Managing Geo Data – Usage of Open Street Map for Own Services and Applications

---

Jörg Roth
Department of Computer Science
Univ. of Applied Sciences Nuremberg
Kesslerplatz 12, 90489 Nuremberg, Germany

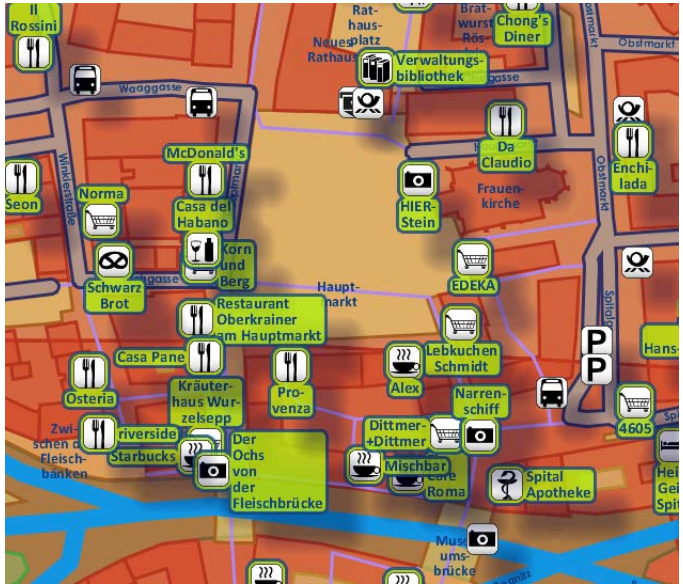
Why geo data?

Applications that consider the user's current location:

- Tour guides
- Route planning (car, tourist)
- Where is…?
- Community services, social networks

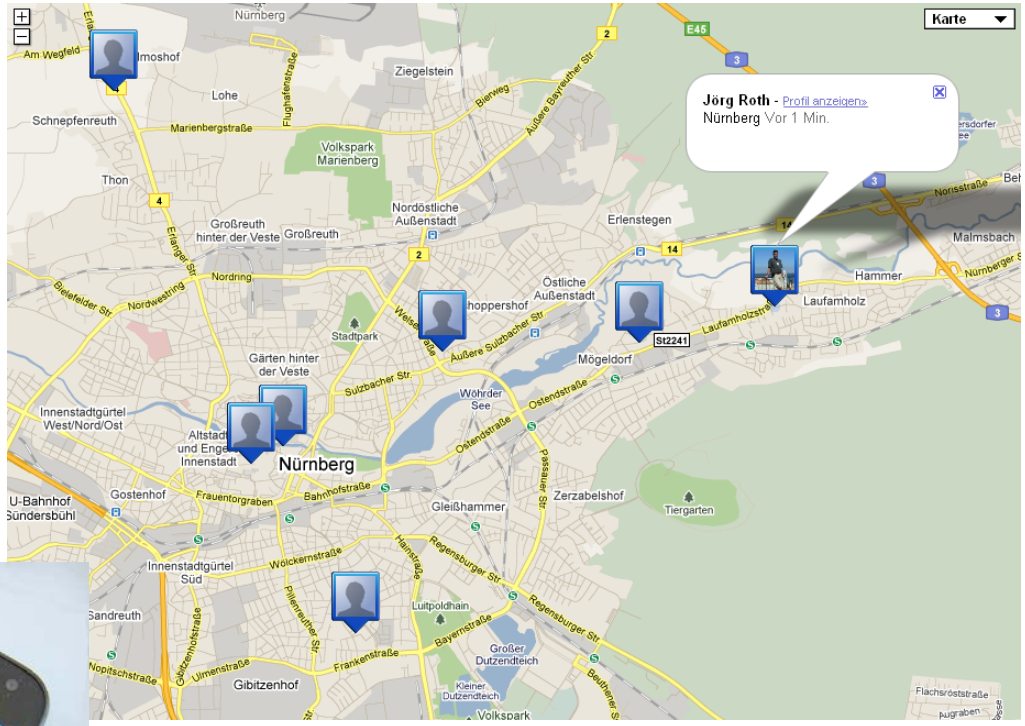
Not only end-consumers:

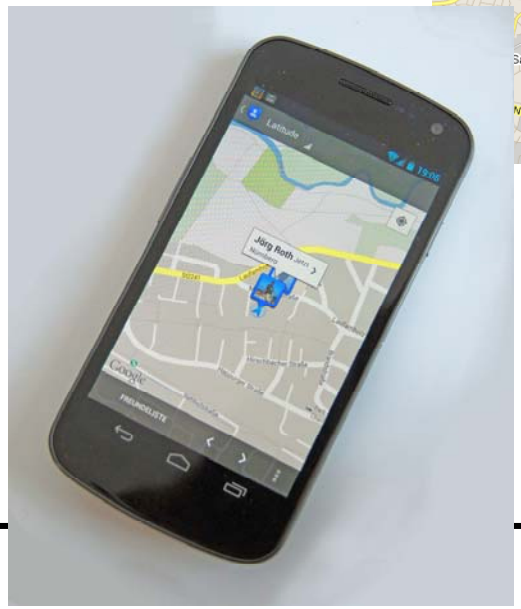
- Market research
- Logistics
- Traffic planning



Service platforms, e.g. *Google Maps*:

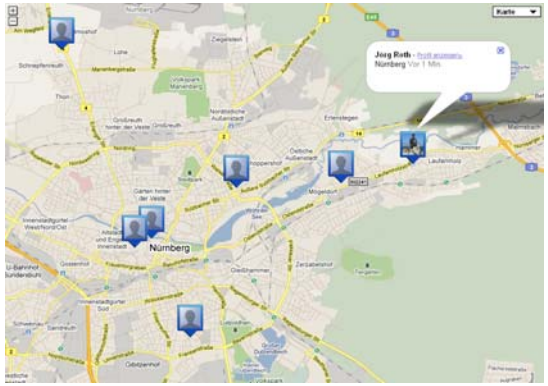
- Map display
- Routing
- Address resolution
- Friend finders
- Mobile support





Why not simply using such a service platform?

- Only services that are available can be used – no modifications possible
- Services can be withdrawn
- Costs, licenses
- Service availability (access via mobile networks often a problem)
- No control over geo data
  - Quality or coverage of geo data
  - Not possible to change or add own data

