

Enhancing Trust in AI-Driven Healthcare: Transparent Data Use with Semantics

Pol Nachtergaele – PhD Student @ KNoWS UGent



Pol.Nachtergaele@UGent.be



+32 497 27 92 75

Table of Contents

➤ Context

- AI-Driven Healthcare in a solid environment

➤ Problem

- Why is trust important in AI-Driven Healthcare

➤ Solution

➤ Implementation

Patient data comes from many sources

IoT: Wearables, Smart scales, ...



GP, Physiotherapist, ...



Hospital-based HCPs



Consolidate data in personal data pod

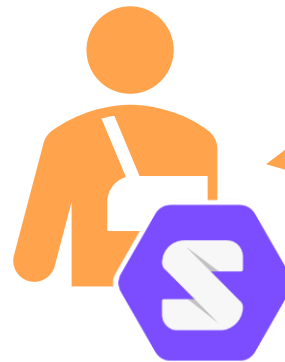
IoT, Wearables, Smart home sensors, ...



GP, Physiotherapist, ...



Hospital-based HCPs



How do we use this data?

AI-Driven Healthcare

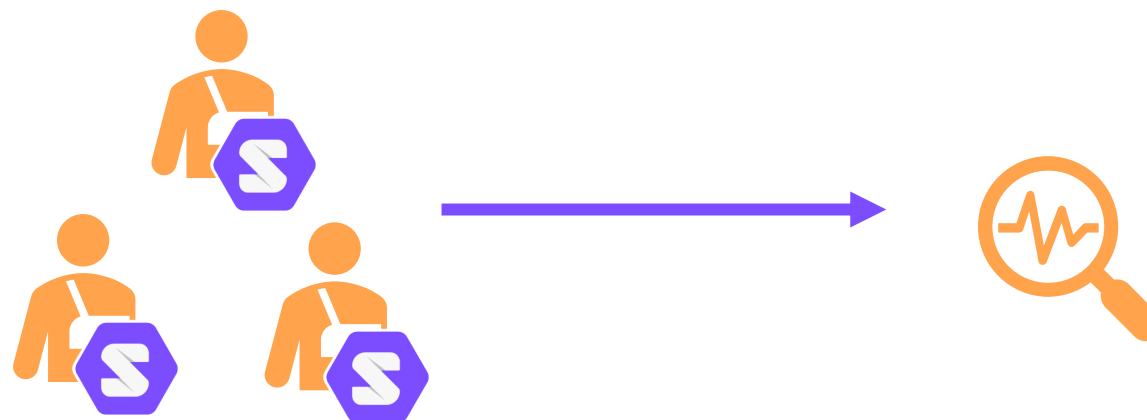
Primary use: direct patient care

- Diagnosis
- Treatment eligibility
- Remote patient monitoring
- ...



Secondary use: beyond direct care

- Medical Research
- Policy-making
- General trends
- ...



