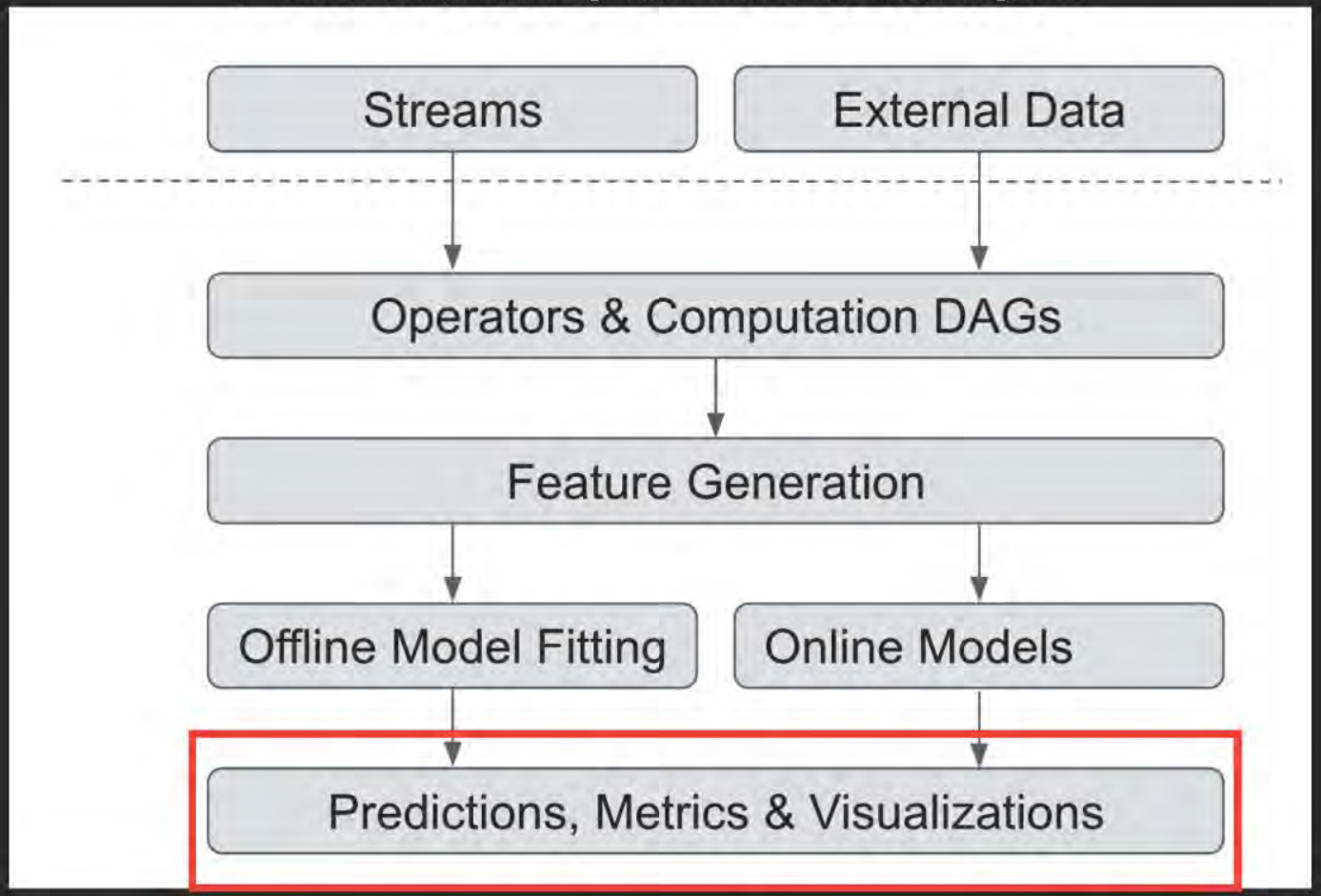
Forecasting as an example



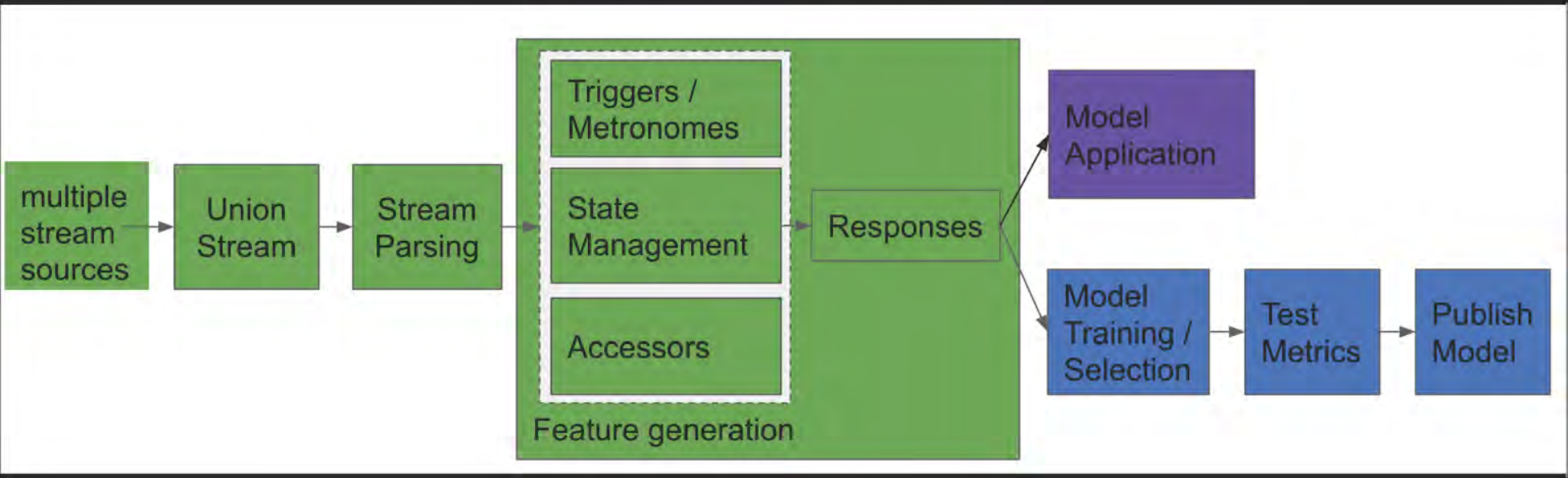
