Intro

The following set of slides gives an introduction to the functionality and usage of gsemantique. It starts with placing gsemantique in the context of foundational works, i.e. mainly the semantique package.

Relationship between semantic EO querying systems

SEMANTIQUE

framework for semantic, knowledge-based querying (van der Meer et al 2022)

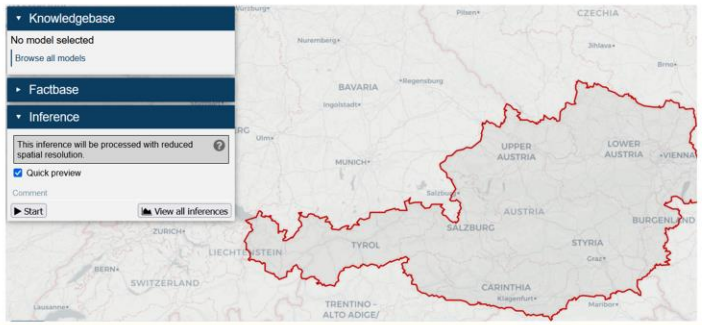
sen2cube.at

Austrian semantic data cube (Sudmanns et al 2021)

gsemantique

On-demand EO data cubes (Kröber et al 2025)

- comprehensive, technology-stack spanning application (SaaS)
- semantically-enriched Sentinel-2 + auxiliary data sets (DEM) with country-level coverage
- processing via central, resource-intensive server without internal scalability mechanisms (chunking)
- primary access via graphical user interface (GUI)



main characteristics & differences

- standalone python library
- pre-configured, diverse, semantic and non-semantic data sets with global coverage
- processing locally or in the cloud with internal scalability through chunking of spatio-temporal extents
- primary access via programmatic interface

```python
# define recipe
recipe = sq.QueryRecipe()
recipe["cloudless"] = (
    sq.entity("valid")
    .filter(sq.entity("cloud").evaluate("not"))
    .reduce("count", "time")
)

# define spatio-temporal extent
res = 500
epsg = 3035
t_start, t_end = '2022-08-01', '2022-09-01'
time = sq.TemporalExtent(pd.Timestamp(t_start), pd.Timestamp(t_end)
aoi = sbg.to_crs(4326)
space = sq.SpatialExtent(aoi)
```

advantages from user's point of view

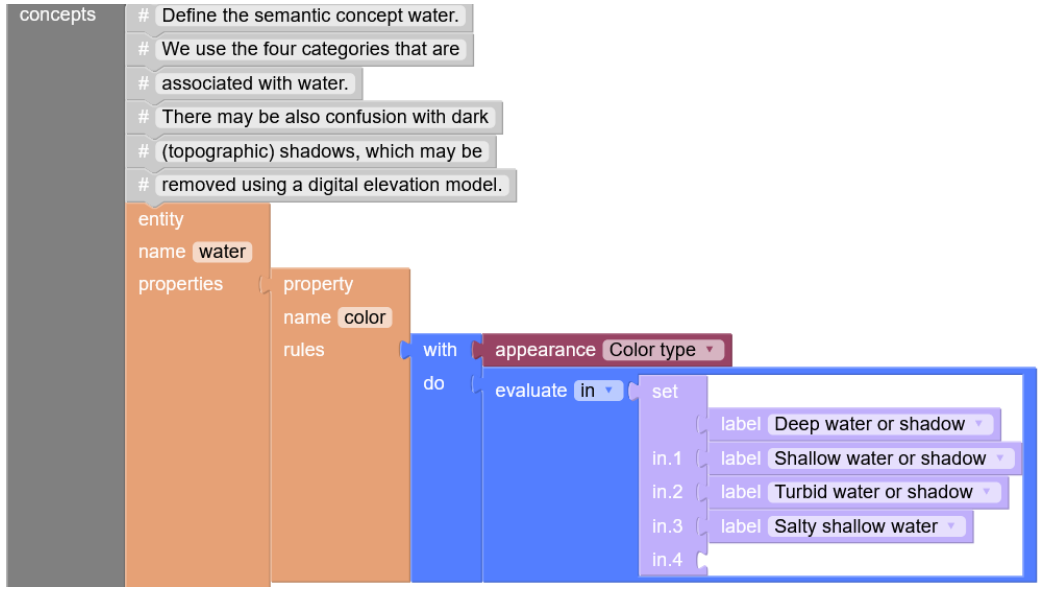
- ✓ accessibility (no programming skills required, no infrastructure needed beyond a web-browser)
- ✓ full semantic capabilities (comprehensive semantic enrichment of all Sentinel-2 imagery)

- ✓ flexibility (analyses anywhere on Earth)
- ✓ extensibility (analyses with any STAC-indexed raster data set)
- ✓ customisability (leveraging user-defined functions)

Semantique

- framework for semantic querying
- underlying processing engine in sen2cube.at
- implemented as an open-source Python package

