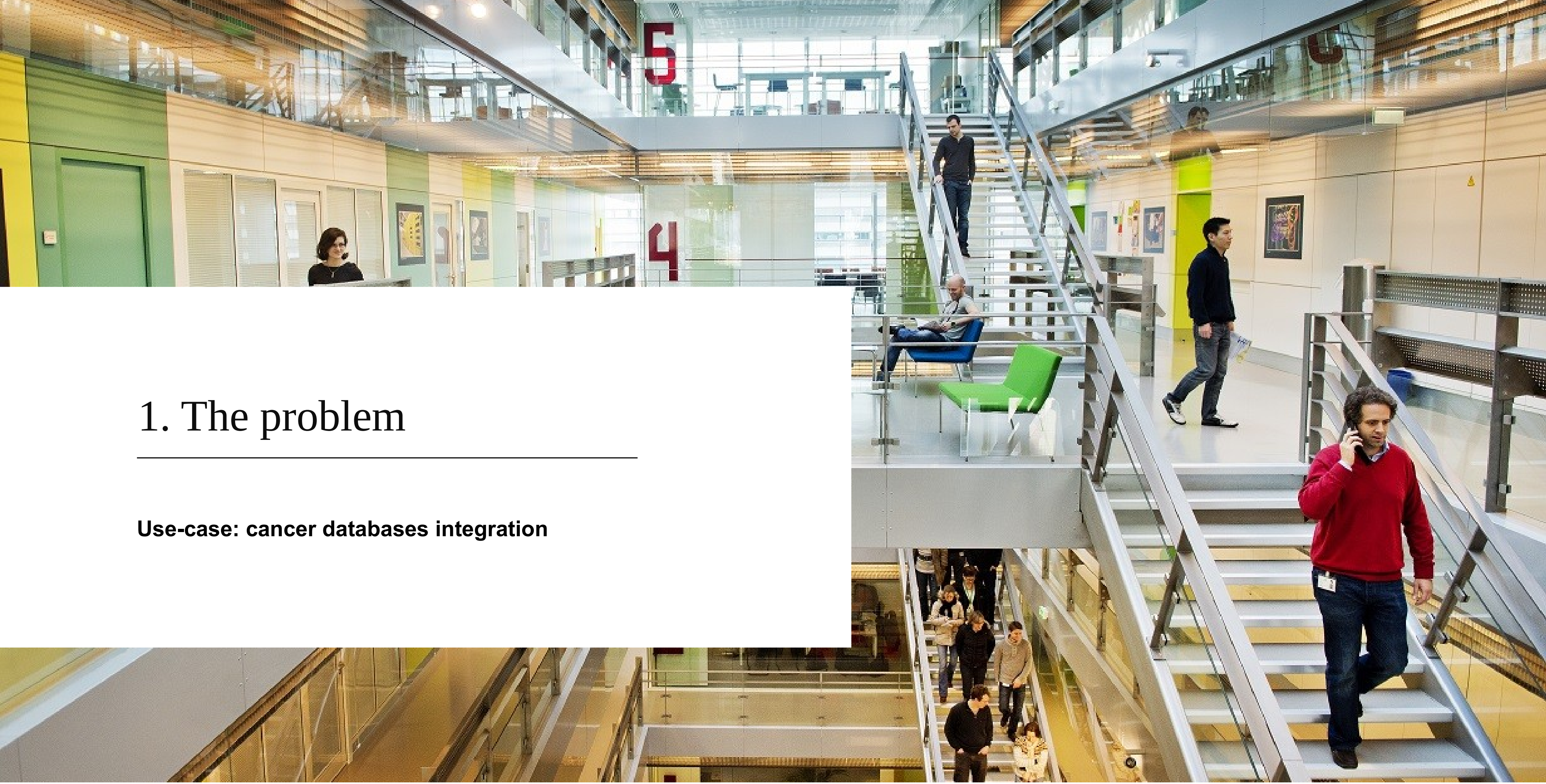
INSTITUT PASTEUR

Johann Dreo, Claire Laudy, Sebastian Lobentanzer, Marko Baric,
Ekaterina Gaydukova, Matthieu Najm, Benno Schwikowski

Intégration reproductible d'informations de haut niveau dans des graphes de connaissances sémantiques avec OntoWeaver et BioCypher, application en oncologie et en écologie

Reproducible Mapping of Tabular Data into Semantic Knowledge Graphs with OntoWeaver and BioCypher, application in oncology and ecology

