# OSM Data Processing with PostgreSQL / PostGIS

Jochen Topf
jochentopf.com

OpenStreetMap

PostgreSQL

Studley Tool Chest | CC-BY-SA | https://www.flickr.com/photos/publicresourceorg/493813720

## osm2pgsql builds ways table but not line table

osm2pgsql

19 Jun, 08:48 SimonPoole ♦ **38.2k**

## how to get way OSM ID lat, lon from postgresql database?

psql imposm postgresql postgres

21 Jan, 07:01 SimonPoole ♦ **38.2k**

## Update osm2pgsql database from apidb database

apidb osm2pgsql osmosis

14 Aug, 11:02 YUNtee **16**

## Nominatim PgSQL database size growing gradually

database nominatim size

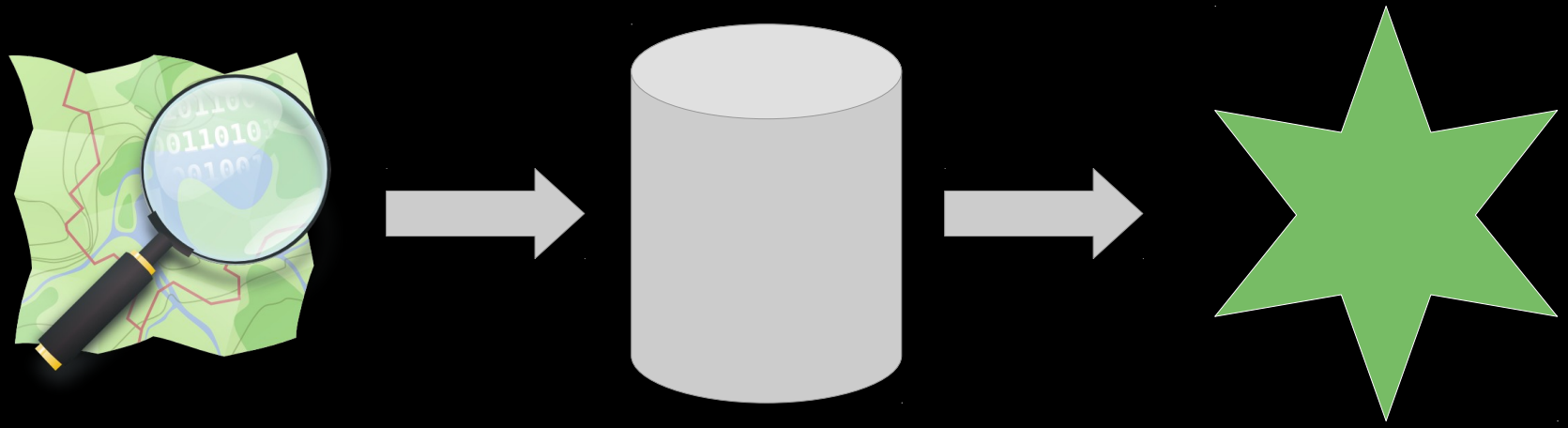
29 Jul '16, 12:36 jot **496**

## osm2pgsql processing a 68MB planet diff for more than 19 hours

planet diff nominatim osm2pgsql updates

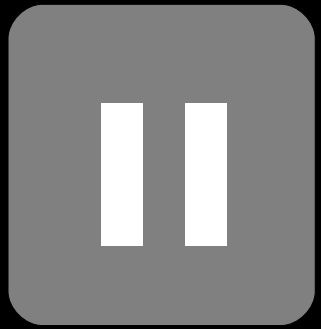
06 Jun, 10:36 SimonPoole ♦ **38.2k**

From help.openstreetmap.org



# What we will talk about...

- Background: Relational Databases, Geodata
- Converting OSM Data
- Use Cases
- Tools
- Tips & Tricks, Odds & Ends



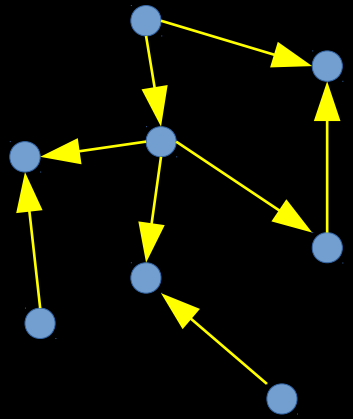
# Background:

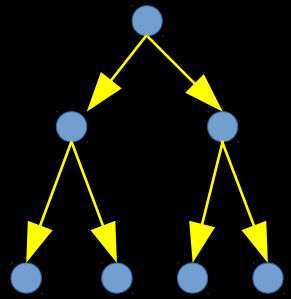
# Relational Databases

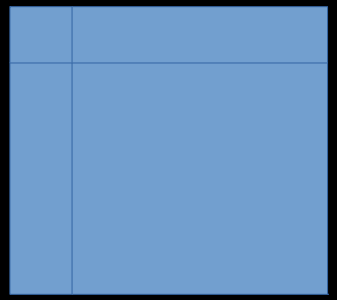
# Databases

Databases store and manipulate data.

There are many different ways to organize data...







# Relational Databases

All data is organized in Tables.

# Relational Databases

**Table Members:**

| ID | Name | Place | Age |
|---|---|---|---|
| 1 | Joe | Sydney | 34 |
| 2 | Jenny | New York | 42 |
| 3 | Jeremy | Moskow | 55 |

# Relational Databases

**Table Members:**

| ID | Name | Place | Age |
|---|---|---|---|
| 1 | Joe | Sydney | 34 |
| 2 | Jenny | New York | 42 |
| 3 | Jeremy | Moskow | 55 |

# Relational Databases

**Table Members:**

| ID | Name | Place | Age |
|---|---|---|---|
| 1 | Joe | Sydney | 34 |
| 2 | Jenny | New York | 42 |
| 3 | Jeremy | Moskow | 55 |

# Relational Databases

**Table Members:**

| ID | Name | Place | Age |
|---|---|---|---|
| 1 | Joe | Sydney | 34 |
| 2 | Jenny | New York | 42 |
| 3 | Jeremy | Moskow | 55 |

# Relational Databases

**Table Members:**

| ID | Name | Place | Age |
|---|---|---|---|
| 1 | Joe | Sydney | 34 |
| 2 | Jenny | New York | 42 |
| 3 | Jeremy | Moskow | 55 |

# Relational Databases

**Table Members:**

| ID | Name | Place | Age |
|---|---|---|---|
| 1 | Joe | Sydney | 34 |
| 2 | Jenny | New York | 42 |
| 3 | Jeremy | Moskow | 55 |

# Data Types

Fields have a type:

Text
Integer
Numeric
Date
...

# Relational Databases

**Table Members:**

| ID | Name | Place | Age |
|---|---|---|---|
| 1 | Joe | Sydney | 34 |
| 2 | Jenny | New York | 42 |
| 3 | Jeremy | Moskow | 55 |

Structured Query Language

# Data Access using SQL

```
SELECT Name
     FROM Members;
```

| Name |
| --- |
| Joe |
| Jenny |
| Jeremy |

# Data Access using SQL

```
SELECT Name, Age
    FROM Members;
```

| Name | Age |
| --- | --- |
| Jenny | 42 |
| Joe | 34 |
| Jeremy | 55 |

# Data Access using SQL

```
SELECT Name, Age
     FROM Members
     ORDER BY Age;
```

| Name | Age |
|---|---|
| Joe | 34 |
| Jenny | 42 |
| Jeremy | 55 |

# Data Access using SQL

```
SELECT Name, Age
     FROM Members
     WHERE Age > 40
     ORDER BY Name;
```

| Name | Age |
| --- | --- |
| Jenny | 42 |
| Jeremy | 55 |

# INSERT, UPDATE, DELETE

```
INSERT INTO Members (Name, Place, Age)
        VALUES (’Julia’, ’London’, 27);

UPDATE Members SET Place = ’Helsinki’
        WHERE Id = 2;

DELETE FROM Members
        WHERE Name = ’Jeremy’;
```

# INSERT, UPDATE, DELETE

```
INSERT INTO Members (Name, Place, Age)
        VALUES (’Julia’, ’London’, 27);

UPDATE Members SET Place = ’Helsinki’
        WHERE Id = 2;

DELETE FROM Members
        WHERE Name = ’Jeremy’;
```

# INSERT, UPDATE, DELETE

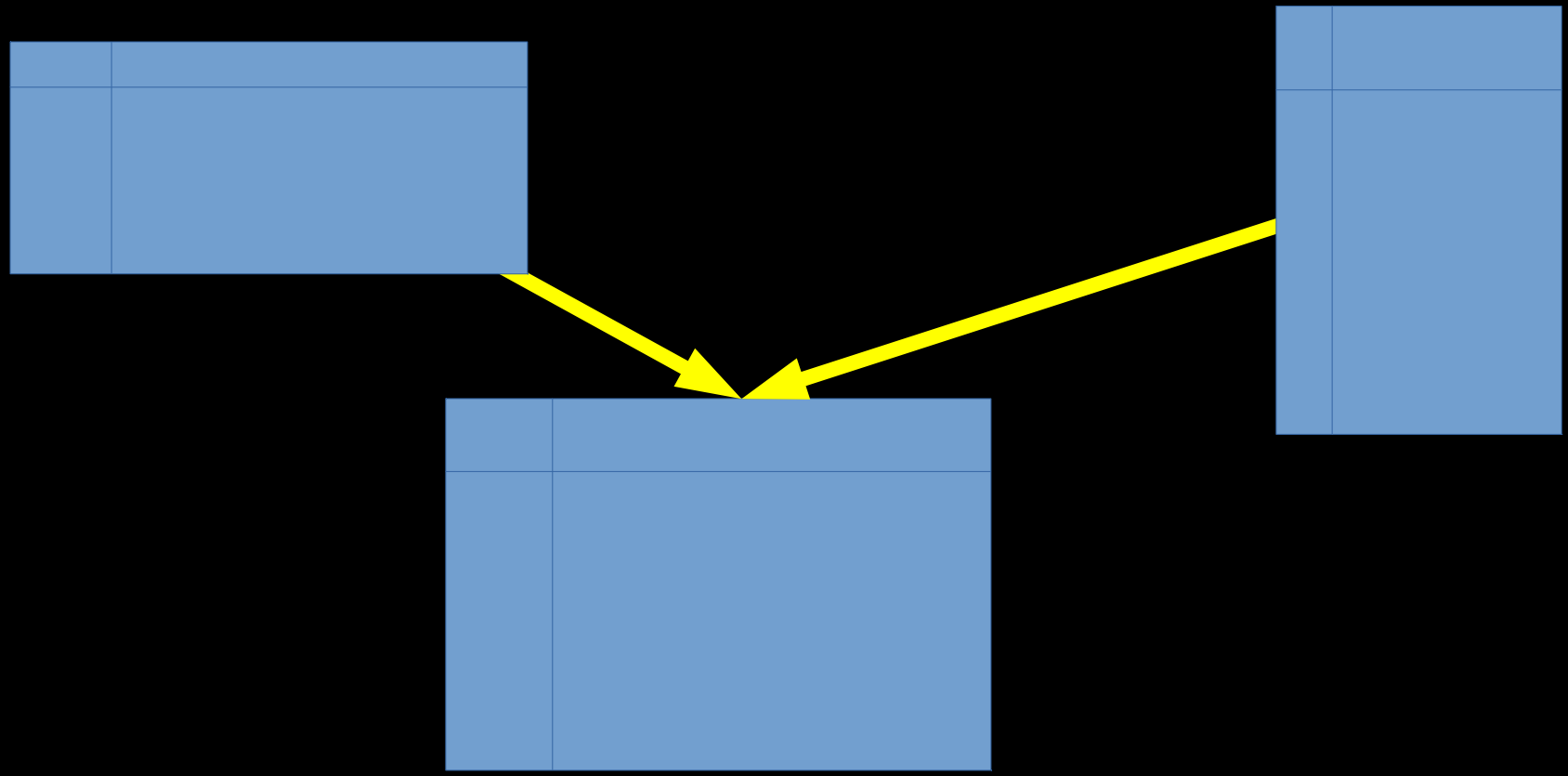
```
INSERT INTO Members (Name, Place, Age)
        VALUES (’Julia’, ’London’, 27);

UPDATE Members SET Place = ’Helsinki’
        WHERE Id = 2;

DELETE FROM Members
        WHERE Name = ’Jeremy’;
```

# Advanced SQL: Aggregate Functions

```
SELECT Avg(Age)
    FROM Members;
```

| Avg |
| --- |
| 43 |

# Advanced SQL: JOIN



# The Magic

You define a structure (“schema“).

