Real-time Processing of Noisy Data from Connected Vehicles

Robin * Slomkowski
Flink Forward 2018-09

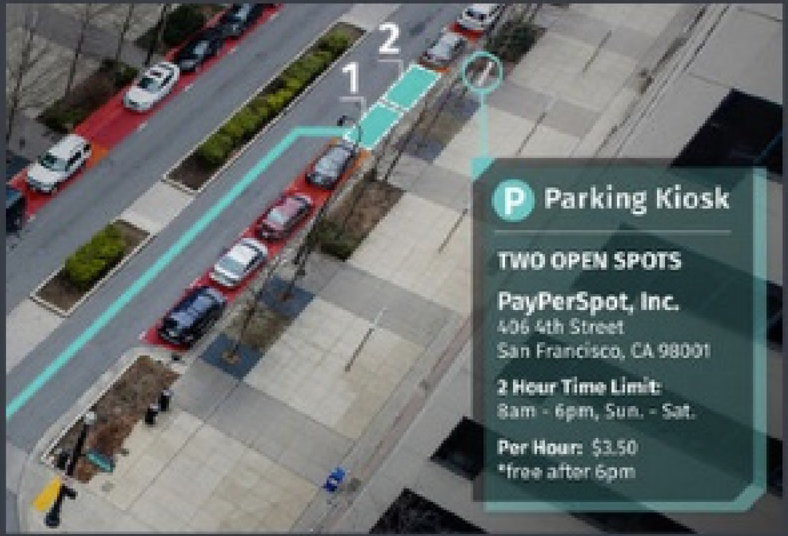
Making Safer Roads

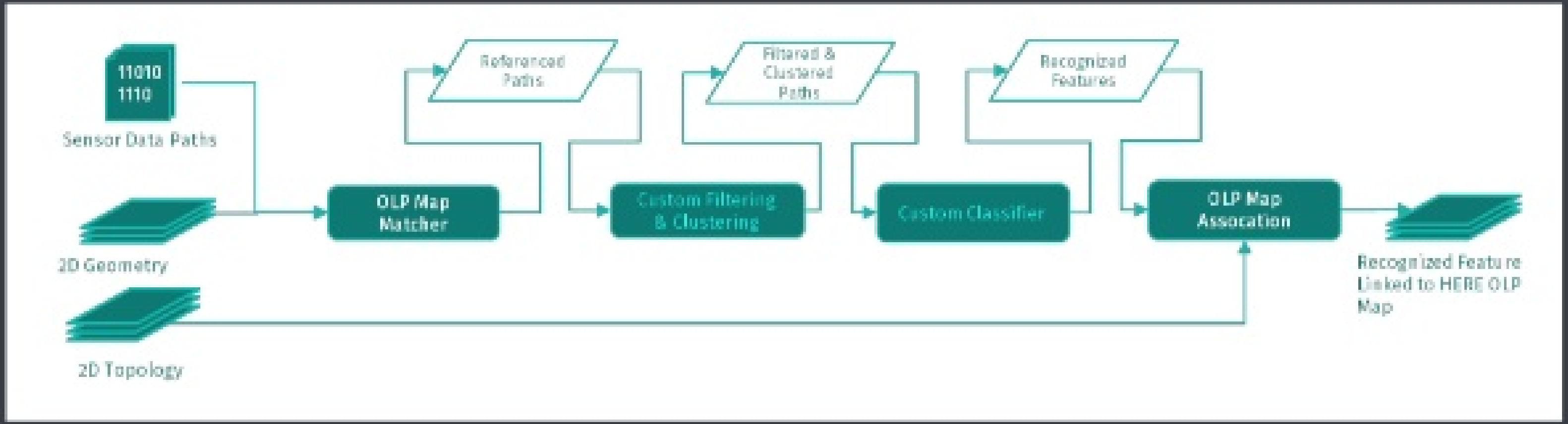
Slippery roads

Variable Signs

Accidents and other Events

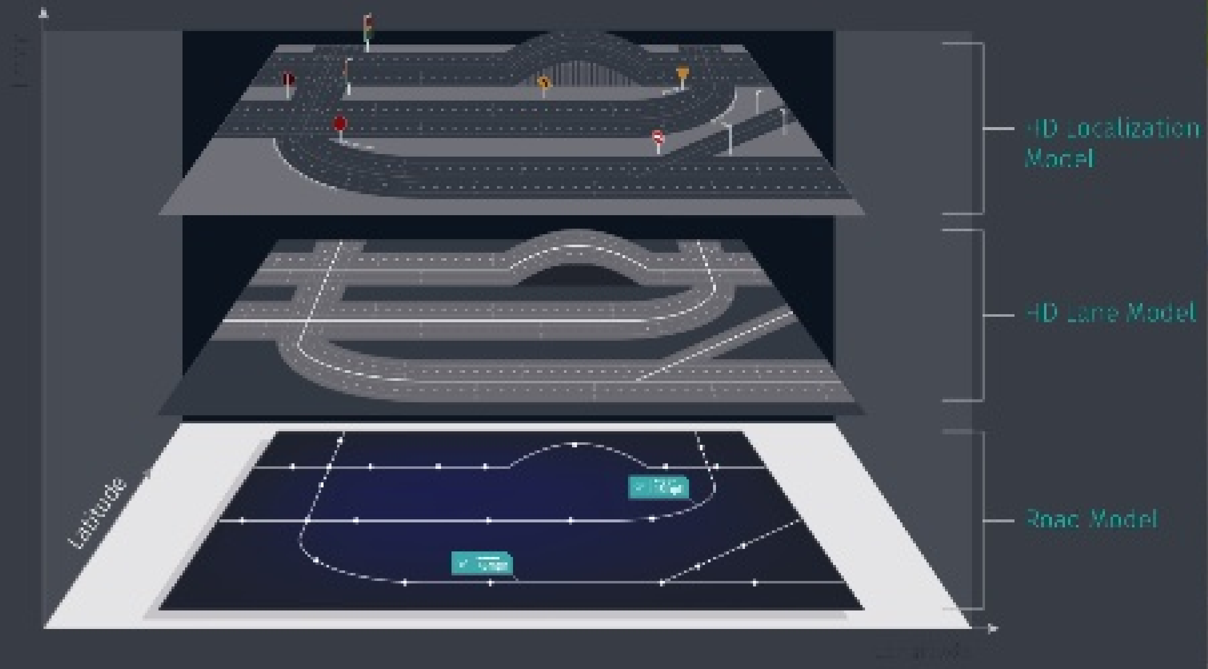
Live Parking Data

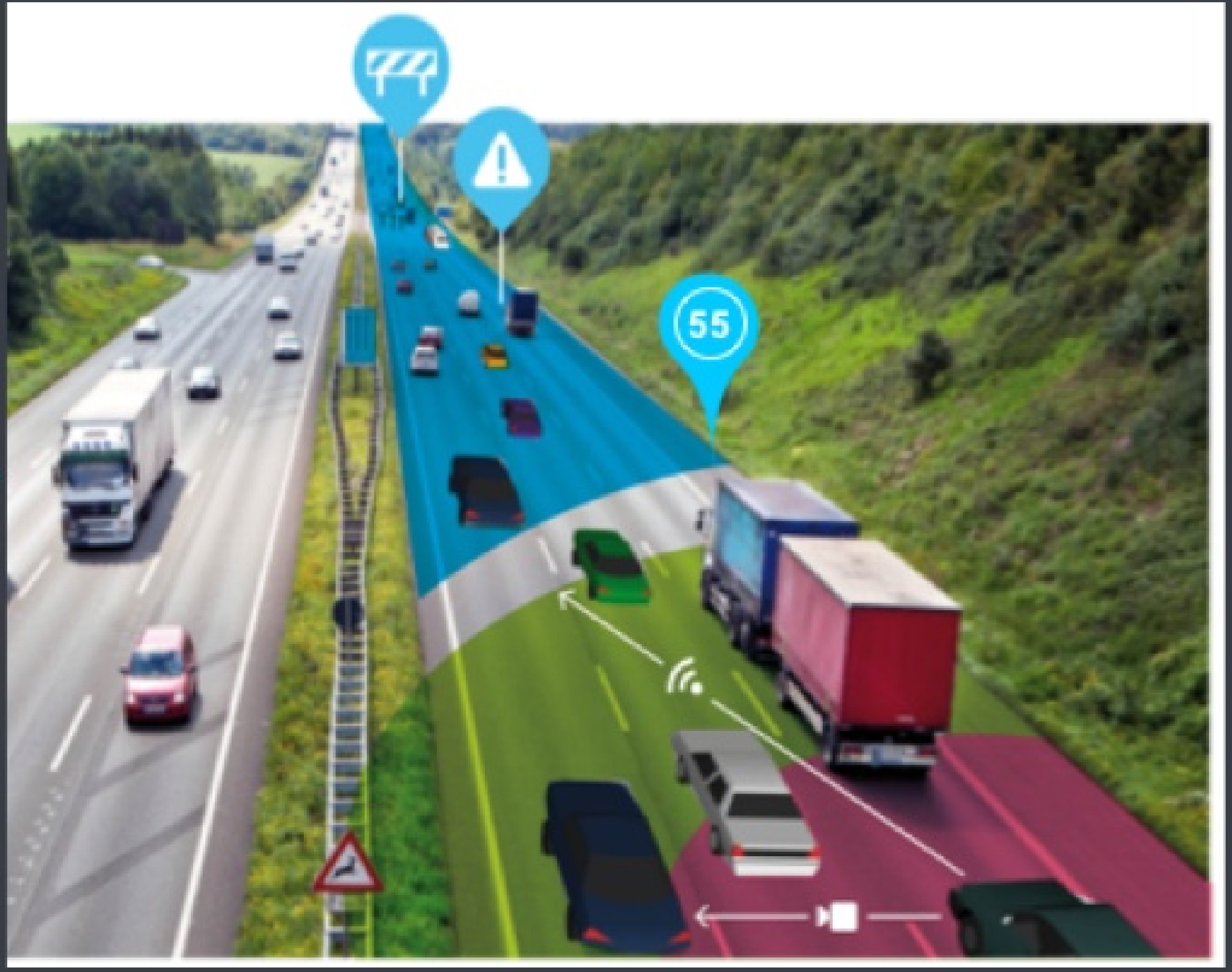




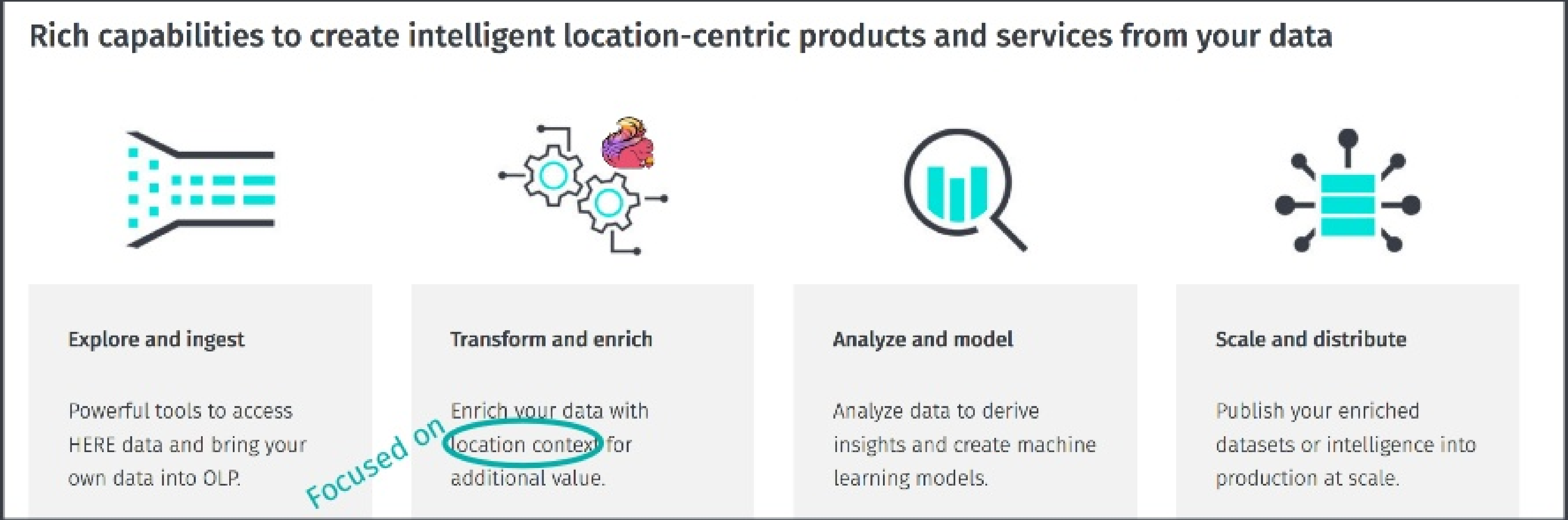
Building Living Maps

Annotating Sensors against an authoritative reference.





HERE Open Location Platform



Rich capabilities to create intelligent location-centric products and services from your data

Explore and ingest

Powerful tools to access HERE data and bring your own data into OLP.

Transform and enrich

Enrich your data with location context for additional value.

Focused on

Analyze and model

Analyze data to derive insights and create machine learning models.

Scale and distribute

Publish your enriched datasets or intelligence into production at scale.