Summary



1. The problem

Use-case: cancer databases integration

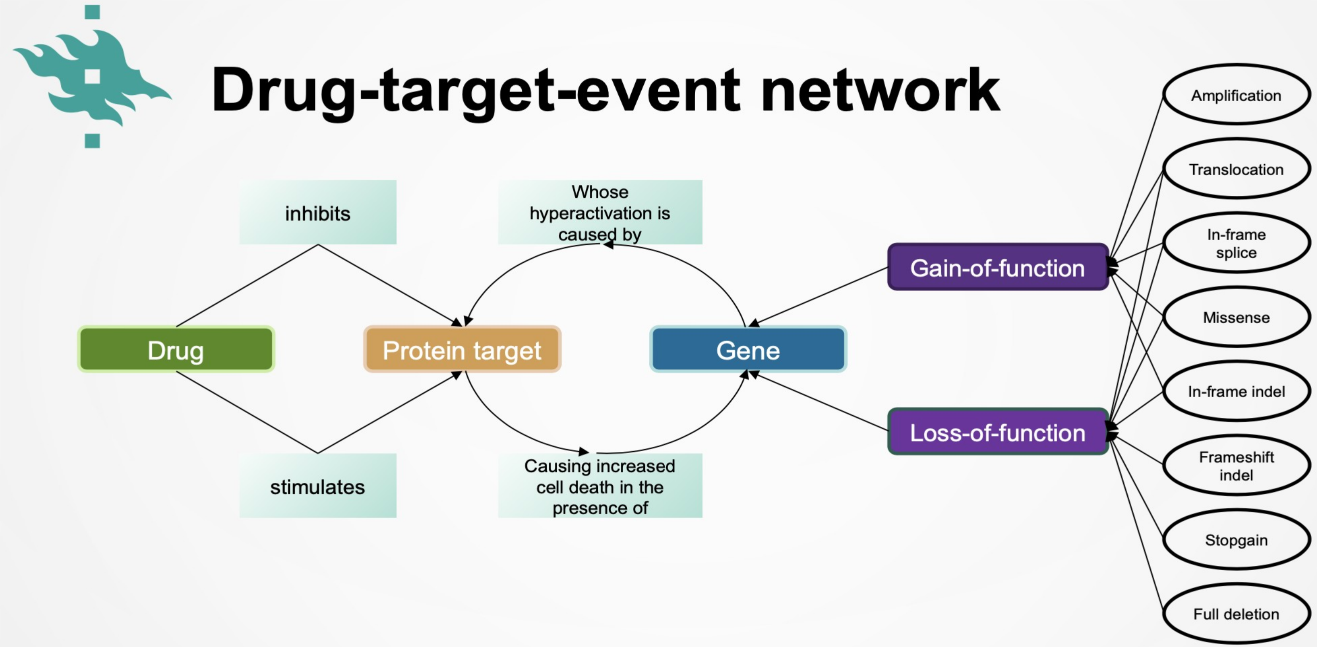
Example Problem

- H2020 *Decider* project

  « develop diagnostic **tools** […] for high-grade serous ovarian **cancer** […] **find effective treatments** to patients with a drug-resistant cancer »

- Oncodash [https://github.com/oncodash]
  - Clinical oncology **tumor board decision support system**
    - End-user = **geneticists**, clinicians
      - Find **drugs actionable** on **patient's alterations**

Actionable drugs



What we search

What we know

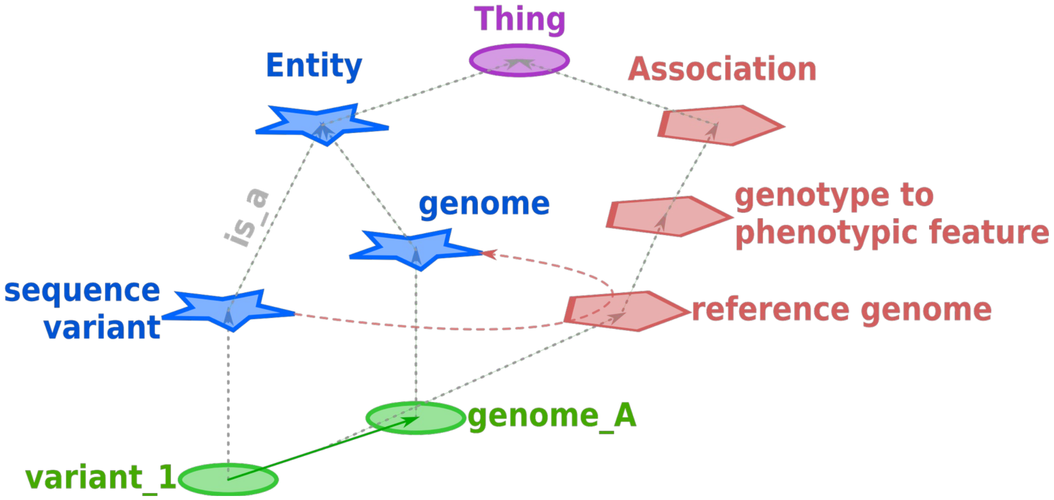
Courtesy of Tary Muranen, research geneticist,
Research Program in Systems Oncology, Research Programs Unit,
Faculty of Medicine, University of Helsinki, Helsinki, Finland

2. How to

Building up Semantic Knowledge Graphs

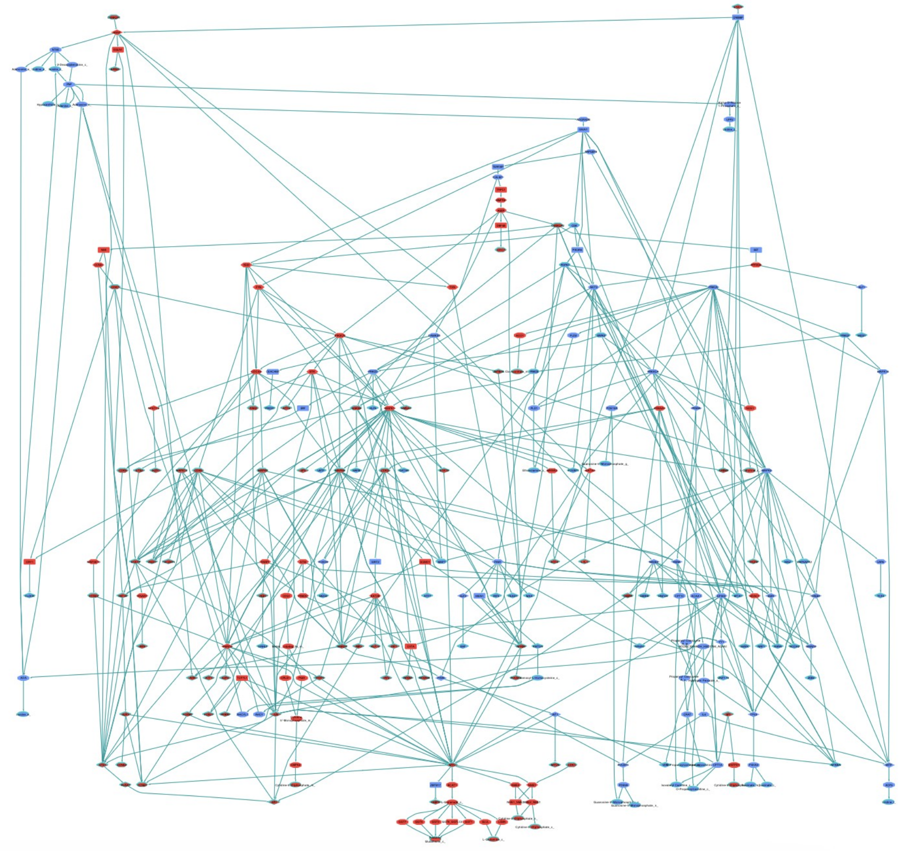
Semantic Knowledge Graphs

- A *graph*
- showing high-level entities (information ≠ data)
- of various *types*