You add data.

You ask for data back.

The database software does everything else.

# Not so magic...

Performance can depend on structure

You still need to know a bit…

# Indexes

Indexes allow faster access
for some queries

Tradeoff: Indexes need space and need to be updated vs. faster queries

# Relational Databases

PostgreSQL

MySQL, MariaDB

SQLite

...

# Relational Databases

PostgreSQL

MySQL, MariaDB

SQLite

...

# PostgreSQL

Open Source

lots of features

good documentation, books, etc.

popular, great eco-system, well-supported

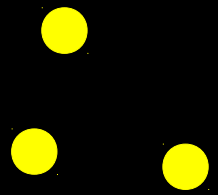
powerful plugin system

# Background:

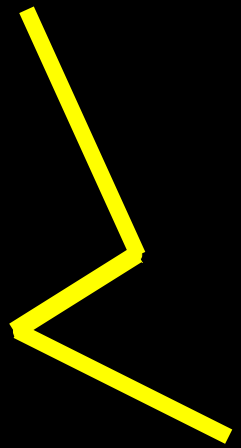
# Geodata

# Simple Feature Model

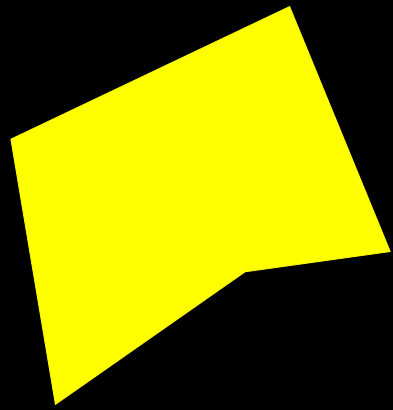
Point



LineString



Polygon



MultiPoint

MultiLineString

MultiPolygon

# We want to store this in a database

# A Naive Approach...

| ID | Name | X | Y |
|---|---|---|---|
| 1 | Joe | 151.22 | -33.85 |
| 2 | Jenny | -74.01 | 40.70 |
| 3 | Jeremy | 37.61 | 55.95 |

# A Better Approach...

Text
Integer
Numeric
Date
Geometry
...

# A Better Approach...

| ID | Name | Geom |
|---|---|---|
| 1 | Joe | POINT(151.22 -33.85) |
| 2 | Jenny | POINT(-74.01 40.70) |
| 3 | Jeremy | POINT(37.61 55.95) |

# PostGIS: Plugin

```
CREATE EXTENSION postgis;
```

# PostGIS: Datatypes

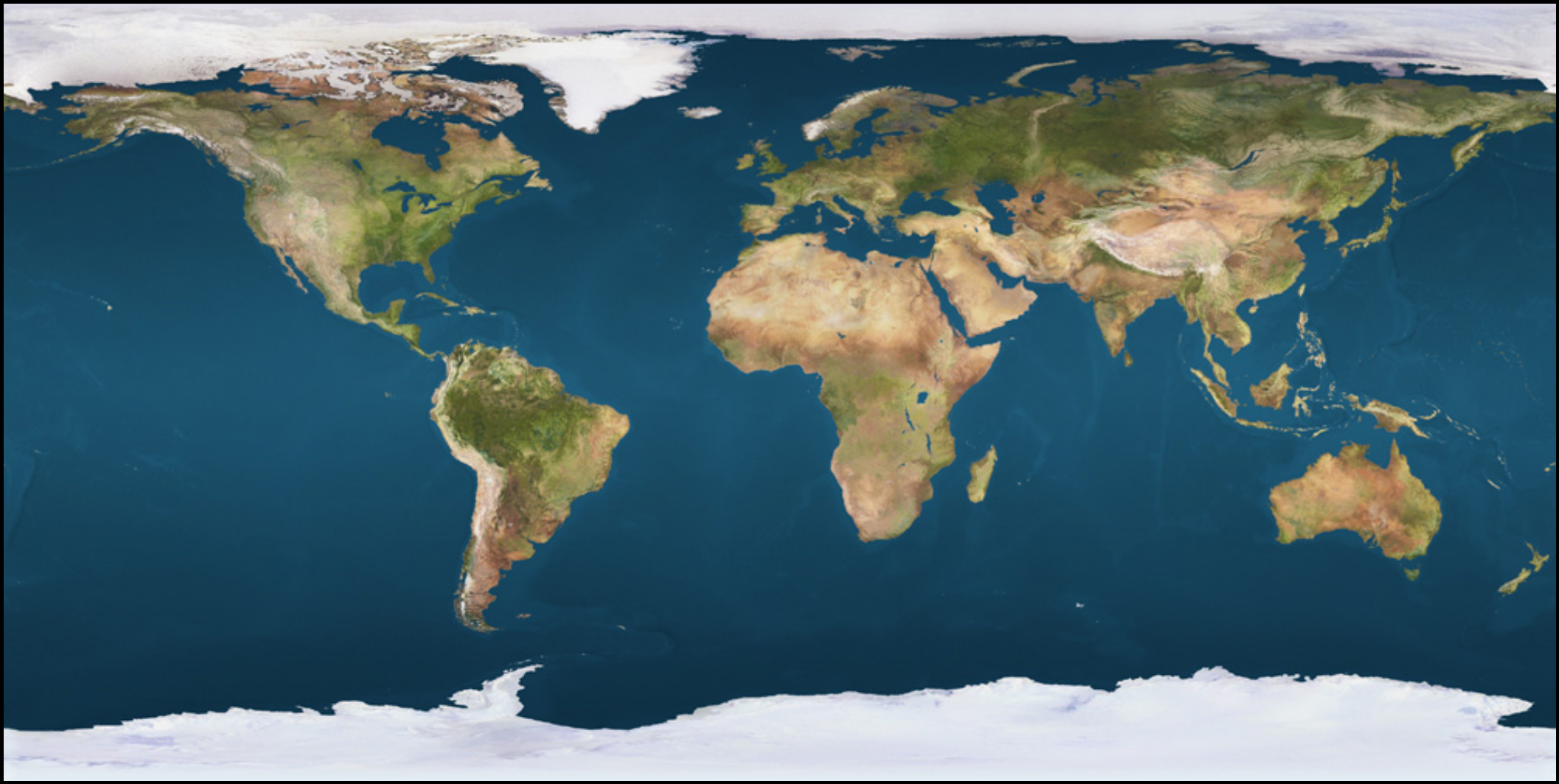
GEOMETRY

GEOMETRY (POINT)
GEOMETRY (LINESTRING)
GEOMETRY (POLYGON)

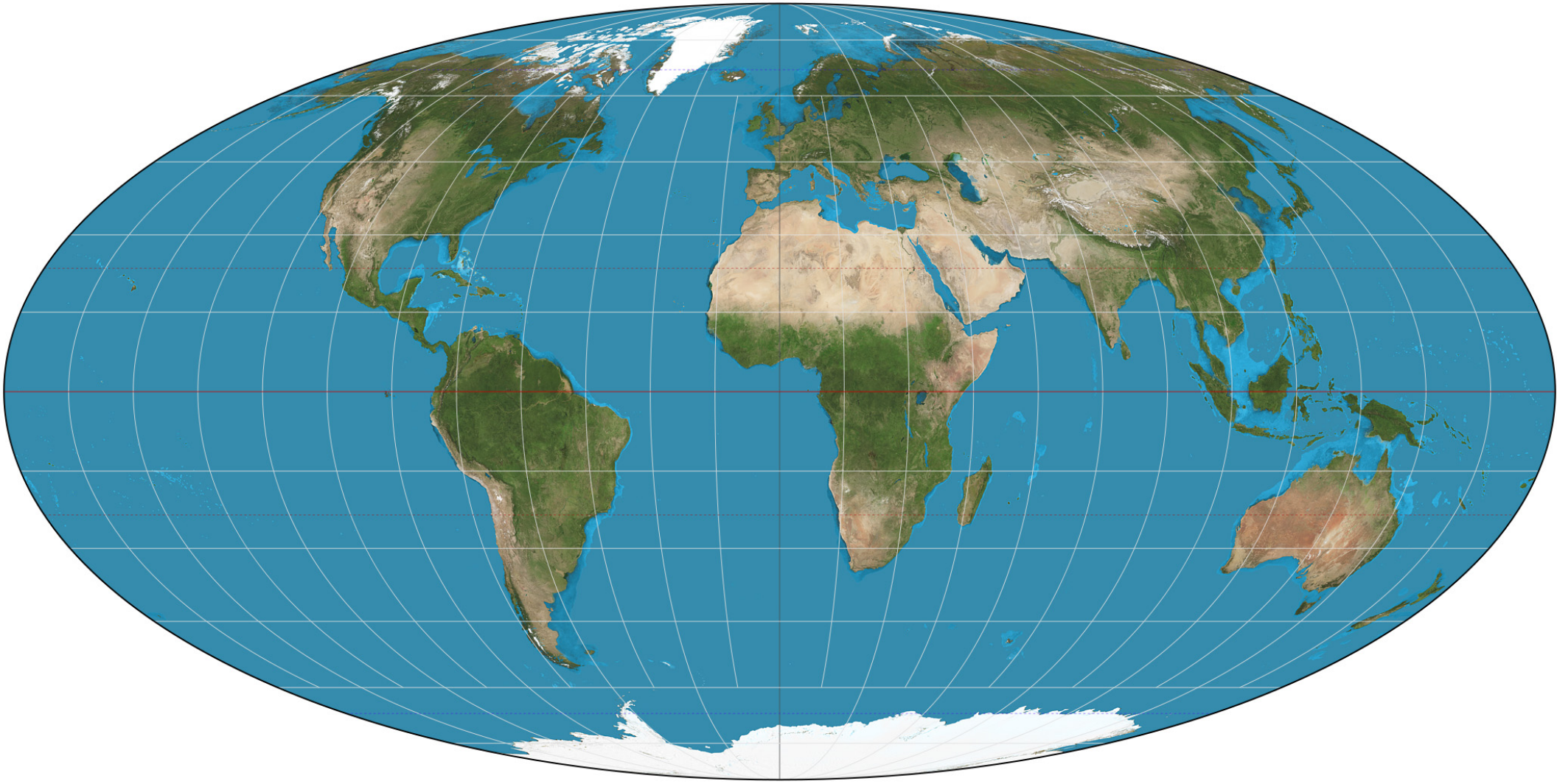
(also: GEOGRAPHY)