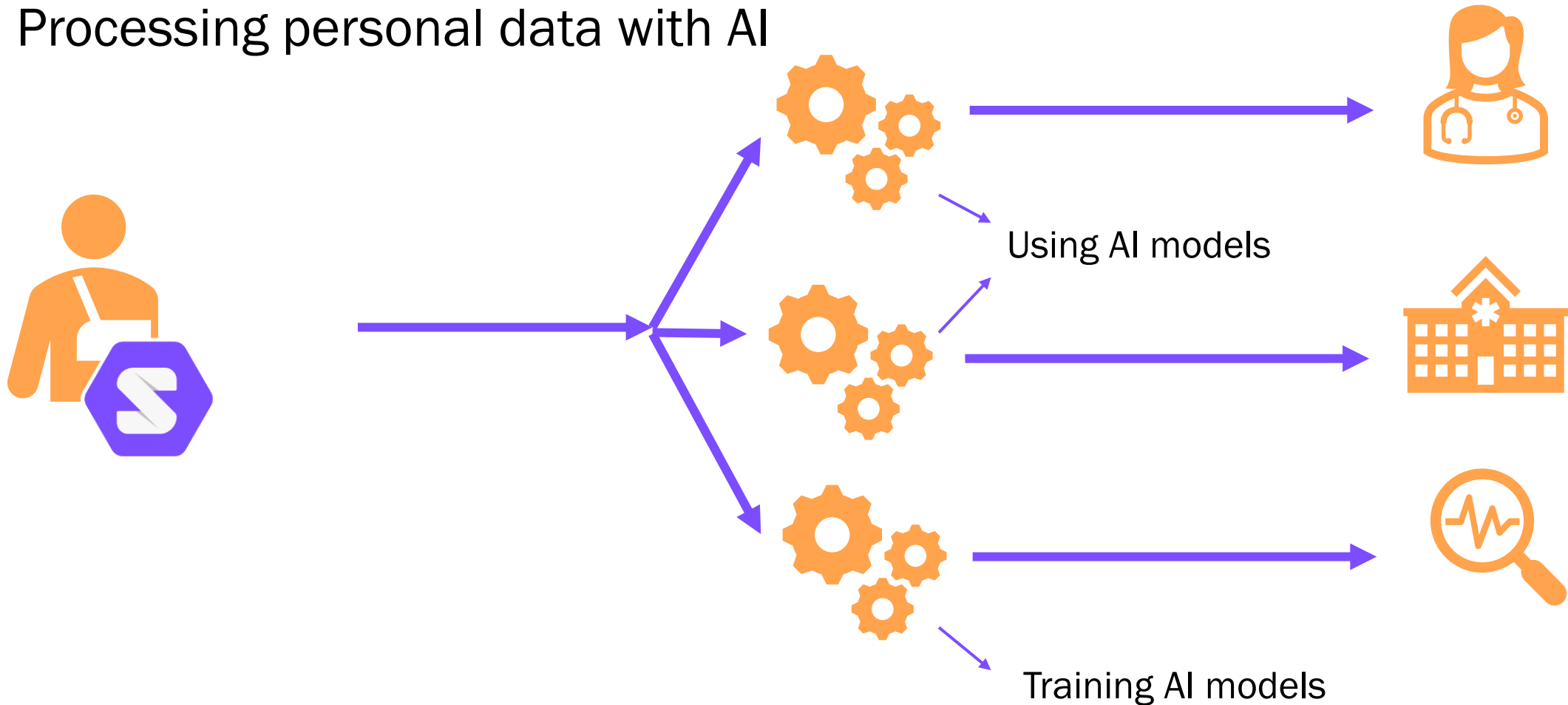
AI-Driven Healthcare

- Many data stakeholders



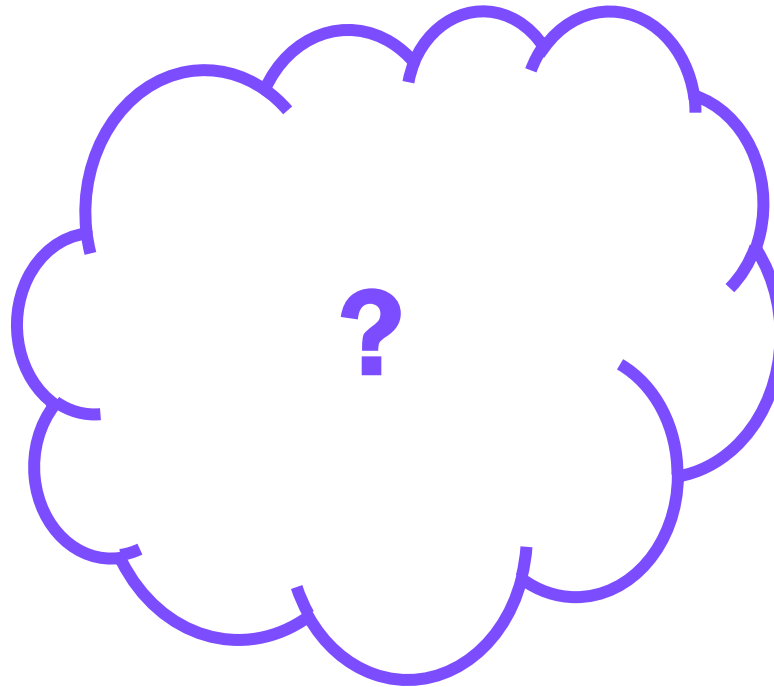
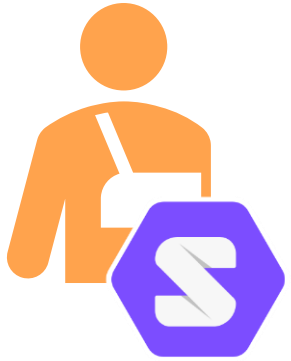
AI-Driven Healthcare