Sen2cube view



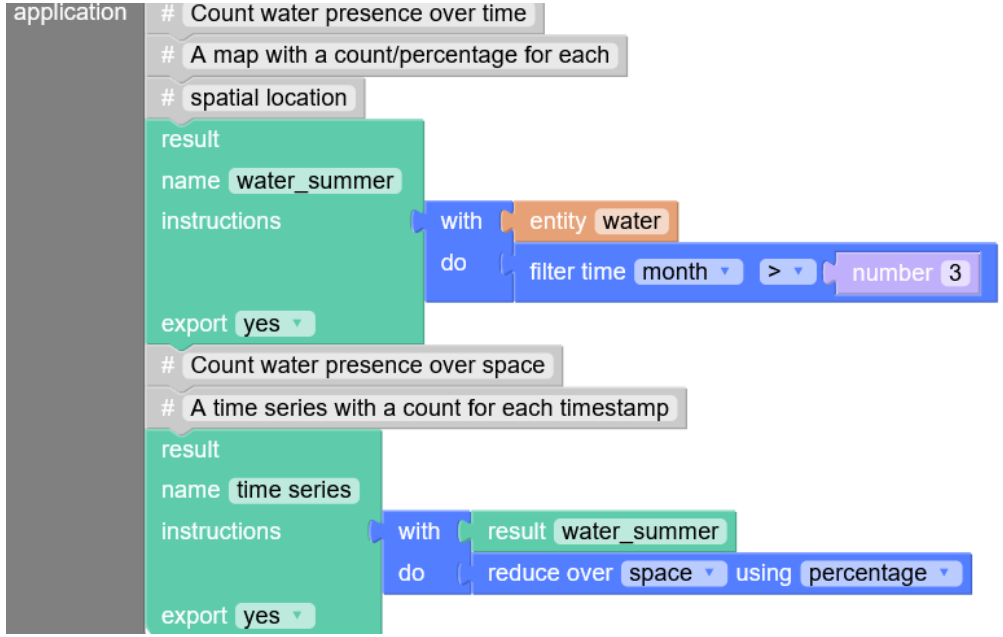
Semantique view

```
mapping = sq.mapping.Semantique()
mapping["entity"] = {}
mapping["entity"]["water"] = {
    "color": (
        sq.appearance("colortype")
            .evaluate("in", [21, 22, 23, 24])
    )
}
```

Semantique

- framework for semantic querying
- underlying processing engine in sen2cube.at
- implemented as an open-source Python package