Source: Wikipedia

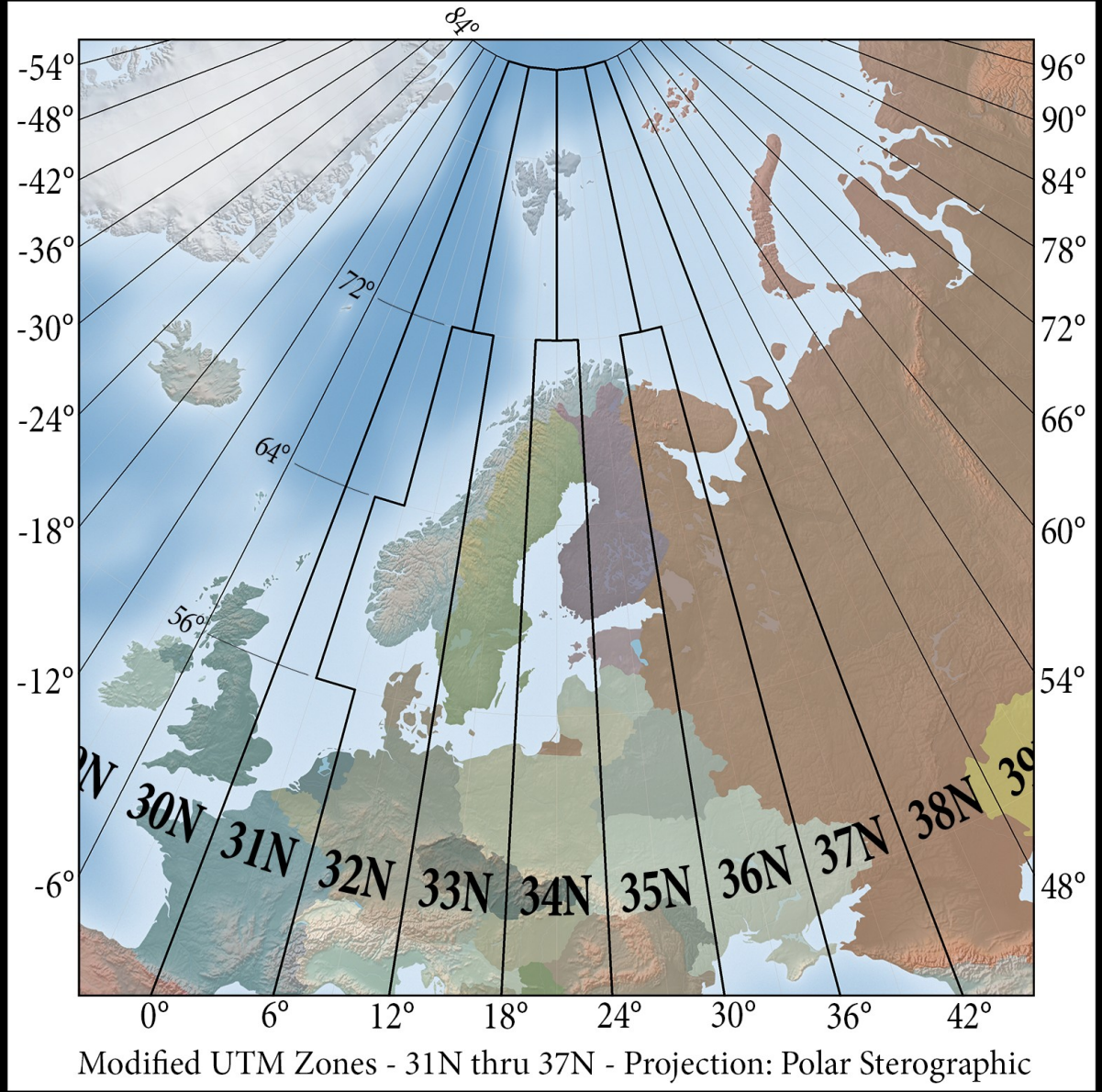
Source: Wikipedia



Source: Wikipedia



Source: Wikipedia




Modified UTM Zones - 31N thru 37N - Projection: Polar Sterographic

Source: Wikipedia

# PostGIS: CRS

PostGIS knowns > 5000 Coordinate Systems (CRS/SRS)

Each Geometry associated with SRID.

Allows Transformations

Mix and match data sources

# Most important CRSes:

WGS84 – EPSG:4326

Web Mercator – EPSG:3857

# PostGIS: Datatypes

GEOMETRY (POINT, 4326)
GEOMETRY (LINESTRING, 4326)
GEOMETRY (POLYGON, 4326)

# Coordinates

Always first X axis, then Y axis
(as in mathematics).

so: longitude first, then latitude.

# Well Known Text (WKT)

POINT(4 3)

LINESTRING(12 4, 3 2, 7, 9)

POLYGON((0 0, 4 0, 4 4, 0 4, 0 0))

MULTIPOINT / -LINESTRING / -POLYGON

# PostGIS: Indexes

Normal indexes are good for
1-dimensional data

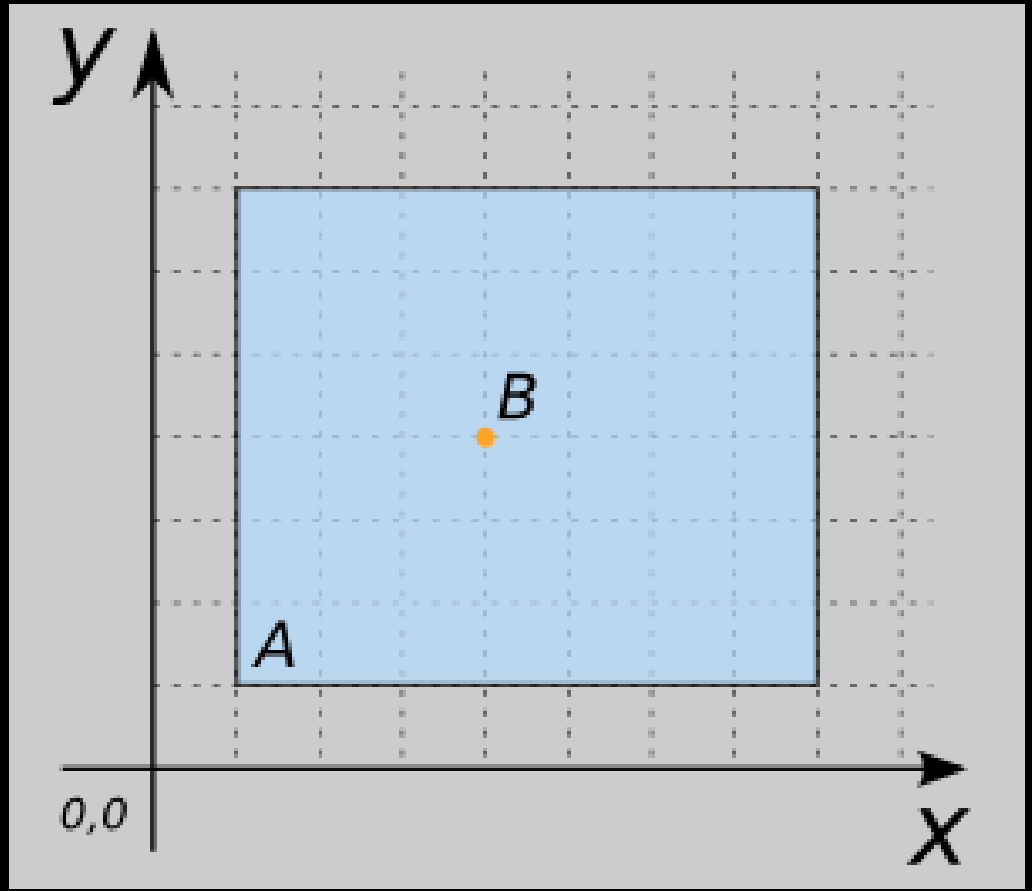
Spatial indexes are good for
2/3-dimensional data
(R -tree)

