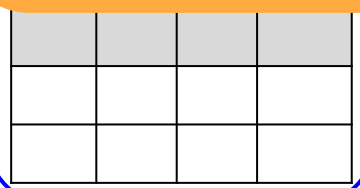
Enabling Moving Features in Multiple Computing Environments

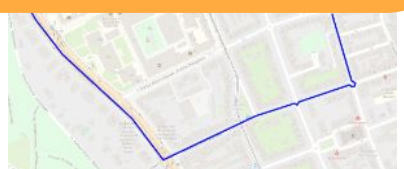
Esteban Zimányi
Université libre de Bruxelles, Belgium

125th OGC Member Meeting, Moving Features Standards Working Group
Frascati, Italy, February 2023

	PostgreSQL	PostGIS	MobilityDB
SQL Optimization	Scalar statistics & selectivity estimation	Grid-based statistics	Spatial grid + period bound histograms
Indexes	B-tree, hash, GiST, SP-GiST, GIN, BRIN	GiST, SP-GiST, BRIN	GiST, SP-GiST
Operations	Comparison, transformation, casting, ...etc	topological, CRS, properties, overlay, ...etc.	trajectory, temporal properties, lifted predicates, aggregations
Type System	numeric, character, date/time, bool, xml, json	Geometry, geography	tgeompoint, tgeogpoint, tint, tfloat, ttext, tbool

