MobilityDuck: Mobility Data Management with DuckDB

Nhu Ngoc Hoang, Ngoc Hoa Pham,
Viet Phuong Hoang, and Esteban Zimányi

BMDA Big Mobility Data Analytics
ULB Université Libre de Bruxelles

Tampere, March 24, 2026

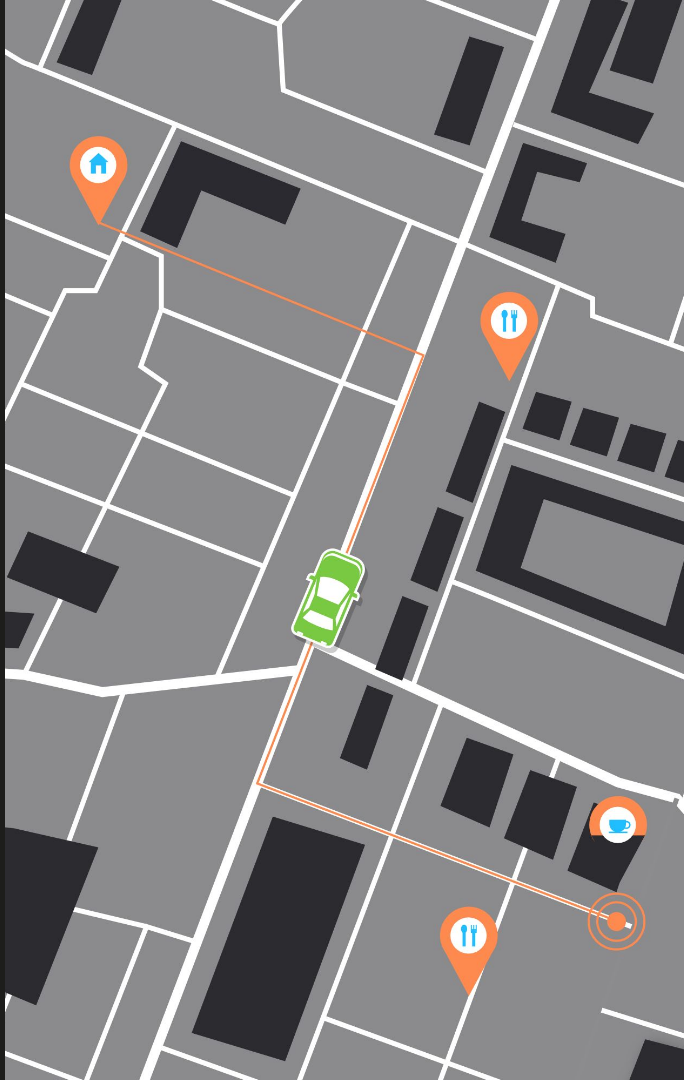
MEOS

MEOS (Mobility Engine, Open Source) is a C library and its associated API for manipulating temporal and spatiotemporal data

MEOS provides temporal types representing the evolution on time of values of some base type

Core component of **MobilityDB**, a trajectory data management & analysis platform built on top of PostgreSQL and PostGIS

Position	Speed	Speed > 50?	Gear	Road type
↓	↓	↓	↓	↓
point	float	bool	int	text
↓	↓	↓	↓	↓
tgeompoint	tfloat	tbool	tint	ttext



Mobility Analytics Trade-Offs

PostgreSQL + MobilityDB

Strengths: expressive moving object database, mature MEOS backbone, full indexing support

Weaknesses: server process, high setup overhead, lack of embeddability

DuckDB

Strengths: in-memory, embeddable, optimized for OLAP

Weaknesses: lack of native support for spatiotemporal* data types and continuous trajectory operators

Goal: create a tool with the expressiveness of a MOD & the speed of an OLAP engine

* Non-temporal geospatial data processing is supported by Spatial extension
https://duckdb.org/docs/stable/core_extensions/spatial/overview