- Many data stakeholders
- Processing personal data with AI



Problem

- Many interested data stakeholders
- Processing personal data with AI



Can I use your data?

How do I trust AI models?
How do I trust results?



How do I exercise my rights?
How do I trust other parties?

Problem

- Without trust:
 - Processed results are meaningless
 - Data doesn't get used
- Increase trust with **transparency**:
 - What will you do with my data?
 - Where does data come from?
 - How was data processed?

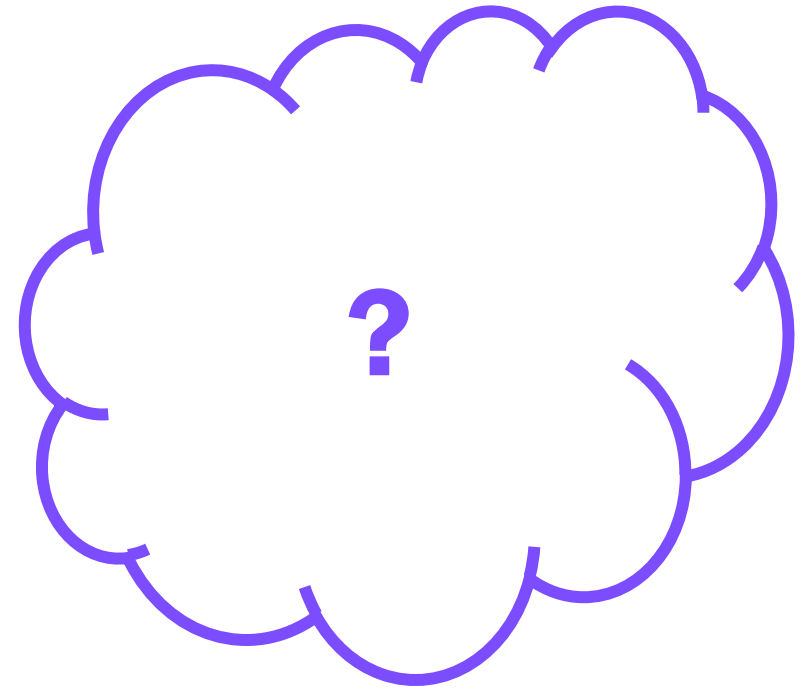
Solution

- Demystify the processing pipeline:
 - Heterogenous components



- Complicated code

```
for i in people.data.users:
    response = client.api.statuses.user_timeline.get(screen_name=i.screen_name)
    print 'Got', len(response.data), 'tweets from', i.screen_name
    if len(response.data) != 0:
        ldate = response.data[0]['created_at']
        ldate2 = datetime.strptime(ldate, '%a %b %d %H:%M:%S +0000 %Y')
        today = datetime.now()
        howlong = (today-ldate2).days
        if howlong < daywindow:
            print i.screen_name, 'has tweeted in the past', daywindow,
            totaltweets += len(response.data)
            for j in response.data:
                if j.entities.urls:
                    for k in j.entities.urls:
                        newurl = k['expanded_url']
                        urlset.add((newurl, j.user.screen_name))
        else:
            print i.screen_name, 'has not tweeted in the past', daywindow
```






Solution

- Adding a **semantic layer** on top of a processing pipeline
 - Provide **understanding** by describing functionality
 - **Function Ontology (FnO)**
 - Ensure **transparency** by describing how and when data is executed
 - **Provenance Ontology (PROV-O)**