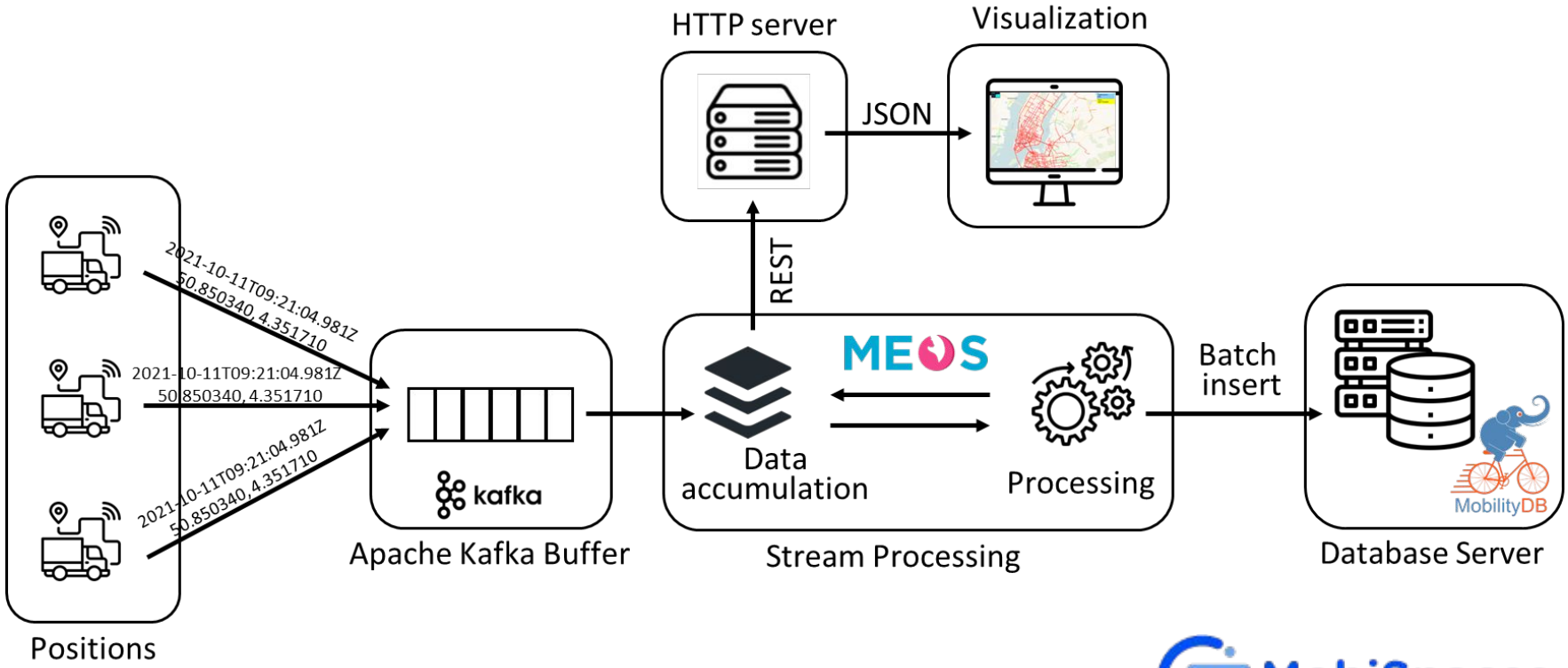
MEOS





https://mobilitydb.com/

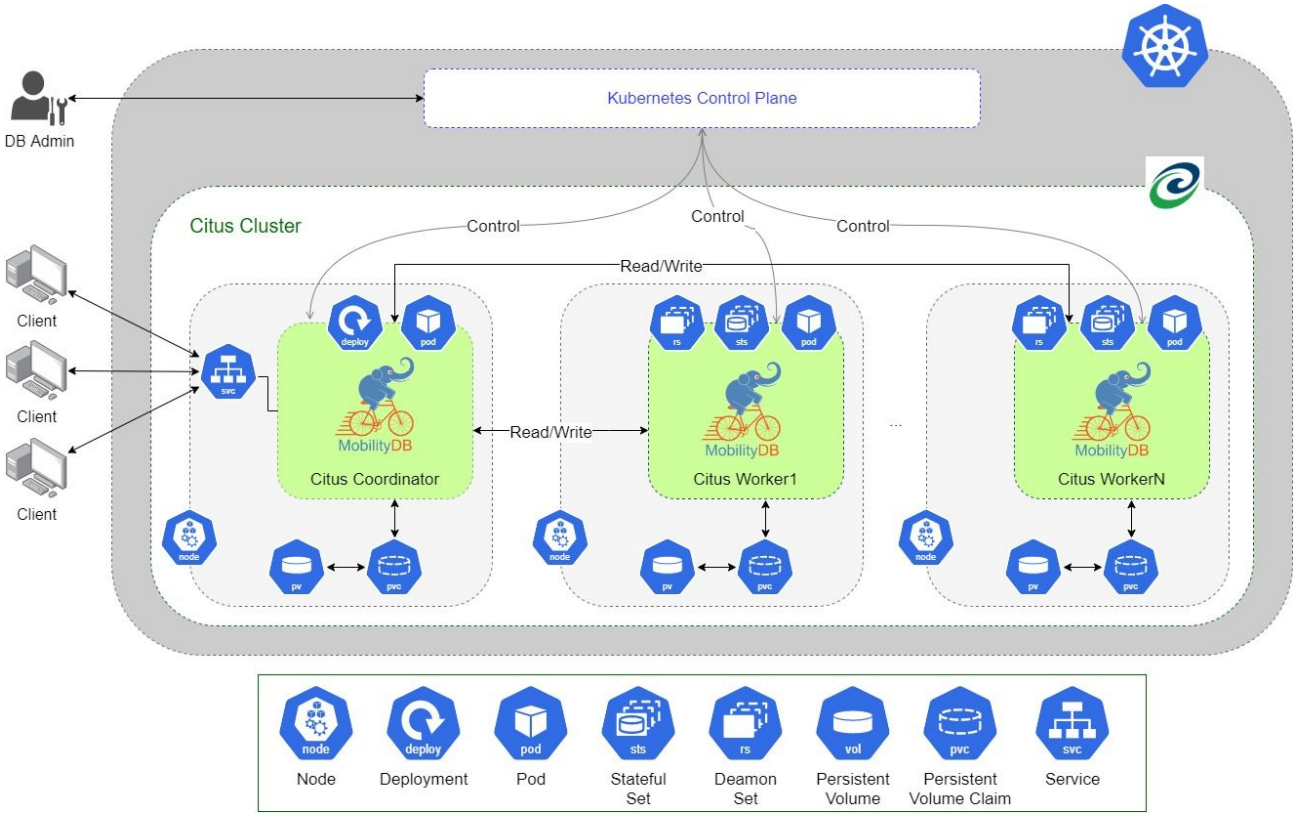
Moving Features in Stream and Edge Processing



