Using binary data serialization for data storage and sharing

Peter Mooney (IE)
Marco Minghini (IT)

European Commission

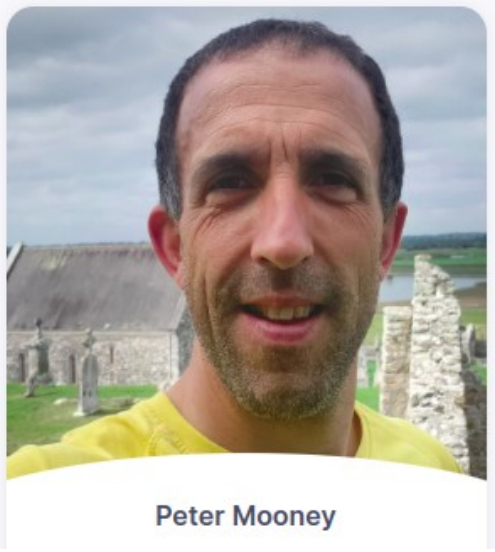
FOSS4G FIRENZE 2022

Maynooth University
National University of Ireland Maynooth

Marco Minghini

Department of Computer Science, Maynooth University, Co. Kildare, Ireland - **peter.mooney@mu.ie**

European Commission, Joint Research Centre (JRC), Ispra, Italy - **marco.minghini@ec.europa.eu**



Peter Mooney

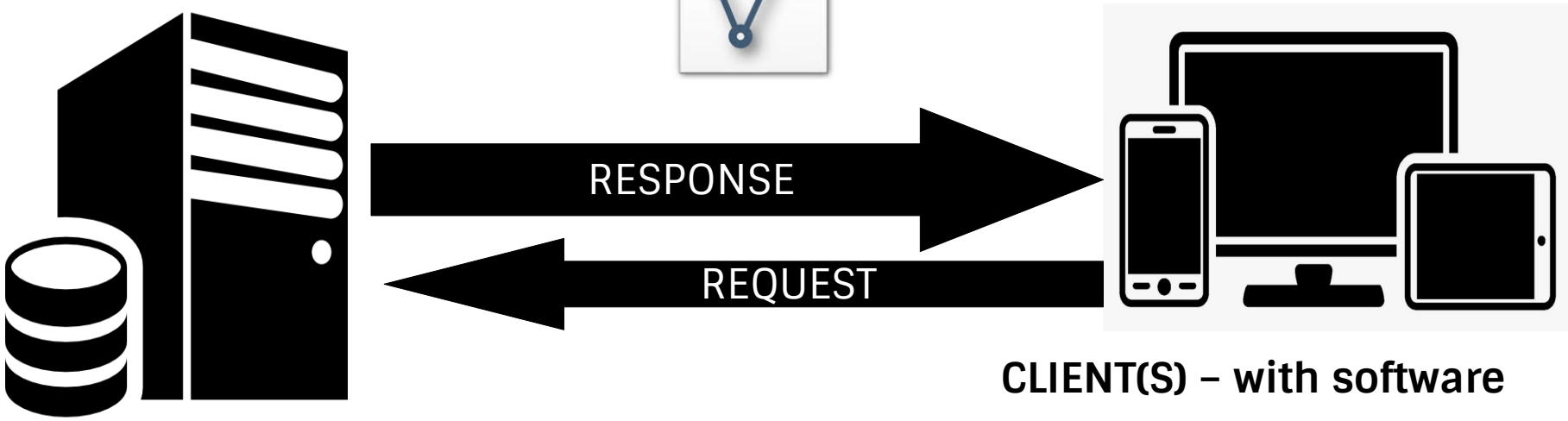
The practicalities of (geo)data exchange

- **Server**
  - **Recieves** request
  - **Prepares** response
  - **Send** response (JSON, XML, csv, SHP, etc)

- **Client tools**
  - **Load** response
  - **Extract** or **Transform**
  - **Process:** Visualise, analyse, integrate, etc.

- **Server**
  - **Preparation of complex responses** (query times)
  - **High traffic, network bandwidth** (uncontrollable)

- **Client tools**
  - **Delayed response**
  - **Loading large responses**
  - Long times: **Extracting, transforming, processing** large responses