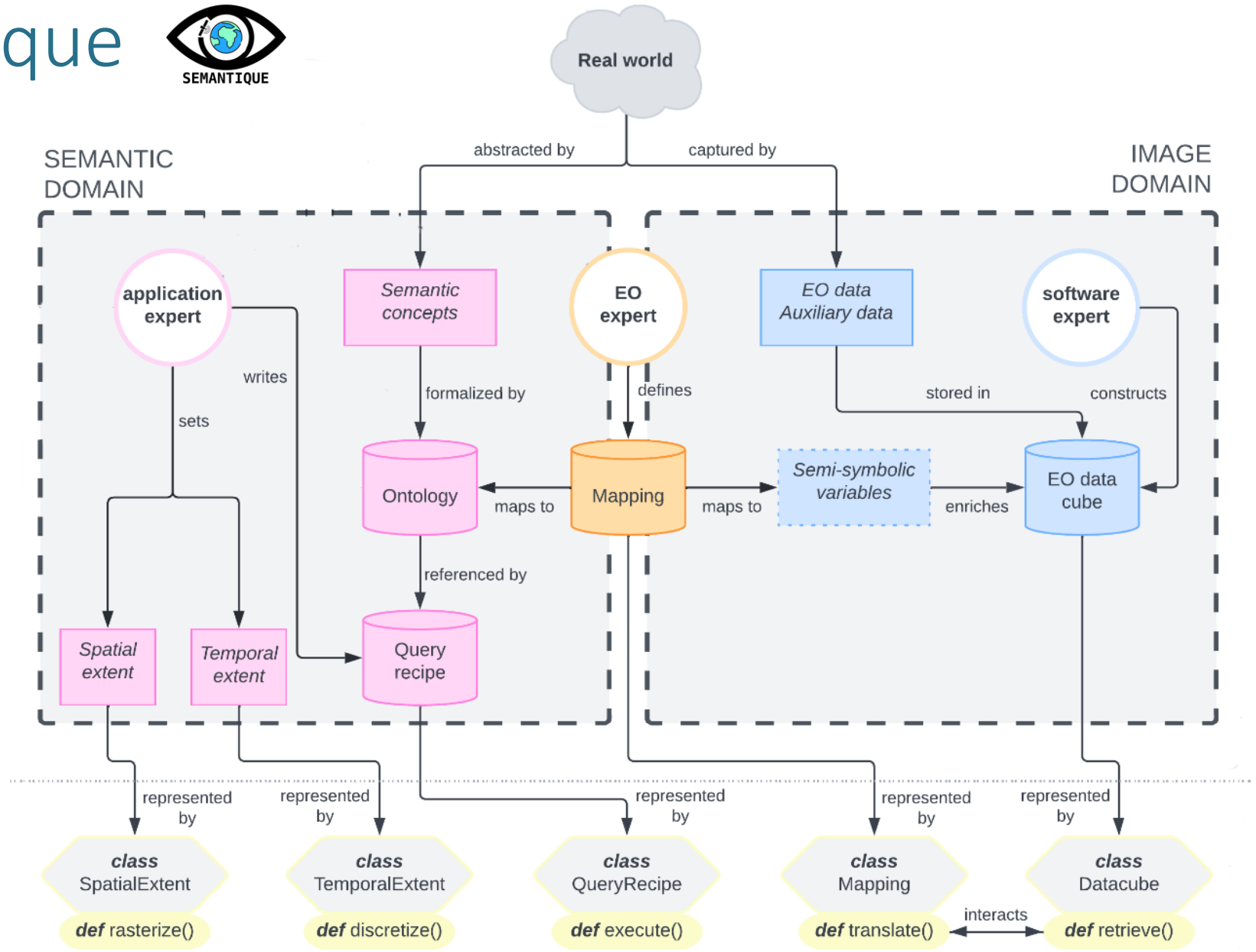
Sen2cube view



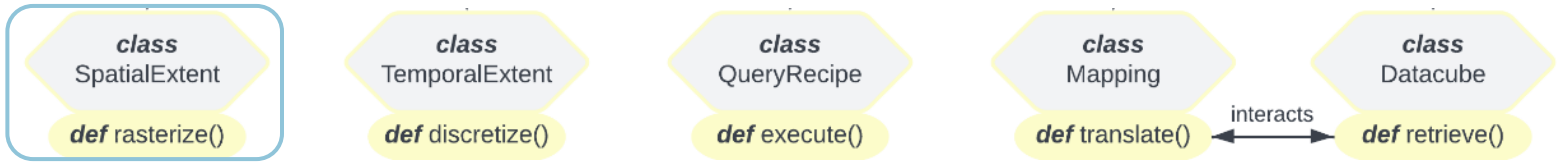
Semantique view

```
recipe = sq.QueryRecipe()
recipe["water_summer"] = (
    sq.entity("water")
    .filter_time("month", "greater", 3)
)
recipe["time series"] = (
    sq.result("water_summer")
    .reduce("percentage", "space")
)
```

Semantique


Real world
abstracted by
captured by
SEMANTIC DOMAIN
IMAGE DOMAIN
application expert
writes
sets
Semantic concepts
formalized by
EO expert
defines
EO data Auxiliary data
stored in
software expert
constructs
Ontology
maps to
Mapping
maps to
Semi-symbolic variables
enriches
EO data cube
referenced by
Spatial extent
Temporal extent
Query recipe
represented by
represented by
represented by
represented by
represented by
class SpatialExtent
def rasterize()
class TemporalExtent
def discretize()
class QueryRecipe
def execute()
class Mapping
def translate()
interacts
class Datacube
def retrieve()

Semantique – individual components



- spatial extents can be bboxes, single or multiple features
- based on geopandas package (GeoDataFrame initializer)

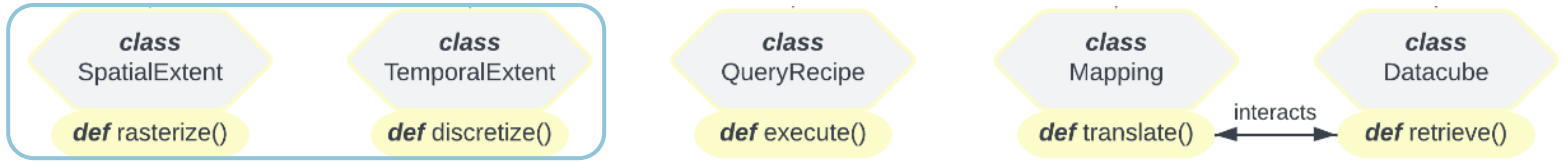
```
import geopandas as gpd
import semantique as sq
from shapely.geometry import box

# define spatial extent from geojson file
gdf = gpd.read_file("files/footprint.geojson")
space = sq.SpatialExtent(gdf)

# define spatial extent from bbox coords
xmin, ymin, xmax, ymax = 9,50,10,51
aoi = box(xmin, ymin, xmax, ymax)
gdf = gpd.GeoDataFrame(geometry=[aoi], crs=4326)
space = sq.SpatialExtent(gdf)
```

Semantique – individual components

- temporal extents can be datetime strings (start & end)
- based on pandas package (Timestamp initializer)

```
import semantique as sq
time = sq.TemporalExtent("2019-01-01", "2020-12-31")
```