# PostGIS: Operations

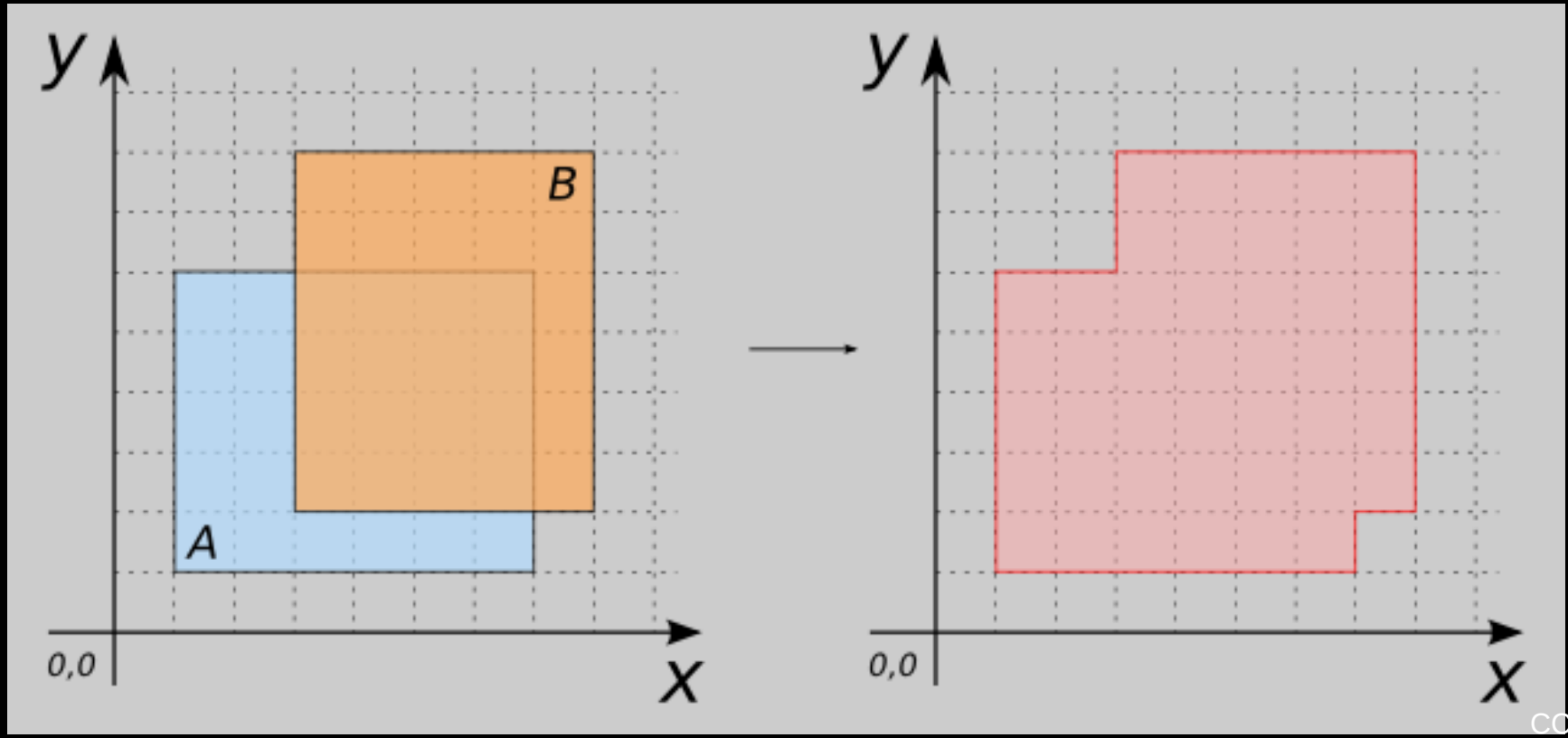
Huge number of

operations on spatial data

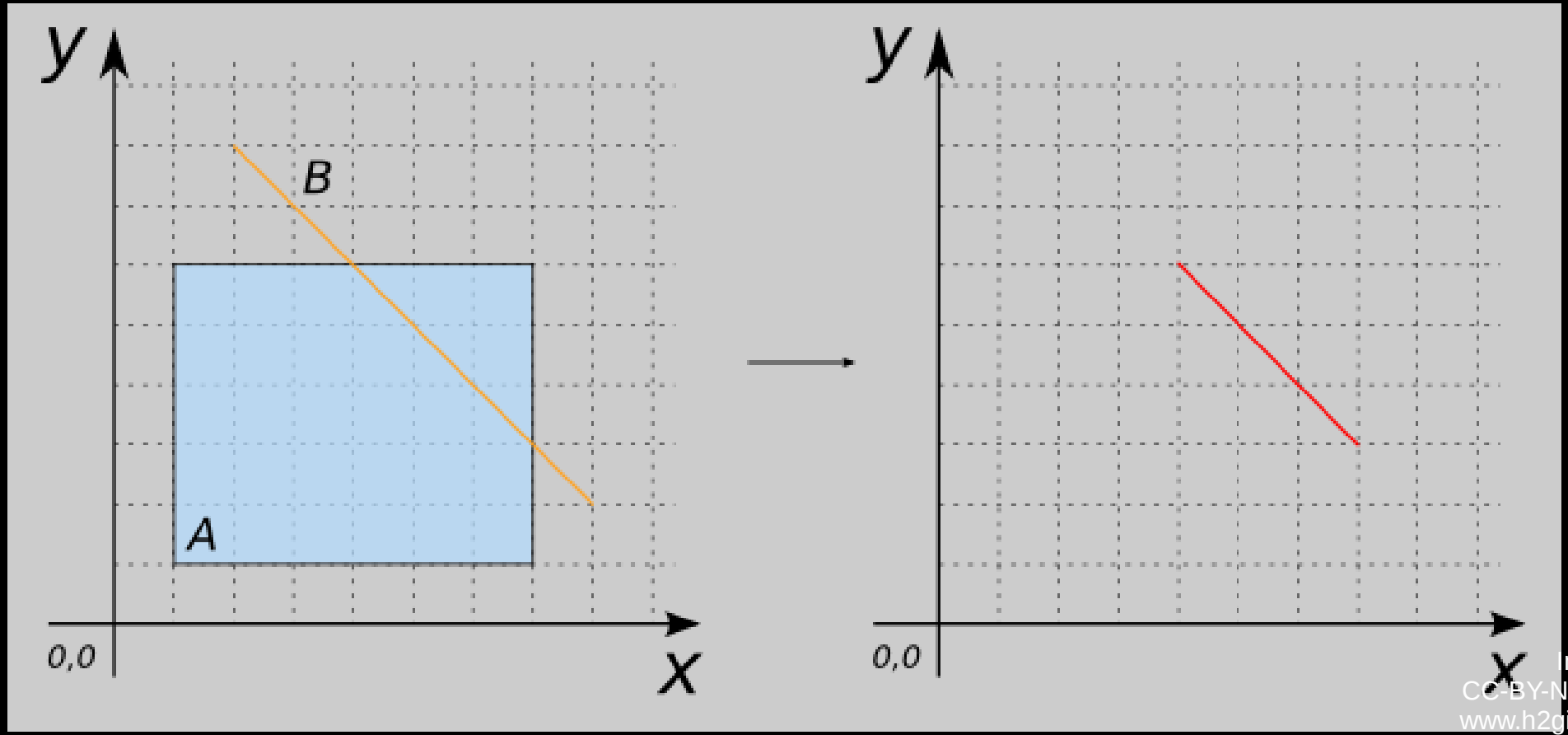
# PostGIS: ST_Contains



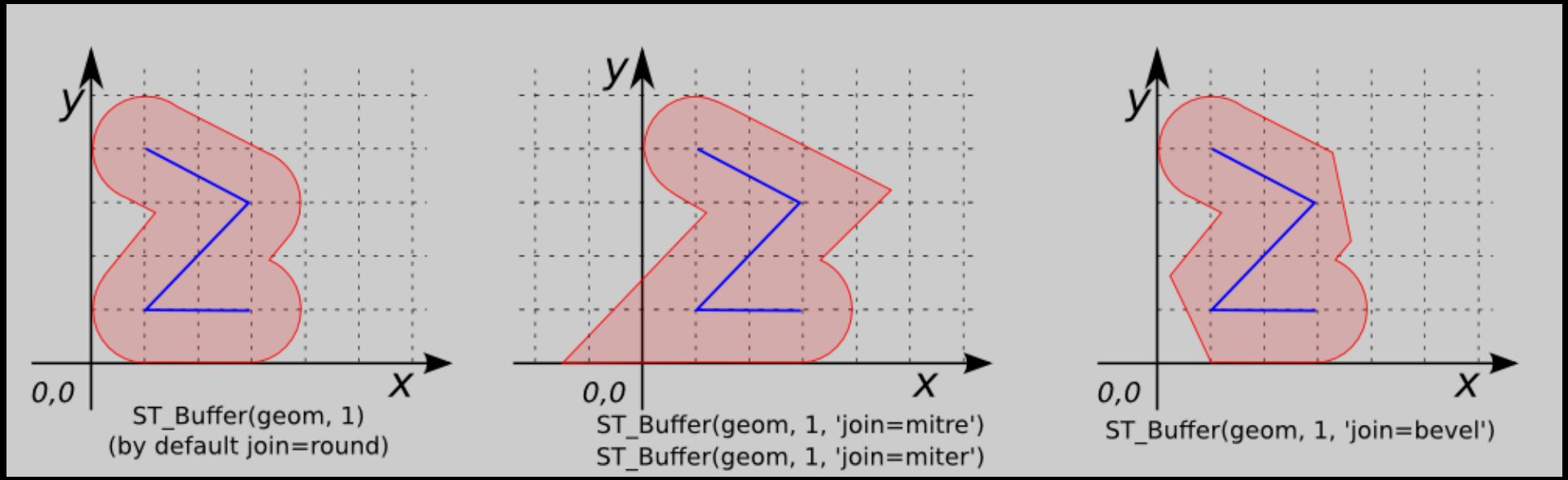
# PostGIS: ST_Union



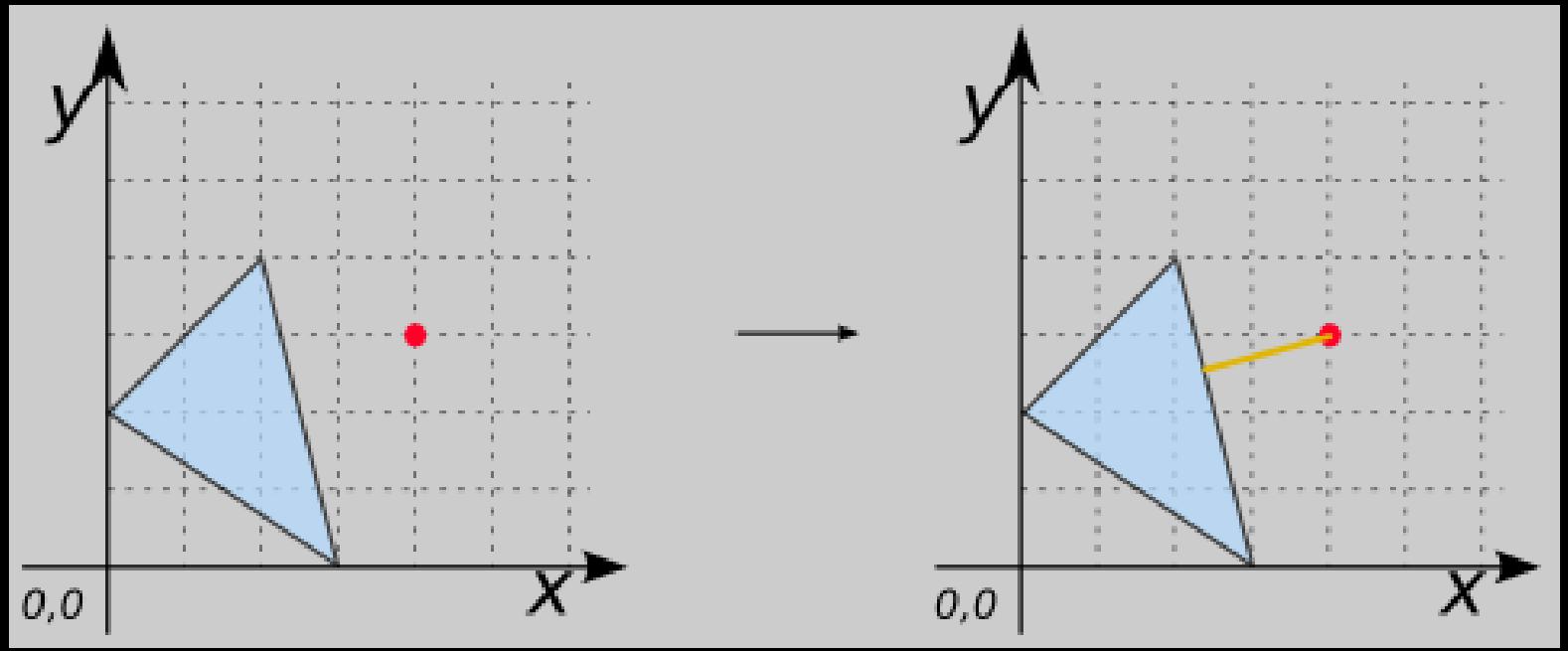
# PostGIS: ST_Intersection



# PostGIS: ST_Buffer

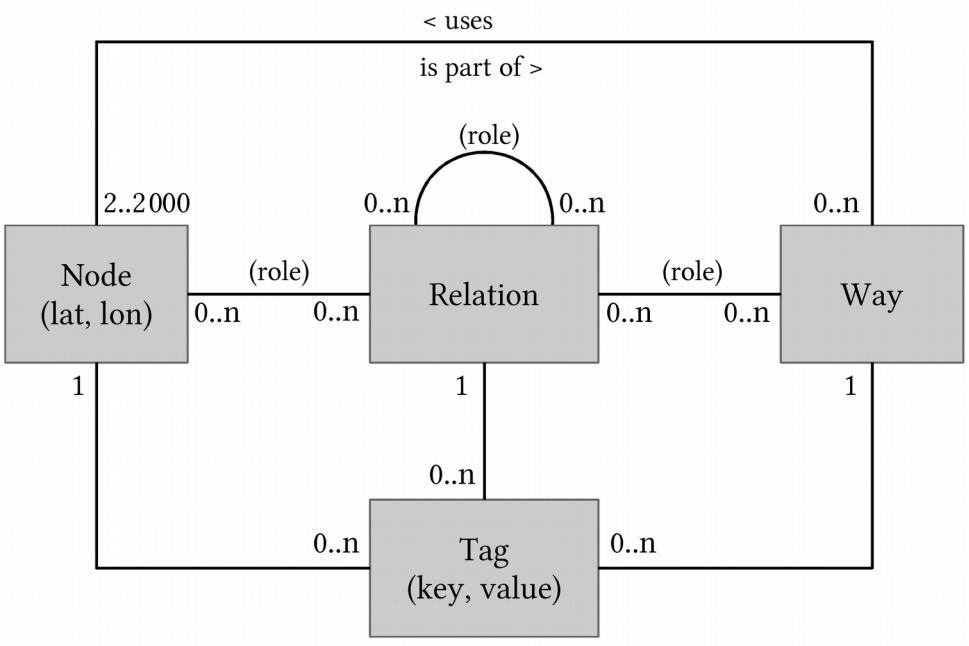


# PostGIS: ST_ShortestLine



# Converting OSM Data

# OSM Data Model



# Mismatch

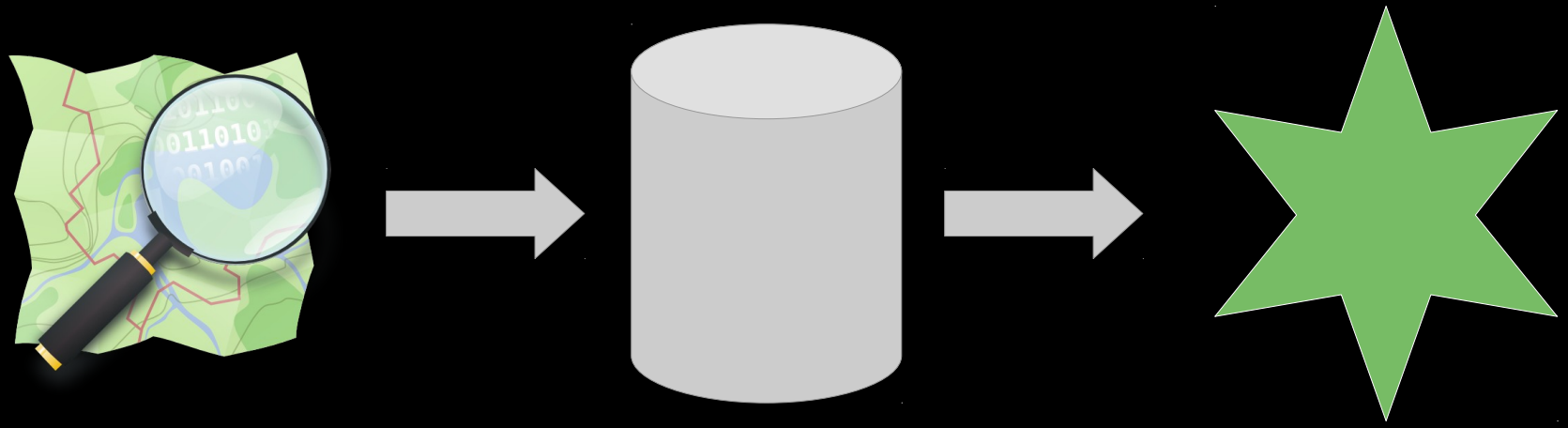
OSM
Data Model

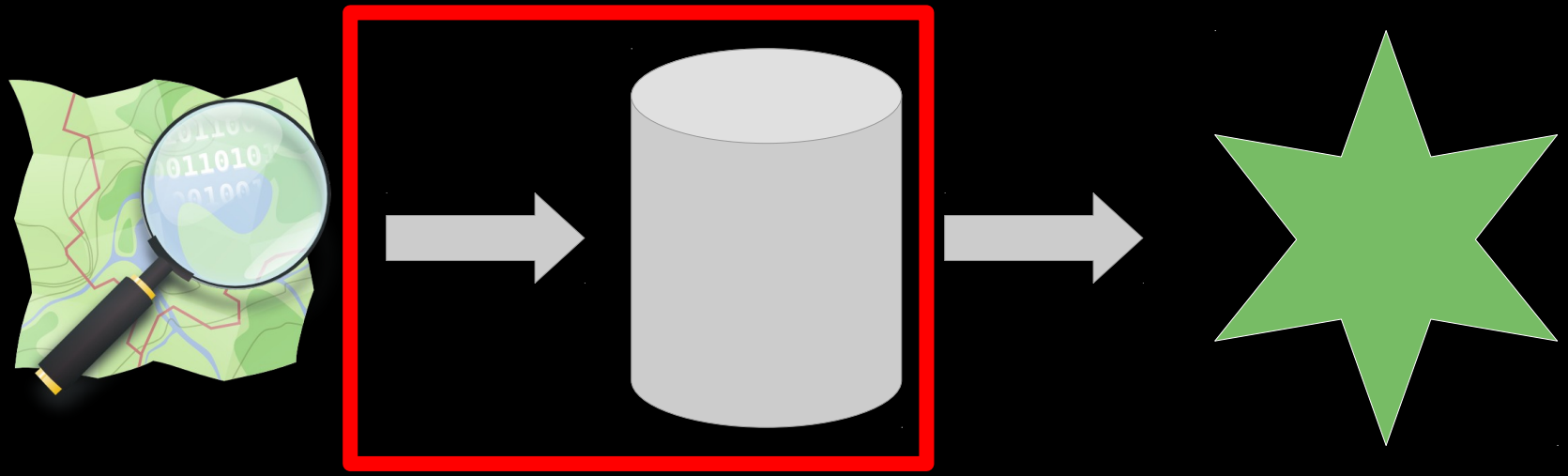
Relational /
Simple Feature
Data Model

# Mismatch

OSM
Data Model


Relational /
Simple Feature
Data Model





# Conversion: Selection

What data do we actually need?

nodes, ways, relations?

user id, timestamp, version, …?

which tags?