(Geo)data SERVER – with API

CLIENT(S) – with software

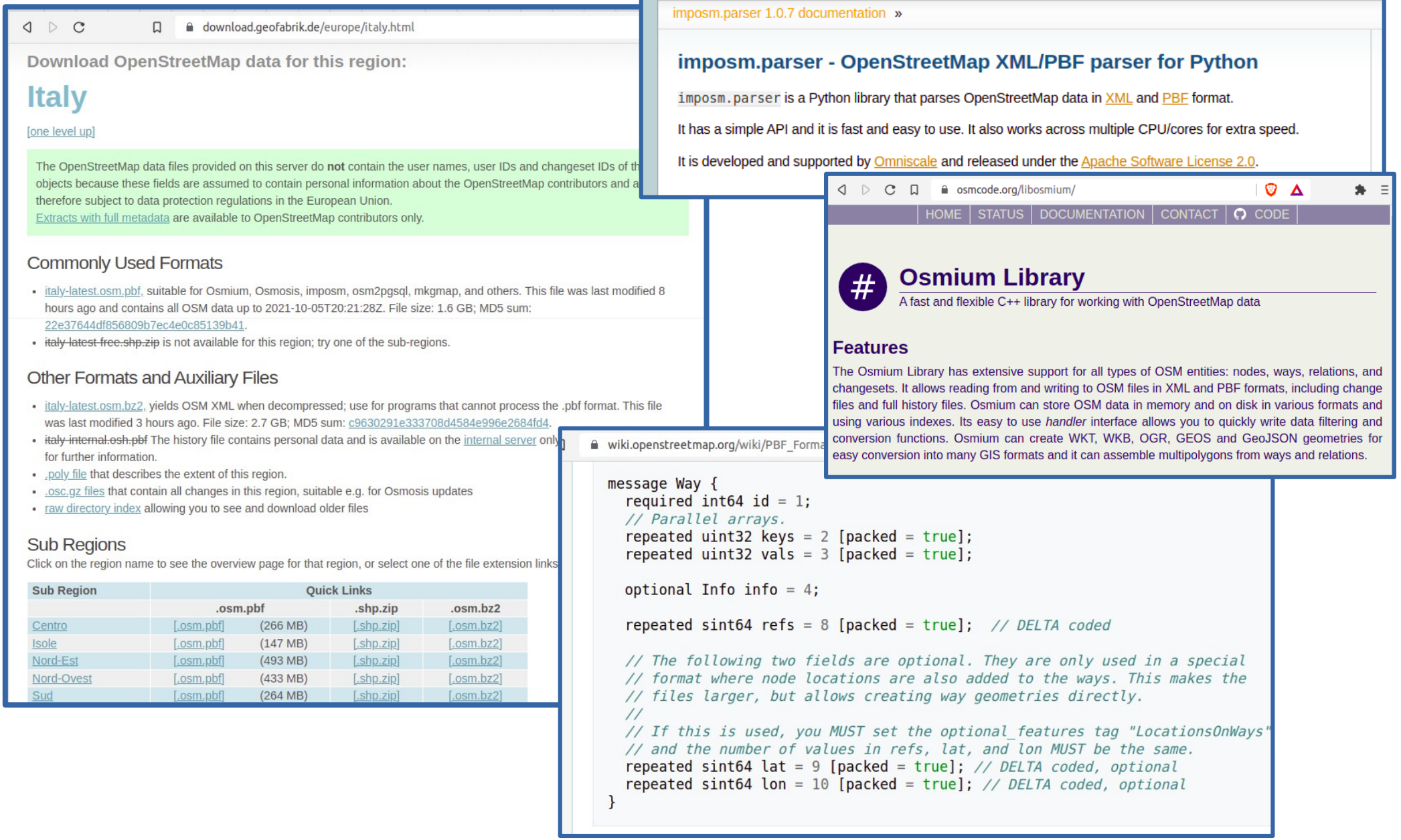
Research Question: Can binary data serialization provide a realistic alternative for geolocation data exchange in 'defacto' standards such as XML or JSON data formats.

- **Not just a "time vrs space" analysis of binary vrs JSON/XML serialization approaches**
- Focus on interoperability, usability, scalability, Open source, open approaches, ease of use ….
- **Investigate conditions where binary serialization could replace or compliment the 'de-facto' standards (JSON, XML, and so on…)**
- **Google Protocol Buffers, Apache Avro**

protobuf
Protocol Buffers

AVRO
Apache Avro

Real world Example: OpenStreetMap – dissemination of data in PBF format



download.geofabrik.de/europe/italy.html

Download OpenStreetMap data for this region:

Italy

[one level up]

The OpenStreetMap data files provided on this server do **not** contain the user names, user IDs and changeset IDs of th objects because these fields are assumed to contain personal information about the OpenStreetMap contributors and a therefore subject to data protection regulations in the European Union.
Extracts with full metadata are available to OpenStreetMap contributors only.

Commonly Used Formats

- italy-latest.osm.pbf, suitable for Osmium, Osmosis, imposm, osm2pgsql, mkgmap, and others. This file was last modified 8 hours ago and contains all OSM data up to 2021-10-05T20:21:28Z. File size: 1.6 GB; MD5 sum: 22e37644df856809b7ec4e0c85139b41.
- ~~italy-latest-free.shp.zip~~ is not available for this region; try one of the sub-regions.

Other Formats and Auxiliary Files

- italy-latest.osm.bz2, yields OSM XML when decompressed; use for programs that cannot process the .pbf format. This file was last modified 3 hours ago. File size: 2.7 GB; MD5 sum: c9630291e333708d4584e996e2684fd4.
- ~~italy-internal.osh.pbf~~ The history file contains personal data and is available on the internal server only for further information.
- .poly file that describes the extent of this region.
- .osc.gz files that contain all changes in this region, suitable e.g. for Osmosis updates
- raw directory index allowing you to see and download older files

Sub Regions

Click on the region name to see the overview page for that region, or select one of the file extension links

Sub Region	Quick Links			
	.osm.pbf		.shp.zip	.osm.bz2
Centro	[.osm.pbf]	(266 MB)	[.shp.zip]	[.osm.bz2]
Isole	[.osm.pbf]	(147 MB)	[.shp.zip]	[.osm.bz2]
Nord-Est	[.osm.pbf]	(493 MB)	[.shp.zip]	[.osm.bz2]
Nord-Ovest	[.osm.pbf]	(433 MB)	[.shp.zip]	[.osm.bz2]
Sud	[.osm.pbf]	(264 MB)	[.shp.zip]	[.osm.bz2]

imposm.org/docs/imposm.parser/latest/

imposm.parser 1.0.7 documentation »

imposm.parser - OpenStreetMap XML/PBF parser for Python

`imposm.parser` is a Python library that parses OpenStreetMap data in XML and PBF format.

It has a simple API and it is fast and easy to use. It also works across multiple CPU/cores for extra speed.