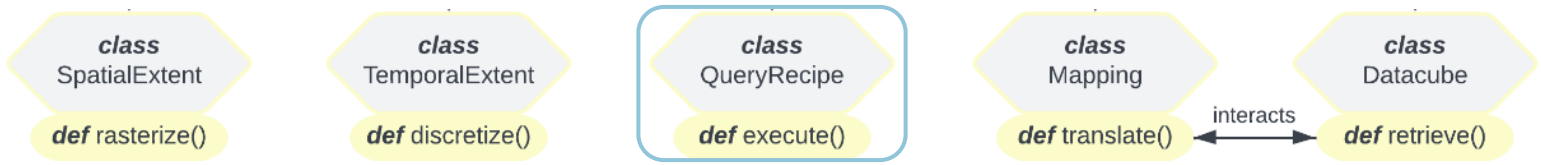
Semantique – individual components



- full description of spatio-temporal extents requires some additional information to be specified
  - spatial resolution *(required)*
  - coordinate reference system *(optional)*
  - timezone *(optional)*

```
from semantique.processor.utils import parse_extent

extent = parse_extent(
    space,
    time,
    spatial_resolution = [-10, 10],
    crs = 3035,
    tz = "UTC"
)
```

Semantique – individual components



- application model references semantic concepts
- based on a Python dictionary

```python
import semantique as sq

recipe = sq.QueryRecipe()
recipe["water_summer"] = (
    sq.entity("water")
    .filter_time("month", "greater", 3)
)
recipe["time series"] = (
    sq.result("water_summer")
    .reduce("percentage", "space")
)
```

```python
import json

# to persist the recipe on disk
with open("recipe.json", "w") as file:
    json.dump(recipe, file, indent = 2)

# to read it back into memory
with open("recipe.json", "r") as file:
    recipe = sq.QueryRecipe(json.load(file))
```

recipe.json

```json
{
  "water_summer": {
    "type": "processing_chain",
    "with": {
      "type": "concept",
      "reference": [
        "entity",
        "water"
      ]
    },
    "do": [
      {
        "type": "verb",
        "name": "filter",
        "params": {
          "filterer": {
            "type": "processing_chain",
            "with": {
              "type": "self"
            },
            ...
```

Semantique – individual components

- mapping links each semantic concept to data values in an EO data cube
- based on a Python dictionary

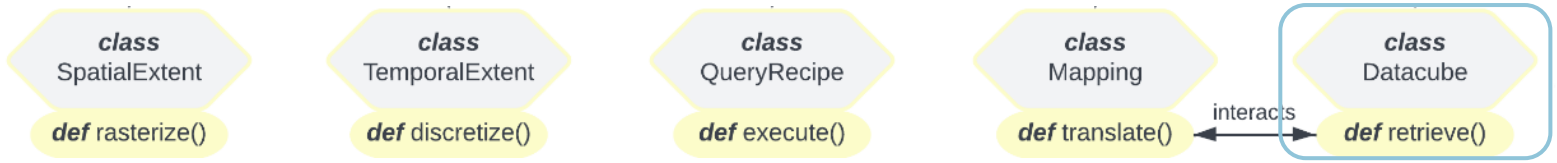
```python
import semantique as sq

mapping = sq.mapping.Mapping()
mapping["entity"] = {}
mapping["entity"]["water"] = {
    "color": (
        sq.appearance("colortype")
          .evaluate("in", [21, 22, 23, 24])
    )
}
```

```python
import json

# to persist the mapping on disk
with open("mapping.json", "w") as file:
    json.dump(mapping, file, indent = 2)

# to read it back into memory
with open("mapping.json", "r") as file:
    mapping = sq.mapping.Semantique(json.load(file))
```

mapping.json

```json
{
  "entity": {
    "water": {
      "color": {
        "type": "processing_chain",
        "with": {
          "type": "layer",
          "reference": [
            "appearance",
            "colortype"
          ]
        },
        "do": [
          {
            "type": "verb",
            "name": "evaluate",
            "params": {
              "operator": "in",
              "y": {
                "type": "set",
                "content": [
                  21,
                  ...
```

Semantique – individual components



- multi-dimensional array structure storing the data
- 3 different configurations of data cube objects available

How is the cube is organized?