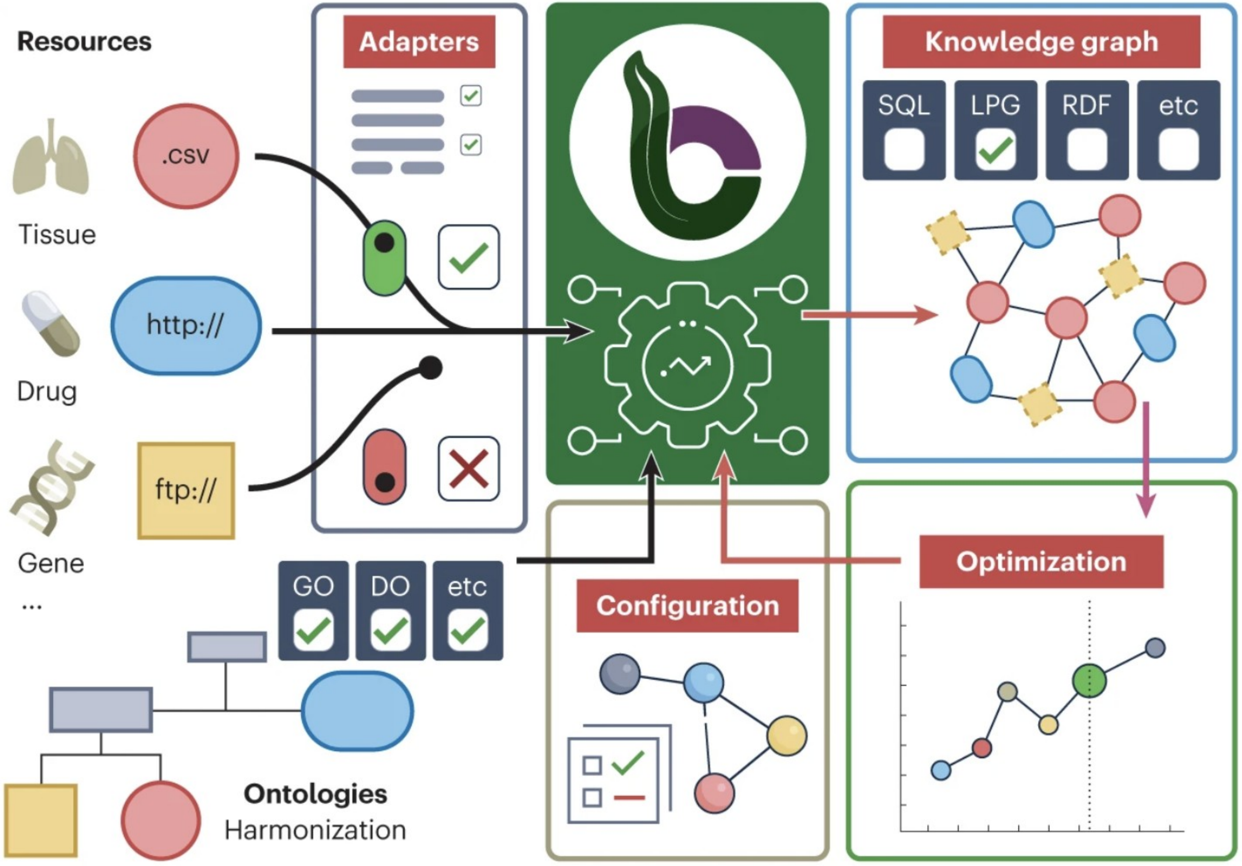
Scalability?

- Biological knowledge is huge.
- The graph needs to be designed specifically for each use-case.
- Doing so by hand cannot scale.
- We need reproducibility tooling.



From **Causal integration of multi-omics data with prior knowledge to generate mechanistic hypotheses**
Aurelien Dugourd et al. Mol Syst Biol (2021) 17: e9730
Causal network summarising the mechanistic hypotheses systematically generated by CARNIVAL. The network comprises 449 edges. It represents the propagation of signals connecting the deregulated kinases, phosphatases, TFs and metabolites in kidney cancer.

biocypher
a unifying framework for biomedical knowledge graphs



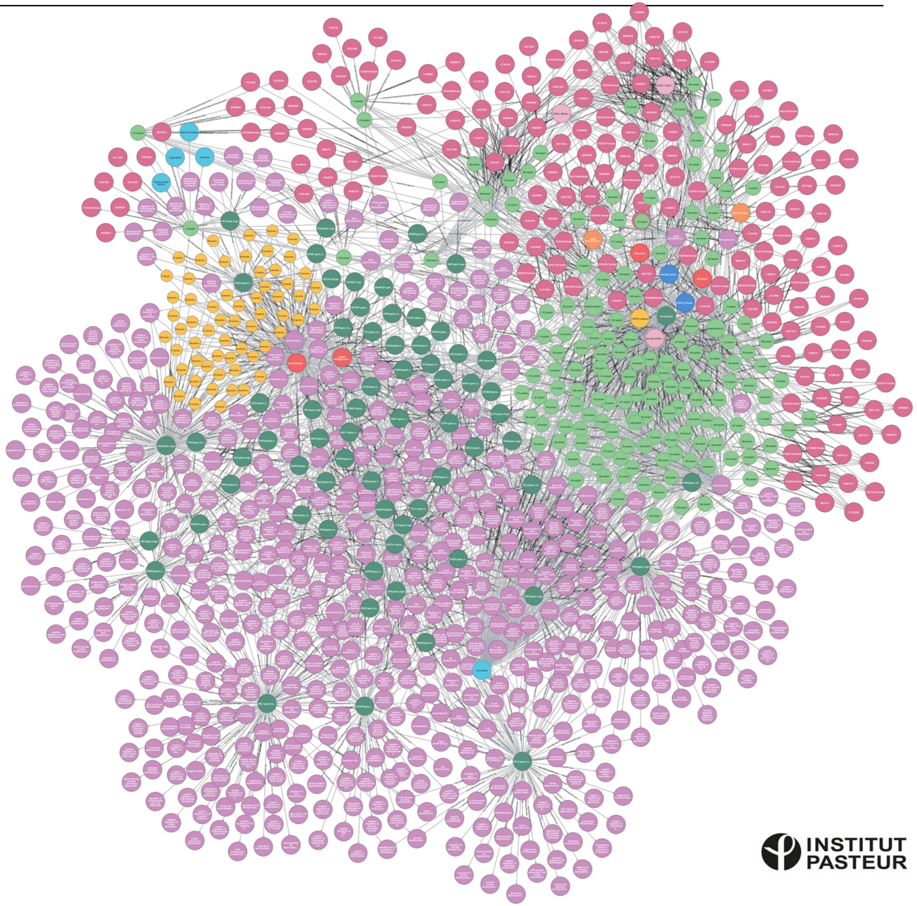
Democratizing knowledge representation with BioCypher