It is developed and supported by Omniscale and released under the Apache Software License 2.0.

osmcode.org/libosmium/

HOME | STATUS | DOCUMENTATION | CONTACT | CODE

Osmium Library

A fast and flexible C++ library for working with OpenStreetMap data

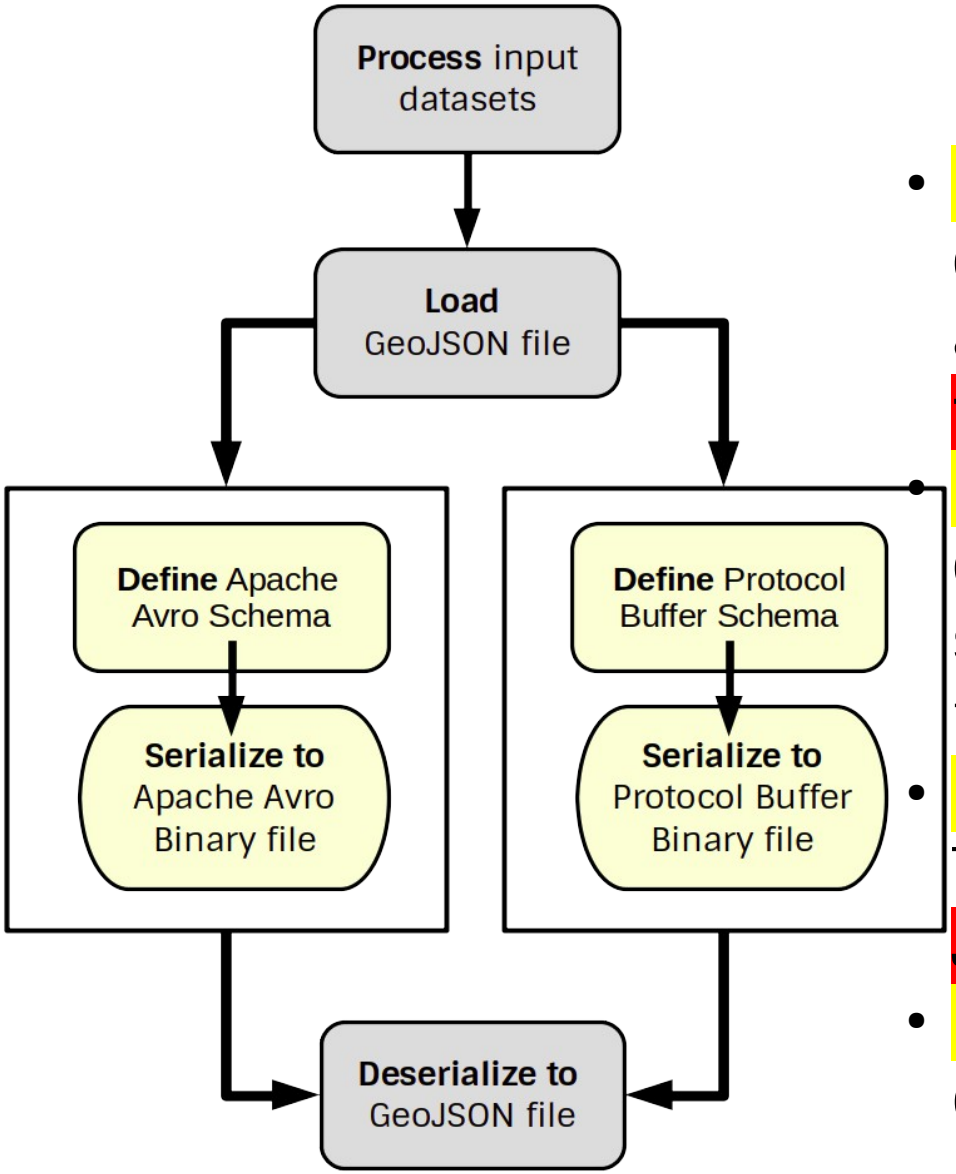
Features

The Osmium Library has extensive support for all types of OSM entities: nodes, ways, relations, and changesets. It allows reading from and writing to OSM files in XML and PBF formats, including change files and full history files. Osmium can store OSM data in memory and on disk in various formats and using various indexes. Its easy to use *handler* interface allows you to quickly write data filtering and conversion functions. Osmium can create WKT, WKB, OGR, GEOS and GeoJSON geometries for easy conversion into many GIS formats and it can assemble multipolygons from ways and relations.