(a)

GeoTIFF

enables to interact with zipped archive of multiple GeoTIFF files

only for demo purposes

(b)

OPEN DATA CUBE

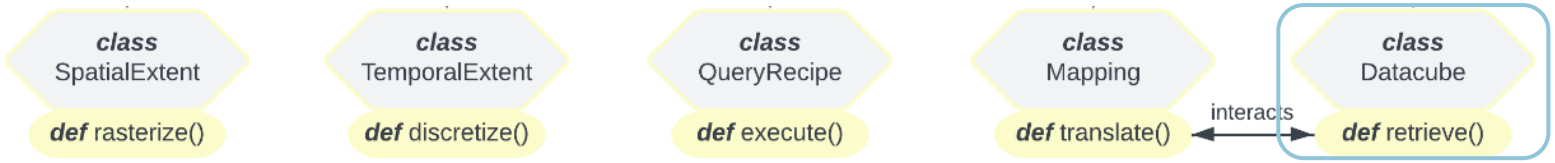
enables to interact with EO data cube instances deployed using ODC

(c)

STAC

enables to interact with STAC metadata search results

Semantique – individual components



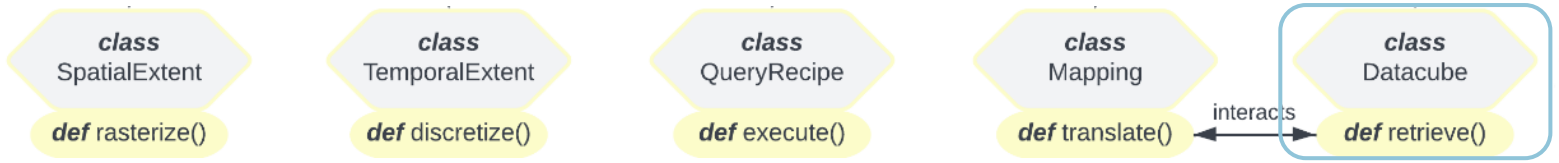
- all data cubes are represented by their layout
- layout intended to be created once by software/EO expert
- specific layout attributes exist for individual data cube configurations
- based on a Python dictionary & stored as a JSON-structured file

```
{
    "reflectance": {
        "s2_band01": {
            "name": "coastal",
            "description": "Coastal aerosol band (~443nm; 60m spatial
            resolution). Top of atmosphere (TOA) reflectance captured by
            the multi-spectral instrument (MSI) of Sentinel-2A or
            Sentinel-2B as band 1.",
            "type": "continuous",
            "values": {
                "min": 0,
                "max": 1,
                "precision": 1
            },
            "file": "coastal.geotiff",
            "copyright": "Contains modified Copernicus data."
        },
        "s2_band02": {
            "name": "...",
```

layout.json
(GeoTiffArchive)

```
import json
import semantique as sq

with open("files/layout_gtiff.json", "r") as file:
    dc = sq.datacube.GeotiffArchive(
        json.load(file),
        src = "files/layers_gtiff.zip"
    )
```

Semantique – individual components



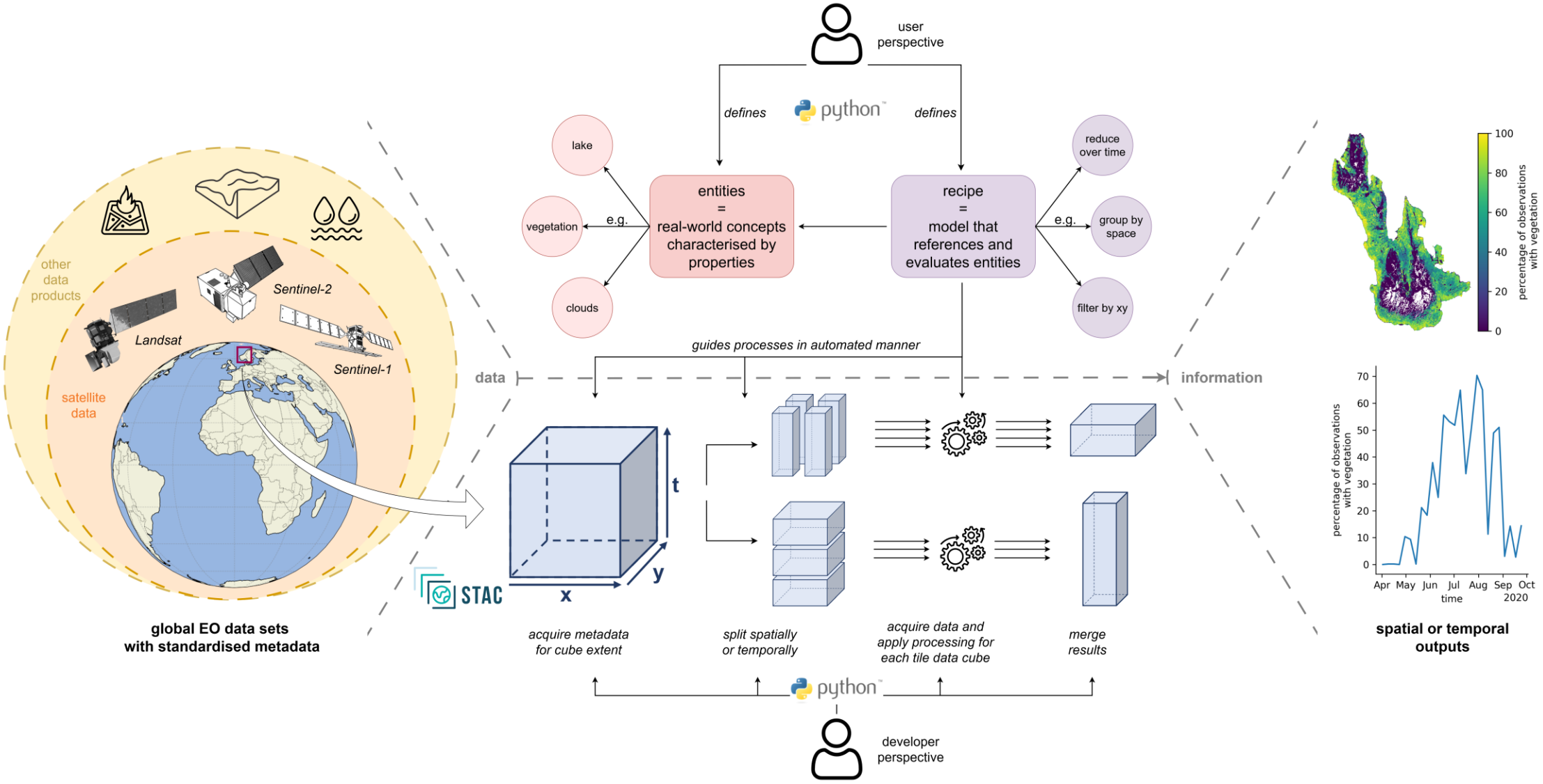
- all data cubes are represented by their layout
- layout intended to be created once by software/EO expert
- specific layout attributes exist for individual data cube configurations
- based on a Python dictionary & stored as a JSON-structured file

```json
{
    "reflectance": {
        "s2_band01": {
            "name": "coastal",
            "description": "Coastal aerosol band (~443nm; 60m spatial
            resolution). Top of atmosphere (TOA) reflectance captured by
            the multi-spectral instrument (MSI) of Sentinel-2A or
            Sentinel-2B as band 1.",
            "type": "continuous",
            "values": {
                "min": 0,
                "max": 1,
                "precision": 1
            },
            "dtype": "float32",
            "na_value": "NA",
            "copyright": "Contains modified Copernicus data."
        },
        "s2_band02": {
            "name": "..."
```