Forecasting as an example



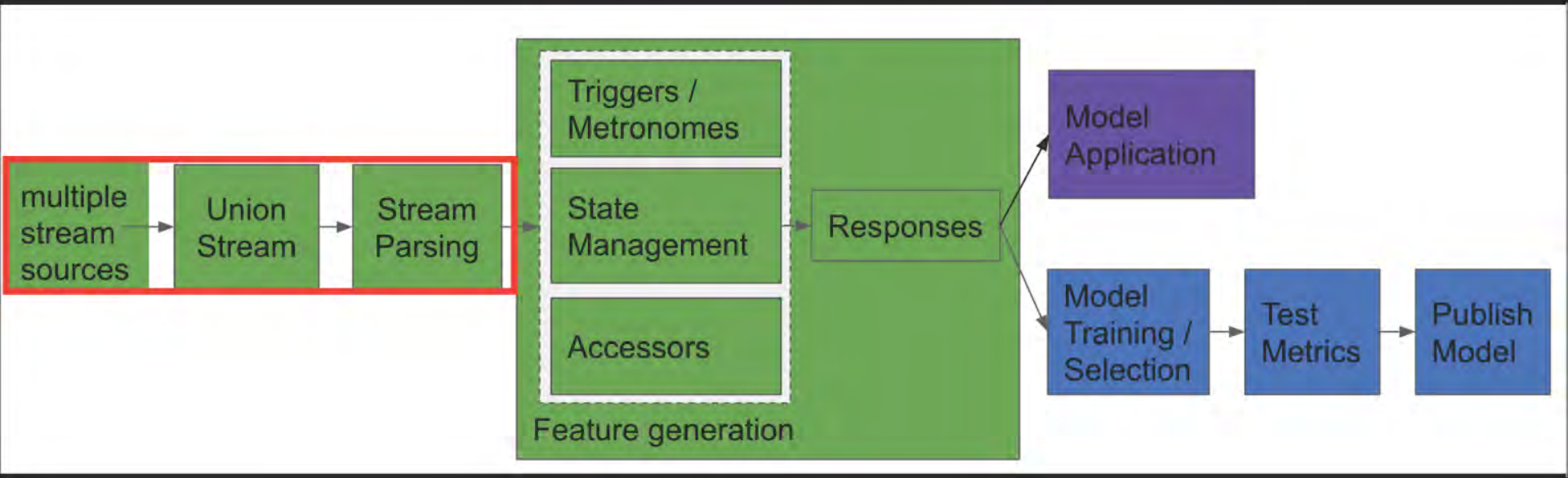
Forecasting as an example