# Conversion: Data Types

tags in OSM:
key → value (both text)

Map to:
text, integer, boolean, enums, ...

# Conversion: Tags → Attributes

tags in OSM are flexible, table columns are fixed

| Highway | Name | Oneway |
|---|---|---|
| primary | Main St | false |
| residential | Elm St | true |
| trunk | | true |

# Conversion: Tags → Attributes

tags in OSM are flexible, table columns are fixed

| Highway | Name | Oneway |
|---|---|---|
| primary | Main St | false |
| residential | Elm St | true |
| trunk | | true |

# Conversion: Tags → Attributes

tags in OSM are flexible, table columns are fixed

| Highway | Name | Oneway |
|---|---|---|
| primary | Main St | false |
| residential | Elm St | true |
| trunk | | true |

# Conversion: hstore and JSON

| Place | Name |
| --- | --- |
| city | de: München, en: Munich |
| city | de: Aachen, fr: Aix-la-Chapelle |
| village | de: Lübben, hsb: Lubin |

# Conversion: Tables

split data into tables…

few tables vs. many tables

by geometry type and/or by subject type

# Split by Geometry Type

Tables:

nodes
ways
areas

# Split by Geometry/Feature Class

Tables:

| | | |
|---|---|---|
| restaurants | highways | lakes |
| bus_stops | railways | forests |
| addresses | rivers | countries |
| places | powerlines | buildings |
| ... | ... | ... |

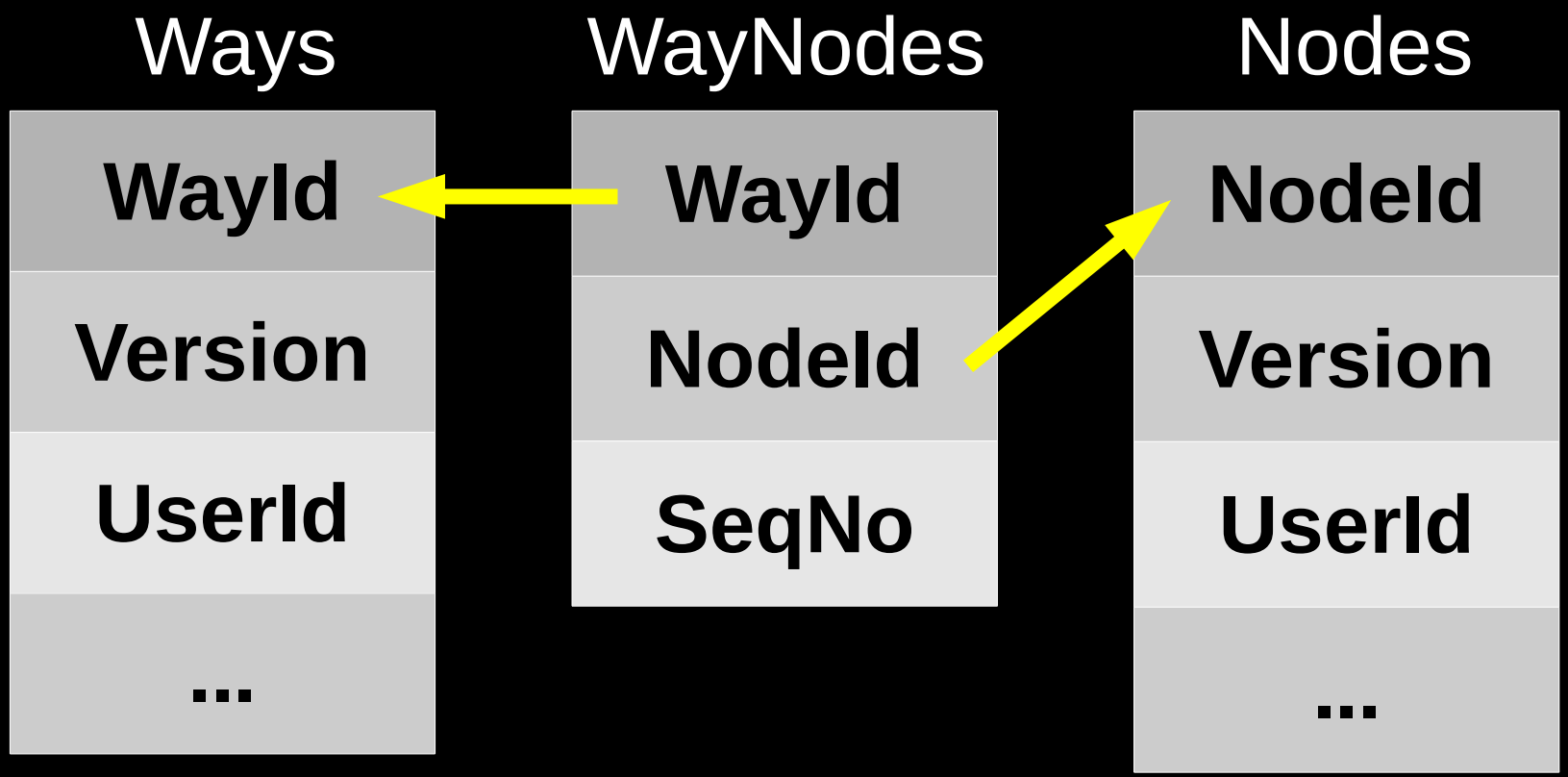
# Conversion: Handling lists

nodes in ways

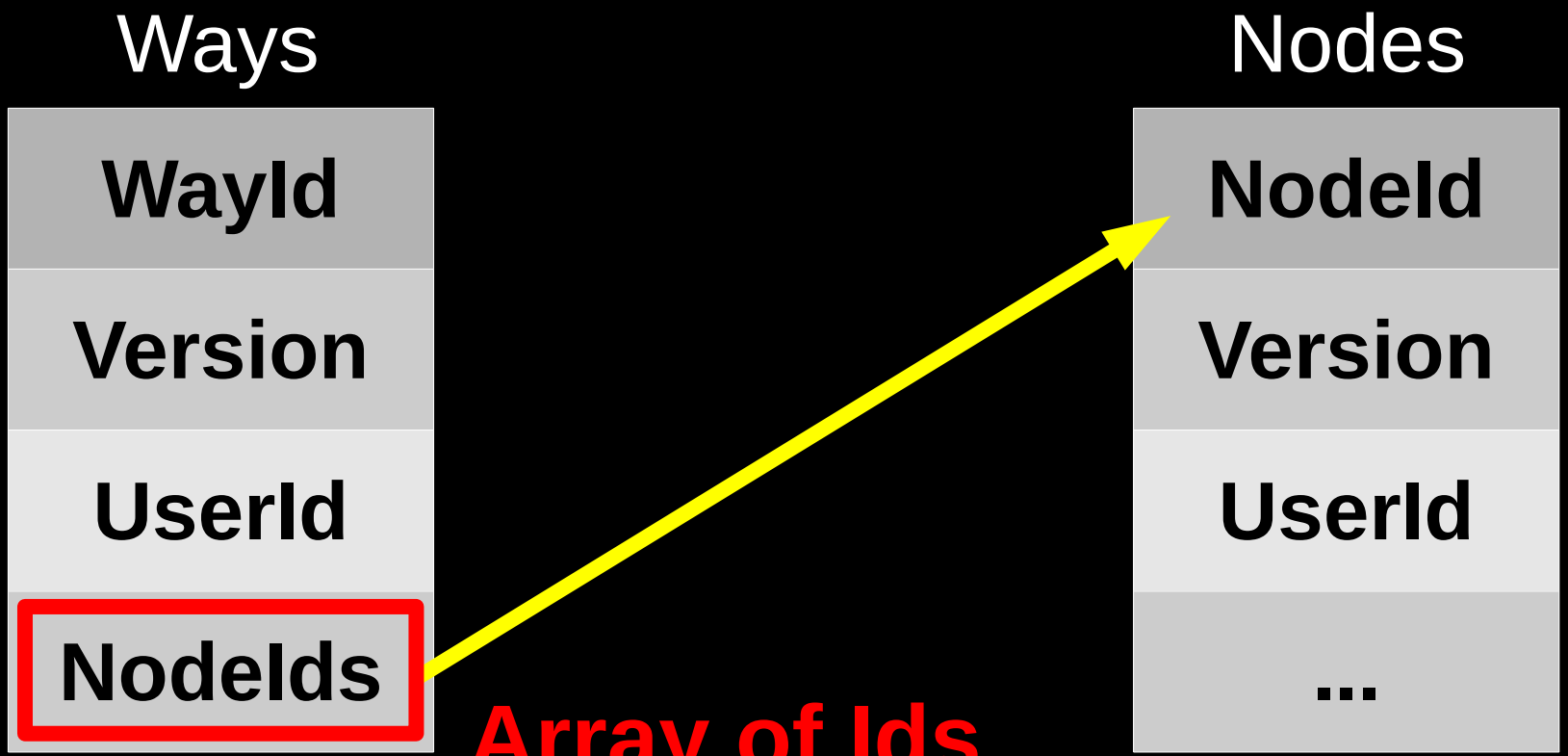
members in relations

tags in nodes, ways, or relations

# Conversion: Way Nodes



# Conversion: Way Nodes



# Conversion: Relation Members

similar to way nodes

but

array of tuple (type, id, role)

# Conversion: Geometry

Nodes → Points

Ways → LineStrings / Polygons

Multipolygon relations → Polygons

Route relations → MultiLineStrings

...