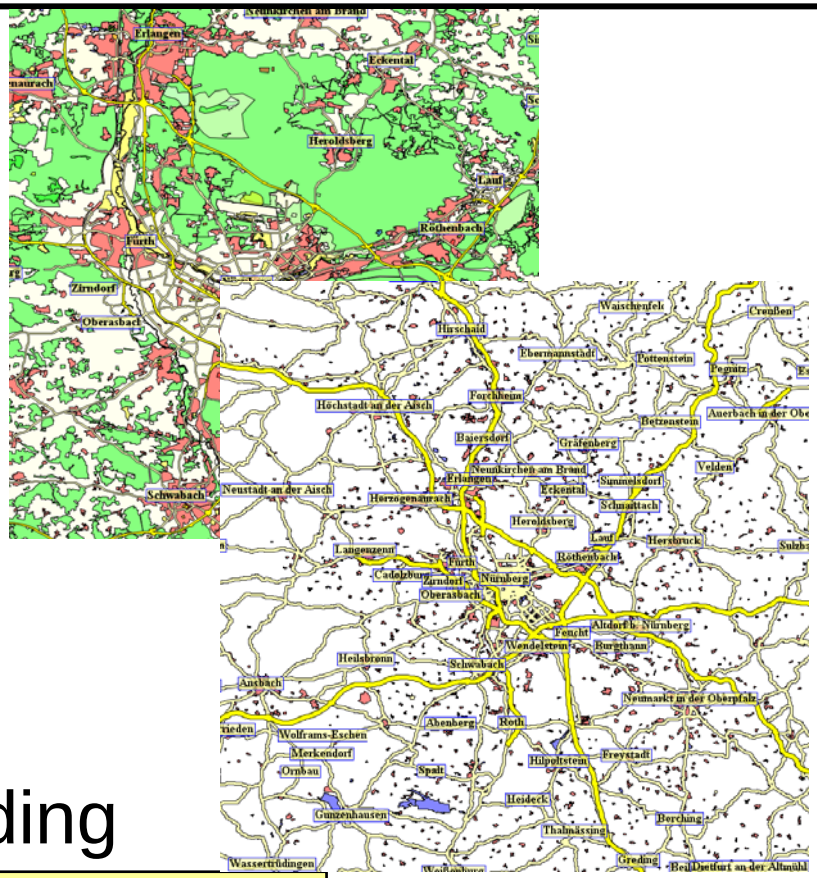


Sometimes, we do not want to rely on other services

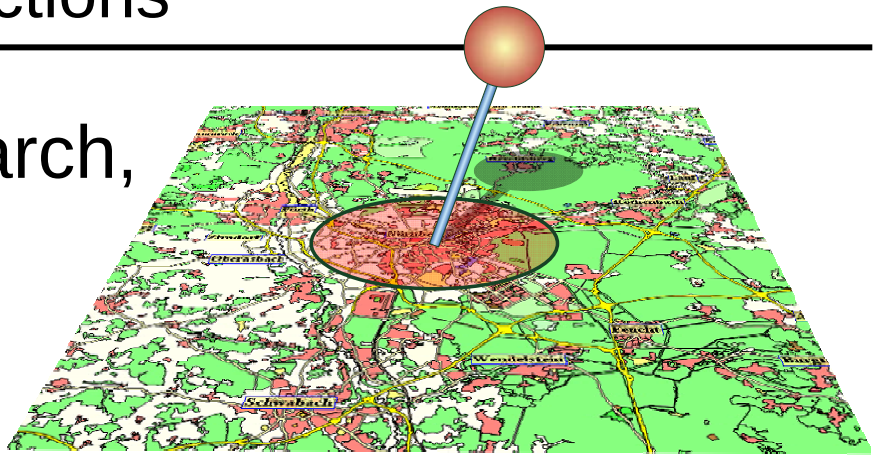
Typical functions

What to do with geo data?

- Paint maps
- Routing
  - shortest, fastest
  - car, pedestrian
  - also useful: train, bus (problem: timetables)
- Geocoding, reverse geocoding
  - *geocoding*: geo object → coordinate
  - *reverse geocoding*: coordinate → geo object (or → postal address)



Typical functions



- Radius search, nearby search, city search:
  - *Where is the nearest fuel station*?
  - *Where are hotels with a distance not exceeding 5 km*?
  - *Give me all parks in Lisbon*.
- Spatial join:
  - *Give me all open air baths in Bavaria that are close to a train station (nearer than 1km)*.
  - *How many cities in Europe have rivers running through them*?

Three major properties of geo data:

- Geometry: what is the shape and location of the object?
- Topology: how is an object related to other objects? (most important: street routing network)
- Thematic properties:
  - Object type, e.g. street, forest, lake, church, bus stop, bistro, tree, artwork…
  - Names
    (in different languages, for different purposes)
  - Further properties: opening hours, max. speed, tree species, restaurant type

We now focus on *vector* data

- Object geometries are points, line strings or polygons
- Also possible: Bezier curves or splines
  (usually not supported)
- Typically 2D (only plane) or 2.5D (height is an attribute), *not* full 3D

Open Street Map:

- A community project to collect, process and distribute world-wide geo data
- First ideas in 2004
- 2006: *Open Street Map Foundation*, operable infrastructure
- 2011: 1.2 billion points, 116 million ways, 1.1 million relations

Sources:

- Privately collected GPS tracks, entered, processed and classified by participating users
- Open geo databases, e.g. the TIGER databases
- Aerial images used to manually georeference objects
- Users can enter objects without any GPS measurement only in relation to existing objects

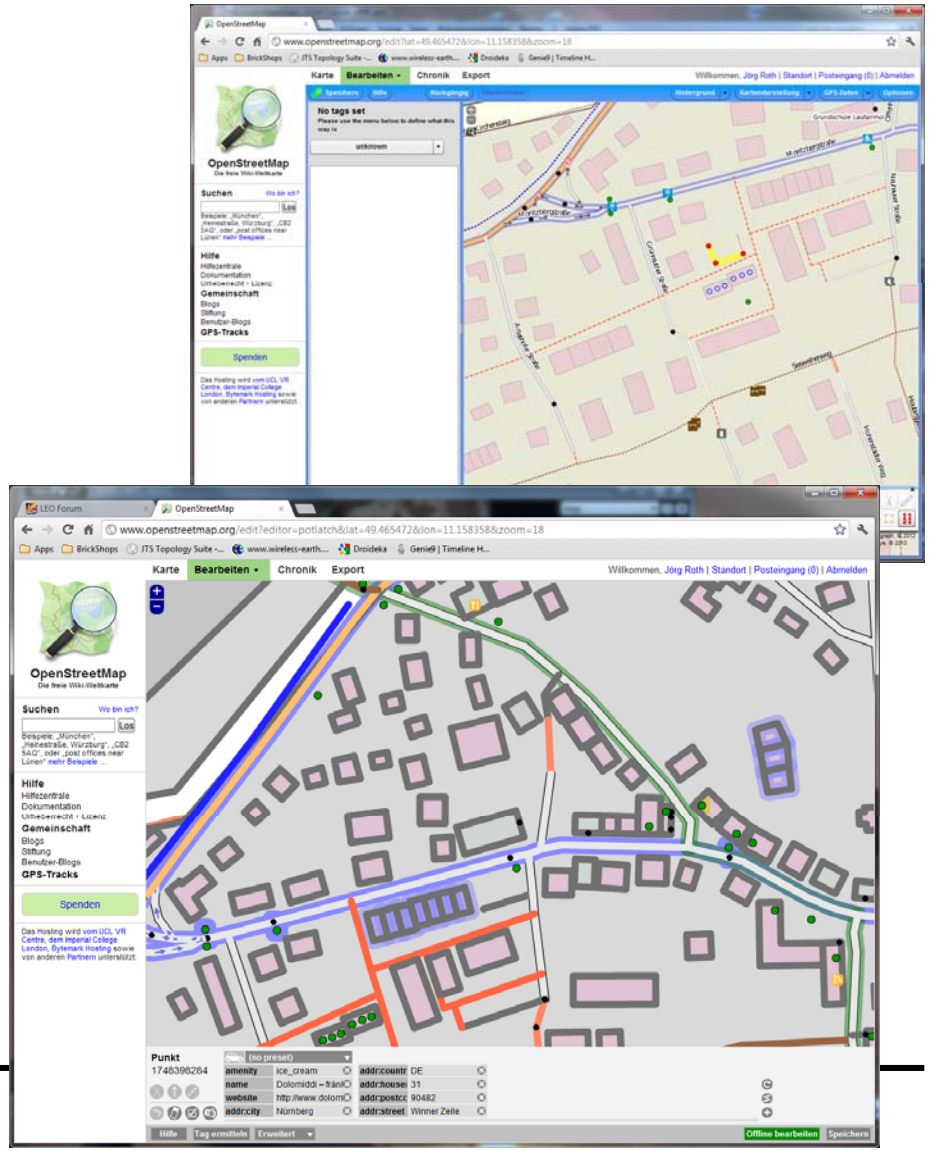
Open Street Map

Ways to add data to Open Street Map:

- Online editors
- Stand alone editors
- Registration required

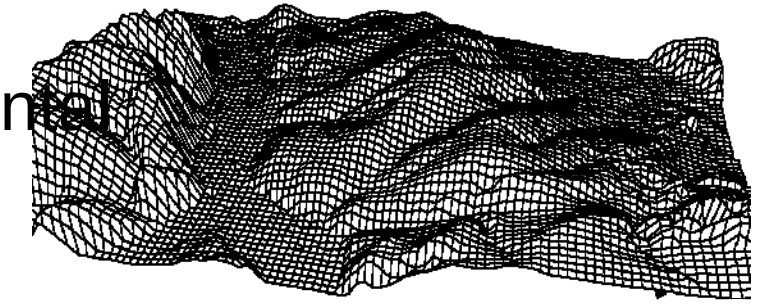
Online discussion (Wiki):