layout.json (STACCube)

```python
import pystac
import semantique as sq
from pystac_client import Client
from shapely.geometry import box

# define temporal & spatial range to perform STAC query
xmin, ymin, xmax, ymax = -2.75, 47.25, -2.25, 47.75
aoi = box(xmin, ymin, xmax, ymax)
t_range = ["2020-07-15", "2020-08-01"]

# STAC-based metadata retrieval
catalog = Client.open("https://earth-search.aws.element84.com/v1")
query = catalog.search(
    collections="sentinel-2-l2a",
    datetime=t_range,
    limit=100,
    intersects=aoi
)
item_coll = query.item_collection()

# define datacube
with open("files/layout_stac.json", "r") as file:
    dc = sq.datacube.STACCube(
        json.load(file),
        src = item_coll
    )
```

Gsemantique – Towards on-demand cubes



Gsemantique: On-demand EO data cubes

Core functionalities (I)

✓ Pre-configured access to a variety of EO datasets with global coverage (e.g. Sentinel-1, Sentinel-2, Landsat)

provider	collection	category	temporality
Planetary	sentinel-1-rtc	SAR	s
Planetary	sentinel-2-l2a	multispectral	s
Planetary	landsat-c2-l2	multispectral	s
Planetary	esa-worldcover	landcover	Y
Planetary	io-lulc-annual-v02	landcover	Y
Planetary	nasadem	DEM	None
Planetary	cop-dem-glo-30	DSM	None
Planetary	modis-64A1-061	fire detection	M
Planetary	modis-14A2-061	fire detection	D
Planetary	jrc-gsw	hydrogeography	None
Element84	sentinel-2-l2a	multispectral	s
ASF	sentinel-1-global-coherence	SAR	3M
ASF	glo-30-hand	hydrogeography	None

Gsemantique: On-demand EO data cubes

Core functionalities (I)

✓ Pre-configured access to a variety of EO datasets with global coverage (e.g. Sentinel-1, Sentinel-2, Landsat)

semantique

```python
import pystac
import semantique as sq
from pystac_client import Client
from shapely.geometry import box

# define temporal & spatial range to perform STAC query
xmin, ymin, xmax, ymax = -2.75, 47.25, -2.25, 47.75
aoi = box(xmin, ymin, xmax, ymax)
t_range = ["2020-07-15", "2020-08-01"]

# STAC-based metadata retrieval
catalog = Client.open("https://earth-search.aws.element84.com/v1")
query = catalog.search(
    collections="sentinel-2-l2a",
    datetime=t_range,
    limit=100,
    intersects=aoi
)
item_coll = query.item_collection()

# define datacube
with open("files/layout_stac.json", "r") as file:
    dc = sq.datacube.STACCube(
        json.load(file),
        src = item_coll
    )
```