Introducing MobilityDuck

The first (to our knowledge) DuckDB extension that enables spatiotemporal analytics leveraging mature functionality of MEOS

Lightweight integration

Native DuckDB extension, preserves embedded deployment model

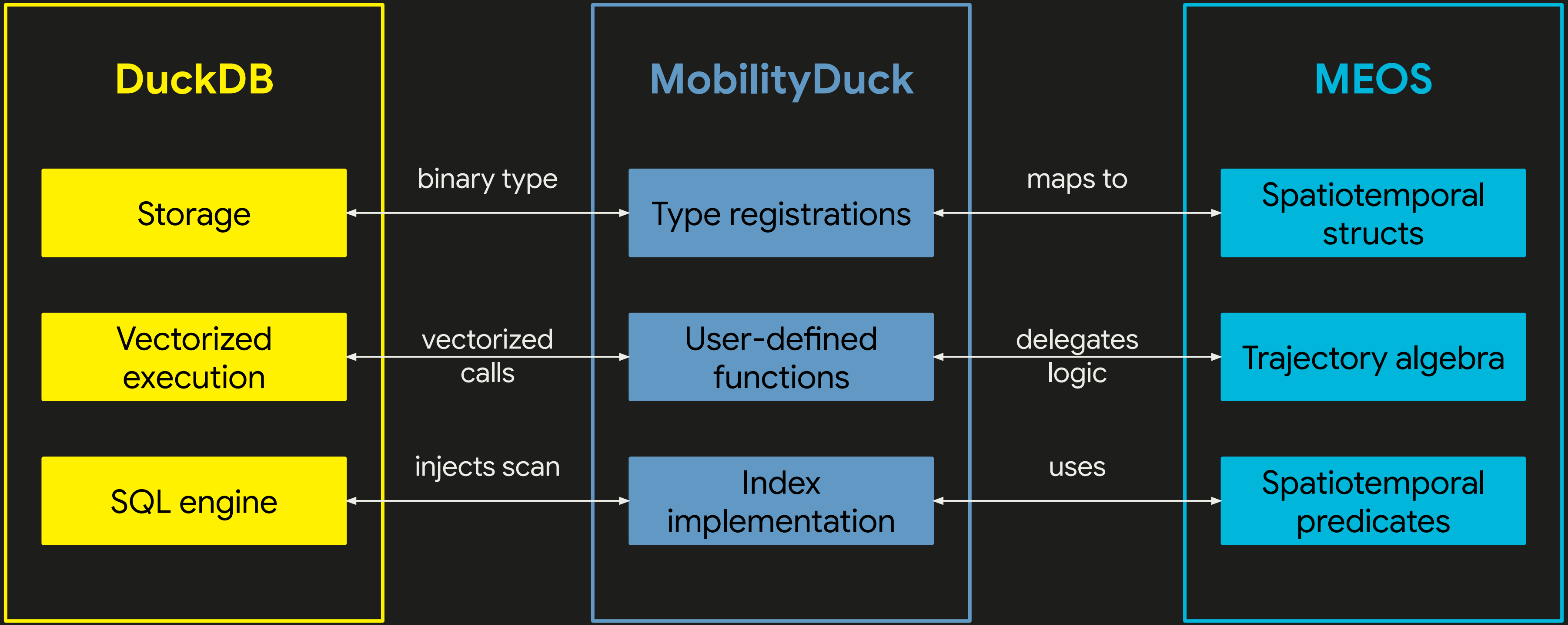
Algebra reuse

Wraps MEOS natively rather than reimplementing logic, ensuring semantic consistency

DuckDB compatibility

Exposes types and functions as user-defined compatible for batch processing

3-layer Architecture Design



Types, Functions, and Operators

Logical types

MEOS types are registered using native type BLOB (Binary Large OBject), providing first-class support for spatiotemporal data types

```
CREATE TABLE Trips(
    trajectory tgeompoint,
    speed tfloat,
    speed_over_50 tbool,
    gear tint,
    road_type ttext
)
```