- Most important issues: How to classify objects? (see later)
- Also: workflow, process models



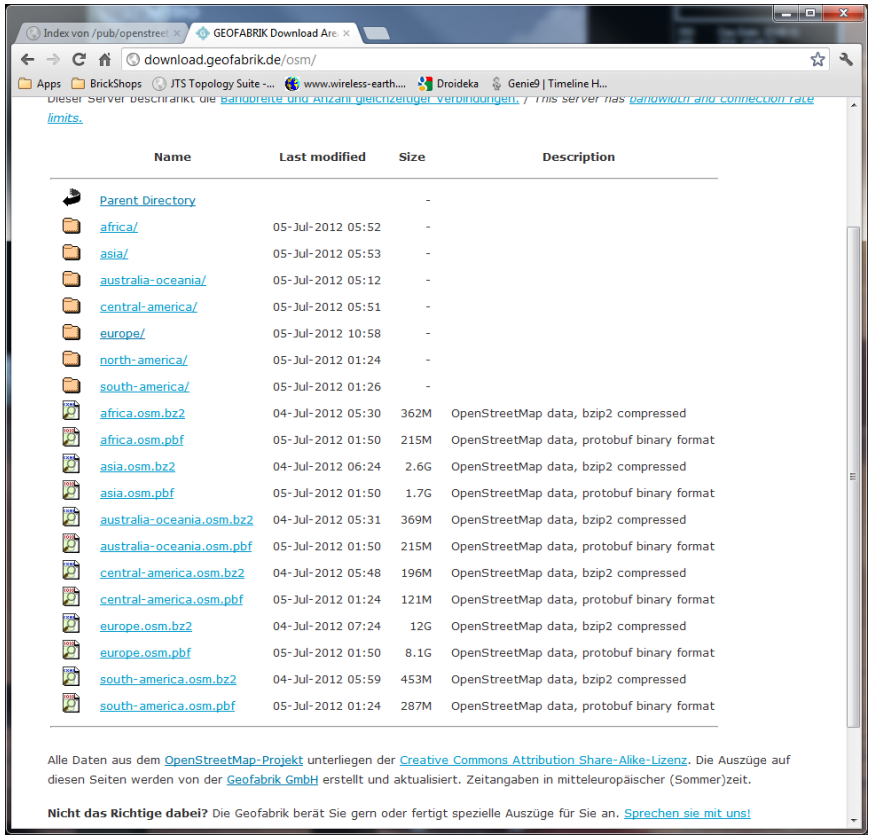
Available data:

- Most important: vector data in 2D
- Map bitmaps (many renderers, also 3rd party) – not the original domain of OSM
- Not or only partly supported:
  - Height profiles: only experimental
  - Street topologies: can be derived from geometries
  - Is-in relationship: only if users enter tags
  - Postal addresses: only if users enter tags



Ways to access vector data from Open Street Map:

- Online via HTTP – only small amounts of data
- *Planet files*:
  - Export of the entire OSM database as XML
  - Also parts in different granularities available: continents, countries, states, districts



OSM files:

- bzip2 compressed
  (zip not possible due to size limitations)
- unpacked: XML
- Main structure **<osm>…</osm>**
- Inside: three types of entries (in this order)
  - Nodes: **<node>…</node>**:
    Point objects and line points
  - Ways: **<way>…</way>**:
    Objects with line string or area geometries
  - Relations: **<relation>…</relation>**:
    Objects that are built up by other objects

Example: Germany.osm (May 2012)

bzip2 file size	2 GB
XML file size	22 GB
XML tags total	318,532,216
nodes	100,475,499
ways	14,996,507
relations	234,273

OSM XML file (nodes)

```
<?xml version='1.0' encoding='UTF-8'?>
<osm version="0.6" generator="pbf2osm">
<node id="1" lat="51.2492152" lon="9.4317166" version="6"
     user="elllit" uid="24852"
     timestamp="2011-08-16T11:26:47Z"/>
<node id="10" lat="51.3806531" lon="9.3599172"
     version="5" user="max60watt" uid="134914"
     timestamp="2011-04-26T20:50:36Z"/>
<node id="12" lat="51.3400316" lon="9.4819956"
     version="2" user="max60watt" uid="134914"
     timestamp="2011-04-28T21:39:02Z"/>
<node id="13" lat="51.3731042" lon="9.5130058"
     version="2" user="max60watt" uid="134914"
     timestamp="2011-05-08T22:06:06Z">
       <tag k="highway" v="bus_stop" />
       <tag k="name" v="Bleichplatz" />
       <tag k="shelter" v="yes" />
</node>
...
```

OSM XML file (ways)

```
<way id="3591699" version="2" user="Bube"
   timestamp="2009-06-13T07:45:37Z">
     <nd ref="17410365"/>
     <nd ref="17410355"/>
     <tag k="created_by" v="JOSM" />
     <tag k="highway" v="track" />
     <tag k="tracktype" v="grade2" />
</way>
<way id="3593390" version="8" user="kanu_guenni"
   timestamp="2010-03-31T18:19:41Z">
     <nd ref="14539664"/>
     <nd ref="14556238"/>
     ...
     <nd ref="14539666"/>
     <tag k="bicycle" v="official" />
     <tag k="foot" v="official" />
     <tag k="highway" v="path" />
</way>
...
```

OSM XML file (relations)

```
<relation id="330" version="6" uid="161619" user="FvGordon"
   timestamp="2012-02-24T23:13:40Z">
     <member type="way" ref="49022711" role="inner"/>
     <member type="way" ref="24808645" role="outer"/>
     <tag k="area" v="yes" />
     <tag k="highway" v="pedestrian" />
     <tag k="name" v="Martin-Luther-Platz" />
     <tag k="type" v="multipolygon" />
</relation>
<relation id="2235" version="6" uid="39381" user="DD1GJ"
   timestamp="2009-11-22T08:15:27Z">
     <member type="way" ref="4917826" role=""/>
     ...
     <member type="way" ref="7942741" role=""/>
     <tag k="name" v="Lichtentaler Allee" />
     <tag k="route" v="road" />
     <tag k="type" v="route" />
</relation>
...
</osm>
```

OSM nodes

Nodes:

- Major property: latitude/longitude, *no* extent
- Two types (only implicitly defined):
  - Point-like objects (often called *Point of Interests*): Objects as such, e.g. shop, restaurant, traffic light, postbox, waste container
  - Part of a line string (i.e. only a coordinate)
- Point of Interests must have further properties to be useful
  - at least the object type

```
<node id="13" lat="51.3731042" lon="9.5130058"
     version="2" user="max60watt" uid="134914"
     timestamp="2011-05-08T22:06:06Z">
       <tag k="highway" v="bus_stop" />
       <tag k="name" v="Bleichplatz" />
       <tag k="shelter" v="yes" />
</node>
```

Tagging objects

Tags k, v:

- **k="…"** and **v="…"** are used to express non-geometric properties, e.g.

```
<tag k="highway" v="bus_stop" />
<tag k="name" v="Bleichplatz" />
<tag k="shelter" v="yes"
```