gsemantique

Layout file

Finder class

```python
import json
import os
import semantique as sq
import gsemantique as gsq
from gsemantique.data.search import Finder

# get layout file
package_dir = os.path.split(gsq.__file__)[0]
layout_path = os.path.join(package_dir, "data", "layout.json")

# search for items in recipe
with open(layout_path, "r") as file:
    dc = sq.datacube.STACCube(
        json.load(file),
        src = [],
    )

fdr = Finder(ds_catalog, t_start, t_end, aoi_bbox)
fdr.search_auto(recipe, mapping, dc)

# define datacube
with open(layout_path, "r") as file:
    dc = sq.datacube.STACCube(
        json.load(file),
        src = fdr.item_coll
    )
```

Gsemantique: On-demand EO data cubes

Core functionalities (II)

✓ Retrieval & storage mechanisms for the data to persist data cube inputs locally

Downloader class

```
# download items that have been found
out_dir = "files/data"
dwn = Downloader(fdr.item_coll, out_dir)
dwn.run()
```

```
landsat-c2-l2 (collection 1/1)
Not enough items to estimate size. Skipping preview run.
Downloading EO data: 168MB [00:18, 9.55MB/s]
```

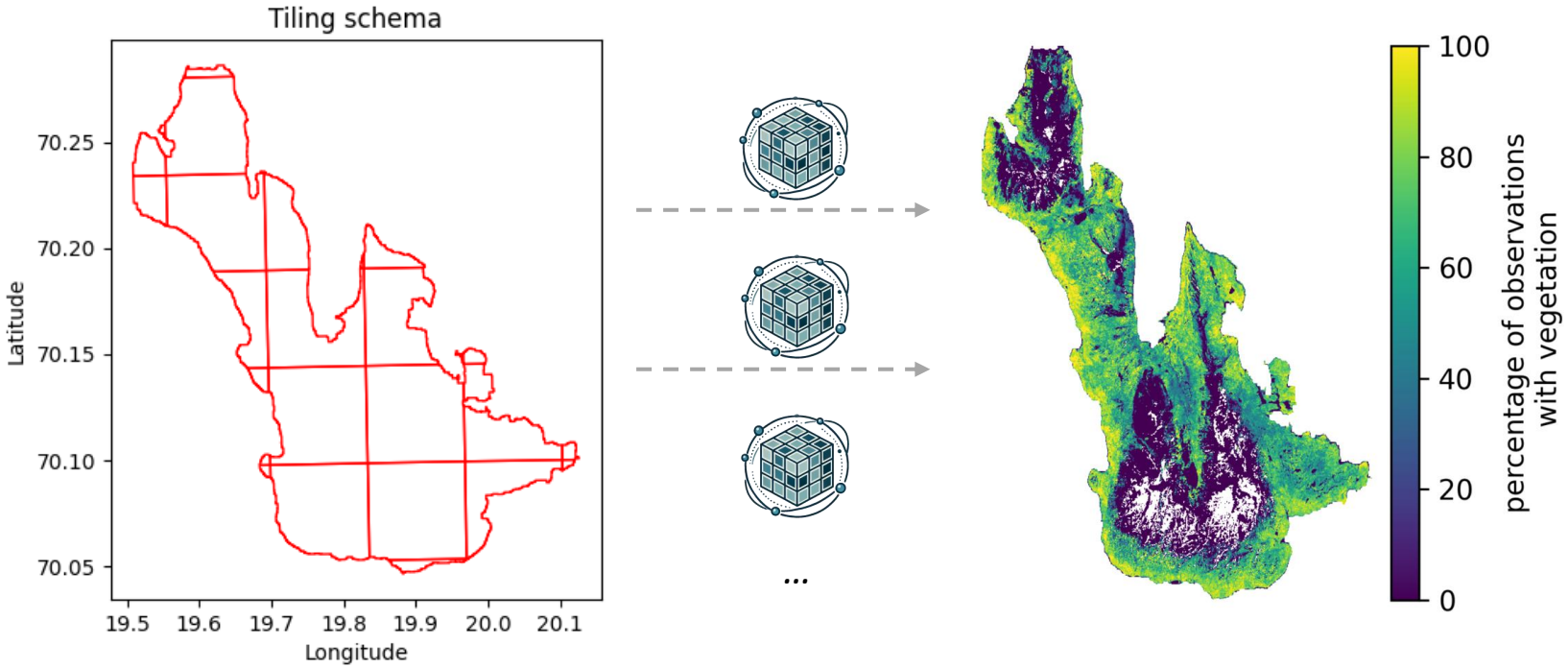
Gsemantique: On-demand EO data cubes

Core functionalities (III)

✓ Scaling mechanisms that allow to evaluate recipes for large spatio-temporal extents up to the mesoscale