# Conversion: Geometry

Generalized geometries

For lower zoom levels / small scales

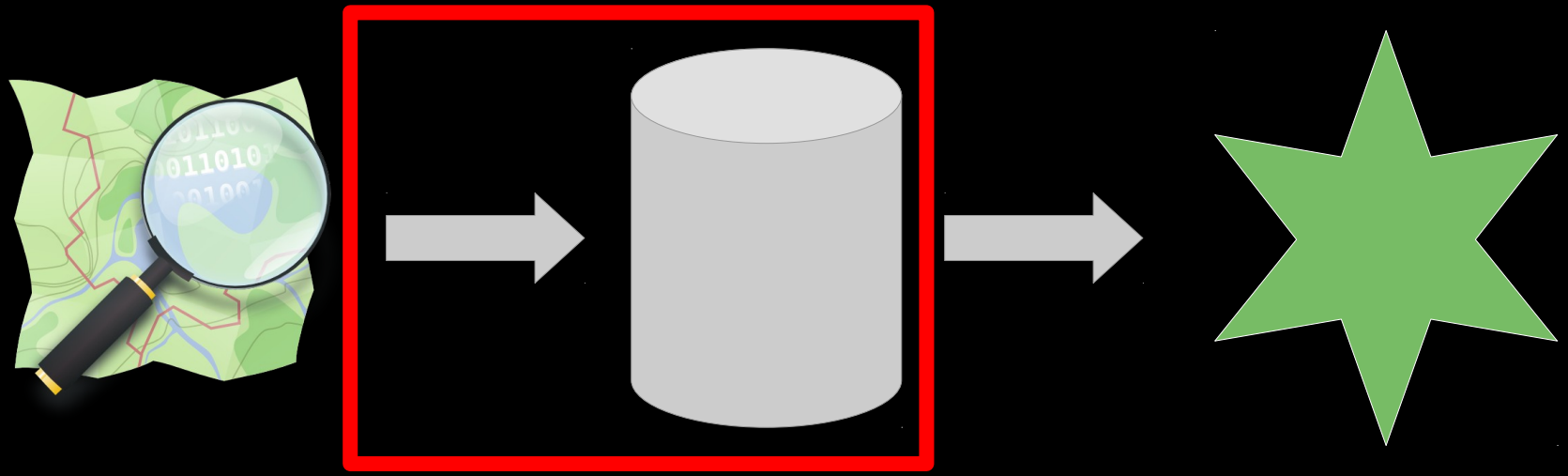
Selection – Merging – Simplification

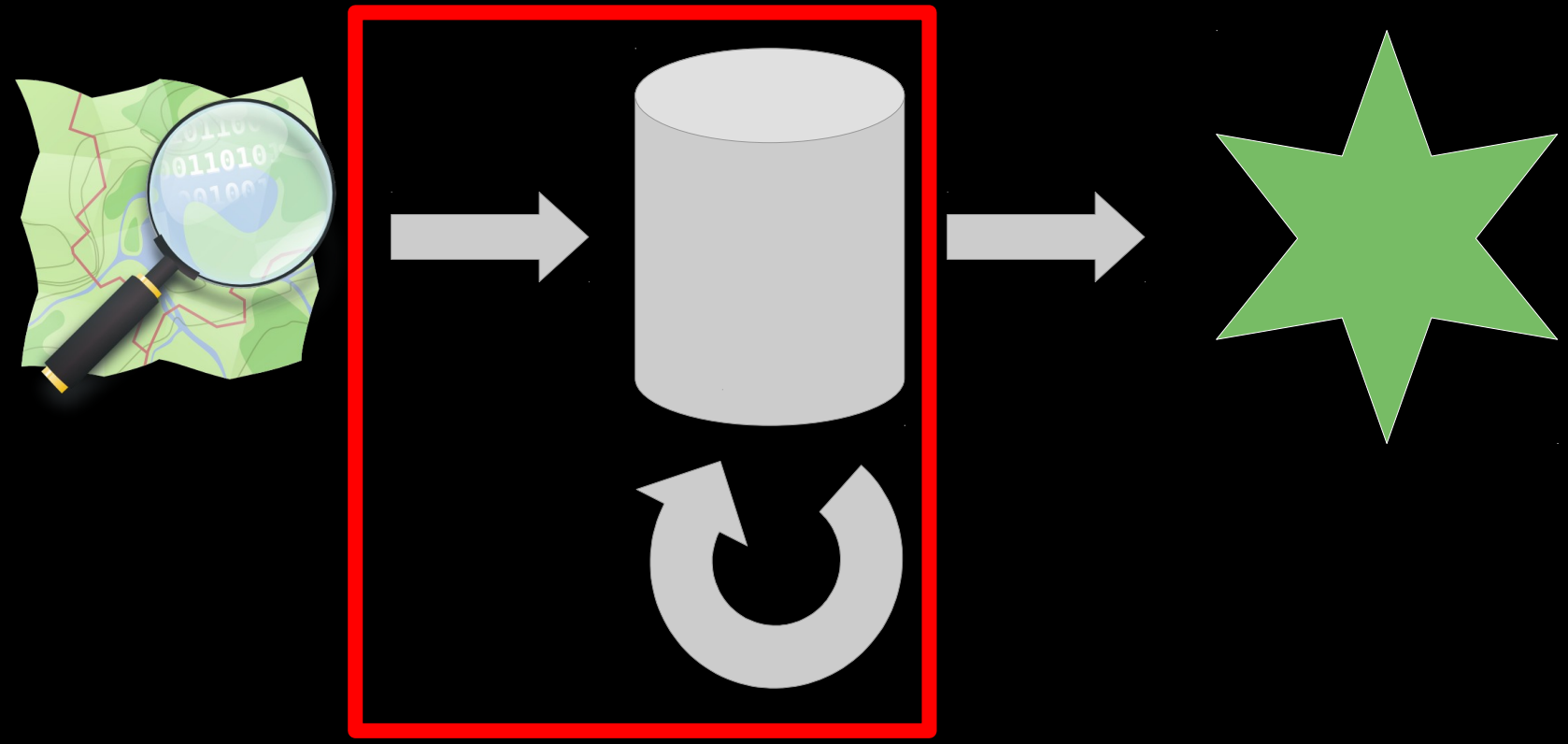
# Conversion: Where?

Conversion can happen

1. before import in code

2. after import in the DB





# Conversion: Where?

Conversion can happen

1. before import in code

2. after import in the DB

# Conversion: Where?

Conversion can happen

1. before import in code

2. after import in the DB

**fast**

**flexible**

# Conversion: Assemble Lines

Take Locations from Nodes

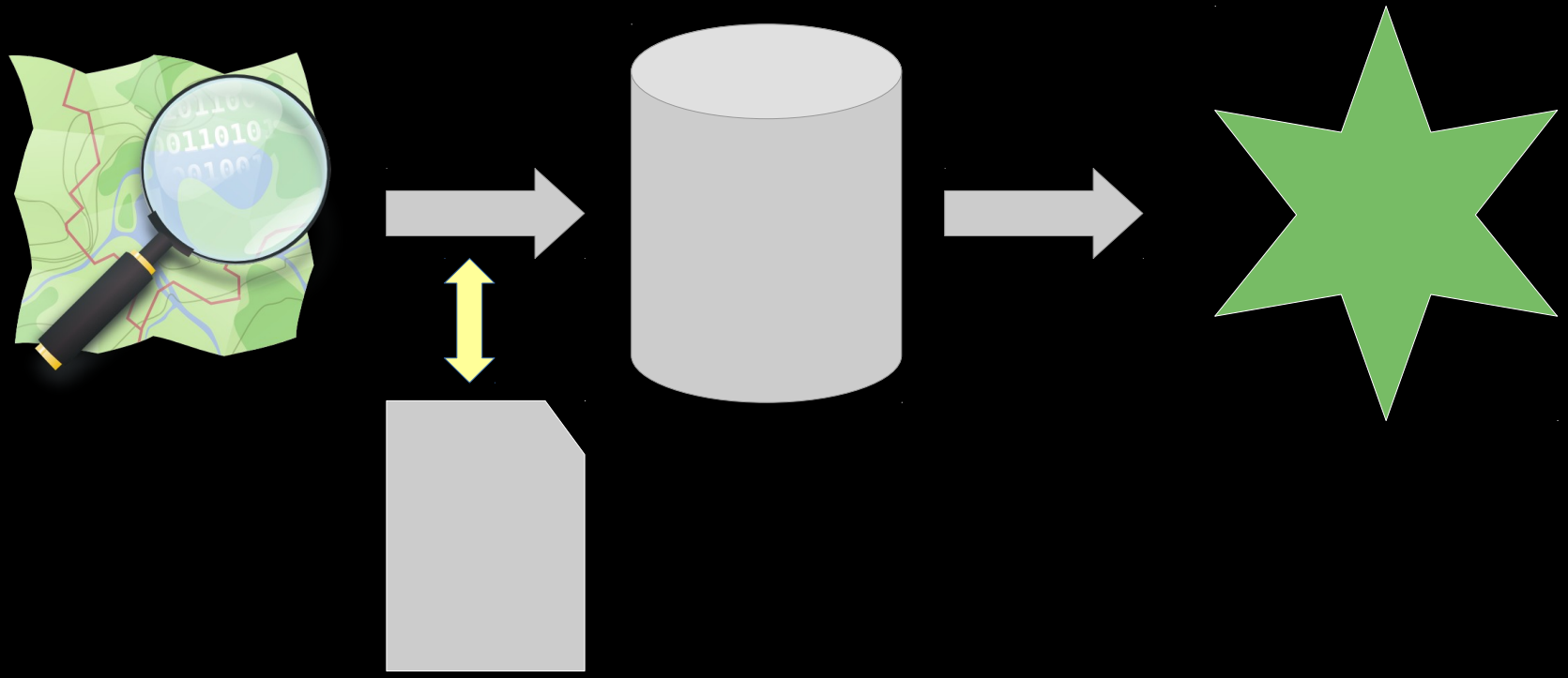
Assemble them into LineStrings

# Conversion: Node locations

Where to store node locations?

1. in the database

2. in specialized index



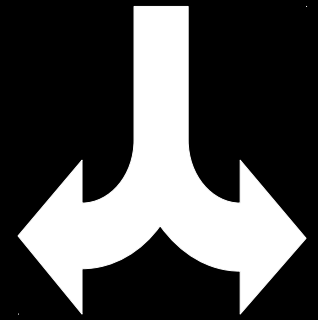
Node Location Store

# Conversion: Polygons

Assemble (Multi)Polygons

from Ways/Relations

**One-off Import**

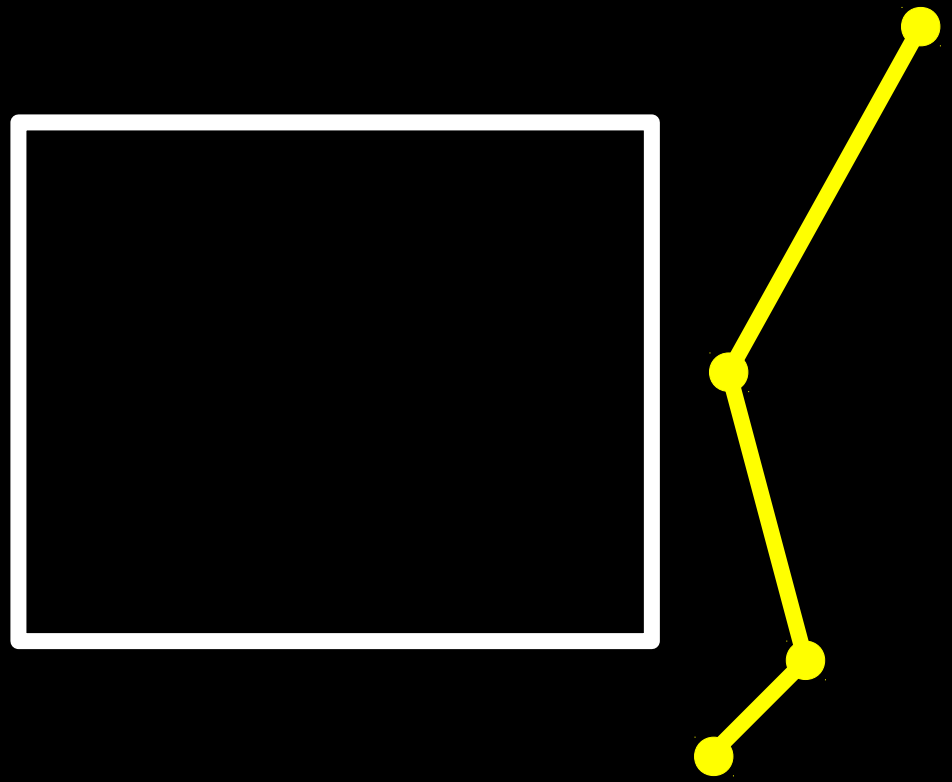


**Import + Updates**

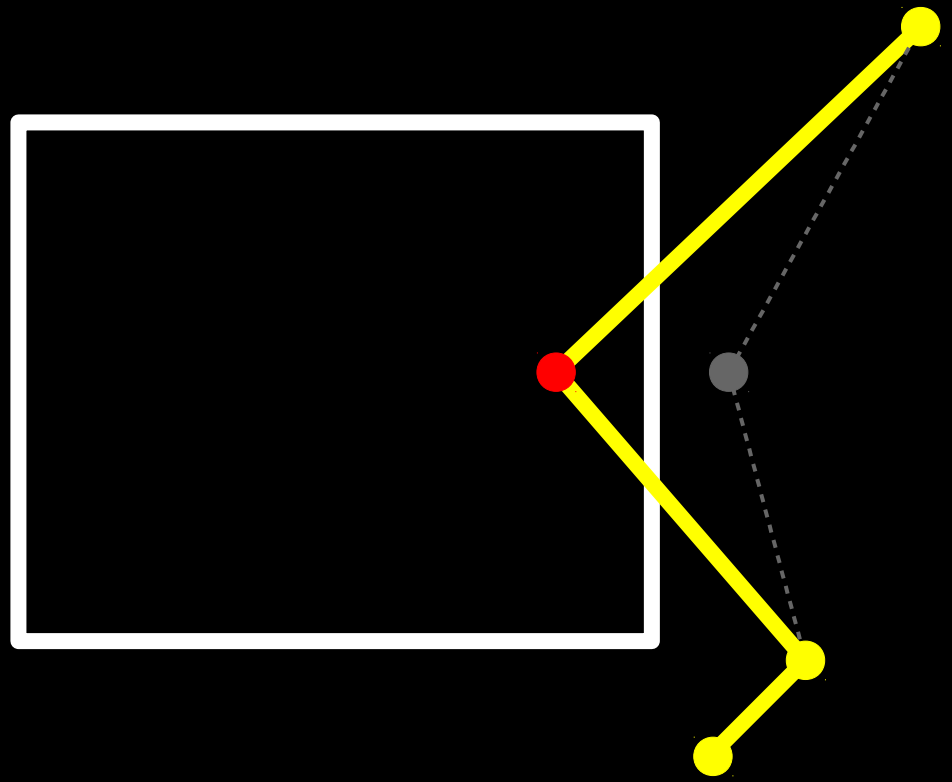
lots of trade-offs

not all software/schemas support updates

# Complete data needed for updates



# Complete data needed for updates



# Complete data needed for updates

Two kinds of data:

1. The data you need for you application
2. The data needed to allow updating

# Complete data needed for updates

Two kinds of data:

1. The data you need for you application
2. The data needed to allow updating

Where? Database? External Storage?