Sebastian Lobentanzer, ..., Johann Dreo, ..., Benno Schwikowski, ..., et al.

Nature Biotechnology
volume 41, pp. 1056–1059 (2023).

FAIR SKG building

- Automated “adapters” scripts
  - Findable
  - Accessible
  - Interoperable
  - Reproducible
- Make a tailored SKG = enable a set of adapters.



BioCypher, but...

- An adapter is a Python script:
    - allow accessing any kind of data,
    - requires programming skills.

```python
    def get_nodes(self):
        """
        Returns a generator of node tuples for node types specified in the
        adapter constructor.
        """

        logger.info("Generating nodes.")

        # get unique patients as list
        patients = (
            self.patients[[AllVariantsAdapterPatientField.ID.value]]
            .drop_duplicates()[AllVariantsAdapterPatientField.ID.value]
            .tolist()
        )

        # get unique samples
        samples = (
            self.variants[[AllVariantsAdapterSampleField.ID.value]]
            .drop_duplicates()[AllVariantsAdapterSampleField.ID.value]
            .tolist()
        )

        # yield tuples for patients
        for patient in patients:
            yield (
                patient,
                "patient",
                {},
            )

        # yield tuples for samples
        for sample in samples:
            yield (
                sample,
                "sample",
                {},
            )

        # VARIANTS: for each node (row), yield a 3-tuple of node id (the 'ID'
        # column), node label (hardcode to 'variant' for now), and node
        # properties (dict of column names and values, except the 'ID')

        # first remove columns in _drop_columns and drop duplicates
        unique_variants = self.variants.drop(
            self._drop_columns, axis=1
        ).drop_duplicates()

        for _, node in unique_variants.iterrows():
            yield (
                node["ID"],
                "variant",
                node.drop("ID").to_dict(),
            )

    def get_edges(self):
        """
        Returns a generator of edge tuples for edge types specified in the
        adapter constructor.
        """

        logger.info("Generating edges.")

        # PATIENT - SAMPLE
        # yield 5-tuple of edge id (hash of patient and sample ids), source
        # node id, target node id, edge label (hardcode to 'patient_has_sample'
        # for now), and edge properties (empty dict for now)
        for _, row in self.patients.iterrows():
            p_id = row[AllVariantsAdapterPatientField.ID.value]
            s_id = row[AllVariantsAdapterSampleField.ID.value]
            _id = hashlib.md5((p_id + s_id).encode("utf-8")).hexdigest()
            yield (
                _id,
                p_id,
                s_id,
                "patient_has_sample",
                {},
            )

        # SAMPLE - VARIANT
        # yield 5-tuple of edge id (hash of sample and variant ids), source node
        # id, target node id, edge label (hardcode to 'sample_has_variant' for
        # now), and edge properties (empty dict for now)
        for _, row in self.variants.iterrows():
            s_id = row[AllVariantsAdapterSampleField.ID.value]
            v_id = row[AllVariantsAdapterVariantField.ID.value]
            _id = hashlib.md5((s_id + v_id).encode("utf-8")).hexdigest()
            yield (
                _id,
                s_id,
                v_id,
                "sample_has_variant",
                {},
            )

        # VARIANT - GENE
        # yield 5-tuple of edge id, source node id, target node id, edge label
        # (hardcode to 'variant_in_gene' for now), and edge properties (empty
        # dict for now)

        # first remove all columns except the variant id and gene and
        # deduplicate
        unique_variant_gene = self.variants[
            [
                AllVariantsAdapterVariantField.ID.value,
                "Gene",  # expanded above
            ]
        ].drop_duplicates()

        # remove all 'NONE' genes
        unique_variant_gene = unique_variant_gene[
            unique_variant_gene["Gene"] != "NONE"
        ]

        for _, row in unique_variant_gene.iterrows():
            v_id = row[AllVariantsAdapterVariantField.ID.value]
            g_id = f"hgnc:{row['Gene']}"
            _id = hashlib.md5((v_id + g_id).encode("utf-8")).hexdigest()
            yield (
                _id,
                v_id,
                g_id,
                "variant_in_gene",
                {},
            )
```