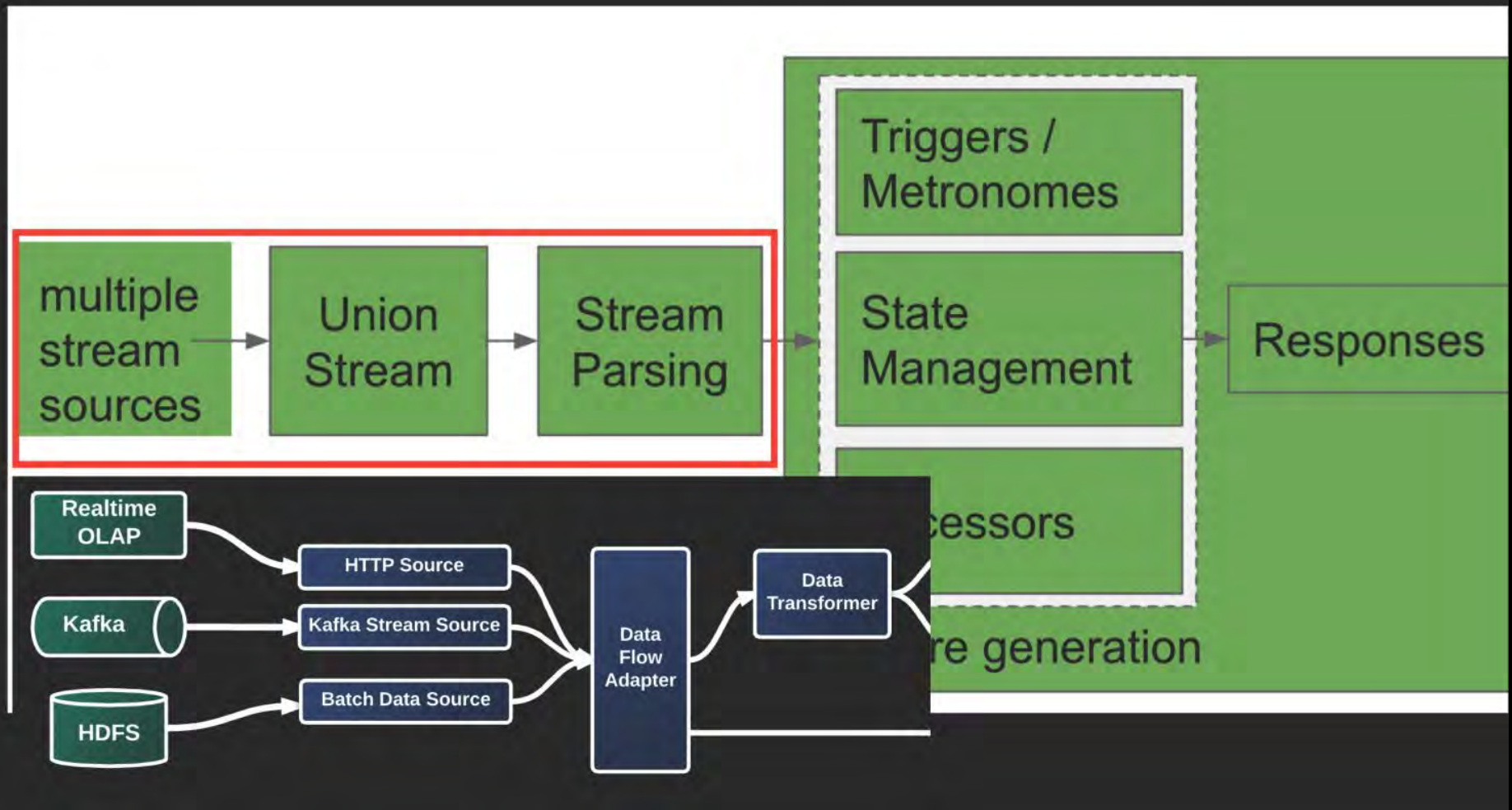
Forecasting as an example



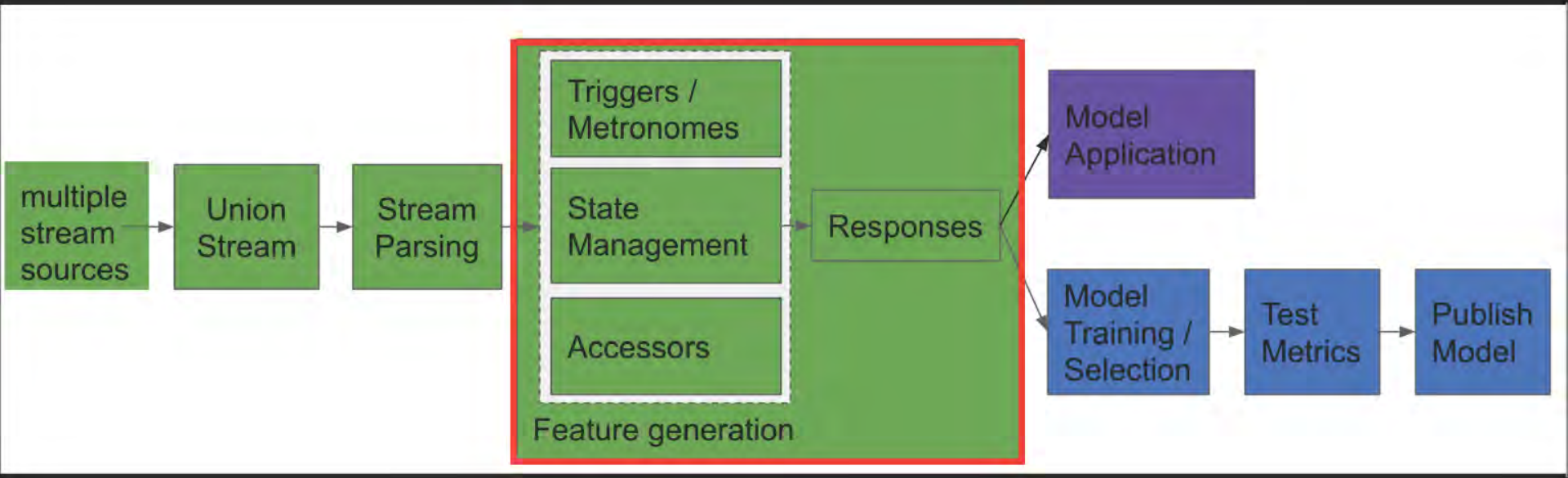