wiki.openstreetmap.org/wiki/PBF_Forma

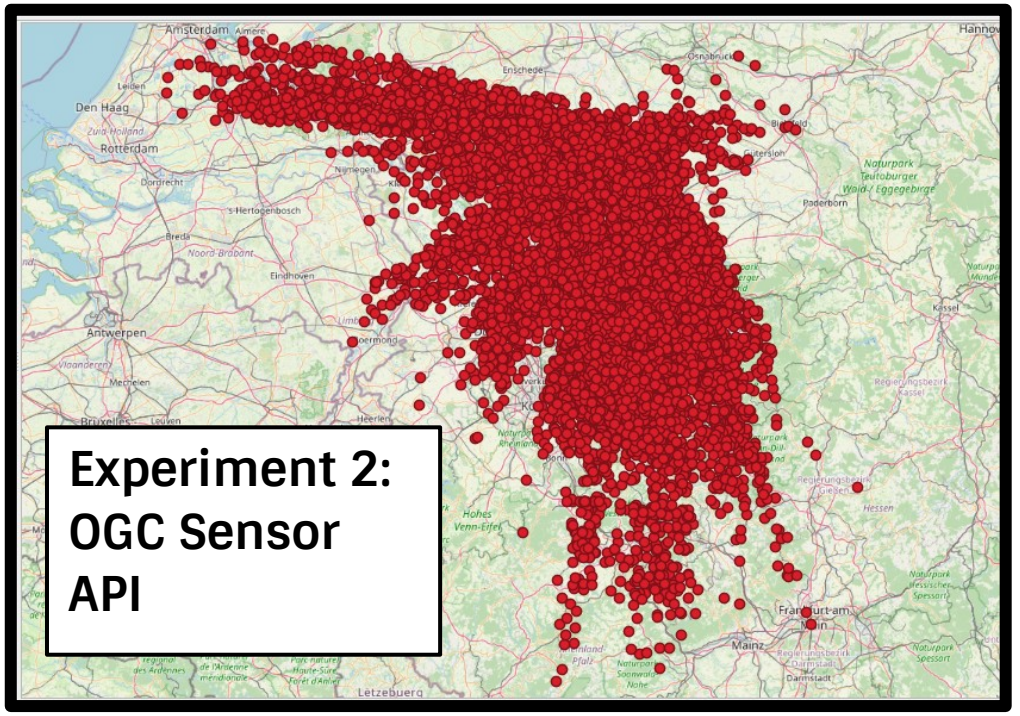
```
message Way {
  required int64 id = 1;
  // Parallel arrays.
  repeated uint32 keys = 2 [packed = true];
  repeated uint32 vals = 3 [packed = true];

  optional Info info = 4;

  repeated sint64 refs = 8 [packed = true];  // DELTA coded

  // The following two fields are optional. They are only used in a special
  // format where node locations are also added to the ways. This makes the
  // files larger, but allows creating way geometries directly.
  //
  // If this is used, you MUST set the optional_features tag "LocationsOnWays"
  // and the number of values in refs, lat, and lon MUST be the same.
  repeated sint64 lat = 9 [packed = true]; // DELTA coded, optional
  repeated sint64 lon = 10 [packed = true]; // DELTA coded, optional
}
```

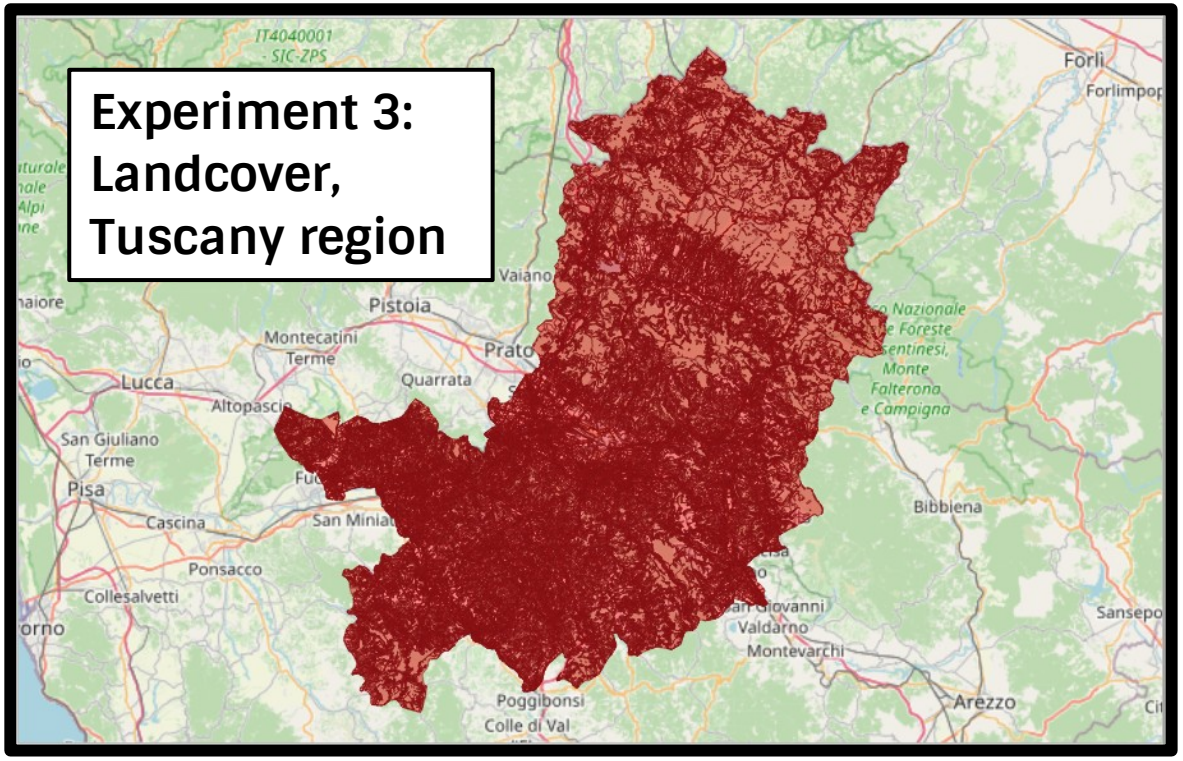
Experimental Setup – Workflow



- **Experiment 1** – "Large" static GIS dataset (NLS Finland and OSM addresses, POINT, 1.9M features, **GPKG file) [300Mb]**
- **Experiment 1a** – "small" statistic GIS dataset (randomly generated data, same as experiment 1, POINT, 20,000 features, **GPKG file) 20Mb**
- **Experiment 2** – An API (OGC Sensor Things API – POINT, 20,000 features, **JSON response) [13Mb]**
- **Experiment 3** – "Large" static GIS dataset (Landcover, Tuscany Italy, POLYGON, 161,191 features, **GPKG file) [200Mb]**