Semantic Annotations

Provide **understanding**

-  FnO Structure
-  From Source Code to FnO
-  Describing Complex Pipelines

Ensure **transparency**

-  PROV-O Structure
-  Execute FnO with Provenance Capture

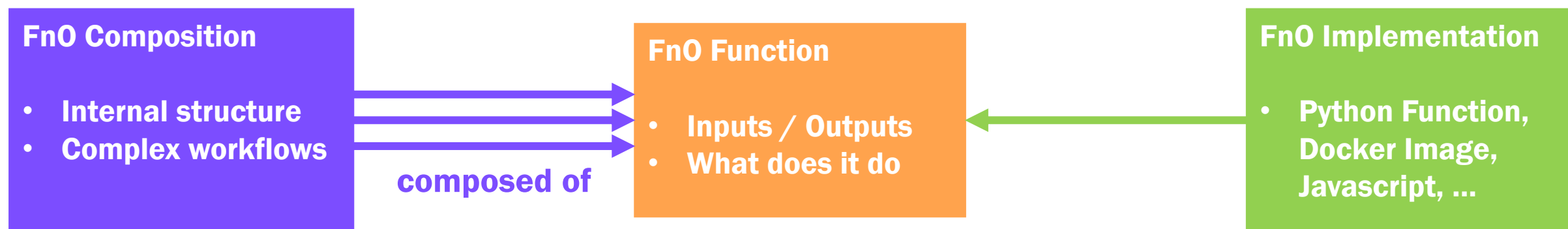
Providing understanding

How to show a person what will be done with their data



FnO Structure

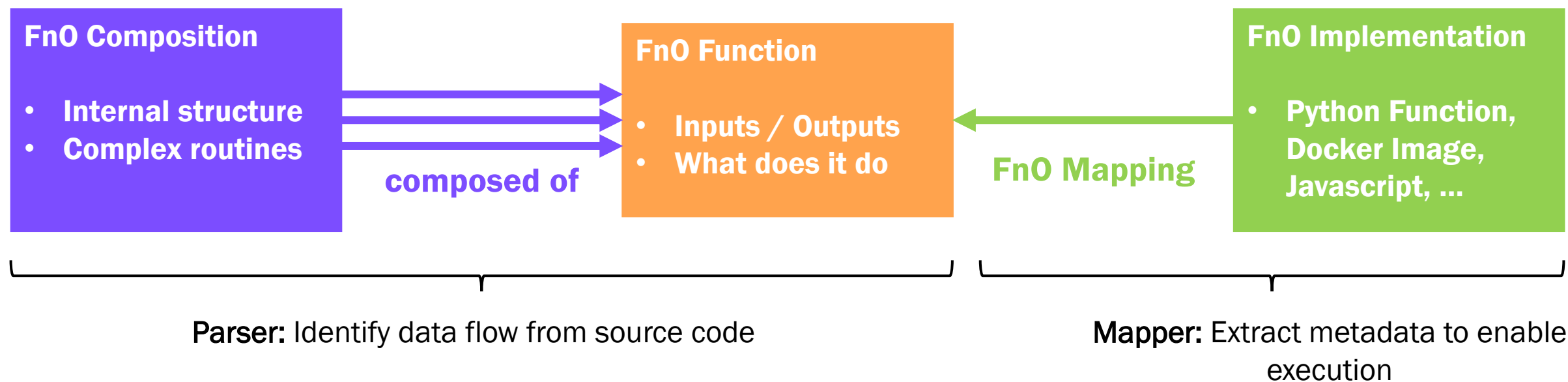
- Describe the data flow independent of the implementation framework
- Three main components:





From Source Code to FnO

- Developers do not “care” about transparency
- Write code → Automatic annotation with FnO → Transparent code





From Source Code to FnO

- Python example
 - Parser: **Identify function calls**

```
def controller(conf, act_dict_inverse, model, data):  
    data = process_data(conf, data)  
    if (data is not None):  
        if (len(data) > conf['min_duration_s'] * conf['frequency']):  
            print('Received enough data to proceed with making predictions')  
            data.set_index("Timestamp", inplace = True)  
  
            fs_dict = {}  
            for name in data.columns:  
                fs_dict[name] = conf['frequency']  
  
            chunks = chunk_data(data=data, fs_dict=fs_dict, min_chunk_dur="15s")
```