Forecasting as an example



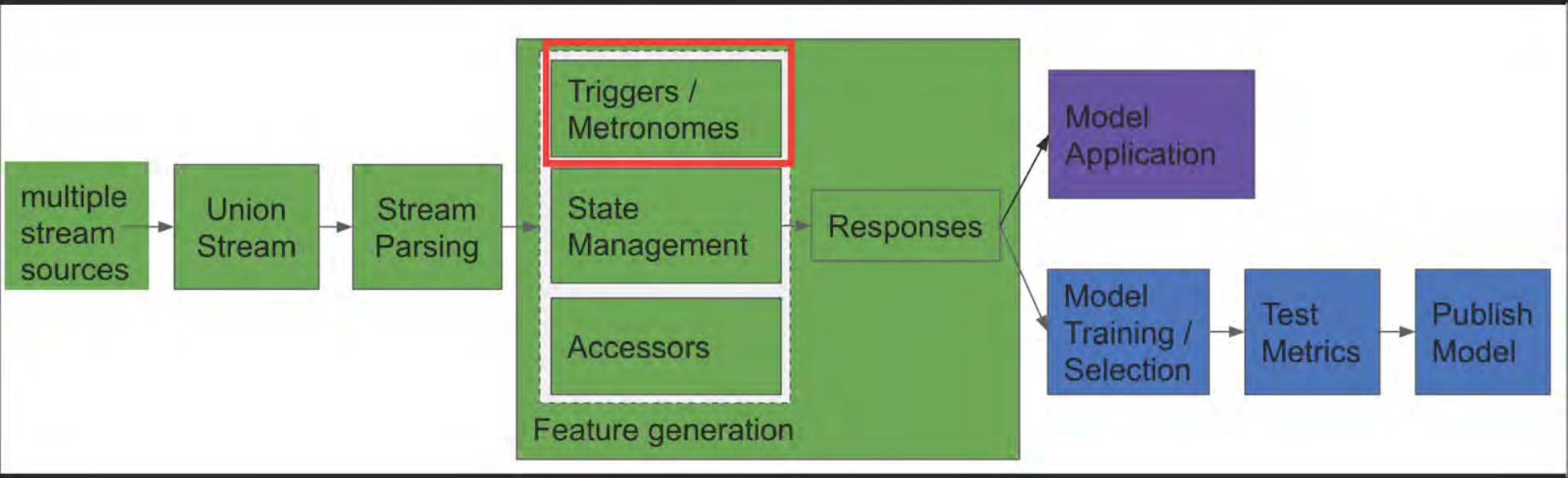
Forecasting as an example