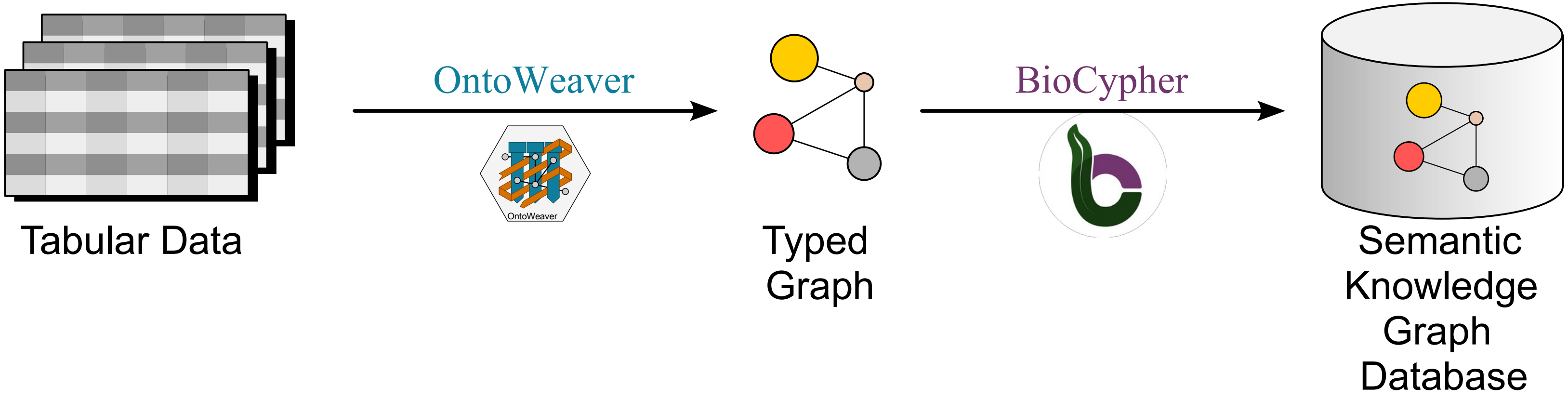
- However:
    - most of our data is tabular in nature,
    - we would want end-users to participate.

3. Our solution

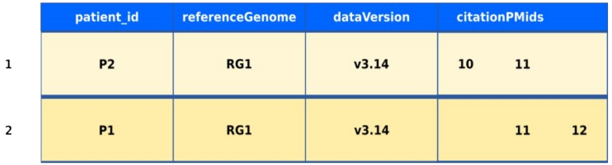
Declarative mapping of tabular data

OntoWeaver[https://github.com/oncodash/ontoweaver]

- Transmogrify *tabular data* into a *SKG*
- Using a (simple) “mapping” declaration



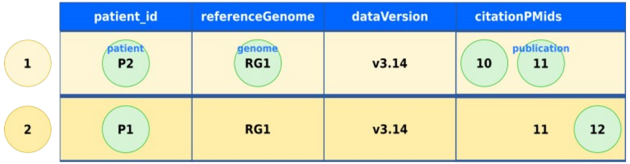
Given a table, ...



	patient_id	referenceGenome	dataVersion	citationPMids
1	P2	RG1	v3.14	10 11
2	P1	RG1	v3.14	11 12

```
row:
  rowIndex:
    to_subject: variant
metadata:
  source: "OntoWeaver"
transformers:
  - map:
    columns:
      - patient_id
    to_object: patient
    via_relation: patient_has_variant
  - split:
    columns:
      - citationPMids
    to_object: publication
    via_relation: reference_genome
    separator: ","
  - map:
    columns:
      - dataVersion
    to_property:
      - version
    for_objects:
      - genome
```

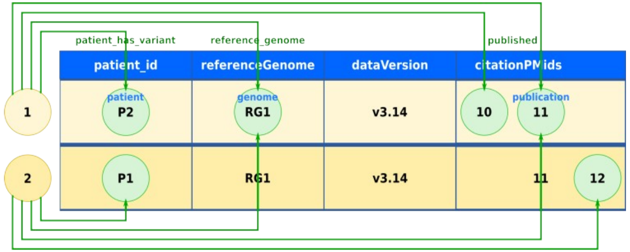
.. map columns to nodes, ...



	patient_id	referenceGenome	dataVersion	citationPMids
1	P2	RG1	v3.14	10 11
2	P1	RG1	v3.14	11 12

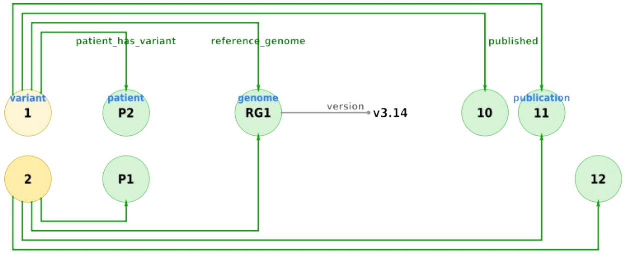
```
row:
  rowIndex:
    to_subject: variant
metadata:
  source: "OntoWeaver"
transformers:
  - map:
    columns:
      - patient_id
    to_object: patient
    via_relation: patient_has_variant
  - split:
    columns:
      - citationPMids
    to_object: publication
    via_relation: reference_genome
    separator: ","
  - map:
    columns:
      - dataVersion
    to_property:
      - version
    for_objects:
      - genome
```