Functional interface

- Cast functions: convert between logical types
- Scalar functions: mapping MEOS-defined operations to vectorized interface
- Operators: exposing spatiotemporal predicates to SQL parser

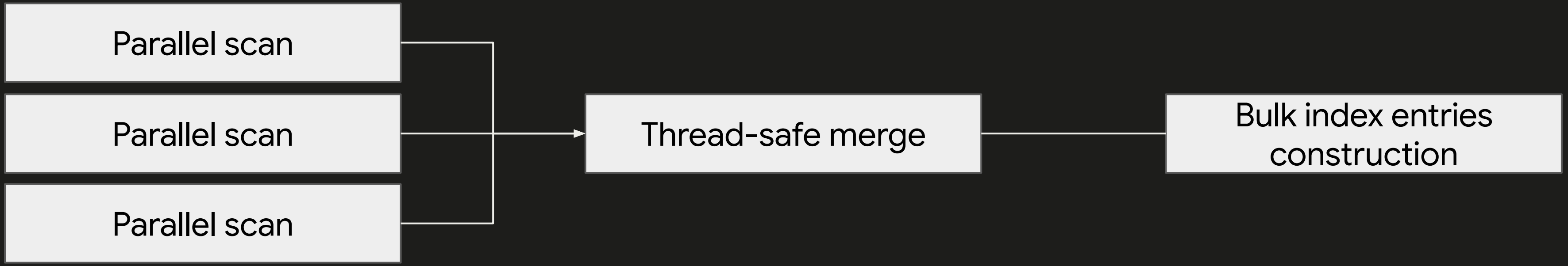
```
SELECT '100@2026-01-15+02'::tint;
-- 100@2026-01-14 22:00:00+00

SELECT duration('{1@2026-01-01 8:00:00, 1@2026-01-03
14:00:00}'::TINT, true);
-- 2 days 06:00:00

SELECT tgeompoint '{[Point(1 1)@2026-01-01, Point(2
2)@2026-01-02]}' && stbox 'STBOX XT(((1.0,2.0),
(10.0,20.0)),[2026-01-01,2026-01-02])';
-- true
```