MobiSpaces
new data spaces for green mobility

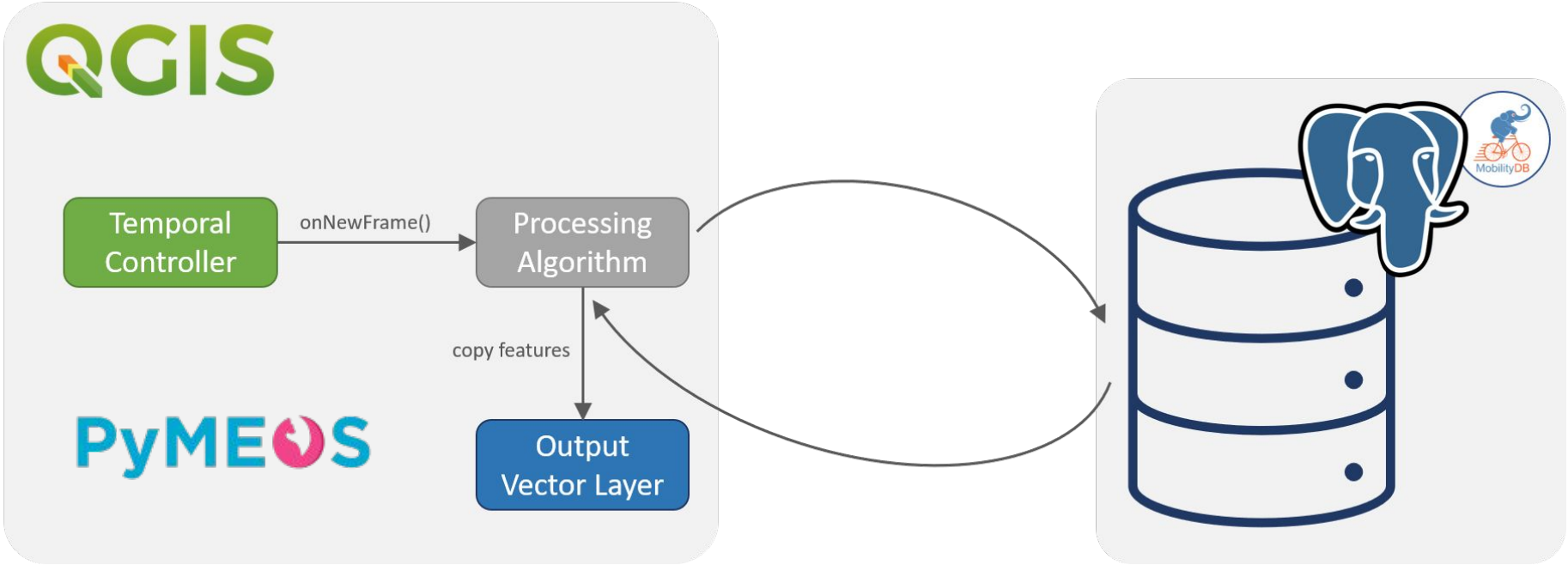
https://mobispaces.eu/

Moving Features in the Cloud



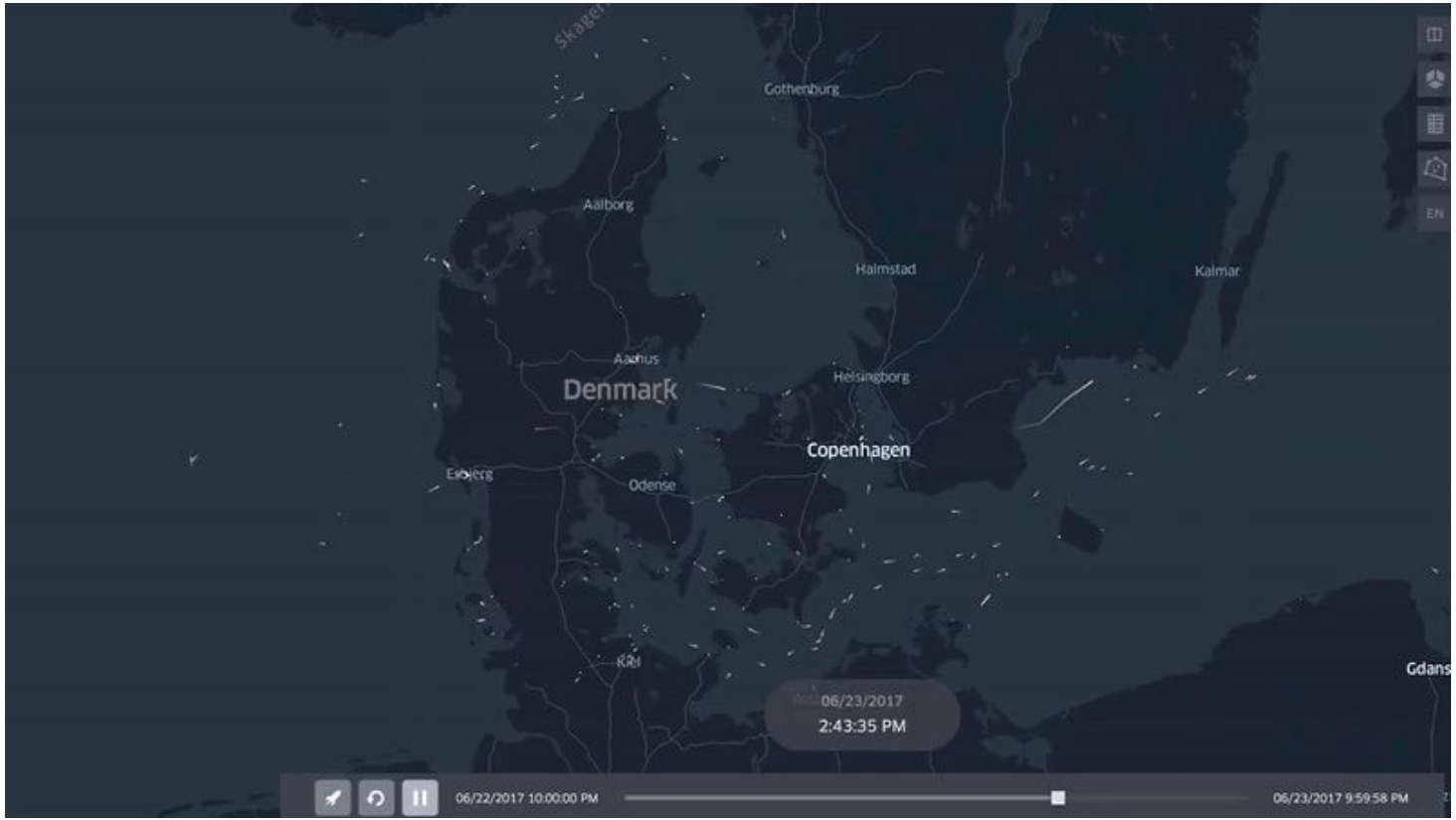
https://github.com/MobilityDB/MobilityDB-Azure