http://openlocation.here.com

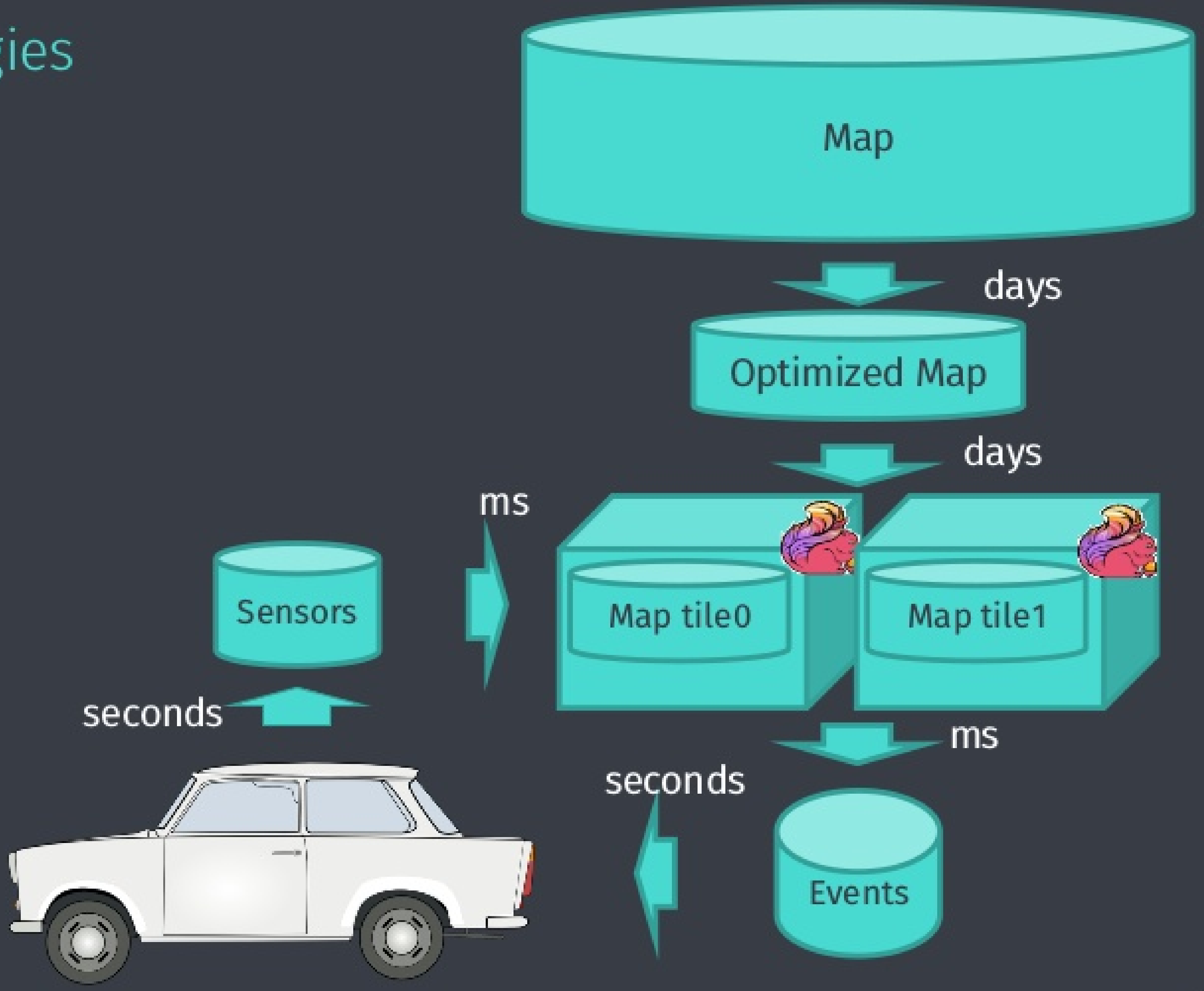
Stream Processing Strategies

Goals

- Lower Latency
- Less Engineering

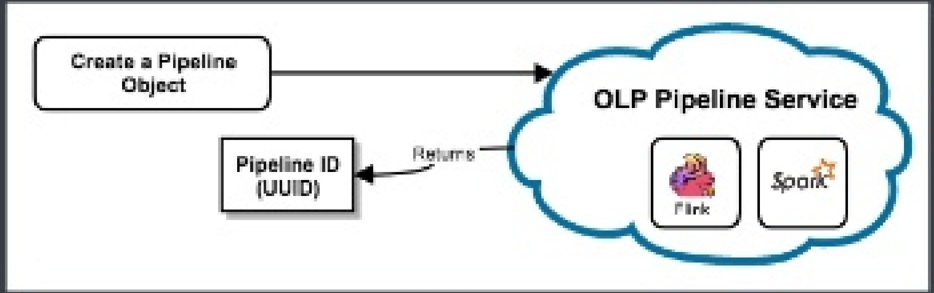
Techniques

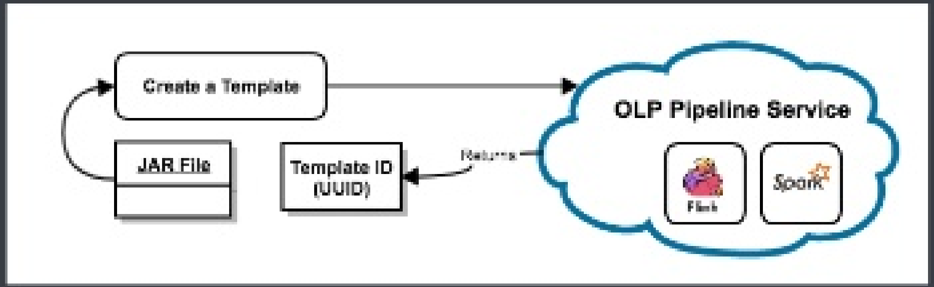
- Pre-Computing
- Deterministic Data Access
- Extensible Infrastructure Optimized Functions

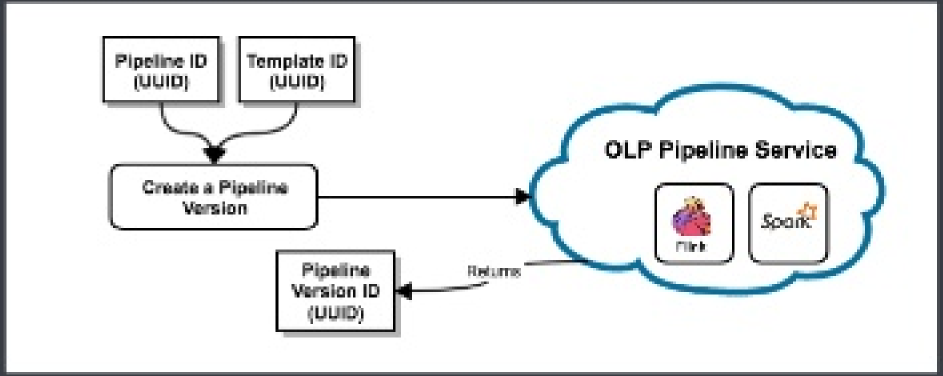


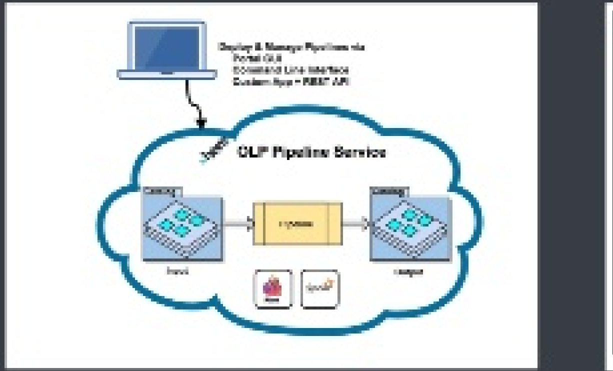
Managing Pipelines on the HERE’s Platform

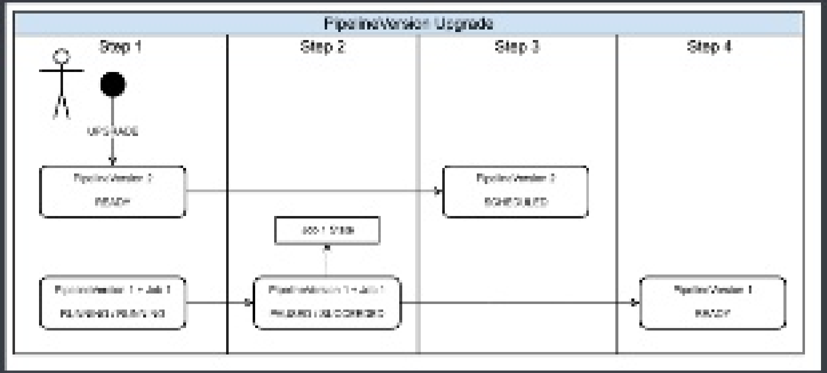
1. Prepare
   1. Create a Pipeline (Framework)
   2. Create a Pipeline Template (Jar)
   3. Create a Pipeline Version (Config)
2. Deploy
   1. Activate (Pipeline Version)
3. Run
   1. Cancel, Pause, Resume, Upgrade