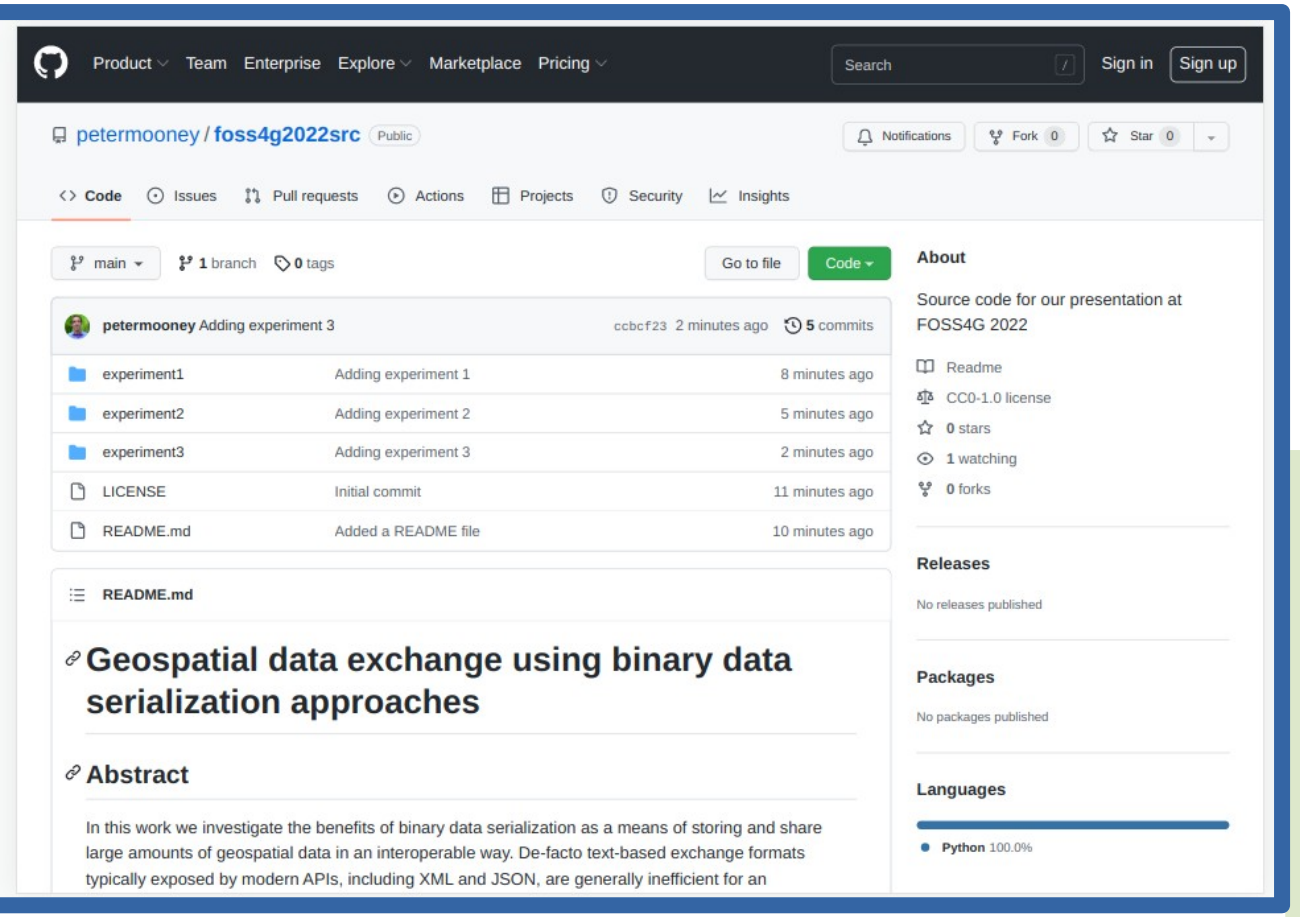

Experiment 1:
OSM + NLS
Finland


Experiment 2:
OGC Sensor
API


Experiment 3:
Landcover,
Tuscany region

Experimental Setup – Software

https://github.com/petermooney/foss4g2022src



SCAN ME

python 3

Software setup

- **Fully reproducible code** (GitHub)
- No “hacks”
- **Use open source and widely supported Python libraries only**
- Interoperable

Creating Avro and Protobuf Schemas

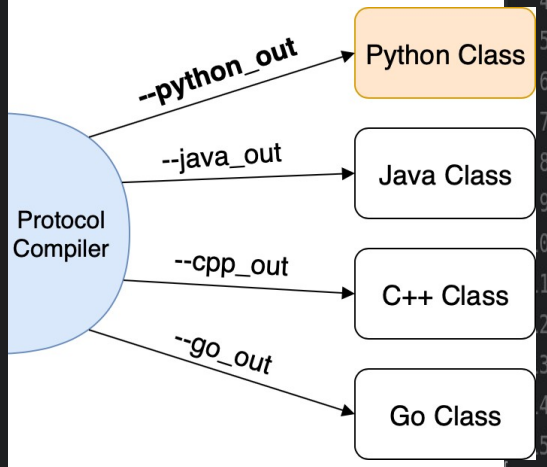
```avro
{
    "name": "FinlandAddresses",
    "namespace": "FI_Addresses",
    "type": "record",
    "fields": [
        {"name": "addrHousenumber", "type": ["string","null"]},
        {"name": "addrStreet", "type": ["string","null"]},
        {"name": "addrCity", "type": ["string","null"]},
        {"name": "fid", "type": "int"},
        {"name": "source", "type": ["string","null"]},
        {"name": "addrUnit", "type": ["string","null"]},
        {"name": "fullAddress", "type": ["string","null"]},
        {"name": "geometry", "type": ["string","null"]}
    ]
}
```