… link subject and nodes with edges, …

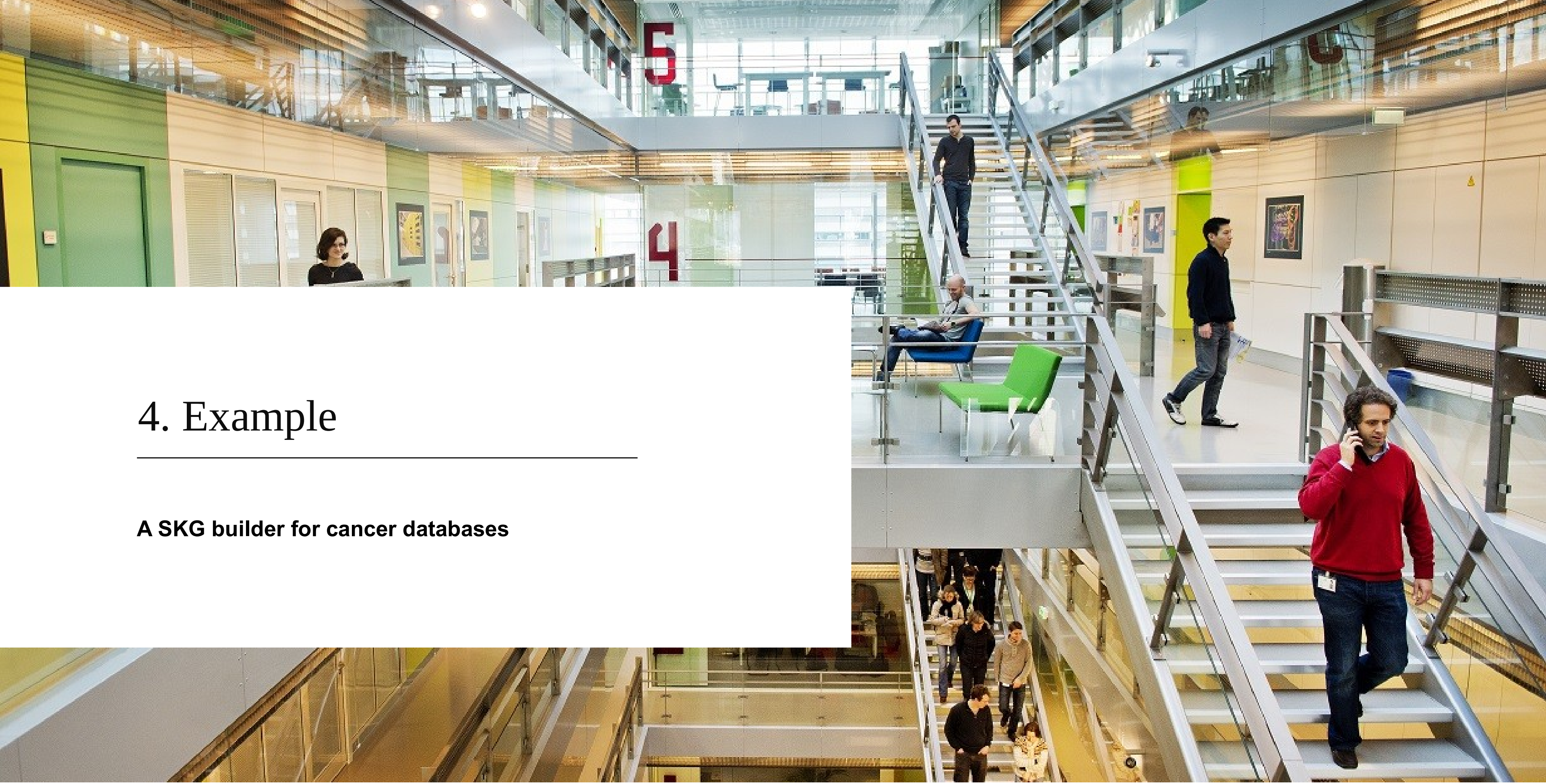


```
row:
  rowIndex:
    to_subject: variant
metadata:
  source: "OntoWeaver"
transformers:
  - map:
    columns:
      - patient_id
    to_object: patient
    via_relation: patient_has_variant
  - split:
    columns:
      - citationPMids
    to_object: publication
    via_relation: reference_genome
    separator: ","
  - map:
    columns:
      - dataVersion
    to_property:
      - version
    for_objects:
      - genome
```

… you have a typed graph.



```
row:
  rowIndex:
    to_subject: variant
metadata:
  source: "OntoWeaver"
transformers:
  - map:
    columns:
      - patient_id
    to_object: patient
    via_relation: patient_has_variant
  - split:
    columns:
      - citationPMids
    to_object: publication
    via_relation: reference_genome
    separator: ","
  - map:
    columns:
      - dataVersion
    to_property:
      - version
    for_objects:
      - genome
```



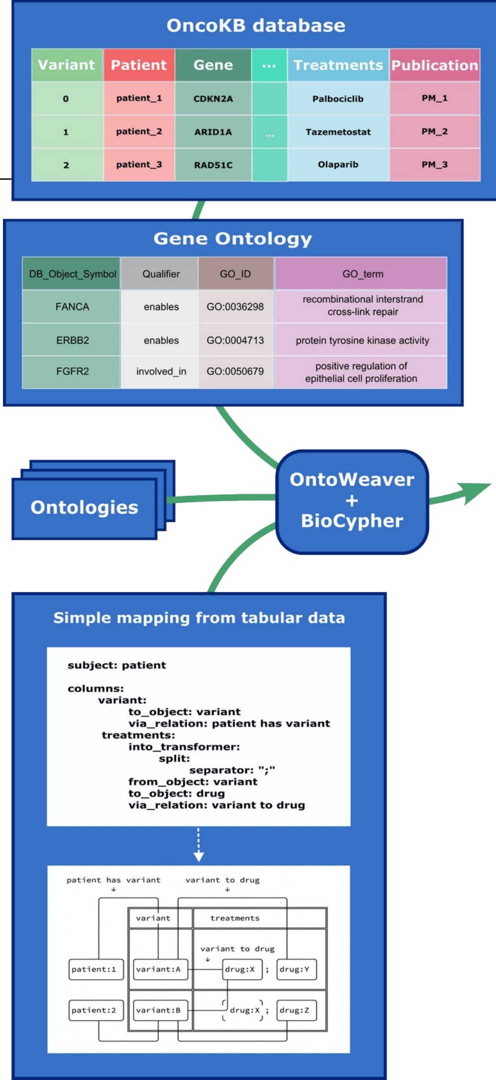
4. Example

A SKG builder for cancer databases

Example

- OncodashKB
  - https://gihutb.com/oncodash/oncodashkb
  - integrate **cancer databases** (OncoKB, CGI)
  - with Gene Ontology

High-level Biomedical Data Integration in a Semantic Knowledge Graph with OncodashKB for finding Personalized Actionable Drugs in Ovarian Cancer, J. Dreo et al., *10th EACR conference on Cancer Genomics, Multiomics and Computational Biology*, Bergamo (2024).