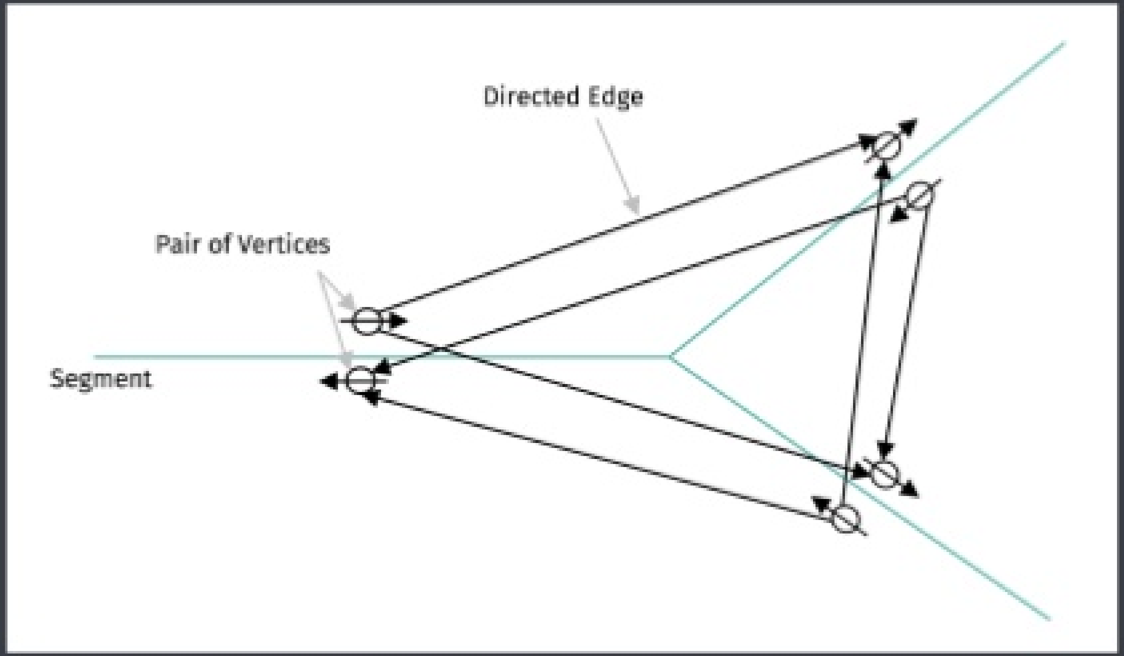


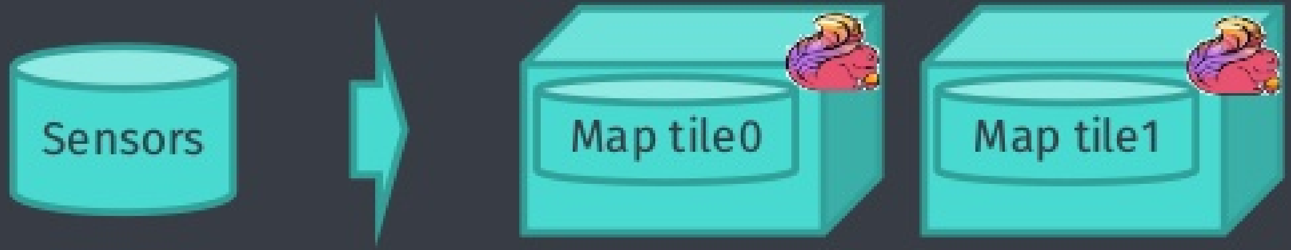


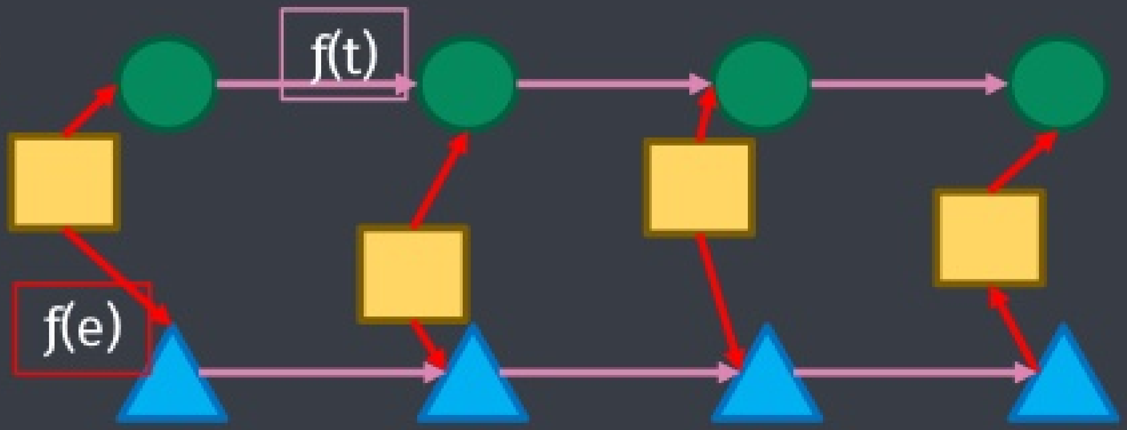


Implementation of Strategy

- Optimize Data for Use Case
  - Don’t need all data for sensor attribution
  - We know how the data will be used
- Key-By Map Tiling
  - Location Data does not arrive randomly, make caches work
- Extensible Map-Matching Implementation
  - We can use stateless evaluation to choose the right result