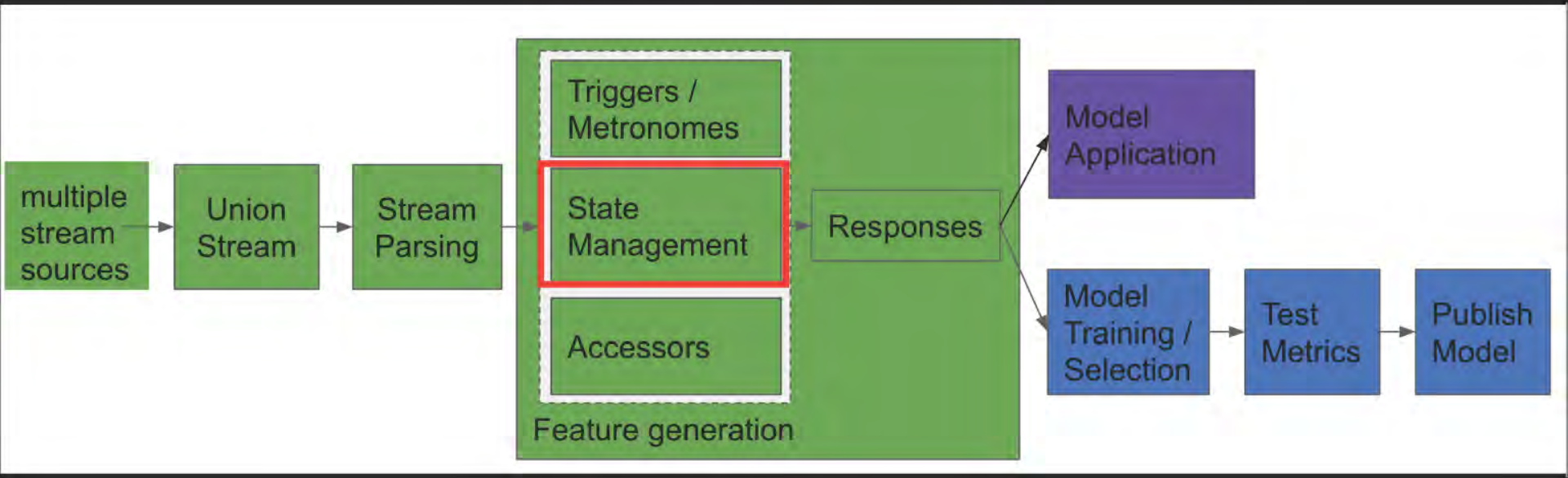
Forecasting as an example



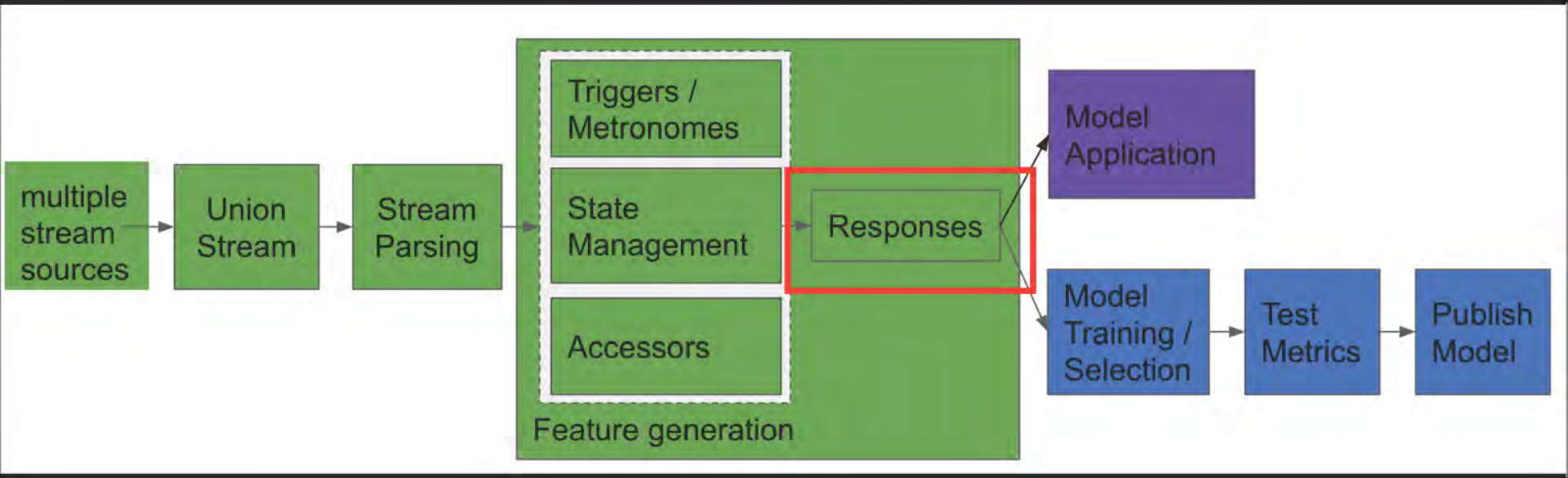
Forecasting as an example



Forecasting as an example



Forecasting as an example



Forecasting as an example