- automated analysis of recipe for possible tiling dimension
- sequential execution of recipe & merging of results aftwerwards

e.g. for spatial tiling



Gsemantique: On-demand EO data cubes

Core functionalities (III)

✓ Scaling mechanisms that allow to evaluate recipes for large spatio-temporal extents up to the mesoscale

semantique

```
context = {
    "datacube": dc,
    "mapping": mapping,
    "space": space,
    "time": time,
    "crs": 3035,
    "tz": "UTC",
    "spatial_resolution": [-10, 10]
}

response = recipe.execute(**context)
```

gsemantique

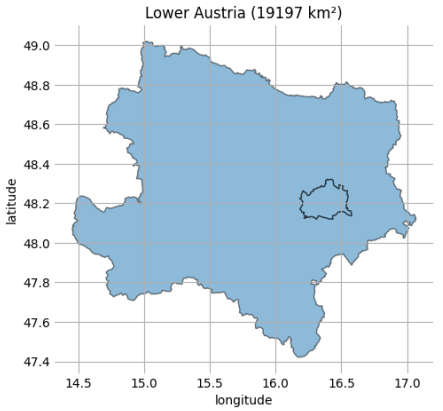
```
# create TileHandler & execute recipe
context = {
    "recipe": recipe,
    "datacube": dc,
    "mapping": mapping,
    "space": space,
    "time": time,
    "crs": 3035,
    "tz": "UTC",
    "spatial_resolution": [-10, 10],
    "chunksize_s": 1024,
    "tile_dim": None,
    "merge_mode": 'merged',
    "out_dir": None,
    "reauth": True,
}
```

optional arguments

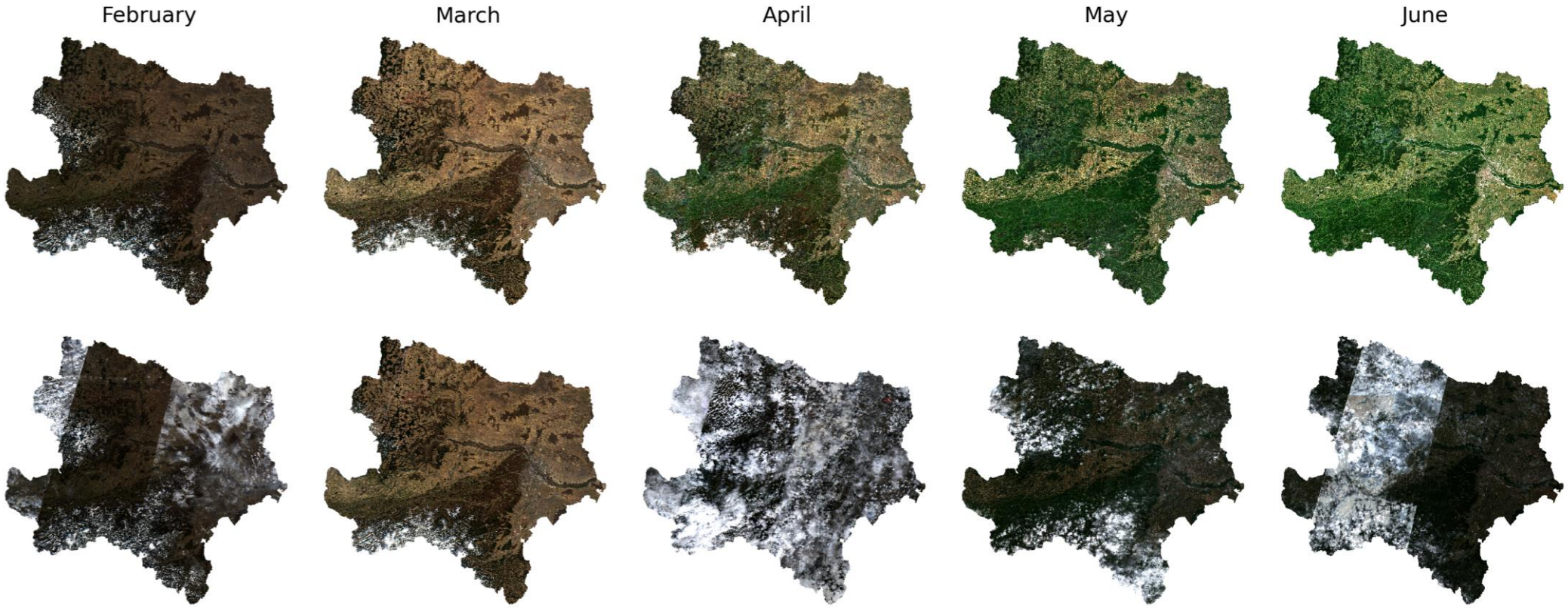
TileHandler class

```
th = TileHandler(**context)
th.execute()
```

What can semantic, on-demand EO cubes be used for?



Cloud-free composites



What can semantic, on-demand EO cubes be used for?

Forest disturbance detection

–

Model comparisons