OncoKB database

Variant	Patient	Gene	…	Treatments	Publication
0	patient_1	CDKN2A		Palbociclib	PM_1
1	patient_2	ARID1A	…	Tazemetostat	PM_2
2	patient_3	RAD51C		Olaparib	PM_3

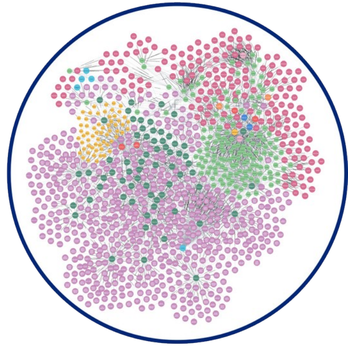
Gene Ontology

DB_Object_Symbol	Qualifier	GO_ID	GO_term
FANCA	enables	GO:0036298	recombinational interstrand cross-link repair
ERBB2	enables	GO:0004713	protein tyrosine kinase activity
FGFR2	involved_in	GO:0050679	positive regulation of epithelial cell proliferation

Simple mapping from tabular data

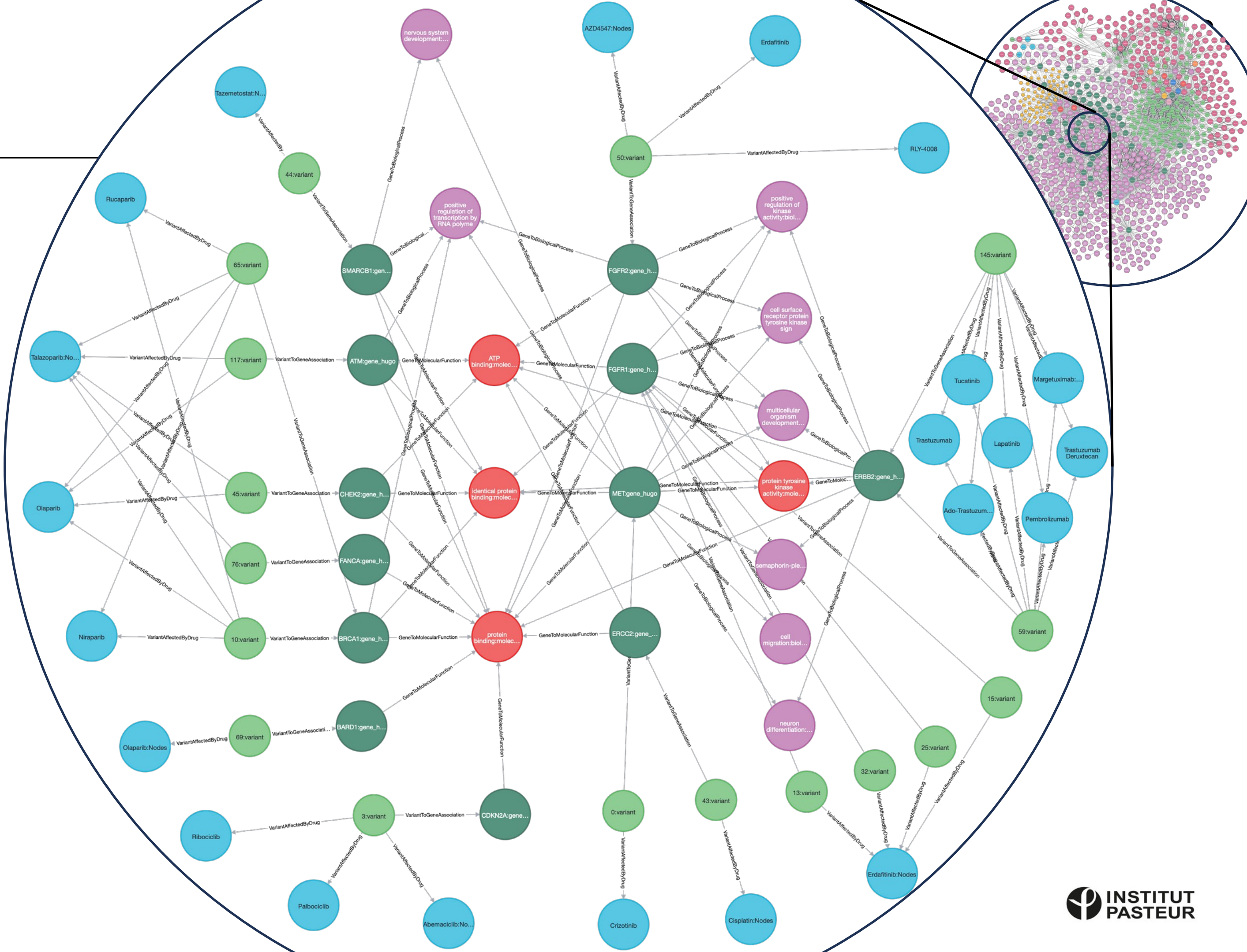
```
subject: patient

columns:
        variant:
                to_object: variant
                via_relation: patient has variant
         treatments:
                into_transformer:
                        split:
                                separator: ";"
                from_object: variant
                to_object: drug
                via_relation: variant to drug
```