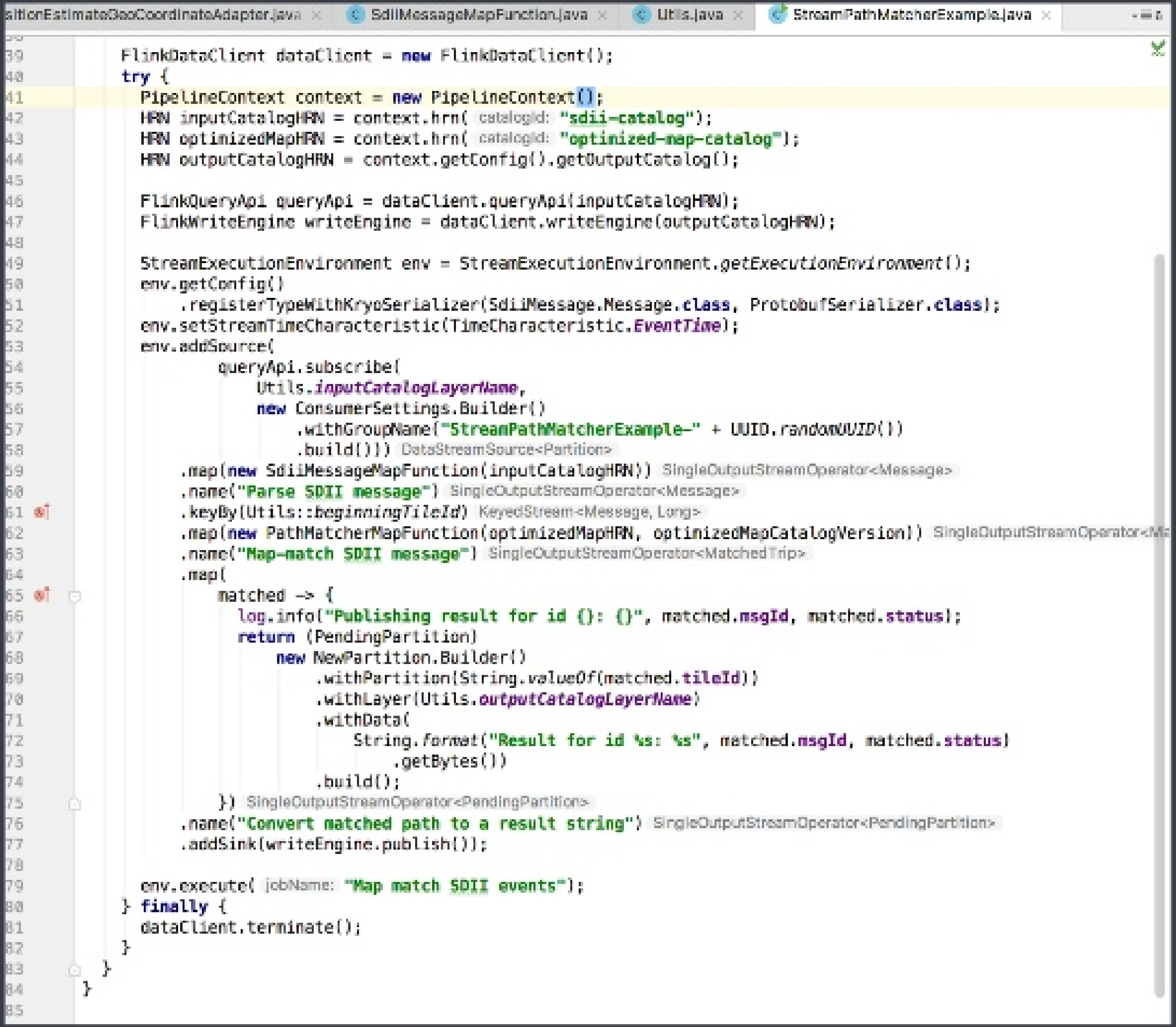
Map Matching Code

Data Client gets the Optimized Map

Data Client gets the Stream for Flink

Parse SDII proto

Default Path Map Matcher

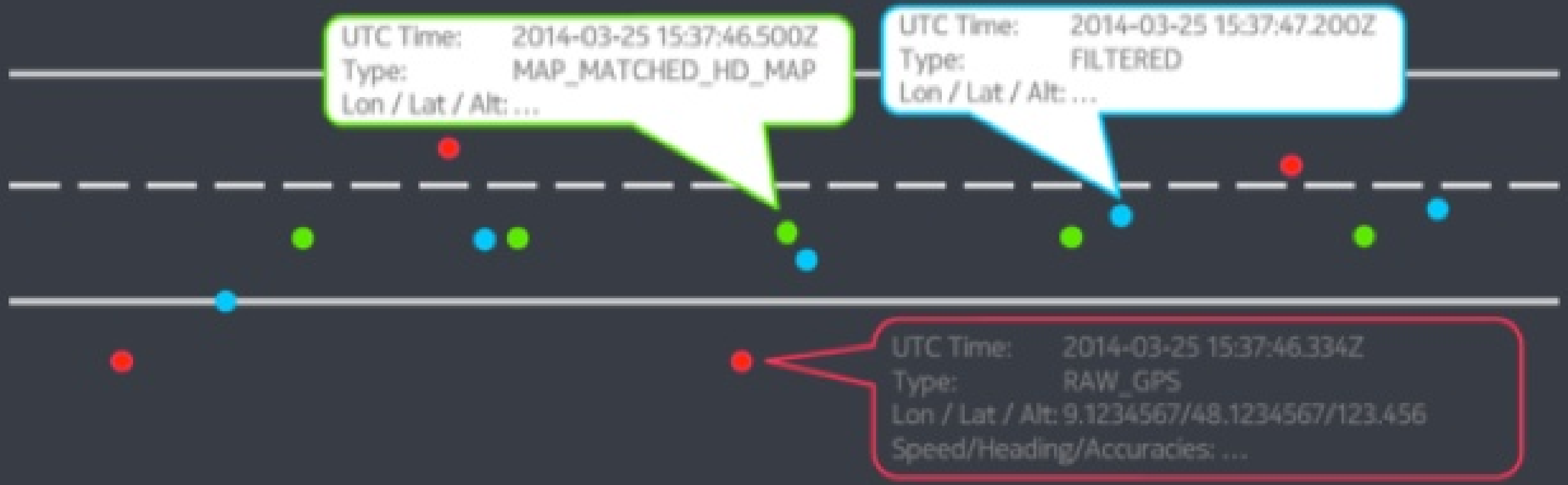

sitionEstimateGeoCoordinateAdapter.java
SdiiMessageMapFunction.java
Utils.java
StreamPathMatcherExample.java

```
    FlinkDataClient dataClient = new FlinkDataClient();
    try {
      PipelineContext context = new PipelineContext();
      HRN inputCatalogHRN = context.hrn( catalogId: "sdii-catalog");
      HRN optimizedMapHRN = context.hrn( catalogId: "optimized-map-catalog");
      HRN outputCatalogHRN = context.getConfig().getOutputCatalog();

      FlinkQueryApi queryApi = dataClient.queryApi(inputCatalogHRN);
      FlinkWriteEngine writeEngine = dataClient.writeEngine(outputCatalogHRN);

      StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
      env.getConfig()
          .registerTypeWithKryoSerializer(SdiiMessage.Message.class, ProtobufSerializer.class);
      env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);
      env.addSource(
              queryApi.subscribe(
                  Utils.inputCatalogLayerName,
                  new ConsumerSettings.Builder()
                      .withGroupName("StreamPathMatcherExample-" + UUID.randomUUID())
                      .build())) DataStreamSource<Partition>
          .map(new SdiiMessageMapFunction(inputCatalogHRN)) SingleOutputStreamOperator<Message>
          .name("Parse SDII message") SingleOutputStreamOperator<Message>
          .keyBy(Utils::beginningTileId) KeyedStream<Message, Long>
          .map(new PathMatcherMapFunction(optimizedMapHRN, optimizedMapCatalogVersion)) SingleOutputStreamOperator<Ma
          .name("Map-match SDII message") SingleOutputStreamOperator<MatchedTrip>
          .map(
              matched -> {
                log.info("Publishing result for id {}: {}", matched.msgId, matched.status);
                return (PendingPartition)
                    new NewPartition.Builder()
                        .withPartition(String.valueOf(matched.tileId))
                        .withLayer(Utils.outputCatalogLayerName)
                        .withData(
                            String.format("Result for id %s: %s", matched.msgId, matched.status)
                                .getBytes())
                        .build();
              }) SingleOutputStreamOperator<PendingPartition>
          .name("Convert matched path to a result string") SingleOutputStreamOperator<PendingPartition>
          .addSink(writeEngine.publish());

      env.execute( jobName: "Map match SDII events");
    } finally {
      dataClient.terminate();
    }
  }
}
```

What Kind of Data is SDII?

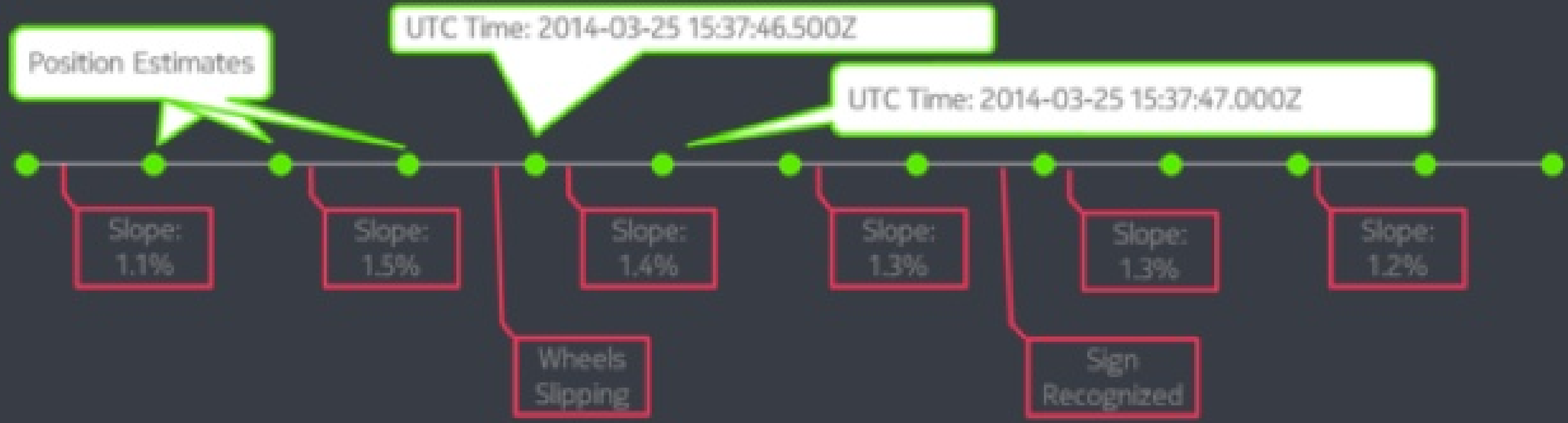
Series of Timestamped KV Pairs

- A series of GPS values with time allows identifying a point better
- A collection of other sensors, slope, wheels, signs, etc… with time


UTC Time: 2014-03-25 15:37:46.500Z
Type: MAP_MATCHED_HD_MAP
Lon / Lat / Alt: …
UTC Time: 2014-03-25 15:37:47.200Z
Type: FILTERED
Lon / Lat / Alt: …
UTC Time: 2014-03-25 15:37:46.334Z
Type: RAW_GPS
Lon / Lat / Alt: 9.1234567/48.1234567/123.456
Speed/Heading/Accuracies: …

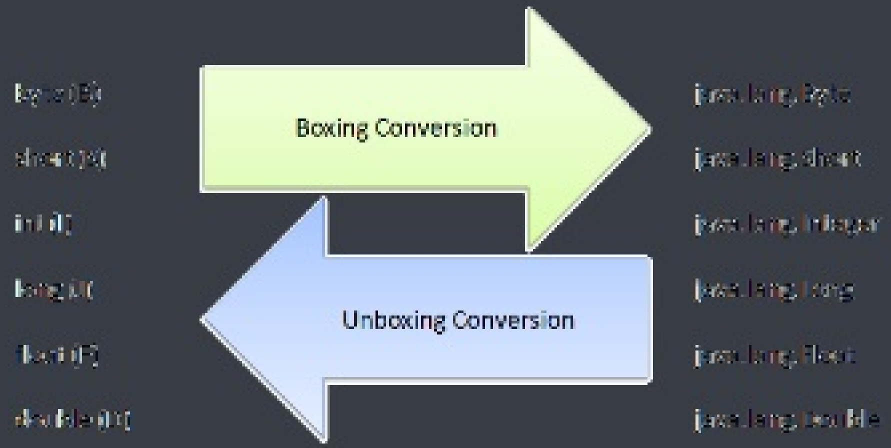
GPS Point to Sensor Estimation is the User's Problem

- DBSCAN clustering is supported by HERE, but custom logic may be added.