# Snapshot vs. History

most use cases only need
current OSM data

some need
history of OSM data

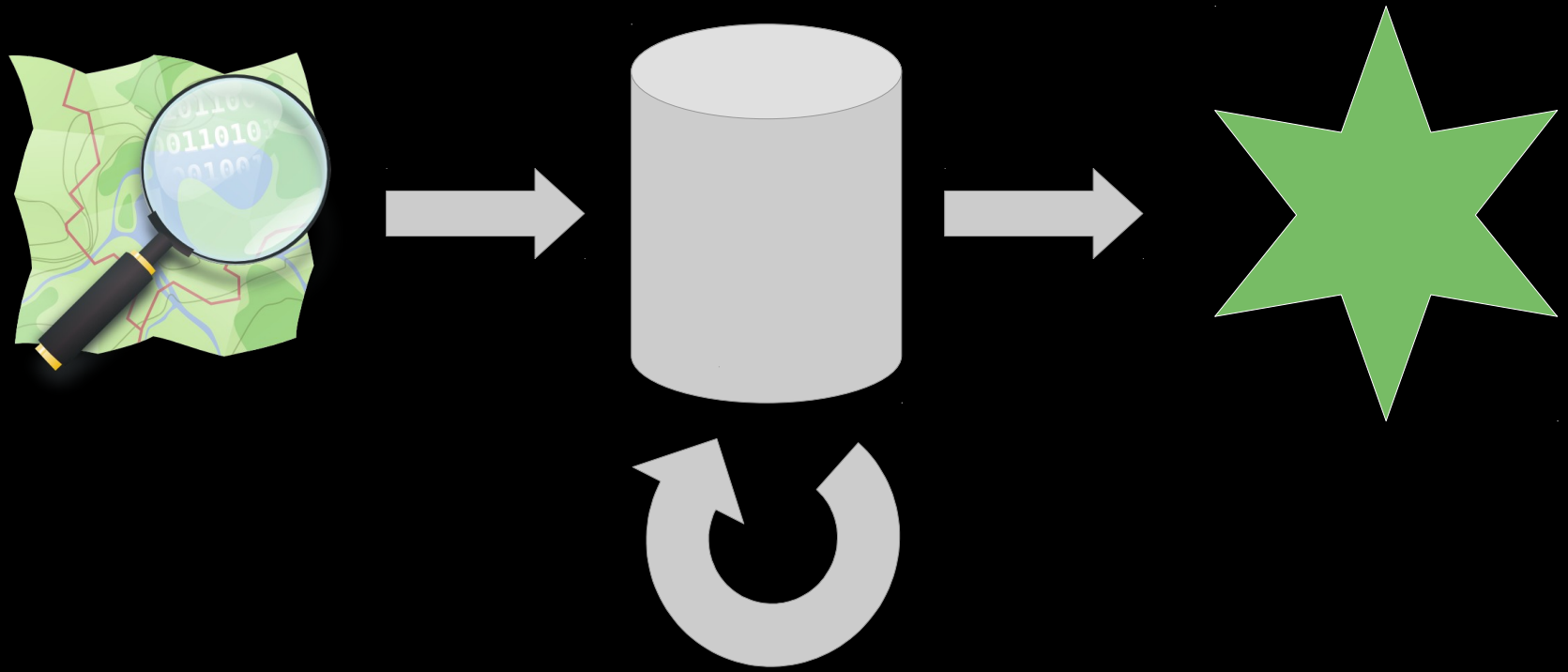
# Snapshot vs. History

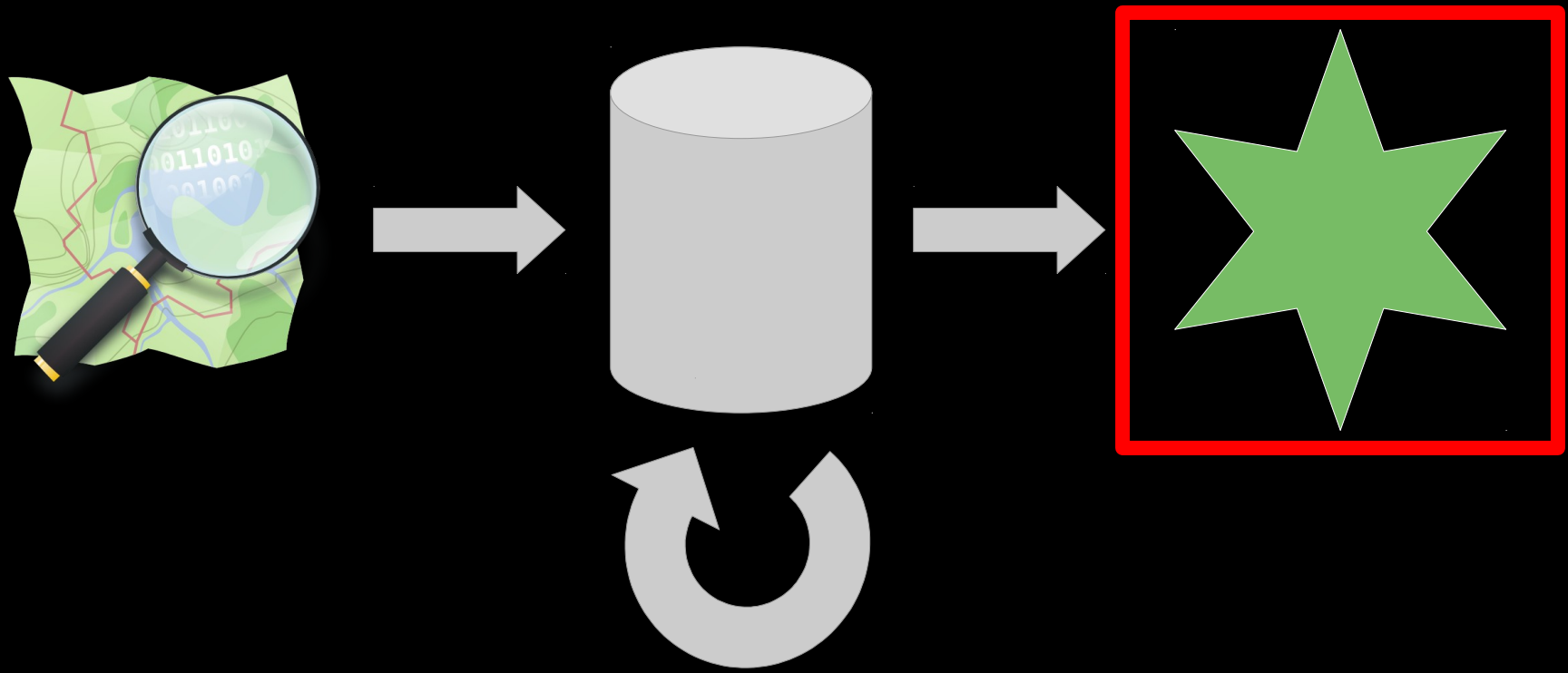
most use cases only need
current OSM data

some need
history of OSM data

**Much more effort needed !**

# Use Cases





# Use Cases

API DB

Rendering

Geocoding

Routing

Analytics

# API DB

Schema used in the main OSM database

PostgreSQL - No PostGIS !

Normal access via HTTP API

You can run your own

# API DB

Needs all (also historical) data

Multiple writers, transactions

Allow bounding-box download

Allow read/write access

Create full dumps and replication diffs

# Rendering

Turning data into maps

Render into bitmap, vector tiles, etc.

# Rendering

Get all data for an area quickly

Multiple layers

Create generalized geometries

One writer, multiple reader

Regular updates

# Geocoding

“Search“

Geocoding – Address to Location

Reverse Geocoding – Location to Address

# Geocoding

Build address hierarchy

Quick “fuzzy“ search

One writer, multiple reader

Regular updates

# Routing

Using PostgreSQL plugin PgRouting

Flexible, but slow

# Routing

Build network of streets

Calculate weights

Find route through network

# Analytics

Statistics

Comparing data

Conflating data

Many diverse needs

# Example 1: Wind Power

Find a place that …

… has lots of steady winds
… is near existing high voltage lines
… is far from residential areas

# Example 2: Public Transport

How far is the nearest public transport stop?

How many people live where the nearest stop is more than x meters away?

Where should a new bus route go?

# Example 3: OSM Contributors

Who are the most active OSM contributors?

What kinds of things do they map?

Where do they do their mapping?

# Analytics

Flexible data model

Use of many geometric operations

Batch processing in multiple steps

# Tools

# psql

File Edit View Terminal Tabs Help

```
sqrt=# select osm_id, osm_type, highway, substr(ST_AsText(geom), 1, 46) as geom
from highway_motorway limit 10;
 osm_id  | osm_type | highway  |                      geom
---------+----------+----------+-----------------------------------------------
 3700602 | way      | motorway | LINESTRING(12.3957525 51.4045029,12.3940079 51
 4068262 | way      | motorway | LINESTRING(12.1950993 51.4221162,12.195996 51.
 4068272 | way      | motorway | LINESTRING(12.1956903 51.4236168,12.194645 51.
 4068276 | way      | motorway | LINESTRING(12.2079569 51.426431,12.2111331 51.
 4071131 | way      | motorway | LINESTRING(12.375885 51.4084059,12.3754508 51.
 4071132 | way      | motorway | LINESTRING(12.355407 51.4120578,12.353102 51.4
 4071145 | way      | motorway | LINESTRING(12.2989367 51.4155116,12.2967172 51
 4071159 | way      | motorway | LINESTRING(12.1805927 51.3958729,12.1805298 51
 4100106 | way      | motorway | LINESTRING(12.1999562 51.4310166,12.1979877 51
 4259290 | way      | motorway | LINESTRING(12.5483554 51.3050722,12.5492398 51
(10 rows)

sqrt=#
```