R-tree Indexing Support

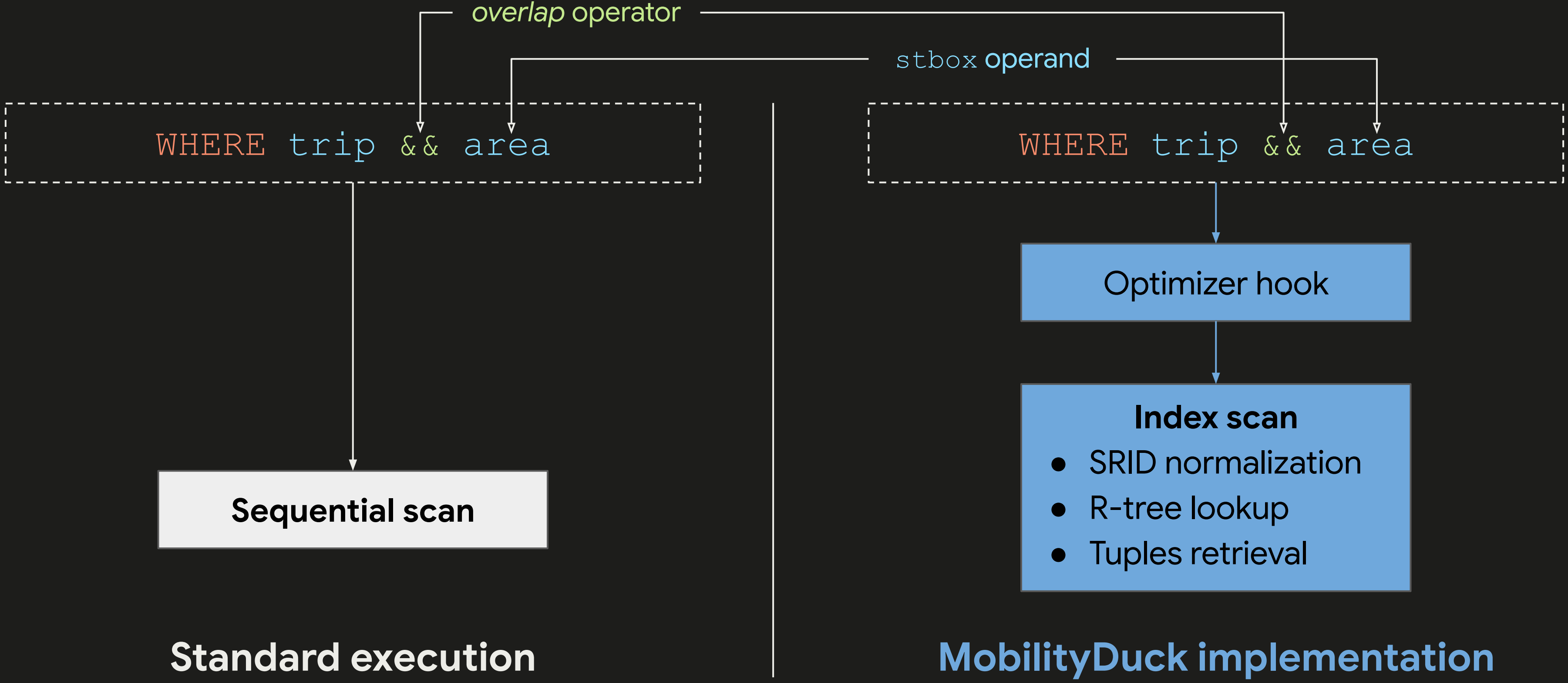
Currently support R-tree index over spatiotemporal bounding boxes (`stbox`), organizing data using topological containment relations

Strategy 1: Incremental construction (index-first)

Strategy 2: Bulk construction (data-first)