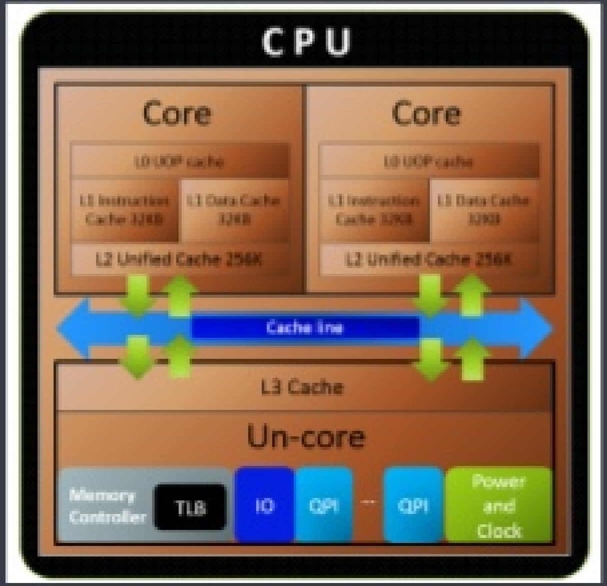


Optimized Map

Java Integer Arrays to avoid scala boxing/unboxing



Add attributed Index for constant time access to in memory data structures



Why so much effort?

- These data must be scanned thousands of times to build emission and transition probabilities.
- Series of points are near each other so cache hits work well
- We want stream processing light as possible, do this work 1x
- Multiple Orders of magnitude faster than the naïve approach

What is Map Tiling?

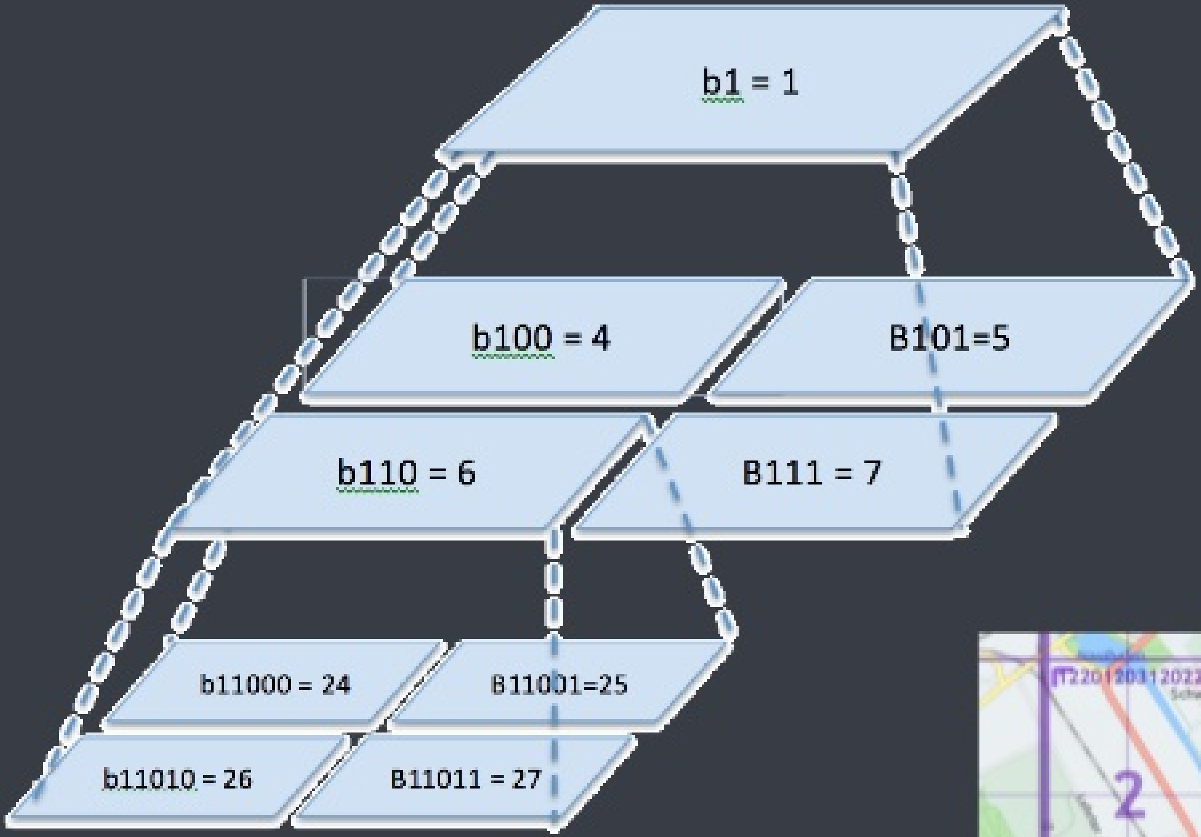
HERE Tile

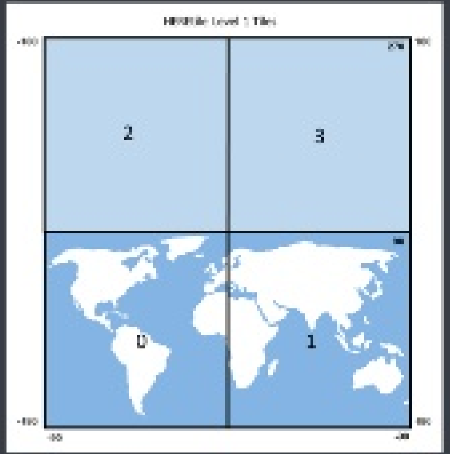
Strategies for categorizing data

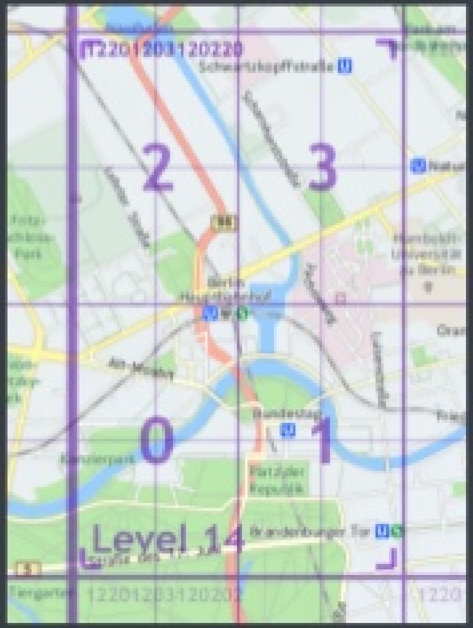
- Partitioning at arbitrary depth

Why Key-By makes sense

- People do not jump randomly between streets or around the world, so you tend to get multiple results near by each other.
- Enhances cache hits immensely
- Yes we talk about origin so given envelope may cross tiles, but the caching works better this way anyway.