Visualizing Moving Features in QGIS



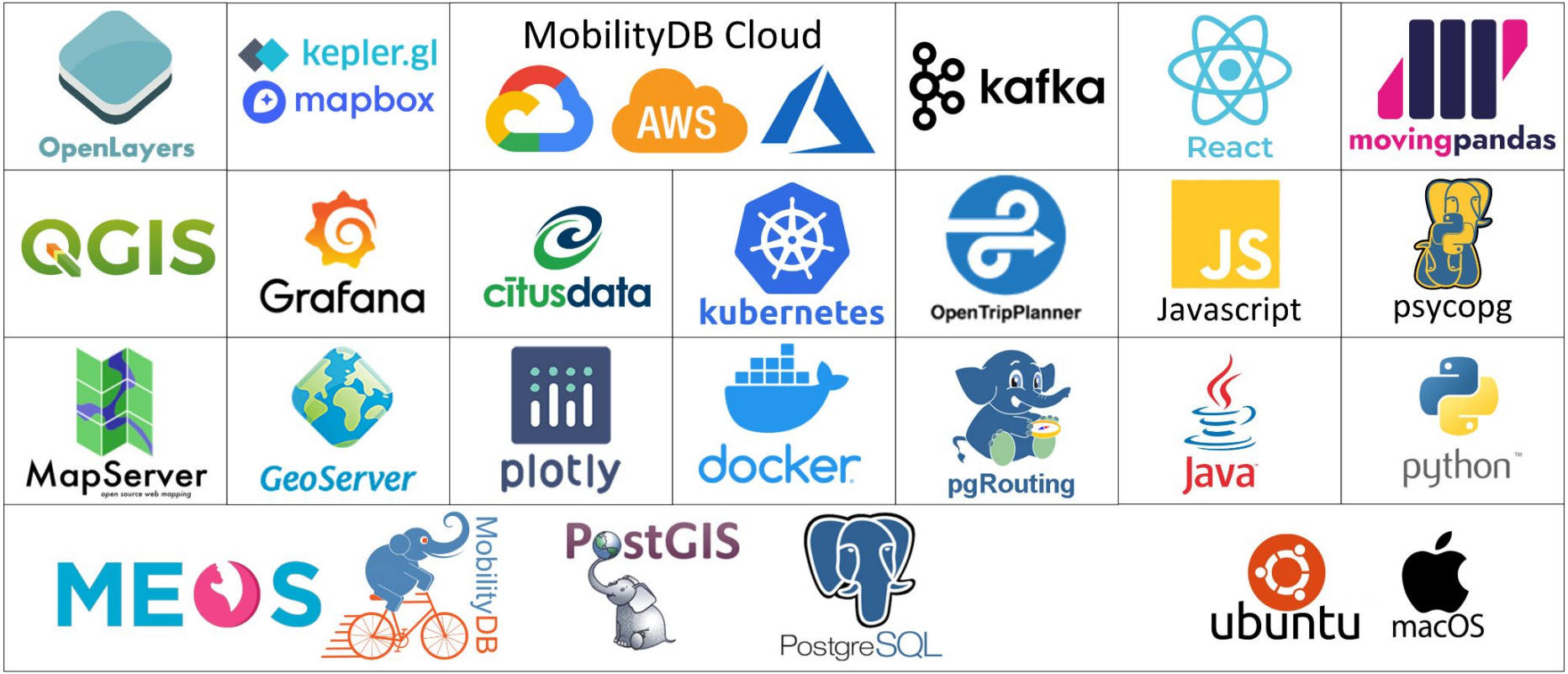
https://github.com/MobilityDB/MobilityDB-QGIS