Query Optimization and Index Scan Injection

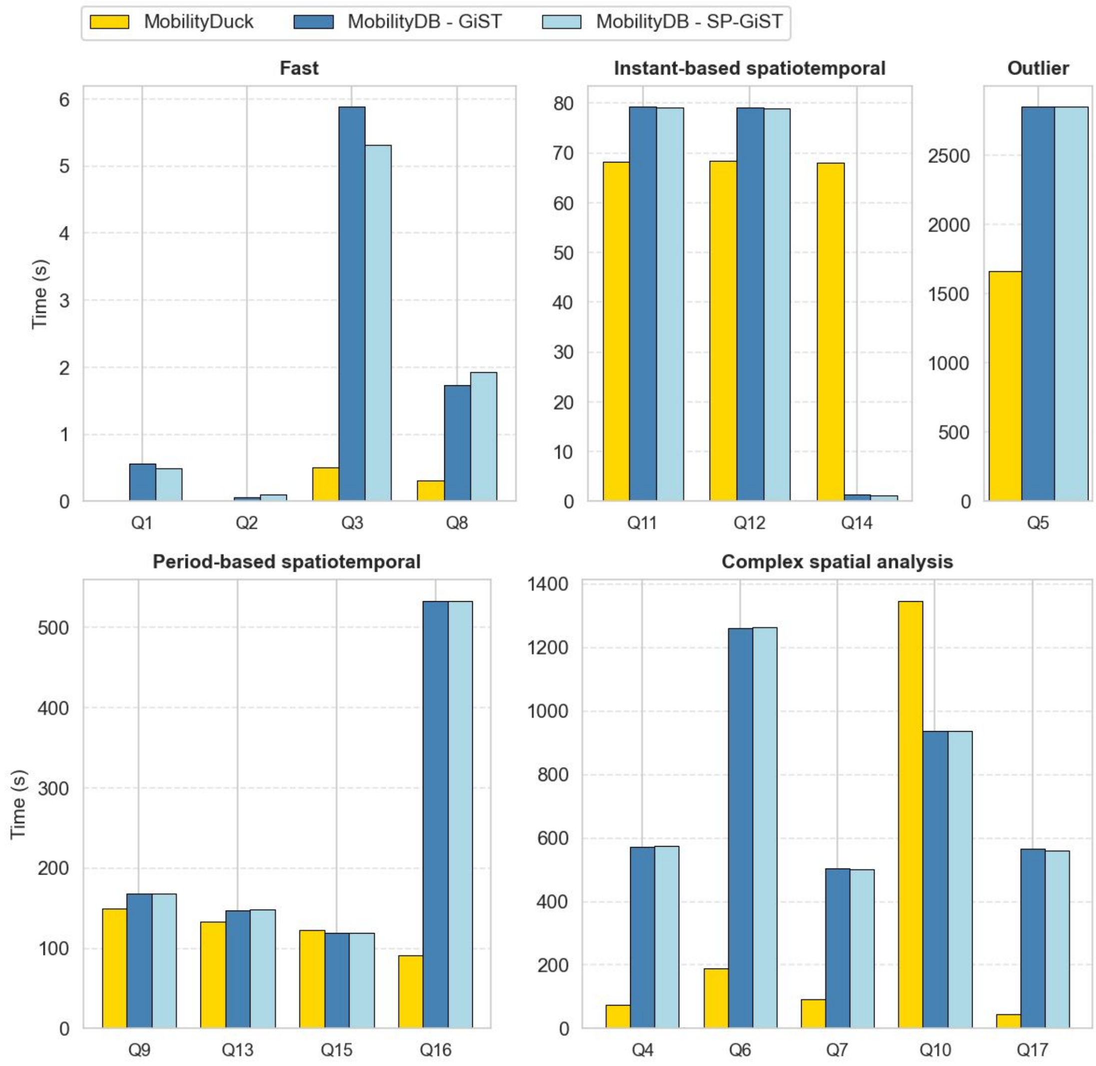


Benchmarking Context: BerlinMOD-Hanoi

- An adaptation of the standard **BerlinMOD** benchmark
- **Motivation**: moving beyond the standard Europe-centric convention to a high-density Asian urban context
- **Road network**: real OpenStreetMap data from Hanoi, Vietnam
- **Mobility model**: synthetic trips based on population statistics
- **Output**: scalable datasets of realistic commuting trips represented as `tgeompoint` trajectories
- **Queries**: 17 range-style queries
- **Scale factors**: 4 scales, ranging from ~9,500 trips (35.7M GPS points) to ~35,000 trips (131.2M GPS points)

Benchmarking Results

- Experimental setup: MobilityDuck w/o index vs. MobilityDB with R-tree index (GiST) & quad-tree index (SP-GiST)
- MobilityDuck outperforms in 14/17 queries
- Q10 & Q14:
    - Complex spatial joins
    - Heavily call upon indexed columns



Scalability on Commodity Hardware

- Tested up to SF-0.2 ~ 20GB data on disk
- Configurations: Ubuntu 20.04, 4 CPUs, 24GB RAM, 20GB swap memory
- SF-0.3 & SF-0.5 (~ 300M GPS points) caused process termination due to resource exhaustion
- Spatiotemporal queries create high-cardinality or variable-length spatial geometries that are not always spillable to disk
- Limitation of the test environment, not an inherent DuckDB ceiling; larger SFs are feasible on higher-memory environments