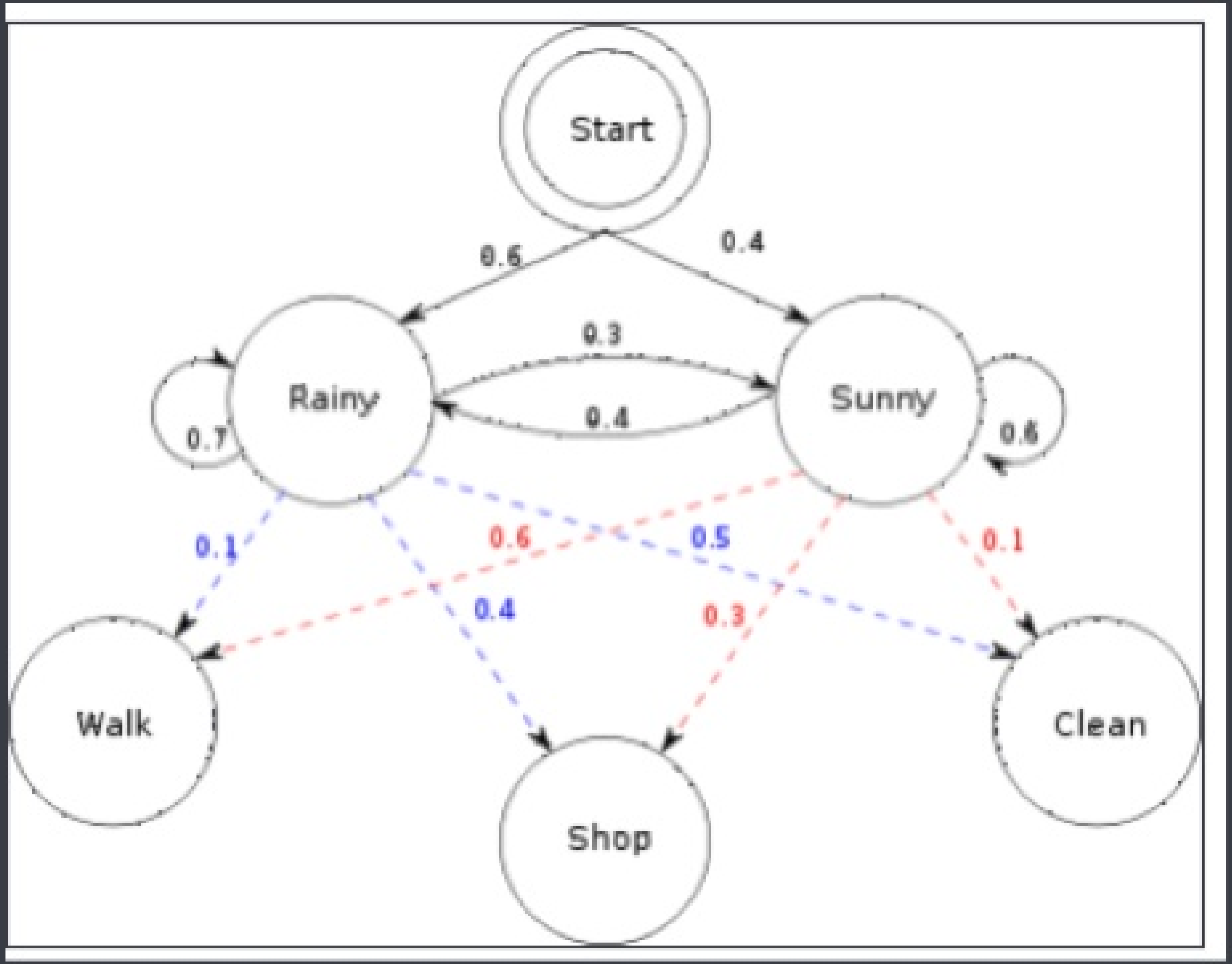






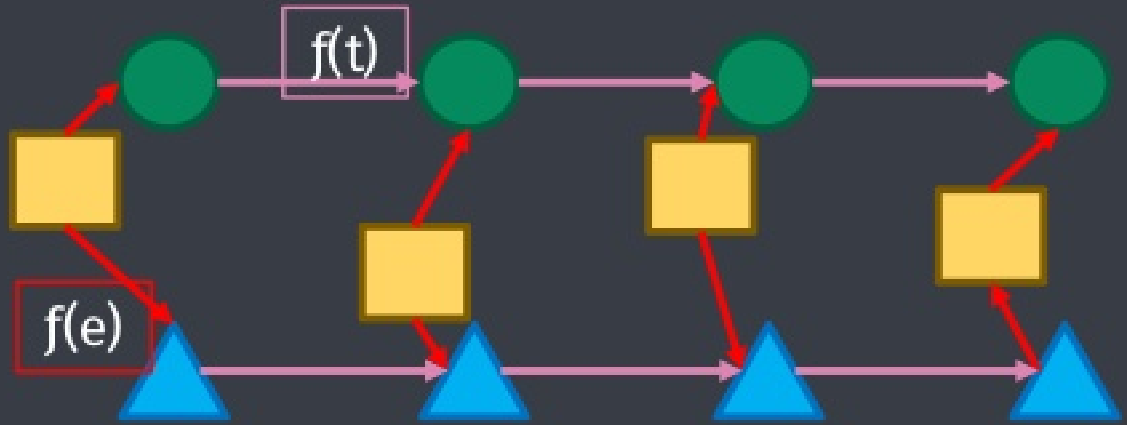
Map Matching: Maintaining Algorithms

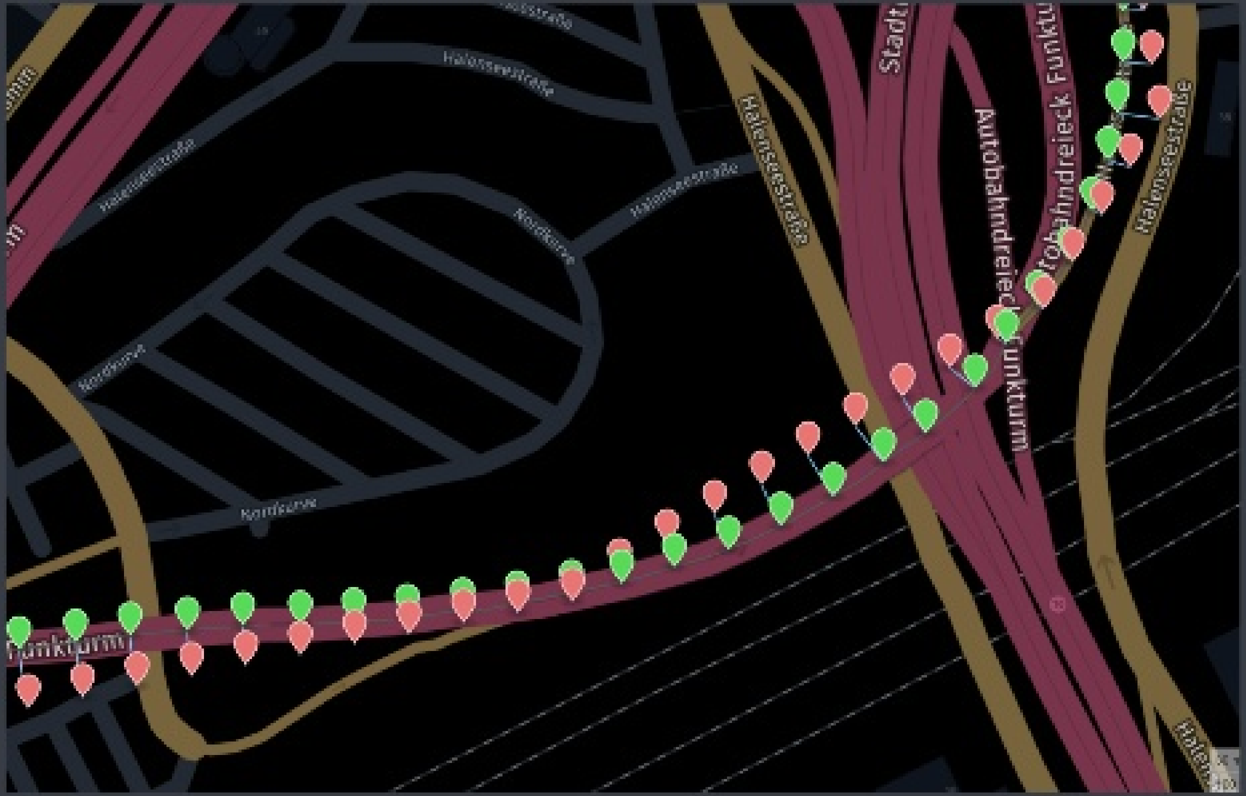
- Hidden Markov Model
  - Yes this is the best practice solution.
  - Optimizing the data needed to make the decisions is where you get the performance from
- Data Model Agnostic
  - Can be applied to any Vector Data
- Customizable
  - Emission Probability
  - Transition Probability



Map Matching: Paths and Points

- Why are points together?
- Optimizing matching based on logical paths?
- What about anomalies?
- What about emission and transition logic?