- Details of tagging see later
- For now: we write

  abc="xyz"

  instead of

  <tag k="abc" v="xyz" />

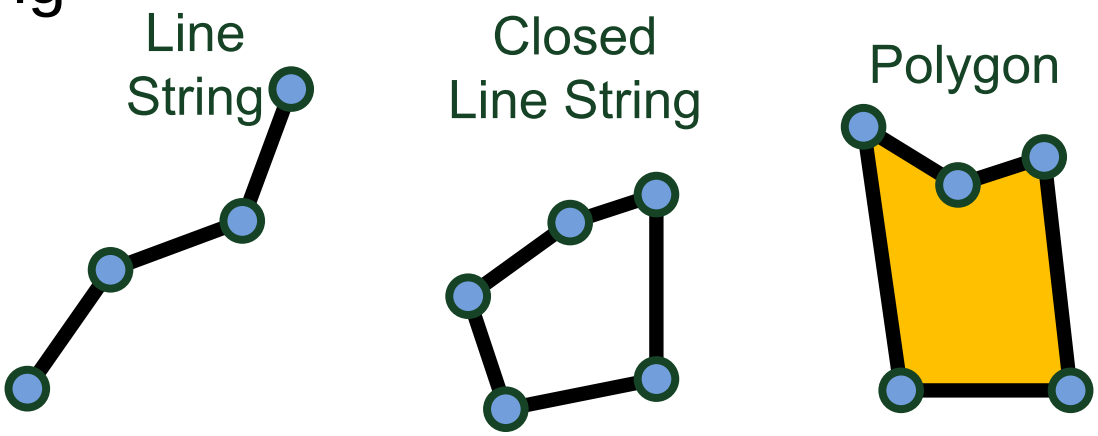
OSM ways

Ways:

- A sequence of nodes, referred by their ID

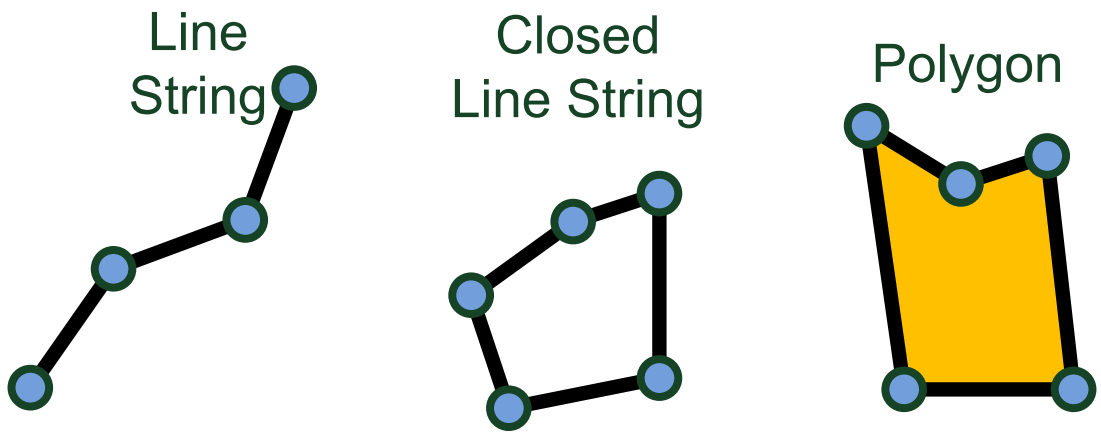
```
<way id="3591699" version="2" user="Bube"
   timestamp="2009-06-13T07:45:37Z">
     <nd ref="17410365"/>
     <nd ref="17410355"/>
     <tag k="created_by" v="JOSM" />
     <tag k="highway" v="track" />
     <tag k="tracktype" v="grade2" />
</way>
```

- Usually complete geo objects, i.e. *with* properties
- Three possible geometries:
  - (open) line string
  - closed line string
  - polygon



OSM ways

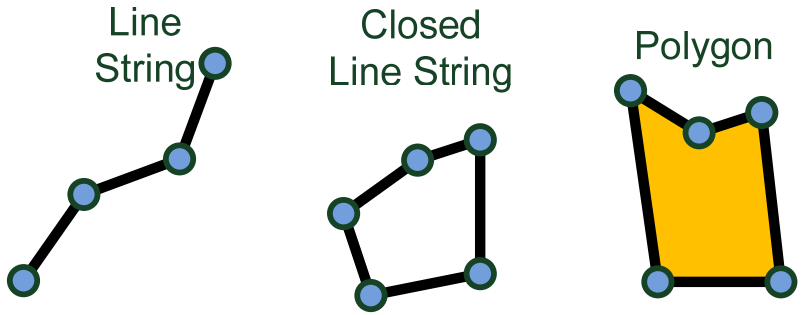
- Open line string: geo objects such as streets
- Closed line string: the actual object is the line string, *not* the content, example: roundabout
- Polygon: the actual object is the *content area*, not the border line. Examples: park, forest, lake



OSM ways

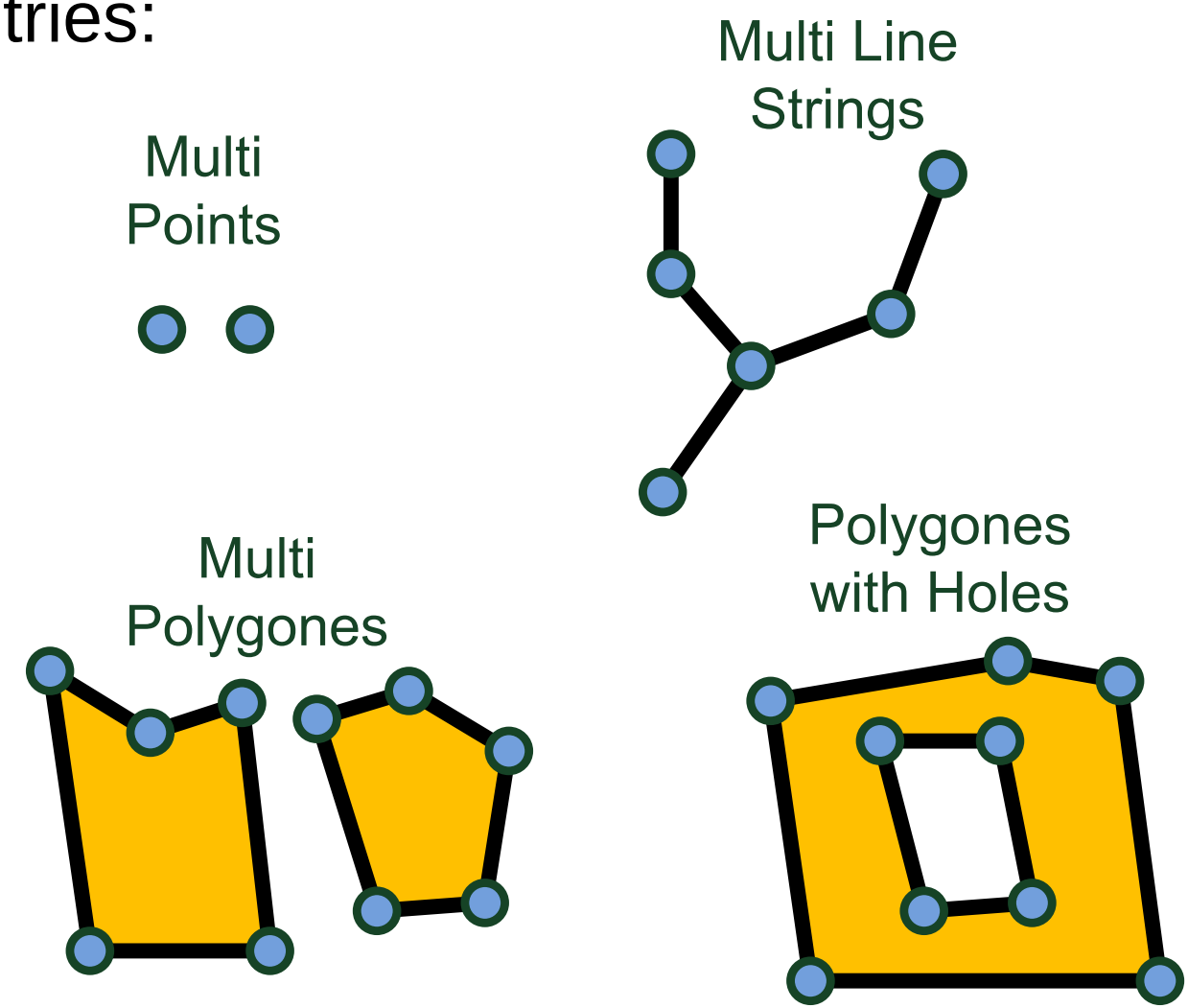
How to distinguish these geometries:

- Open line string: first and last node are not equal
- Closed line string vs. polygon (area):
  - no simple rule
  - sometimes distinguishable by their type
    - **`highway`** usually is closed line string
    - **`building`** usually is polygon
  - sometimes optional tag **`area="yes"`**



Further geometries

- OSM nodes and ways cannot express these geometries:



OSM relations

```
<relation id="330" version="6" uid="161619"
   user="FvGordon" timestamp="2012-02-24T23:13:40Z">
     <member type="way" ref="49022711" role="inner"/>
     <member type="way" ref="24808645" role="outer"/>
     <tag k="area" v="yes" />
     <tag k="highway" v="pedestrian" />
     <tag k="name" v="Martin-Luther-Platz" />
     <tag k="type" v="multipolygon" />
</relation>
```

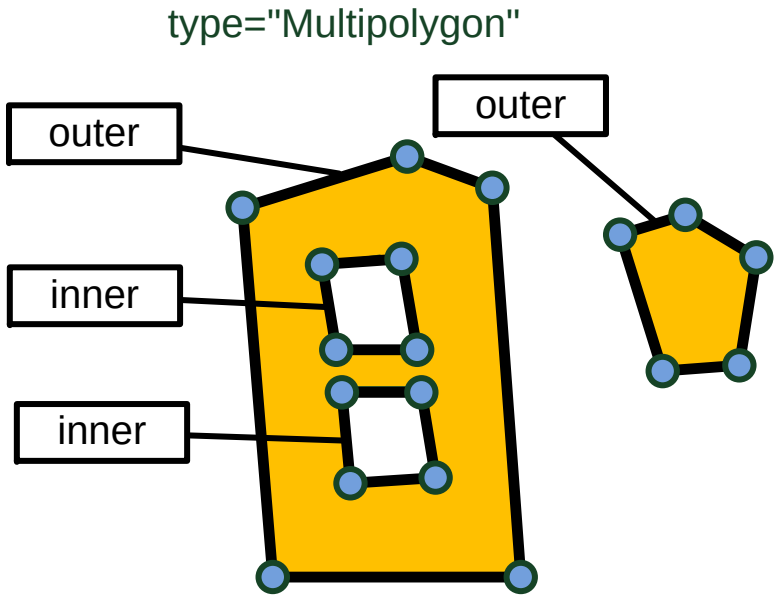
Relations:

- Increase the ability to express complex objects and complex geometries
- Types of relations:

Complex geometries	e.g. polygons with holes, multiple polygons
Objects that share parts to avoid redundancy	e.g. shared borders of two cities
Combine objects to 'bigger' objects	e.g. *'Hiking trail Frankonia to Baltic Sea*' that contains smaller hiking trails
Objects with piecewise defined properties	e.g. a highway with different speed limits (note: a way entry can only have a single set of properties)
Relationship between objects	e.g. a turn restriction between roads at a certain crossing

Multipolygon and polygons with holes:

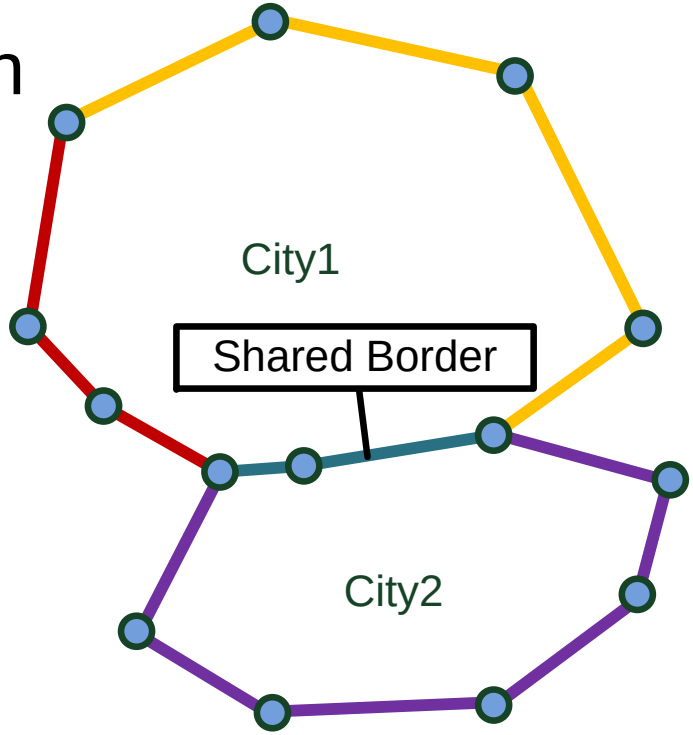
- Relations with tag **`type="multipolygon"`**
- Each member must be a closed linestring
- Each member should have **`role="outer"`** (a shell) or **`role="inner"`** (a hole)
- Unfortunately: no expression which hole belongs to which shell



OSM relations – complex geometries

Borders:

- Borders (e.g. city borders) are often a collection of lines, each of it representing a part of the border
- Reason: shared borders should only be stored once
- The problem: neither ordering nor orientation are specified in the member description
- Difficult to create a city *area* from borders



OSM object IDs:

- nodes, ways and relations have unique IDs
- They do not change over time, i.e. can be used to identify objects between imports
- Note: they are only unique inside a type (nodes, ways, relations)
  - node with ID 1 *and* way with ID 1 possible
  - to have unique object IDs for all types you have to artificially distinguish types, e.g.

$ownID = ID \cdot 3 + 0$	for nodes
$ownID = ID \cdot 3 + 1$	for ways
$ownID = ID \cdot 3 + 2$	for relations

Tagging objects

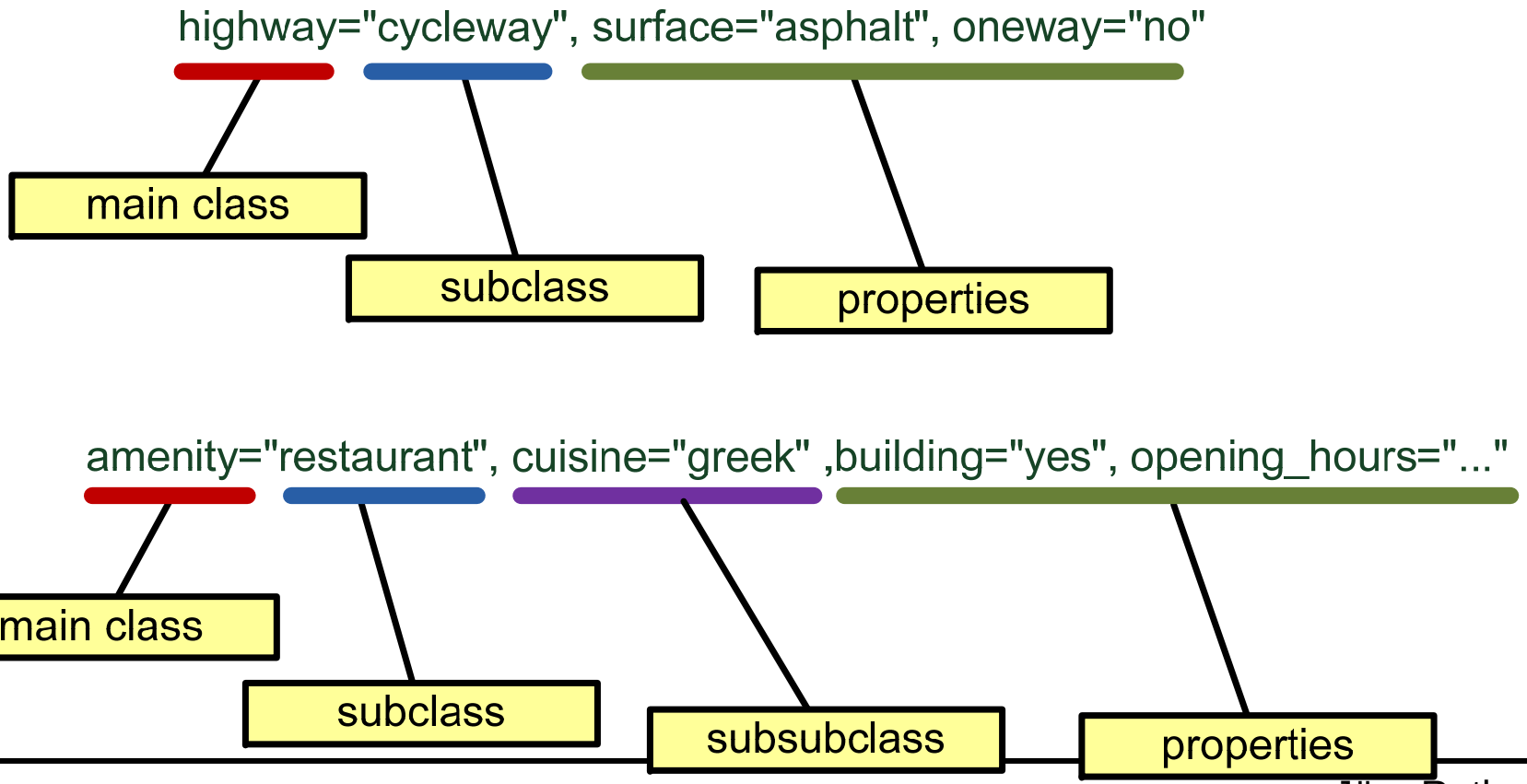
Tagging:

- Users can assign *arbitrary* pairs of key/value to objects
  - no limits for number of pairs
  - no obligatory keys, to superset of keys
  - all values allowed that can be expressed as string
- Only recommendations:
  - you may ask the editor for useful keys
  - no technical check, if entries are useful
- Currently, one problem for automatic classification

Tagging objects

However, there is some structure:

- usually there is a most important key/value
- unfortunately, not syntactically indicated



There are keys that define main classes

- Traffic

Main Class	Subclass example	Description
`highway="…"`	`highway="motorway"`	Streets, roads, paths
`railway="…"`	`railway="station"`	Train stuff
`cycleway="…"`	`cycleway="lane"`	Cycling
`waterway="…"`	`waterway="river"`	Water
`aeroway="…"`	`aeroway="terminal"`	Flying
`junction="…"`	`junction="roundabout"`	Junctions

OSM object classification

- Buildings, amenities, shops etc.

Main Class	Subclass example	Description
`amenity="…"`	`amenity="restaurant"`	Amenities
`shop="…"`	`shop="hairdresser"`	Shops
`craft="…"`	`craft="plumber"`	Craft
`office="…"`	`office="lawyer"`	Offices
`barrier="…"`	`barrier="bollard"`	Barriers, walls
`man_made="…"`	`man_made="water_tower"`	Special buildings
`power="…"`	`power="generator"`	Electricity
`military="…"`	`military="bunker"`	Military

OSM object classification

- Countryside, nature

Main Class	Subclass example	Description
`natural="…"`	`natural="rock"`	Anything natural
`landuse="…"`	`landuse="farm"`	Agriculture, farms
`geological="…"`	`geological="valley"`	Geological formations

- Non-physical objects

Main Class	Subclass example	Description
`route="…"`	`route="bus"`	Sequence of ways
`boundary="…"`	`boundary="postal_code"`	Boundaries, e.g. city
`place="…"`	`place="county"`	Captions, labels

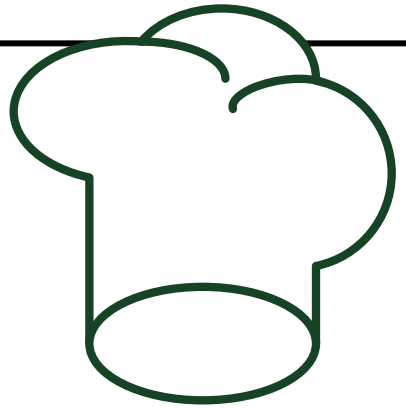
OSM object classification

- Leisure, tourism

Main Class	Subclass example	Description
`sport="…"`	`sport="golf"`	Places for sport
`leisure="…"`	`leisure="park"`	Places for leisure
`tourism="…"`	`tourism="hostel"`	Related to tourism
`historic="…"`	`historic="monument"`	Historic objects

- (list is not complete)

Further classifications



- For some classes, further classes are recommended, e.g. for

  `amenity="restaurant"`

  a further tag

  `cuisine="…"` (e.g. **`cuisine="bistro"`**)

  is expected
- The problem:
  - it is not clear, whether a tag is meant as classification (such as 'cuisine') or as property (such as 'opening hours')
  - rule-based analyses required to map classes to objects

Ambiguity:

- Three ways to express 'bistro'
  - **`shop="bistro"`**
  - **`amenity="bistro"`**
  - **`amenity="restaurant",cuisine="bistro"`**
- Two ways to express combined foot/cycleway
  - **`highway="cycleway", footway="yes"`**
  - **`highway="footway", cycleway="yes`**"
- To quickly distinguish object types (e.g. for routing), a simpler classification scheme is required in own databases

Tags to name objects:

Tag	Meaning	Example
`name="…"`	Standard name	*Germany*
`name:de="…"`	Name in different languages	*Deutschland*
`*_name="…"`	**`*`**$\in$ {**`int`**, **`nat`**, **`loc`**, **`reg`**} international, national, local or regional name	*Bayern*
`short_name="…"`	Common abbreviation	*UK*
`official_ name="…"`	Official name	*Principality of Andorra*

Further tags for naming:

Tag	Meaning	Example
`ref="…"`	Name of motorways, bus lines etc.	*A1, Route 66, 23*
`*_ref="…"`	**`*`**$\in$ {**`int`**, **`nat`**, **`loc`**, **`reg`**} international, national, local or regional name	*A1* (without blank), *A 1* (with blank)
`addr:housename="…"`	House name as part of address information	*Hotel 4 Seasons*

Defining postal addresses:

Tag	Meaning	Example
`addr: country="…"`	Country code as used in Internet names	*DE*
`addr: city="…"`	Local city name	*Nürnberg*
`addr: postcode="…"`	Postcode	*90489*
`addr: street="…"`	Street	*Kesslerplatz*
`addr: housenumber="…"`	House number	*12*
`addr: housename="…"`	House name	*Gebäude A*

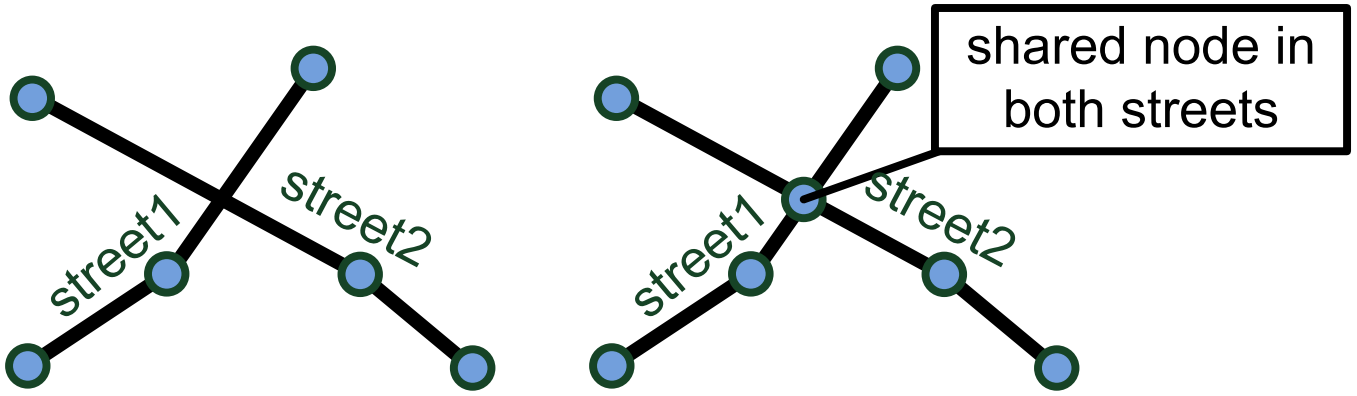
is_in relationship

Where is an object located semantically?