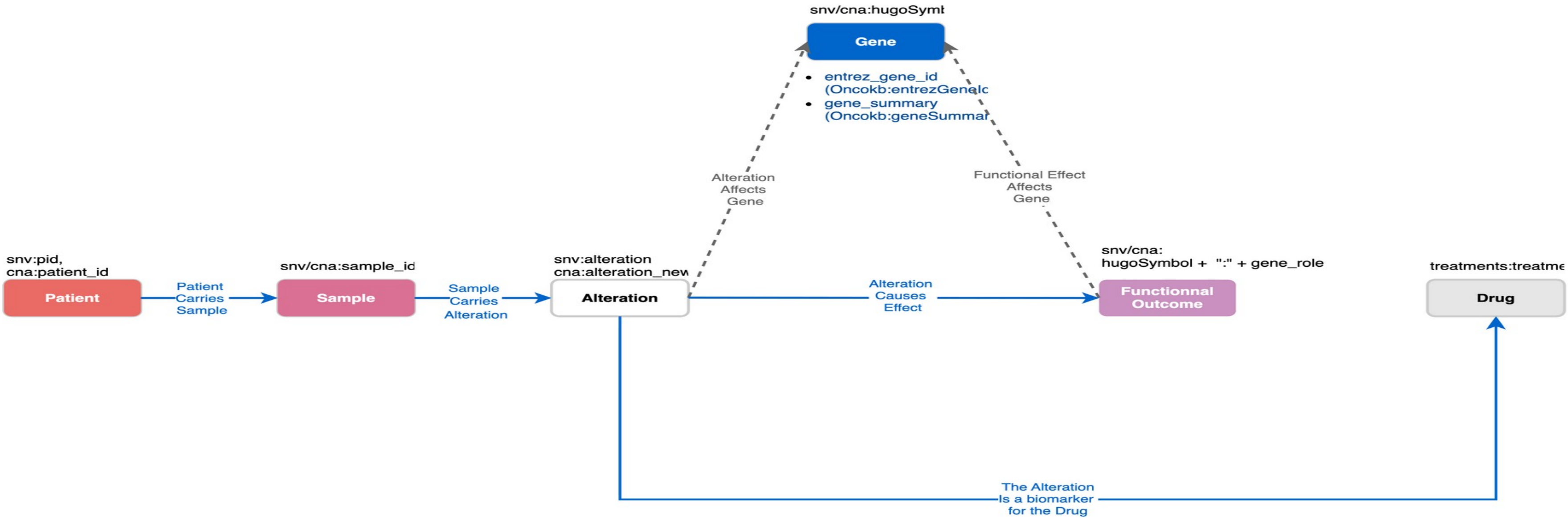
Example

- Queries on SKG.
- Paths between alterations and drugs, *through new pathways*.
- With all the supporting *evidences*.



INSTITUT PASTEUR

OncodashKB information integration



snv/cna:hugoSymb

Gene

- entrez_gene_id (Oncokb:entrezGeneIc
- gene_summary (Oncokb:geneSummar

Alteration Affects Gene

Functional Effect Affects Gene

snv:pid,
cna:patient_id

Patient

Patient Carries Sample

snv/cna:sample_id

Sample

Sample Carries Alteration

snv:alteration
cna:alteration_new

Alteration

Alteration Causes Effect

snv/cna:
hugoSymbol + ":" + gene_role

Functionnal Outcome

treatments:treatme

Drug

The Alteration Is a biomarker for the Drug

5. Conclusion

Just try it

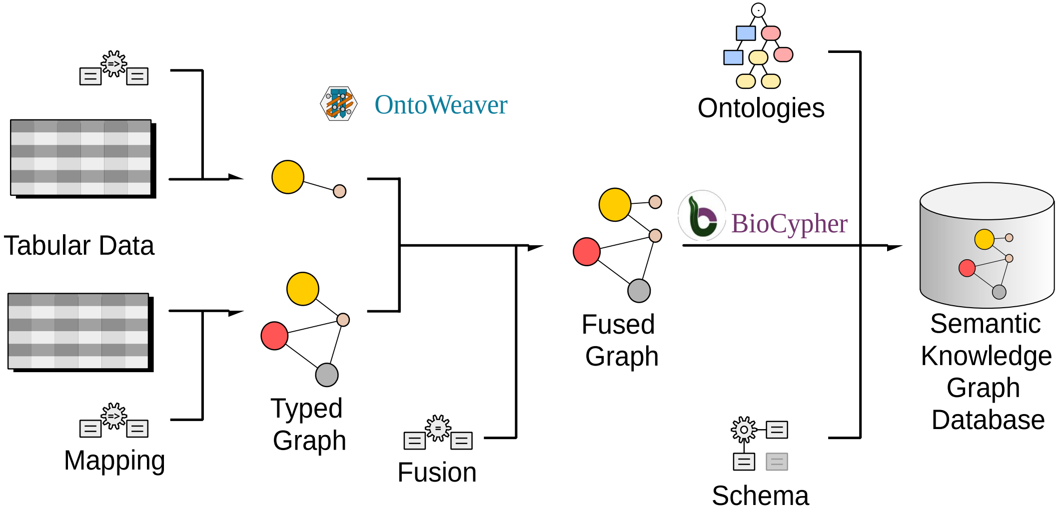
Conclusion

- OntoWeaver:
  - FLOSS software in Python.
  - https://github.com/oncodash/ontoweaver
  - Maps **tabular data** into **Semantic Knowledge Graphs**.
  - Brings **FAIR** SKG building blocks.
  - Relatively **easy** to setup and maintain.
  - Allows to build up **tailored SKG** as needed.

Perspectives

- High-level Information Fusion

```
• BRCA2    {p1:x}
• 139618   {p1:y     , p2:z}
=> BRCA2 {p1:[x,y], p2:z}
```



Reproducible Mapping of Tabular Data into Semantic Knowledge Graphs with OntoWeaver and BioCypher, application in oncology and ecology

https://github.com/oncodash/ontoweaver

johann.dreo@pasteur.fr

https://research.pasteur.fr/en/team/csb/

- **Democratizing knowledge representation with BioCypher,** S. Lobentanzer *et al.*, *Nature Biotechnology*, vol. 41, pp. 1056–1059 (2023).
- **High-level Biomedical Data Integration in a Semantic Knowledge Graph with OncodashKB for finding Personalized Actionable Drugs in Ovarian Cancer**, J. Dreo *et al.*, *10th EACR conference on Cancer Genomics, Multiomics and Computational Biology*, Bergamo (2024).
- **Reproducible Mapping of Tabular Data into Semantic Knowledge Graphs with OntoWeaver and BioCypher**, J. Dreo *et al.*, *27th International Conference on Information Fusion*, Venizia (2024).

Institut Pasteur
25-28 rue du docteur Roux
75724 Paris Cedex 15