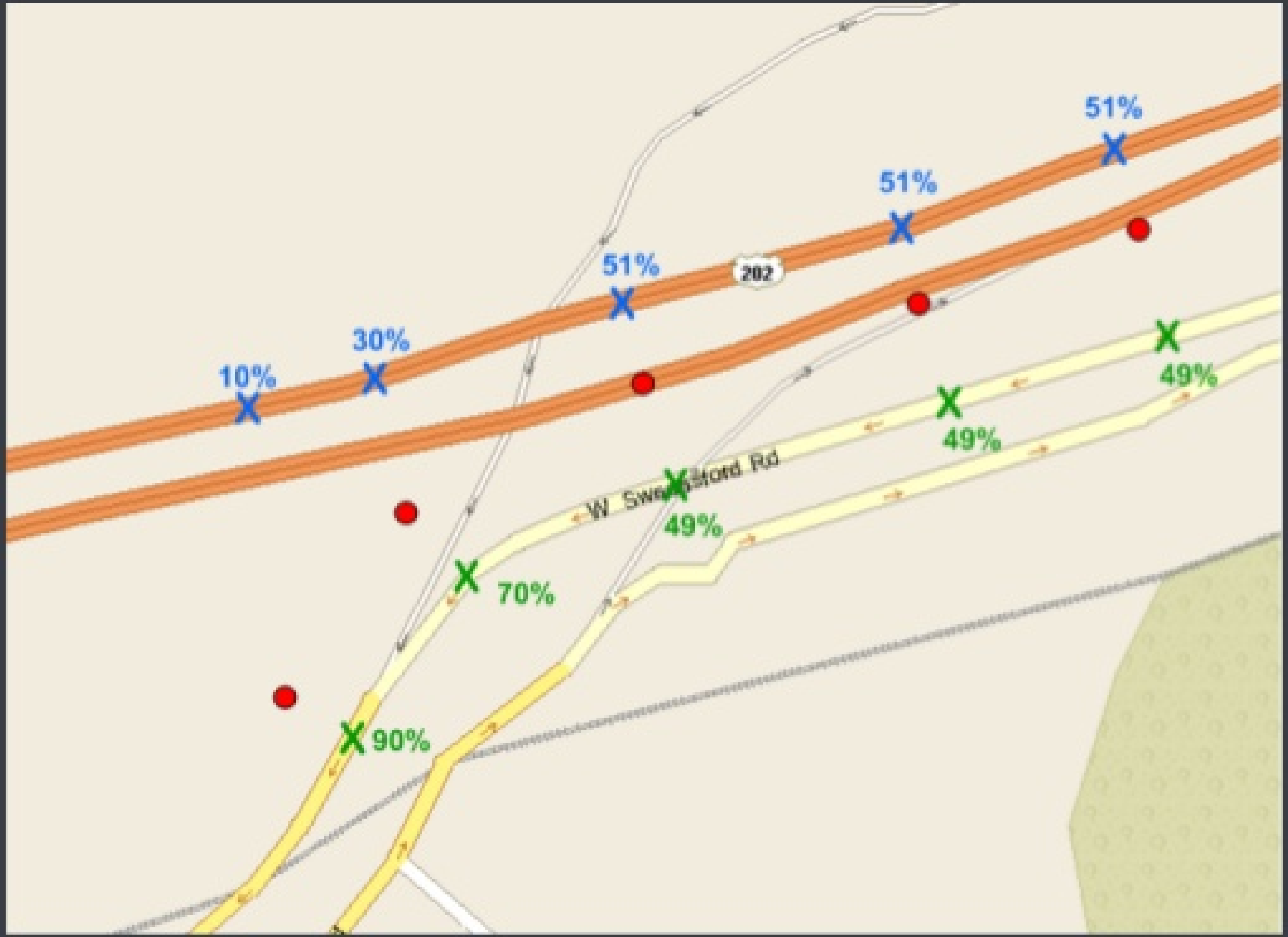


Map Matching: Emission Probability

- Points don't line up
- Shortest path doesn't match vector
- One algorithm to select the emission
- A second to select the route, to define the point.

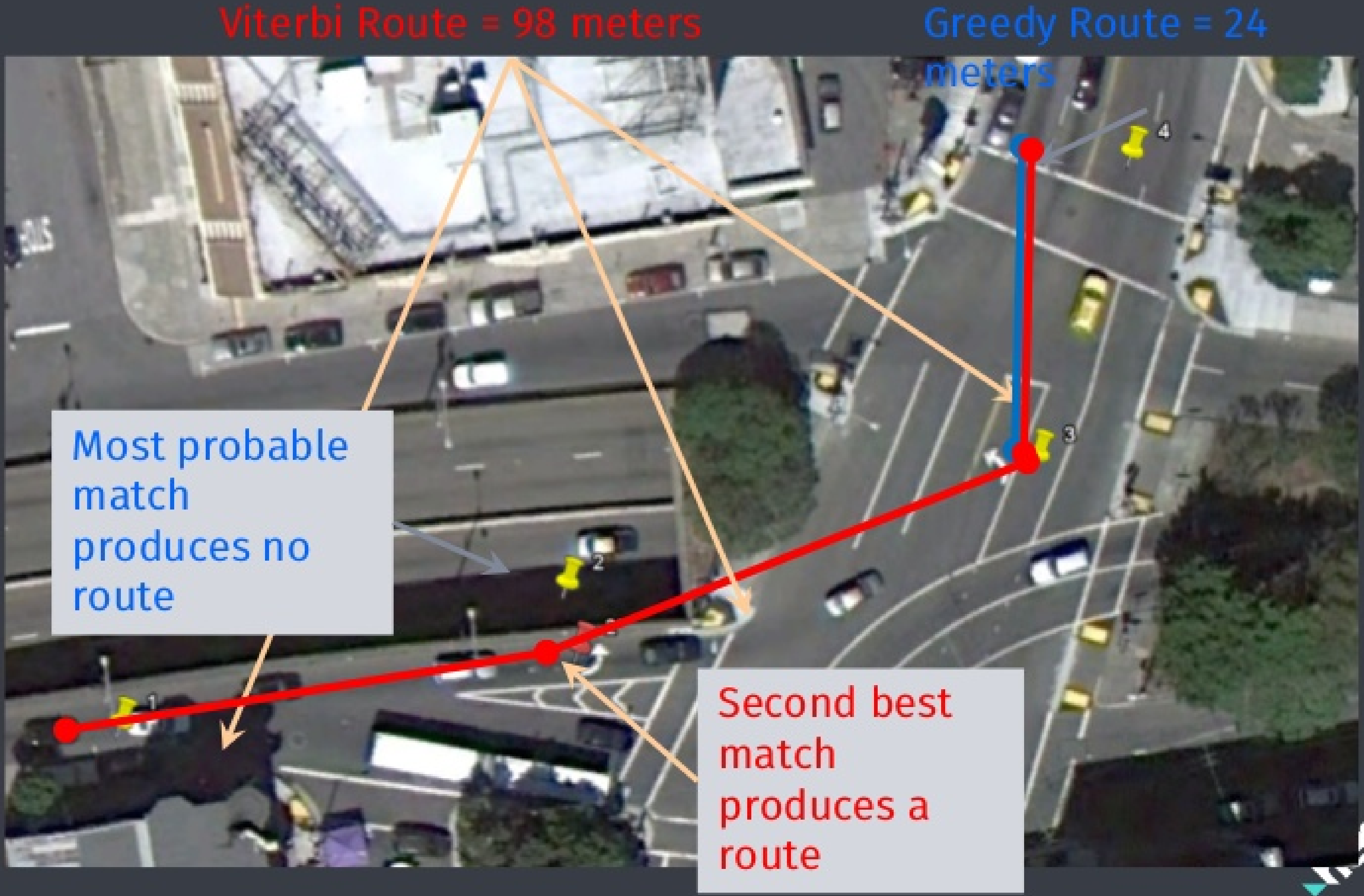
Route 202
0.51 * 0.51 * 0.51 * 0.3 * 0.1 => **0.40 %**

West Swedesford Rd.
0.49 * 0.49 * 0.49 * 0.7 *0.9 => **7.40 %**



Map Matching: Transition Probability

- Viterbi as opposed to Greedy Match
- No Right Answer Every Time Bring your own algorithm



Viterbi Route = 98 meters

Greedy Route = 24 meters

Most probable match produces no route

Second best match produces a route

Thank you

http://openlocation.here.com

Contact