- In which city, state, country resides an object?
- Can in principle derived by
  - border geometries: area required, geometric check is time consuming
  - postal addresses: not every relationship is encoded in the postal address (e.g. suburbs, districts in town, villages)
- Object can have explicit **`is_in`** tags, e.g. **`is_in="Nuremberg"`**
  - controversially discussed
  - not consistently used

Street topologies:

- Not actually supported by OSM
- The street network can be derived from street geometries:
  - if streets are connected, they must have a shared node
  - if crossing streets that are *not* connected (bridges, tunnels), they *must not* have a shared node



Tags that affect route planning:

Tag	Meaning	Example
`oneway="yes"`	Driving only possible in the given direction	
`maxspeed="…"`	Speed limit	*50, 60 mph*
`highway="…"`	Street type	*residential, secondary, motorway*

In addition:

- Information about traffic signs, traffic lights
- Relations can identify turn restrictions

Public transportation

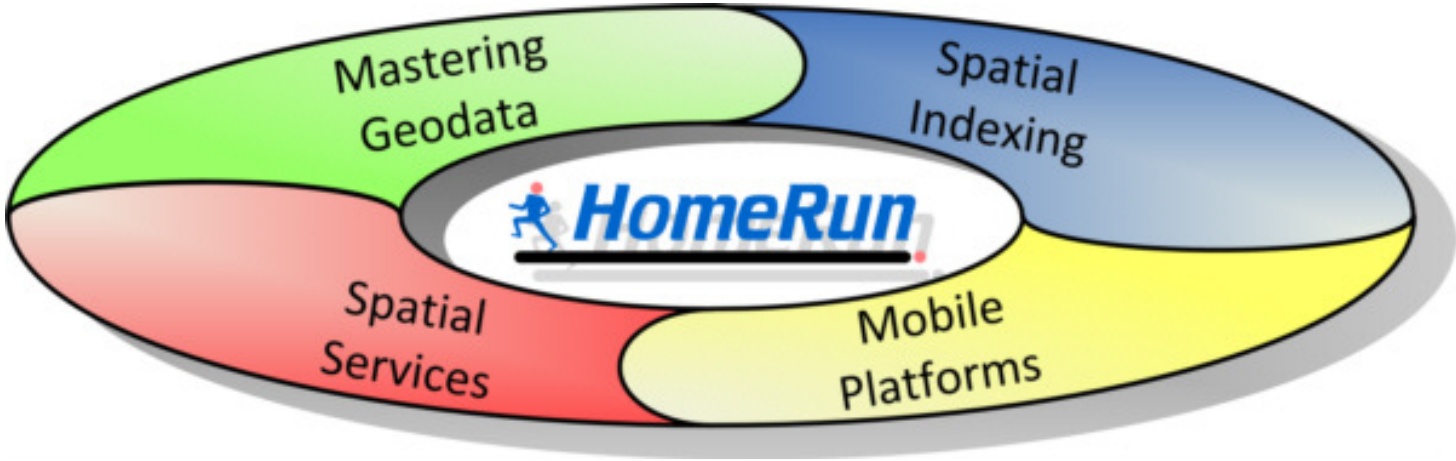
Public Transportation:

- Stops and stations are stored
- Connections are also available
  - typically as relations of streets or tracks
- The problem:
  no timetables available

Example tool chain – HomeRun

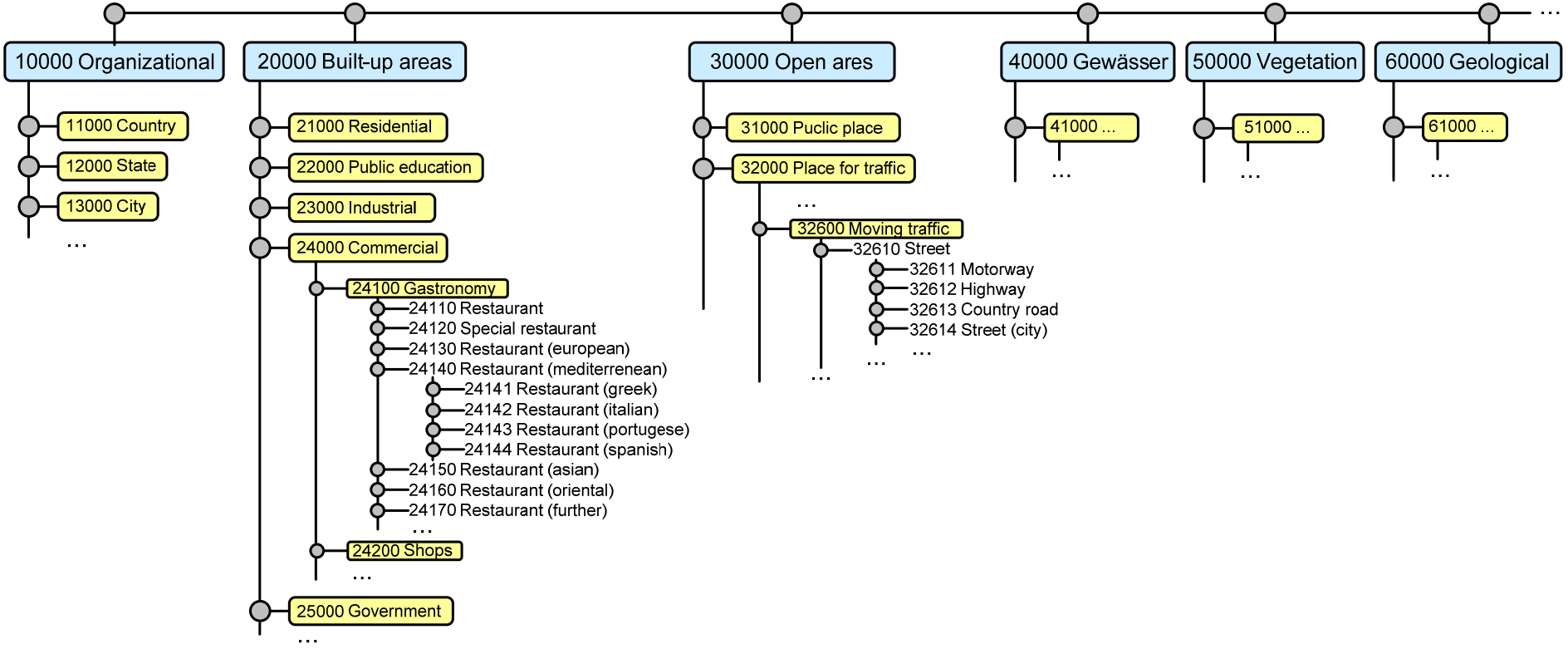
The *HomeRun* environment:

- Import, management, access of large amounts of geo data
- Also: foundation for spatial services, support for mobile platforms
  - *dorenda* map renderer and viewer
  - *donavio* navigation environment

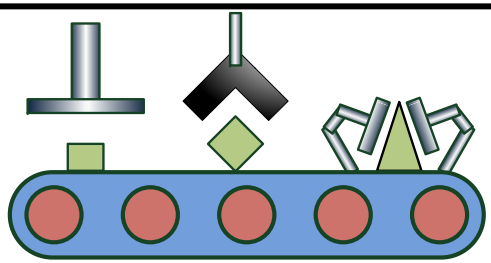


HomeRun object classification

- Objects are classified by a 5 digit number
- Implicitly: a tree



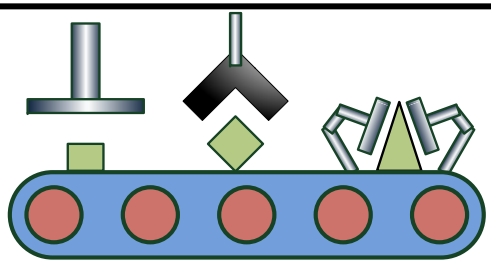
HomeRun OSM import



The HomeRun import chain:

- Parse XML
- Replace all references to nodes and ways by their geometry
  - Following references during a query is too time consuming
- Classify objects (rule-based):
  - Define the classification number
  - Decide geometry (esp. closed line string vs. area)
  - Find out the appropriate (best) name
  - Distinguish tags (names, address, is-in, links, organizational)