Visualizing Moving Features in Deck.gl



https://github.com/MobilityDB/MobilityDB-Deck

MobilityDB Ecosystem



MEOS

MEOS (Mobility Engine, Open Source) is a C library and its associated API for manipulating temporal and spatiotemporal data. It is the core component of MobilityDB, an open source geospatial trajectory data management & analysis platform built on top of PostgreSQL and PostGIS.

MEOS extends the ISO 19141:2008 standard (Geographic information — Schema for moving features) for representing the change of non-spatial attributes of features. It also takes into account the fact that when collecting mobility data it is necessary to represent "temporal gaps", that is, when for some period of time no observations were collected due, for instance, to signal loss.

MEOS is heavily inspired by a similar library called GEOS (Geometry Engine, Open Source) — hence the name. A first version of the MEOS library written in C++ has been proposed by Krishna Chaitanya Bommakanti. However, due to the fact that MEOS codebase is actually a subset of MobilityDB codebase, which is written in C and in SQL, the current version of the library allows us to evolve both programming environments simultaneously.

MEOS aims to be the base library on which other projects can be built. For example, the following projects are built on top of MEOS:

- PyMEOS is a Python binding to MEOS using CFFI
- MobilityDB is a PostgreSQL extension that enables storing and manipulating the temporal types provided by MEOS.

Other projects can built on top of MEOS, for example, Java or C# drivers for MEOS or implementing MEOS on other DBMSs such as MySQL.

https://libmeos.org

GEOS

GEOS is a C/C++ library for computational geometry with a focus on algorithms used in geographic information systems (GIS) software. It implements the OGC *Simple Features* geometry model and provides all the spatial functions in that standard as well as many others. GEOS is a core dependency of PostGIS, QGIS, GDAL, and Shapely.

If you need support using the GEOS library or would like to get involved in the community check out the Support page.

Capabilities

Spatial Model and Functions

- **Geometry Model**: Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, GeometryCollection
- **Predicates**: Intersects, Touches, Disjoint, Crosses, Within, Contains, Overlaps, Equals, Covers
- **Operations**: Union, Distance, Intersection, Symmetric Difference, Convex Hull, Envelope, Buffer, Simplify, Polygon Assembly, Valid, Area, Length,
- **Prepared geometry** (using internal spatial indexes)
- **Spatial Indexes**: STR (Sort-Tile-Recursive) packed R-tree spatial index
- **Input/Output**: OGC Well Known Text (WKT) and Well Known Binary (WKB) readers and writers.

API Features

https://libgeos.org

MobilityDB, MEOS, PyMEOS, and Friends

- MEOS enables a single code base for manipulating moving features in multiple programming environments and languages
- Thin wrappers for several programming languages

6.4.1 Input/Output Functions

- Get the Well-Known Text (WKT) representation

  `asText({tpoint,tpoint[],geo[]}) → {text,text[]}`

```
SELECT asText(tgeompoint 'SRID=4326;[Point(0 0 0)@2001-01-01, Point(1 1 1)@2001-01-02)');
-- [POINT Z (0 0 0)@2001-01-01, POINT Z (1 1 1)@2001-01-02)
SELECT asText(ARRAY[geometry 'Point(0 0)', 'Point(1 1)']);
-- {"POINT(0 0)","POINT(1 1)"}
```