Original GPKG

Layer Properties — original-dataset-experiment1 test-geopackage — Fields

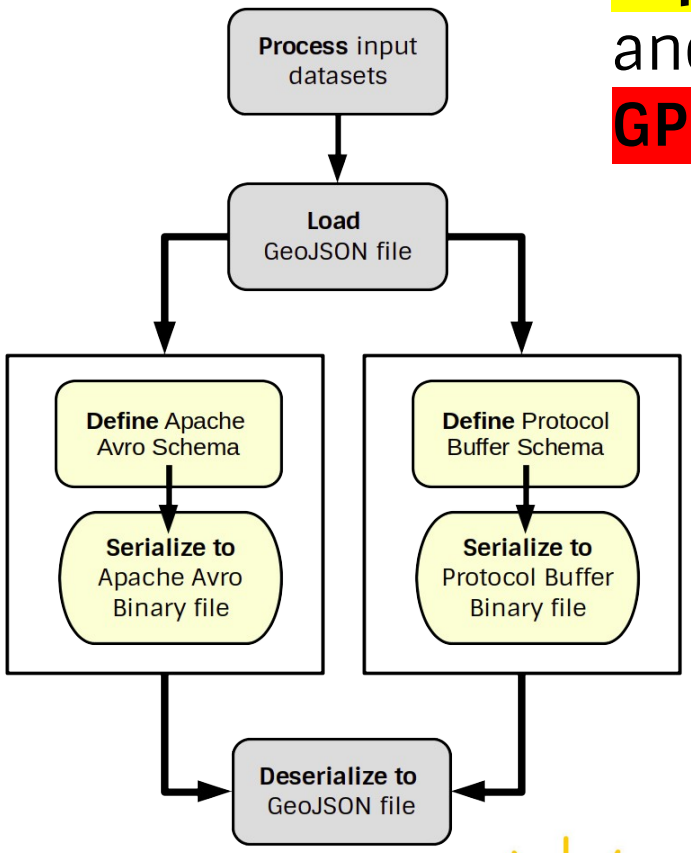
Id	Name	Alias	Type	Type name	Length	Precision	Comment
123 0	fid		qlonglong	Integer64	0	0	
abc 1	addr:housenumber		QString	String	0	0	
abc 2	addr:street		QString	String	0	0	
abc 3	addr:country		QString	String	0	0	
abc 4	addr:city		QString	String	0	0	
abc 5	source		QString	String	0	0	
abc 6	fullAddress		QString	String	0	0	
abc 7	addr:unit		QString	String	0	0	

```proto
syntax = "proto2";

package AddressFI;

message Address_prop {
  required string addrHousenumber = 1;
  required string addrStreet = 2;
  required string addrCity = 3;
  required int32 fid = 4;
  required string source = 5;
  required string addrUnit = 6;
  required string fullAddress = 7;
  required string geometry = 8;
}
message Address {
repeated Address_prop address = 1;
}
```




```python
# -*- coding: utf-8 -*-
# Generated by the protocol buffer compiler.  DO NOT EDIT!
# source: address.proto
"""Generated protocol buffer code."""
from google.protobuf import descriptor as _descriptor
from google.protobuf import message as _message
from google.protobuf import reflection as _reflection
from google.protobuf import symbol_database as _symbol_database
# @@protoc_insertion_point(imports)

_sym_db = _symbol_database.Default()



DESCRIPTOR = _descriptor.FileDescriptor(
  name='address.proto',
  package='AddressFI',
  syntax='proto2',
```

Experiment 1 Results: Conflated addresses dataset

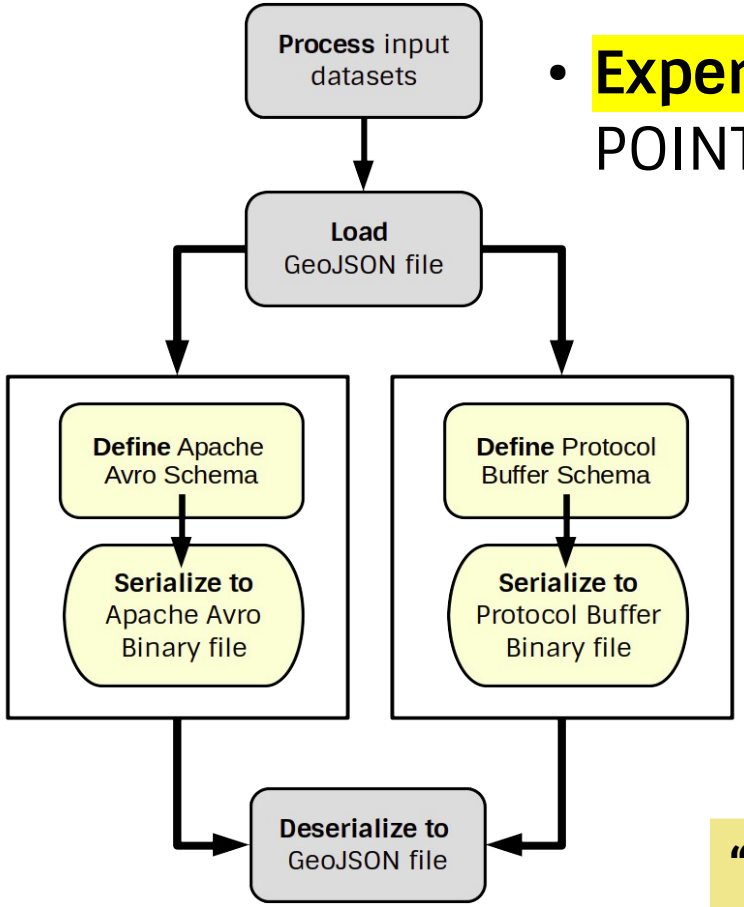


- **Experiment 1** – "Large" GIS dataset (NLS Finland and OSM addresses, POINT, 1.9M features, **GPKG file) [300Mb]**