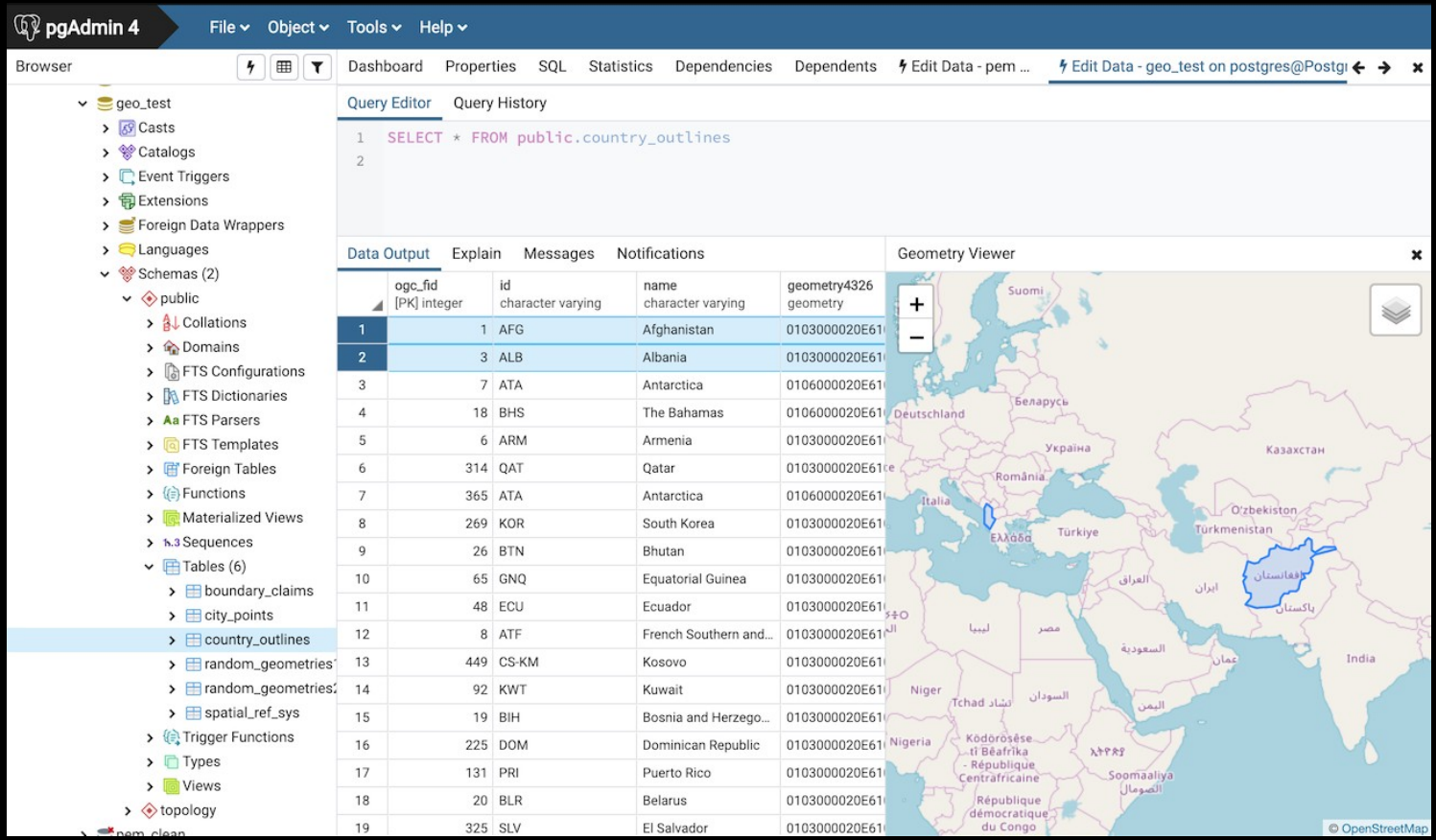
# psql

File Edit View Terminal Tabs Help

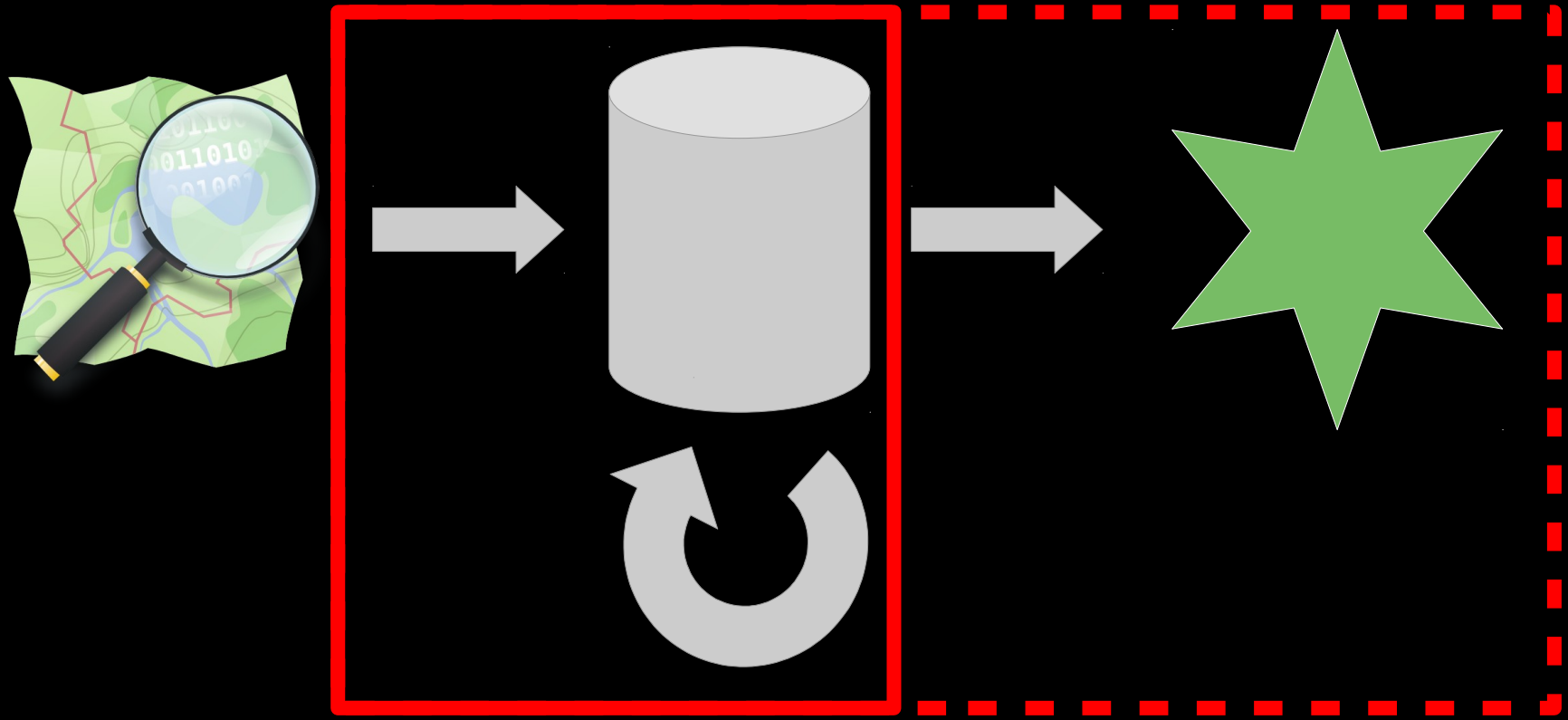
```
sqrt=# select osm_id, osm_type, highway, substr(ST_AsText(geom), 1, 46) as geom
from highway_motorway limit 10;
 osm_id  | osm_type | highway  |                      geom
---------+----------+----------+-----------------------------------------------
 3700602 | way      | motorway | LINESTRING(12.3957525 51.4045029,12.3940079 51
 4068262 | way      | motorway | LINESTRING(12.1950993 51.4221162,12.195996 51.
 4068272 | way      | motorway | LINESTRING(12.1956903 51.4236168,12.194645 51.
 4068276 | way      | motorway | LINESTRING(12.2079569 51.426431,12.2111331 51.
 4071131 | way      | motorway | LINESTRING(12.375885 51.4084059,12.3754508 51.
 4071132 | way      | motorway | LINESTRING(12.355407 51.4120578,12.353102 51.4
 4071145 | way      | motorway | LINESTRING(12.2989367 51.4155116,12.2967172 51
 4071159 | way      | motorway | LINESTRING(12.1805927 51.3958729,12.1805298 51
 4100106 | way      | motorway | LINESTRING(12.1999562 51.4310166,12.1979877 51
 4259290 | way      | motorway | LINESTRING(12.5483554 51.3050722,12.5492398 51
(10 rows)

sqrt=#
```

Editor

# pgadmin


pgAdmin 4
File
Object
Tools
Help
Browser
Dashboard
Properties
SQL
Statistics
Dependencies
Dependents
Edit Data - pem ...
Edit Data - geo_test on postgres@Postgr
Query Editor
Query History
1 SELECT * FROM public.country_outlines
2
Data Output
Explain
Messages
Notifications
Geometry Viewer
geo_test
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Schemas (2)
public
Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized Views
Sequences
Tables (6)
boundary_claims
city_points
country_outlines
random_geometries1
random_geometries2
spatial_ref_sys
Trigger Functions
Types
Views
topology
ogc_fid [PK] integer
id character varying
name character varying
geometry4326 geometry
1 1 AFG Afghanistan 0103000020E61
2 3 ALB Albania 0103000020E61
3 7 ATA Antarctica 0106000020E61
4 18 BHS The Bahamas 0106000020E61
5 6 ARM Armenia 0103000020E61
6 314 QAT Qatar 0103000020E61
7 365 ATA Antarctica 0106000020E61
8 269 KOR South Korea 0103000020E61
9 26 BTN Bhutan 0103000020E61
10 65 GNQ Equatorial Guinea 0103000020E61
11 48 ECU Ecuador 0103000020E61
12 8 ATF French Southern and... 0103000020E61
13 449 CS-KM Kosovo 0103000020E61
14 92 KWT Kuwait 0103000020E61
15 19 BIH Bosnia and Herzego... 0103000020E61
16 225 DOM Dominican Republic 0103000020E61
17 131 PRI Puerto Rico 0103000020E61
18 20 BLR Belarus 0103000020E61
19 325 SLV El Salvador 0103000020E61
Suomi
Беларусь
Deutschland
Україна
Казахстан
România
Italia
Ελλάδα
Türkiye
O'zbekiston
Türkmenistan
افغانستان
پاکستان
ایران
العراق
ليبيا
مصر
السعودية
عمان
India
Niger
Tchad
السودان
اليمن
Nigeria
Soomaaliya
الصومال
République démocratique du Congo
© OpenStreetMap


QGIS



# Osmosis

https://wiki.osm.org/wiki/Osmosis

Use case: API DB, Analytics
Updates: Yes
Schema: Several
Status: Not being maintained

# Osmosis Schemas

API DB (version 0.6)

PostGIS Snapshot Schema (uses hstore)

PostGIS Simple Schema (no hstore)

(API DB MySQL <0.6)

# osm2pgsql

Used in “standard“ OSM rendering toolchain

https://wiki.osm.org/wiki/Osm2pgsql

Use case: Rendering
Updates: Yes
Schema: Few tables (hstore optional)
Status: Maintained

# Imposm3

Alternative rendering toolchain

https://imposm.org

Use case: Rendering
Updates: Yes
Schema: Many tables
Status: Actively maintained

# Nominatim

Standard OSM search/geocoding

https://nominatim.org

Use case: (Reverse) Geocoding
Schema: Optimized for geocoding
Status: Actively maintained

Uses osm2pgsql (with special plugin)

# Osmium

https://osmcode.org/osmium-tool/

Use case: Analytics, (Rendering)

Updates: No
Schema: Simple
Status: Actively maintained
Simple to run for ad-hoc use

# osm-postgresql-experiments

Experimental, very flexible data import

https://github.com/osmcode/osm-postgresql-experiments

Use case: Rendering, Analytics
Updates: (Yes)
Schema: Flexible
Status: Experimental

# osm2pgrouting

Importer für PgRouting

https://github.com/pgrouting/osm2pgrouting

Use case: Routing
Updates: No
Schema: PgRouting
Status: Maintained

# Tips & Tricks

# Odds & Ends

# Quantity Structure

How much disk space do I need?

How much memory do I need?

How long will an import take?

# Quantity Structure

hundreds of Gbytes
for full planet

How much disk space do I need?

How much memory do I need?

How long will an import take?

# Quantity Structure

hundreds of Gbytes for full planet

How much disk space do I need?

How much memory do I need?

More!

How long will an import take?

# Quantity Structure

hundreds of Gbytes
for full planet

How much disk space do I need?

How much memory do I need?

More!

How long will an import take?

many hours
if not days for planet

# Start small...

Do not try to import the whole planet at first!

Start small
(e.g. with data for a city)
and
work your way up

# Minimize data

1. Filter data outside DB if you can

2. Import data into DB

# Importing Data

1. load data

2. create indexes

3. ANALYZE

# Performance Tuning

You will need to tune your PostgreSQL!

Settings in postgresql.conf:

shared_buffers, work_mem, maintenance_work_mem, fsync, synchronous_commit, checkpoint_timeout, checkpoint_completion_target, ...

# Indexes

Learn how indexes work and
when they are used

Also for spatial indexes!

Use EXPLAIN command

# The COPY command

COPY instead of INSERT

more efficient, use it if possible

# Learning Curve

PostgreSQL / PostGIS
is an incredible powerful tool

“Magic“ working of PostgreSQL
can be surprising

Configure the logs and look at them

# Learning Curve

Spatial operations are extra magic!

Again: Start small

# Always growing

Database will grow over time
(not only because of more OSM data)

VACUUM

Still grow more

# Other SQL Databases

MySql, MariaDB

Oracle Spatial

Sqlite (Library, not Server)

# Other SQL Databases

geodata support lacking

MySql, MariaDB

Oracle Spatial

Sqlite (Library, not Server)

# Other SQL Databases

geodata support lacking

MySql, MariaDB

Oracle Spatial

proprietary

Sqlite (Library, not Server)

# Other SQL Databases

geodata support lacking

MySql, MariaDB

proprietary

Oracle Spatial

Sqlite (Library, not Server)

not as powerful, problems with huge datasets,
but can be useful for some applications

https://www.floss-shop.de/de/floss-merchandise/stofftiere/40/postgresql-elefant

# THE END

Jochen Topf
jochentopf.com
jochen@topf.org