- Get the Extended Well-Known Text (EWKT) representation

  `asEWKT({tpoint,tpoint[],geo[]}) → {text,text[]}`

```
SELECT asEWKT(tgeompoint 'SRID=4326;[Point(0 0 0)@2001-01-01, Point(1 1 1)@2001-01-02)');
-- SRID=4326;[POINT Z (0 0 0)@2001-01-01, POINT Z (1 1 1)@2001-01-02)
SELECT asEWKT(ARRAY[geometry 'SRID=5676;Point(0 0)', 'SRID=5676;Point(1 1)']);
-- {"SRID=5676;POINT(0 0)","SRID=5676;POINT(1 1)"}
```

- Get the Moving Features JSON representation

  `asMFJSON(tpoint,options=0,flags=0,maxdecdigits=15) → bytea`

  The `options` argument can be used to add BBOX and/or CRS in MFJSON output:

  - 0: means no option (default value)
  - 1: MFJSON BBOX
  - 2: MFJSON Short CRS (e.g., EPSG:4326)
  - 4: MFJSON Long CRS (e.g., urn:ogc:def:crs:EPSG::4326)

https://docs.mobilitydb.com/MobilityDB/develop/

Connection between MobilityDB and MEOS

MobilityDB SQL definition

```
CREATE FUNCTION asText(tgeompoint)
  RETURNS text
  AS 'MODULE_PATHNAME', 'Tpoint_as_text'
  LANGUAGE C IMMUTABLE STRICT PARALLEL SAFE;
CREATE FUNCTION asText(tgeogpoint)
  RETURNS text
  AS 'MODULE_PATHNAME', 'Tpoint_as_text'
  LANGUAGE C IMMUTABLE STRICT PARALLEL SAFE;
```

MobilityDB C definition

```
PGDLLEXPORT Datum
Tpoint_as_text(PG_FUNCTION_ARGS)
{
  Temporal *temp = PG_GETARG_TEMPORAL_P(0);
  int dbl_dig_for_wkt = OUT_DEFAULT_DECIMAL_DIGITS;
  if (PG_NARGS() > 1 && ! PG_ARGISNULL(1))
    dbl_dig_for_wkt = PG_GETARG_INT32(1);
  char *str = tpoint_as_text(temp, dbl_dig_for_wkt);
  text *result = cstring2text(str);
  pfree(str);
  PG_FREE_IF_COPY(temp, 0);
  PG_RETURN_TEXT_P(result);
}
```

MEOS

https://github.com/MobilityDB/MobilityDB/wiki/Building-MobilityDB-and-MEOS

Connection between PyMEOS and MEOS

PyMEOS Classes

```python
class TPoint(Temporal[shp.Point, TG, TI, TS, TSS], ABC):
    """
    Abstract base class for both Geographic and Geometric types of any temporal subtype.
    """

    def as_wkt(self, precision: int = 15):
        """Return the string representation of the content of ``self``."""
        return tpoint_as_text(self._inner, precision)

    def as_ewkt(self, precision: int = 15):
        return tpoint_as_ewkt(self._inner, precision)
```

PyMEOS

PyMEOS CFFI Interface

```python
def tpoint_as_text(temp: 'const Temporal *', maxdd: int) -> str:
    temp_converted = _ffi.cast('const Temporal *', temp)
    result = _lib.tpoint_as_text(temp_converted, maxdd)
    result = _ffi.string(result).decode('utf-8')
    return result if result != _ffi.NULL else None
```

MEOS

https://github.com/MobilityDB/PyMEOS/wiki/PyMEOS-Architecture

MEOS API

Modules

Functions for PostgreSQL types
Functions for PostgreSQL types.

Functions for PostGIS types
Functions for PostGIS types.

Functions for set and span types
Functions for set and span types.

Functions for box types
Functions for box types.

Functions for temporal types
Functions for temporal types.