Processing steps from Figure 1	Time (s)	Size (Kb)
Convert (GPKG → GeoJSON)	327, 11.3	614,859
Load (GeoJSON)	81, 3.2	614,859
Serialize (GeoJSON → Avro)	301, 3.4	228,102
Serialize (GeoJSON → PBF)	306, 2.9	235,821
Deserialize (PBF → GeoJSON)	378, 2.8	542,878
Deserialize (Avro → GeoJSON)	389, 3.1	546,616

"Long (de)serialization processing times. The difference in GeoJSON file sizes (614MB vrs 546MB) is due to coordinate precision representation in Python libraries. Binary files (Avro and PBF) are ~ 21% smaller"

Experiment 2 Results: OGC Sensor Things API

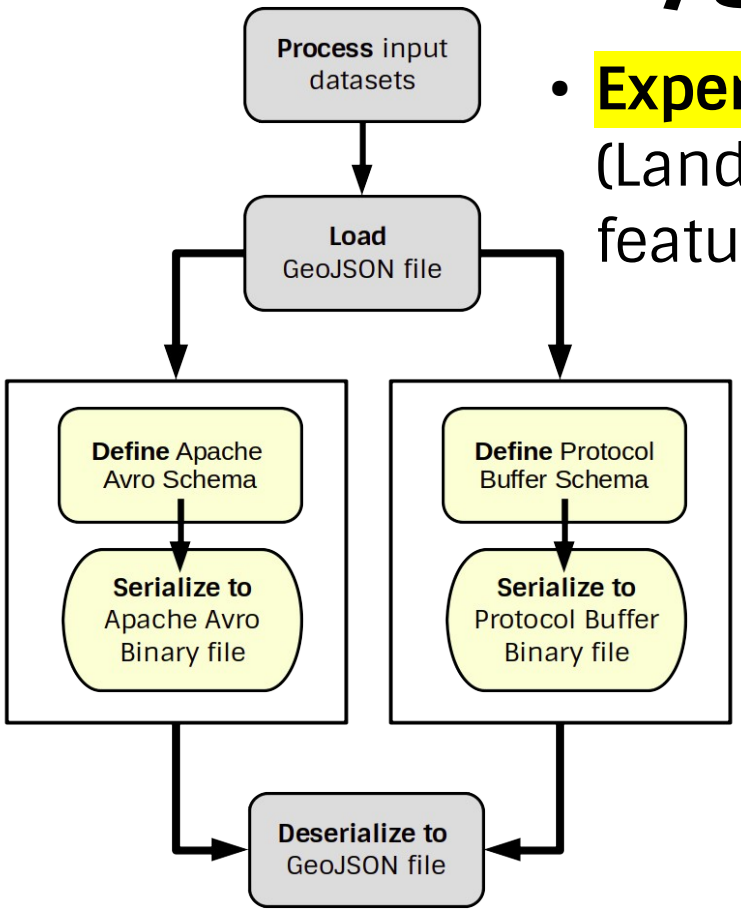


- **Experiment 2** – An API (OGC Sensor Things API – POINT, 20,000 features, **JSON response) [13Mb]**

Processing steps from Figure 1	Time (s)	Size (Kb)
JSON Response download	n/a	12,926
Load (GeoJSON)	1.23, 0.07	11,539
Serialize (GeoJSON → Avro)	0.34, 0.04	7,001
Serialize (GeoJSON → PBF)	0.32, 0.04	7,109
Deserialize (PBF → GeoJSON)	1.14, 0.07	11,515
Deserialize (Avro → GeoJSON)	1.10, 0.03	11,554

"Very positive results. Binary files (Avro and PBF) ~ 46% smaller with serialization very fast. Deserialization is 3.5 times slower and this would need to be improved, perhaps with a more efficient implementation."

Experiment 3 Results: Polygons – Tuscany



- **Experiment 3** – "Large" static GIS dataset (Landcover, Tuscany Italy, POLYGON, 161,191 features, **GPKG file) [200Mb]**

Processing steps from Figure 1	Time (s)	Size (Kb)
Convert (GPKG → GeoJSON)	128.7, 2.30	528,386
Reduce GeoJSON attributes	148.3, 2.2	320,472
Load (GeoJSON - reduced)	48.7, 0.31	320,472
Serialize (GeoJSON → Avro)	61.3, 0.18	264,555
Serialize (GeoJSON → PBF)	58.0, 0.15	264,937
Deserialize (PBF → GeoJSON)	80.4, 1.53	320,472
Deserialize (Avro → GeoJSON)	82.6, 1.93	320,472

"Many redundant attributes in original dataset which we removed. Original dataset expresses coordinates with 15 decimal places whereas Python GeoJSON allows 10. Avro and PBF ~ 17% smaller than the *reduced* GeoJSON file. "

Experimental Results – Technical considerations

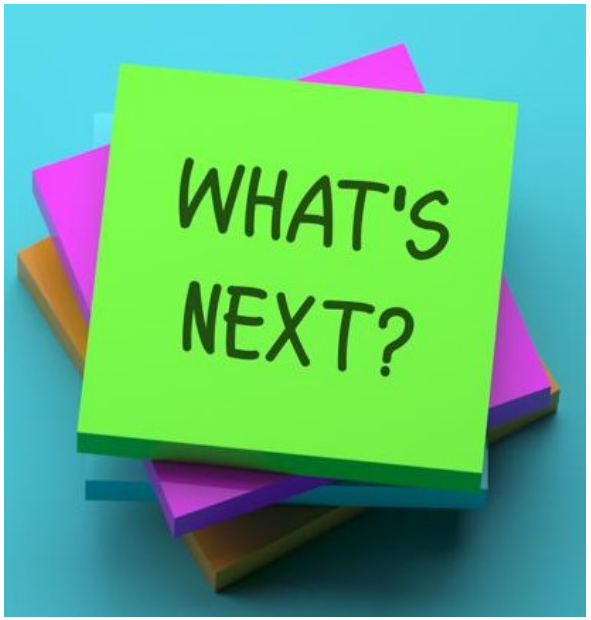
- File sizes: **PBF and Avro were, on average, at least 20% smaller than the GeoJSON files**
- **Deserialization was always slower** (futher investigation needed) than serialization
- Deserialization back to GeoJSON was performed as **binary files lack basic query capabilities**
- **Data model considerations**: Precision of geographic coordinates, inclusion of redundant attributes, and so on must be carefully considered in the source datasets.