From Source Code to Fn0

➤ Python example

➤ Parser: **Identify function calls**

Data flow by connecting variables

```
def controller(conf, act_dict_inverse, model, data):  
    data = process_data(conf, data)  
    if (data is not None):  
        if (len(data) > conf['min_duration_s'] * conf['frequency']):  
            print('Received enough data to proceed with making predictions')  
            data.set_index("Timestamp", inplace = True)  
  
            fs_dict = {}  
            for name in data.columns:  
                fs_dict[name] = conf['frequency']  
  
            chunks = chunk_data(data=data, fs_dict=fs_dict, min_chunk_dur="15s")
```



From Source Code to Fn0

➤ Python example

➤ **Parser:** Identify function calls

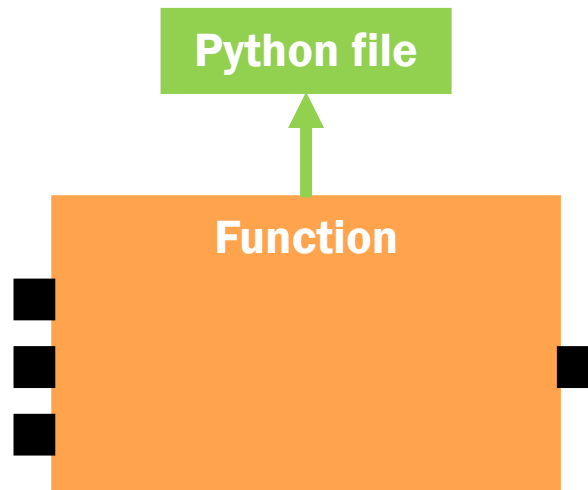
Data flow by connecting variables

➤ **Mapper:** specify file, package, ...

```
def controller(conf, act_dict_inverse, model, data):  
    data = process_data(conf, data)  
    if (data is not None):  
        if (len(data) > conf['min_duration_s'] * conf['frequency']):  
            print('Received enough data to proceed with making predictions')  
            data.set_index("Timestamp", inplace = True)  
  
            fs_dict = {}  
            for name in data.columns:  
                fs_dict[name] = conf['frequency']  
  
            chunks = chunk_data(data=data, fs_dict=fs_dict, min_chunk_dur="15s")
```