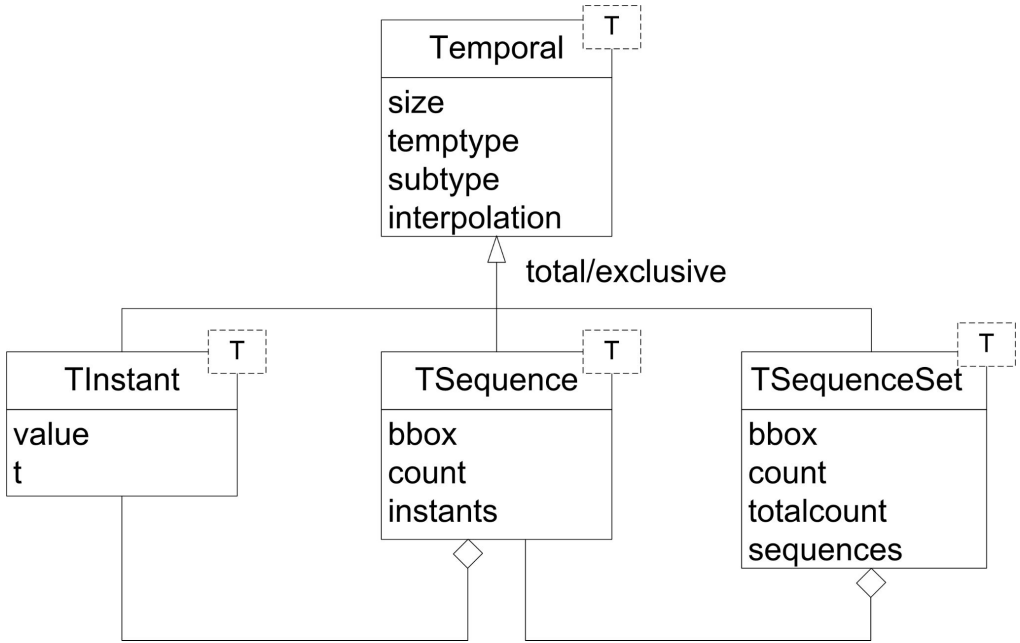
Functions

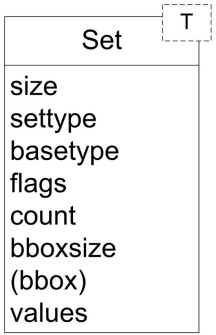
Temporal *	**tbool_in** (const char *str) Return a temporal boolean from its Well-Known Text (WKT) representation. More...
char *	**tbool_out** (const **Temporal** *temp) Return a temporal boolean from its Well-Known Text (WKT) representation. More...
char *	**temporal_as_hexwkb** (const **Temporal** *temp, uint8_t variant, size_t *size_out) Return the WKB representation of a temporal value in hex-encoded ASCII. More...
char *	**temporal_as_mfjson** (const **Temporal** *temp, bool with_bbox, int flags, int precision, char *srs) Return the MF-JSON representation of a temporal value. More...
uint8_t *	**temporal_as_wkb** (const **Temporal** *temp, uint8_t variant, size_t *size_out) Return the WKB representation of a temporal value. More...
Temporal *	**temporal_from_hexwkb** (const char *hexwkb) Return a temporal value from its HexEWKB representation. More...
Temporal *	**temporal_from_mfjson** (const char *mfjson) Return a temporal point from its MF-JSON representation. More...

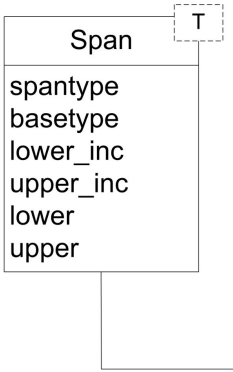
. . .

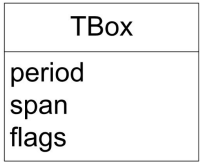
https://docs.mobilitydb.com/libmeos/html/modules.html

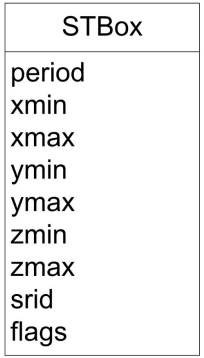
Temporal Model: Data Structures











https://libmeos.org/documentation/datastructures/

MobilityDB and OGC's Moving Features SWG

- MobilityDB aims at being 100% conformant with OGC's MF standards
- MF-JSON support for many years
  Extended for all temporal types: tbool, tint, tfloat, ttext
- OGC's Simple Feature Access WKT and WKB extended for all temporal types
- WKB is essential in distributed environments such as the cloud:
  Processes need an efficient way to exchange information
- MF-API support is being done in the European project EMERALDS
- Essential component for enabling Mobility as a Service (MaaS)
- MobilityDB provides an open-source testbench for designing and implementing MF-SWG standards

https://emeralds-horizon.eu/

EMERALDS

MobilityDB and OGC's Moving Features API

- The API of MobilityDB is peer reviewed and published in two TODS publications:

  MobilityDB: A mobility database based on PostgreSQL and PostGIS
  E Zimányi, M Sakr, A Lesuisse
  ACM Transactions on Database Systems (TODS) 45(4), pp 19:1--19:42, 2020

  A foundation for representing and querying moving objects
  RH Güting, MH Böhlen, M Erwig, CS Jensen, NA Lorentzos, M Schneider, M Vazirgiannis
  ACM Transactions on Database Systems (TODS) 25 (1), 1-42, 2000