HomeRun OSM import

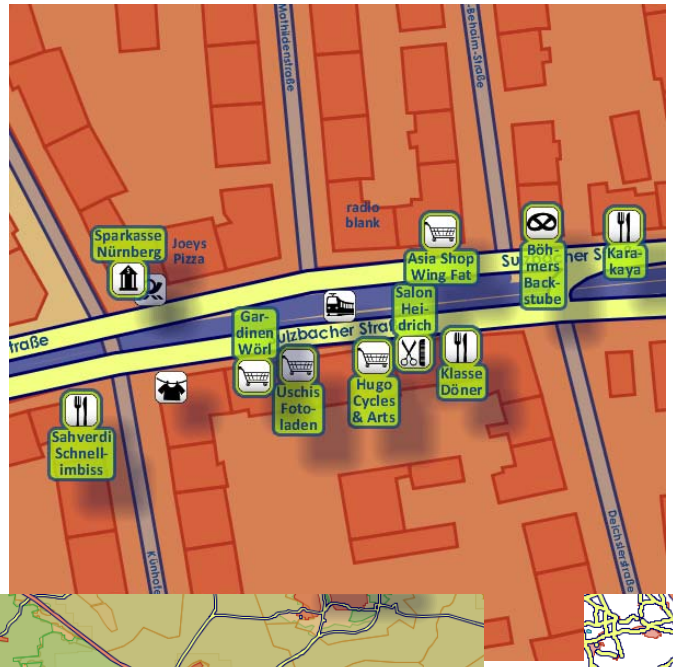


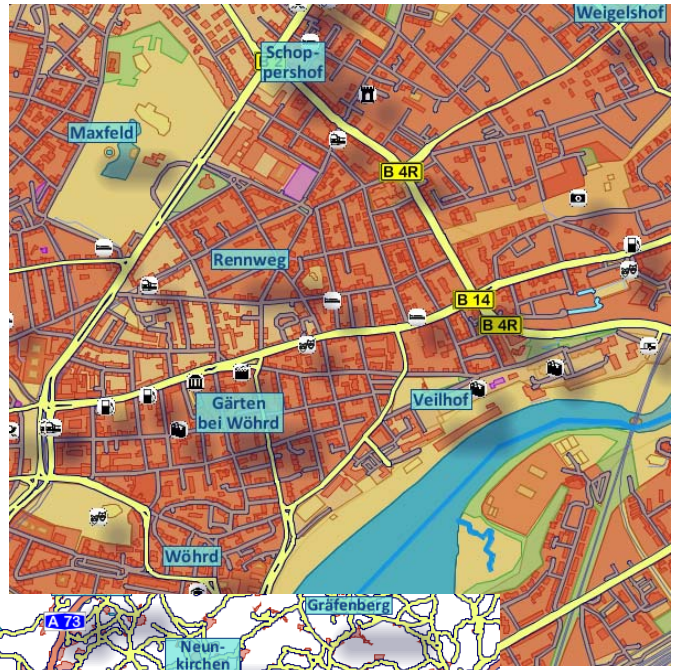
- Resolve relations:
  - Build multipolys with holes
  - Build areas from line string borders
- Prepare route planning:
  - Compute the street topology from geometries
  - Retrieve routing relevant properties (oneway, maxspeed, avgspeed)
- Compute is-in relationship:
  - Detect import larger objects (city, state etc.)
  - Geometrically check if 'inside'

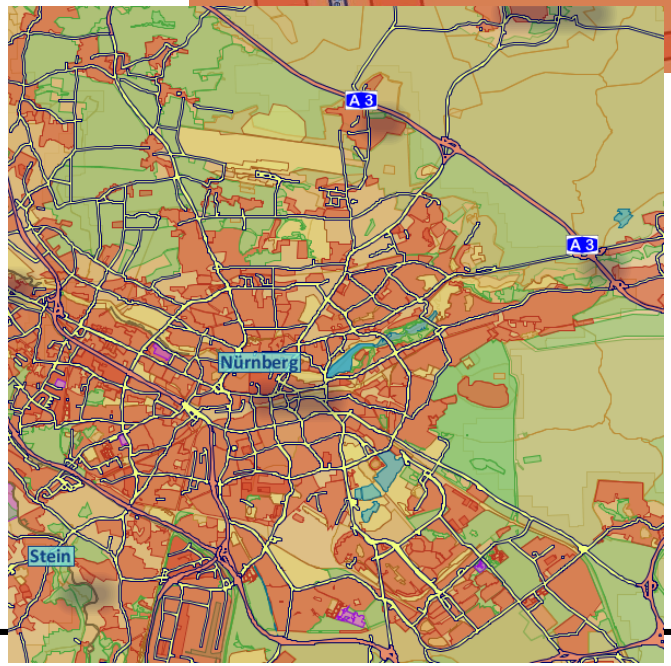
How HomeRun stores geo data:

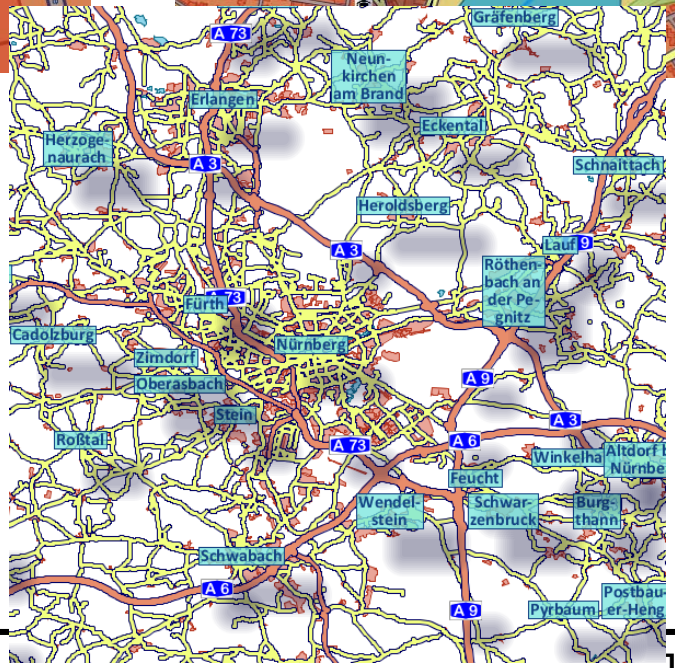
- Postgres database *without* spatial extension
  - geometries stored as *Well-known-binary* (*WKB*) in BLOBs
  - own spatial index (*Extended Split Index*)
- Also conceivable: SQL databases with spatial extension (e.g. PostGIS)
- HomeRun supports mobile devices
  - spatial extension to SQLite
  - spatially indexed virtual memory arrays (called *spatial hashtables*)

Rendered maps with *dorenda*









10 golden rules to start

How to start?

1. *Enter some data into OSM – this is useful to get an idea of the process and structure.*

2. *Browse through the OSM Wiki to learn about object classifications.*

3. *For the first step, concentrate on a specific task, e.g. map painting, route planning.*

4. *Create an OSM XML parser (a simple one is sufficient). Be sure, the parser does not read the entire file before analyzing it.*

10 golden rules to start

5. *Replace references to nodes by their coordinate – following references at runtime is too time consuming.*

6. *From the beginning, use a database (a non-spatial often is sufficient) – using files will only work for small regions.*

7. *From the beginning use a geometry library. Don't reinvent the wheel – geometric computation is challenging.*

10 golden rules to start

8. *Use an own classification schema (e.g. based on a number) – the OSM way to classify objects is too complex at runtime.*

9. *Think about rule-based classification and name finding (simple rule execution will be sufficient).*

10. *In early stages – forget relations. They are very complex to analyze, often inconsistently stored and often not useful.*

Good luck ☺

Jörg Roth

Univ. of Applied Sciences Nuremberg
Joerg.Roth@Ohm-hochschule.de
http://www.wireless-earth.org

Zonezz