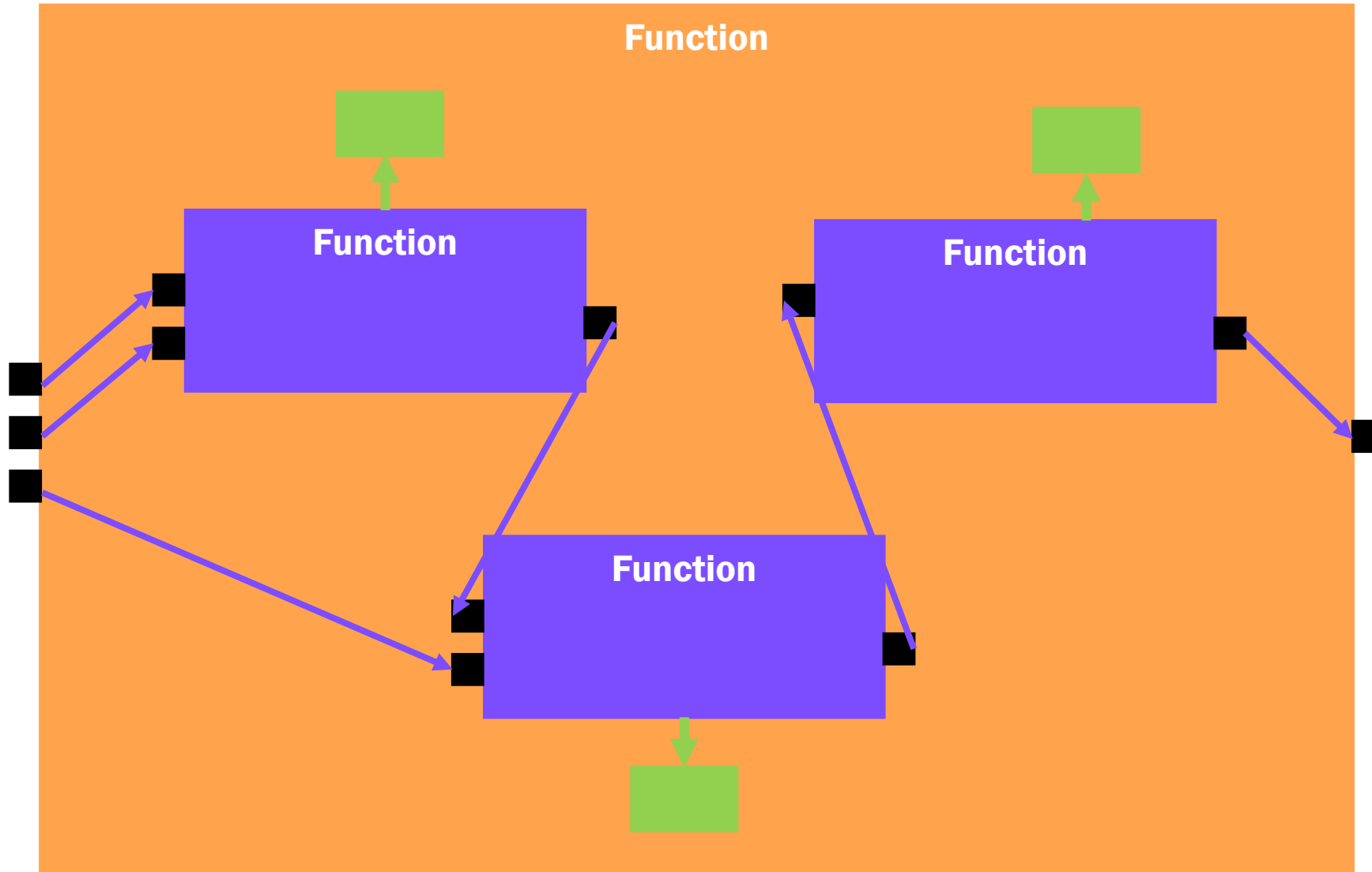


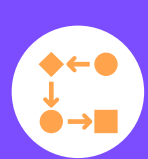
From Source Code to Fn0





From Source Code to Fn0





Describing complex pipelines

- Multiple levels of detail:
 - From pipeline components to individual lines of code
 - Docker container → Python file → line of code → imported function
- Model dataflow across implementation frameworks:
 - Identify points where frameworks meet: `RUN python file.py` in dockerfile
 - Under development