Experimental Results – Practical considerations

- **Positive results but many overheads remain** (non-automated schema generation, specialist programmer knowledge, user community support, wider (geo) tool support)
- **Binary data serialization could work very well for specific scenarios, services and applications** (OSM is the leading example)
- **It still remains a challenge to measure and understand "success"** in regards to the possible replacement of existing 'de-facto' standards with binary data serialization

Opportunities for future work

- **Further computational experimentation** on different types of (geo)data and (geo)services
- **Making binary data approaches user-friendly** like their non-binary data alternatives. This includes more integrated software tool support
- Schema definitions: **automated semantic interoperabilty** using linked geodata



Publications related to this work

European Commission

JRC TECHNICAL REPORT

Emerging approaches for data-driven innovation in Europe

Sandbox experiments on the governance of data and technology

SCAN ME

Granell C., Mooney P., Jirka S., Rieke M., Ostermann F., van den Broecke J., Sarretta A., Verhulst S., Dencik L., Oost H., Micheli M., Minghini M., Kotsev A., Schade S.

2022

The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume XLVIII-4/W1-2022
Free and Open Source Software for Geospatial (FOSS4G) 2022 – Academic Track, 22–28 August 2022, Florence, Italy

GEOSPATIAL DATA EXCHANGE USING BINARY DATA SERIALIZATION APPROACHES

P. Mooney [1,*], M. Minghini [2]

[1] Department of Computer Science, Maynooth University, Co. Kildare, Ireland - peter.mooney@mu.ie
[2] European Commission, Joint Research Centre (JRC), Ispra, Italy - marco.minghini@ec.europa.eu

Commission IV, WG IV/4

KEY WORDS: Binary data serialization, interoperability, geospatial, GeoJSON, Protocol Buffers, Apache Avro.

ABSTRACT:

In this paper we investigate the benefits of binary data serialization as a means of storing and sharing large amounts of geospatial data in an interoperable way. De-facto text-based exchange encodings typically exposed by modern Application Programming Interfaces (APIs), including eXtensible Markup Language (XML) and JavaScript Object Notation (JSON), are generally inefficient for an increasingly higher number of applications due to their inflated volumes of data, low speed and the high computational cost for parsing and processing. In this work we consider comparisons of JSON/Geospatial JSON (GeoJSON) and two popular binary data encodings (Protocol Buffers and Apache Avro) for storing and sharing geospatial data. Using a number of experiments, we illustrate the advantages and disadvantages of both approaches for common workflows that make use of geospatial data encodings such as GeoPackage and GeoJSON. The paper contributes a number of practical recommendations around the potential for binary data serialization for interoperable (geospatial) data storage and sharing in the future.

1. INTRODUCTION AND MOTIVATION

Data-driven innovation has seen recent advances enabled by a dynamic technological context characterised by the continuous influx of data, miniaturization and massive deployment of sensing technology, data-driven algorithms, and the Internet of Things (IoT) (Granell et al., 2022). Data-driven innovation is considered a key part of several policy actions worldwide. The recently published European strategy for data (European Commission, 2020) envisions Europe's digital future in the data economy through the establishment of dedicated data spaces. These would allow the efficient flow of data between actors and sectors so that data-driven innovation is translated into concrete benefits for the economy and society. To exploit Europe's potential in our data-driven society, technologies currently used for the management, exchange and consumption of data, including geospatial data, must be evaluated in terms of their suitability to efficiently scale and adapt to streams of larger data and datasets. The increasing number of users accessing data services through mobile devices is putting pressure on service providers to make larger volumes of data available efficiently to these particular users. For many years, encodings such as JavaScript Object Notation (JSON), Geospatial JSON (GeoJSON), Comma-Separated Values (CSV) and eXtensible Markup Language (XML) have been considered as the *de facto* interoperable standards for data serialisation. The majority of Application Programming Interfaces (APIs) available today facilitate data sharing and exchange using these encodings (Vaccari et al., 2020). Whilst such encodings, mainly JSON and XML, have an almost universal support and many other advantages (e.g. they are human and machine-readable as well as open and standards-based), they also have many limitations. These limitations are particularly pronounced when the volume of data is large, resulting in reduced computational performance when exchanging or managing large data volumes.

As an alternative, binary data serialization approaches also allow for the interoperable exchange of large volumes of data (Vanura and Kriz, 2018). Binary data serialization is a process to transform data structures of an object into a binary stream that can be transmitted (and/or stored) and reconstructured later. Popular distributors of geospatial data have also begun mak-

SCAN ME

perimental analysis for our three common GIS workflow scen-

* Corresponding author

Grazie a tutti – Thanks everyone!

Peter Mooney (IE)
Marco Minghini (IT)

Department of Computer Science, Maynooth University, Co. Kildare, Ireland - **peter.mooney@mu.ie**

European Commission, Joint Research Centre (JRC), Ispra, Italy - **marco.minghini@ec.europa.eu**