Python Integration

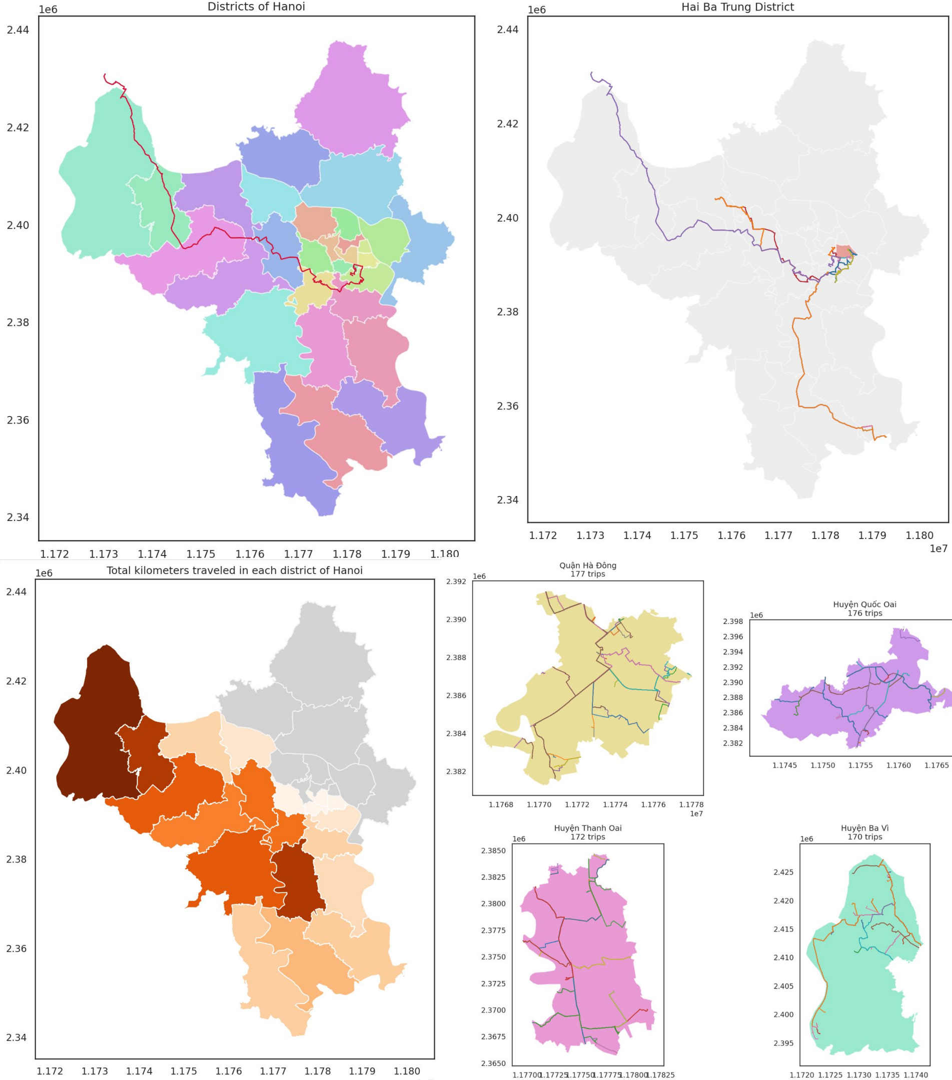
MobilityDuck → Spatiotemporal analytics

↓

DuckDB Python API → Database management

↓

GeoPandas Shapely Plotly → Visualizations



Summary & Future Works

Key takeaways

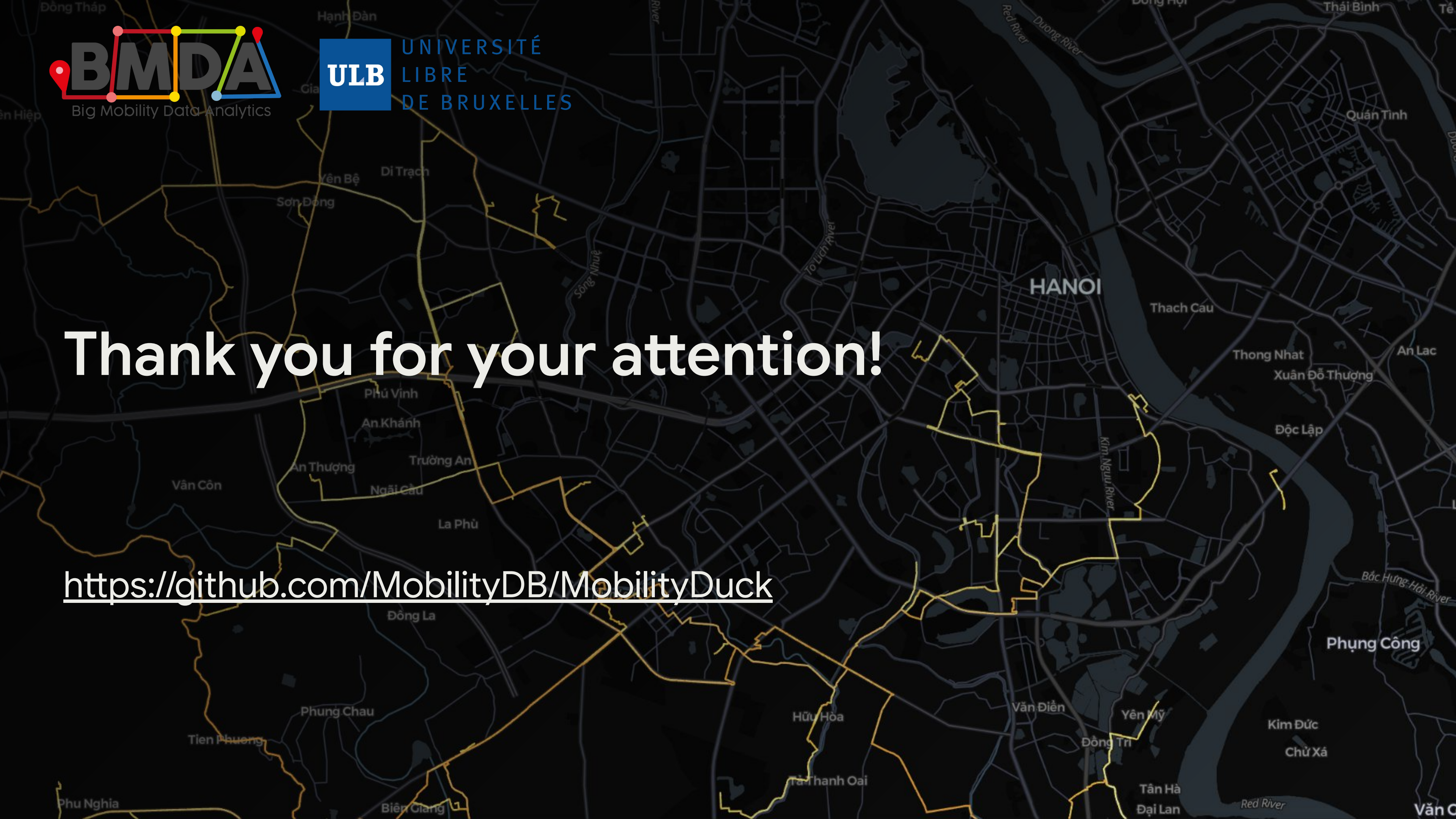
- **Feasibility**: successfully embedded MEOS algebra into DuckDB's vectorized engine
- **Performance**: MobilityDuck outperforms traditional indexing for most analytical queries
- **Contribution**: released BerlinMOD-Hanoi for densely populated urban benchmarking

Future works

- Automating MEOS bindings generation
- Supporting temporal geography type

BMDA
Big Mobility Data Analytics

ULB UNIVERSITÉ LIBRE DE BRUXELLES



Thank you for your attention!

https://github.com/MobilityDB/MobilityDuck