Ensuring transparency

How to show a person what happened with their data



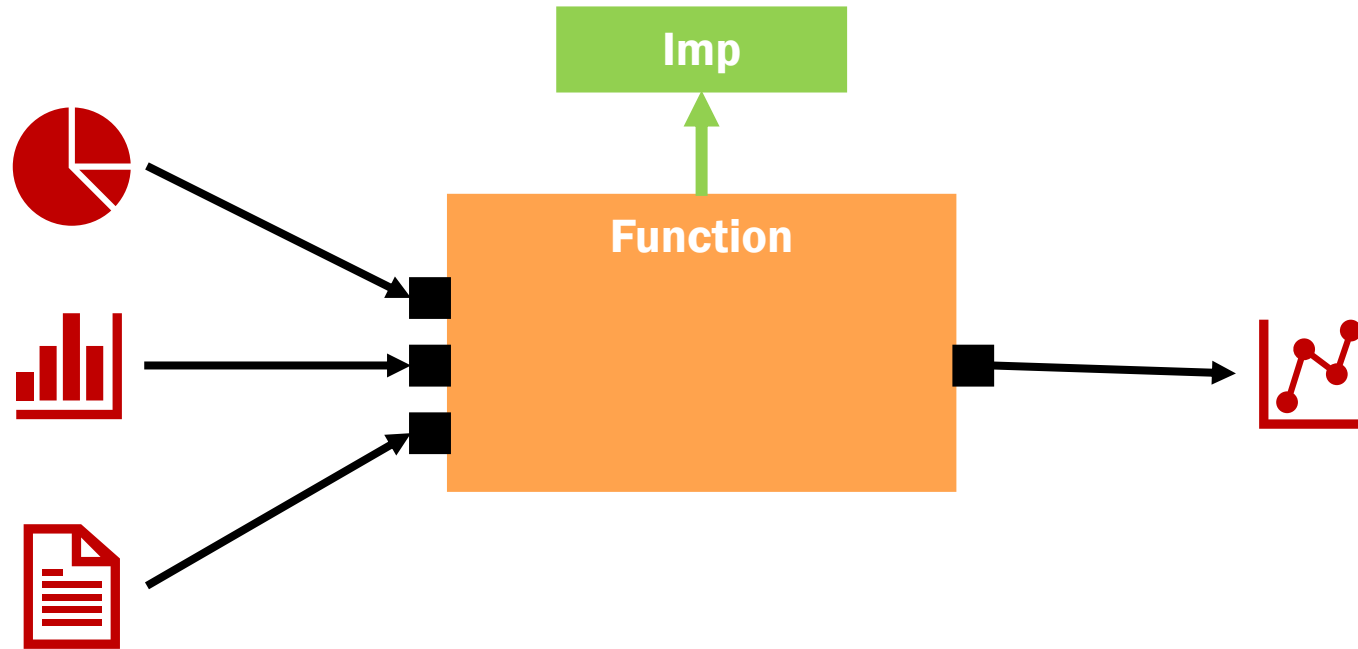
PROV-O Structure

- Describe the lifecycle of data, detailing how and when it was processed
- Three main components:



Execute FnO with Provenance Capture

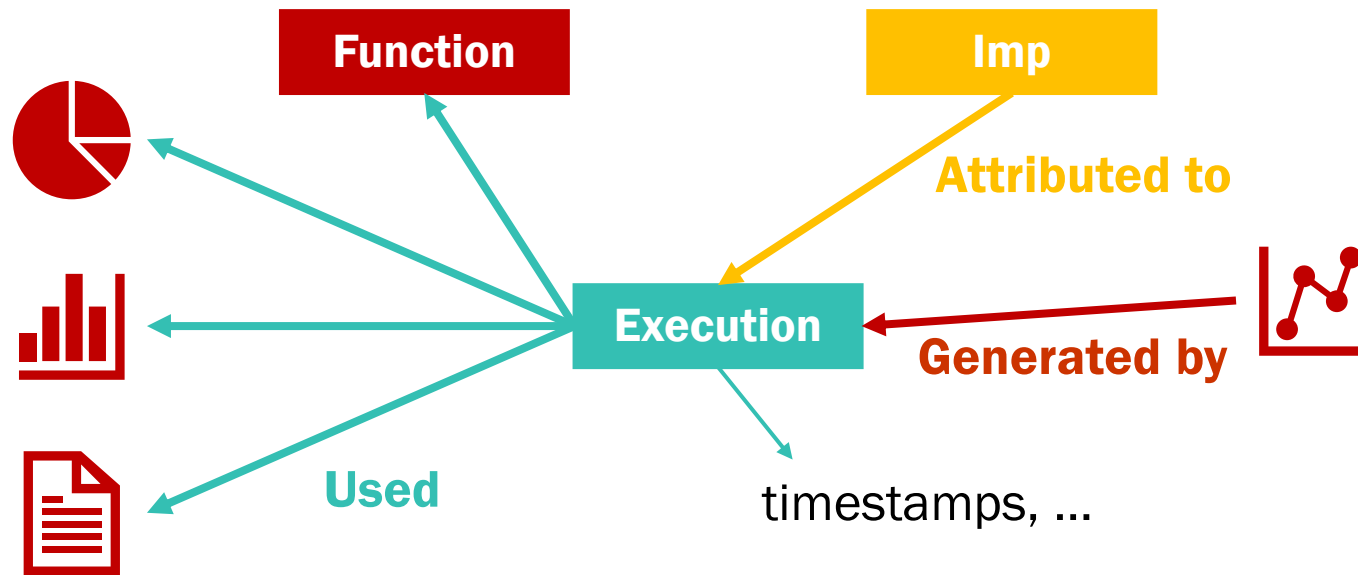
- ➔ Orchestrator: execute an FnO pipeline/function
 1. Look for implementation
 2. Get executable version
 3. Execute with data