- Scientific basis for defining the MF-API

OGC Moving Features API

Rationale: split the development of the standard into three incremental phases

- P1: Retrieval of features and their attributes
  - Corresponding to operations in MF-Access and their temporal counterparts
  - Accessor functions, restriction functions (atX, minusX), from/to MF-JSON, speed, azimuth, …
- P2: Topological operations/predicates between pairs of features, where at least one is a moving feature
  - Lifted topological relationships, simple topological relationships, from/to WKT, WKB
  - Example: “intersection” between a geometry object (e.g., an administrative boundary) and a trajectory of a moving feature (e.g., a car, a person, a vessel, an aircraft, a hurricane).
- P3: Advanced aggregation and analytical operations
  - Temporal aggregates (tcentroid, tmax, tsum, …), static aggregates (twAvg), simplify, …
  - Example: movement patterns between soccer players for defining proper tactics

Home The Project Publications Documentation Installation Tutorials Contact

MobilityDB

An open source geospatial trajectory data management & analysis platform.

View on Github

Many thanks for your attention !

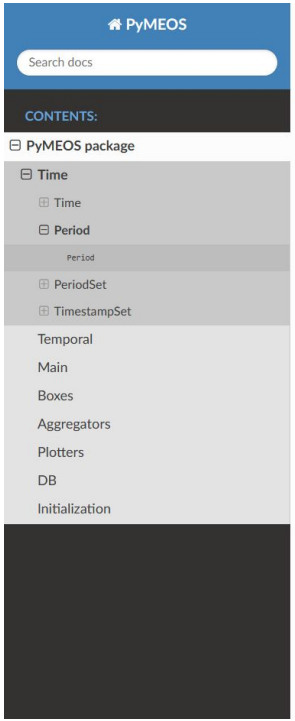
Location tracking devices, such as GPS, are nowadays widely used in smartphones, and in vehicles. As a result, geospatial trajectory data are currently being collected and used in many application domains. MobilityDB provides the necessary database support for storing and querying such geospatial trajectory data

MobilityDB is implemented as an extension to PostgreSQL and PostGIS. It implements persistent database types, and query operations for managing geospatial trajectories and their time-varying properties.

Learn More

OPEN CHAT

PyMEOS API



Period

class **pymeos.time.period.Period**(*string: str | None = None, *, lower: str | datetime | None = None, upper: str | datetime | None = None, lower_inc: bool = True, upper_inc: bool = False, _inner=None*)

Bases: `object`

Class for representing sets of contiguous timestamps between a lower and an upper bound. The bounds may be inclusive or not.

`Period` objects can be created with a single argument of type string as in MobilityDB.

```
>>> Period('(2019-09-08 00:00:00+01, 2019-09-10 00:00:00+01)')
```

Another possibility is to provide the `lower` and `upper` named parameters (of type str or datetime), and optionally indicate whether the bounds are inclusive or exclusive (by default, the lower bound is inclusive and the upper is exclusive):

```
>>> Period(lower='2019-09-08 00:00:00+01', upper='2019-09-10 00:00:00+01')
>>> Period(lower='2019-09-08 00:00:00+01', upper='2019-09-10 00:00:00+01', lower_inc=False, upper_in
>>> Period(lower=parse('2019-09-08 00:00:00+01'), upper=parse('2019-09-10 00:00:00+01'), upper_inc=T
```

static **from_hexwkb**(*hexwkb: str*) → Period

Returns a *Period* from its WKB representation in hex-encoded ASCII. :param hexwkb: WKB representation in hex-encoded ASCII

Returns: A new `Period` instance

MEOS Functions:

span_from_hexwkb

shift(*delta: timedelta*) → Period

Returns a new period that is the result of shifting `self` by `delta`

Examples

```
>>> Period('[2000-01-01, 2000-01-10]').shift(timedelta(days=2))
>>> 'Period([2000-01-03 00:00:00+01, 2000-01-12 00:00:00+01])'
```

Parameters: **delta** – `datetime.timedelta` instance to shift
Returns: A new `Period` instance

MEOS Functions:

period_shift_tscale

tscale(*duration: timedelta*) → Period

Returns a new period that starts as `self` but has duration `duration`

Examples

```
>>> Period('[2000-01-01, 2000-01-10]').tscale(timedelta(days=2))
>>> 'Period([2000-01-01 00:00:00+01, 2000-01-03 00:00:00+01])'
```

Parameters: **duration** – `datetime.timedelta` instance representing the duration of the new period
Returns: A new `Period` instance

MEOS Functions:

period_shift_tscale