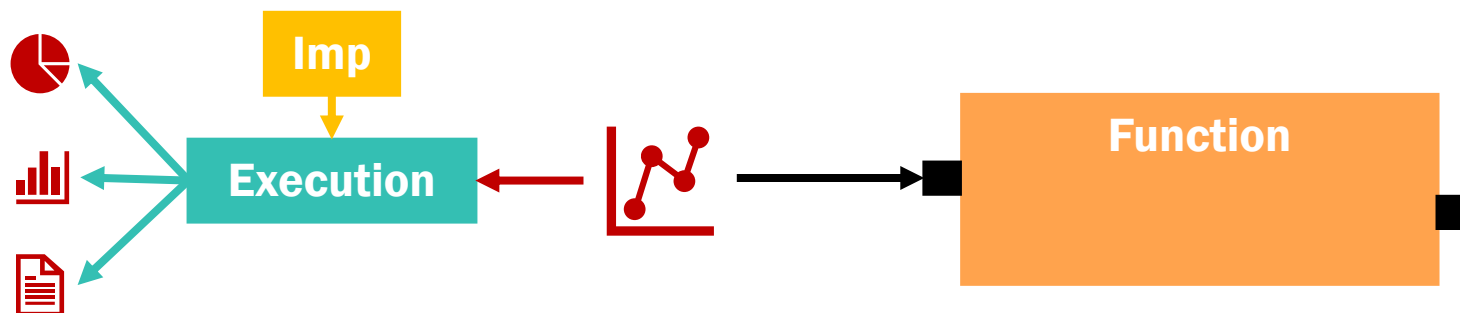
Execute FnO with Provenance Capture

- ➔ Orchestrator: execute an FnO pipeline/function
 1. Look for implementation
 2. Get executable version
 3. Execute with data
 4. Log provenance



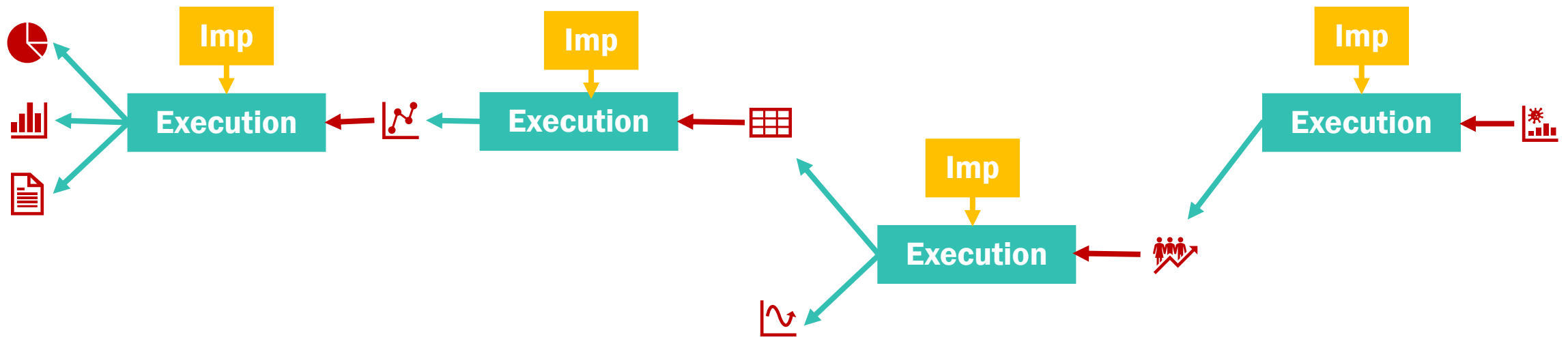
Execute FnO with Provenance Capture

- Orchestrator: execute an FnO pipeline/function
 1. Look for implementation
 2. Get executable version
 3. Execute with data
 4. Log provenance
 5. Propagate output






Execute FnO with Provenance Capture



➔ Result: Provenance tree



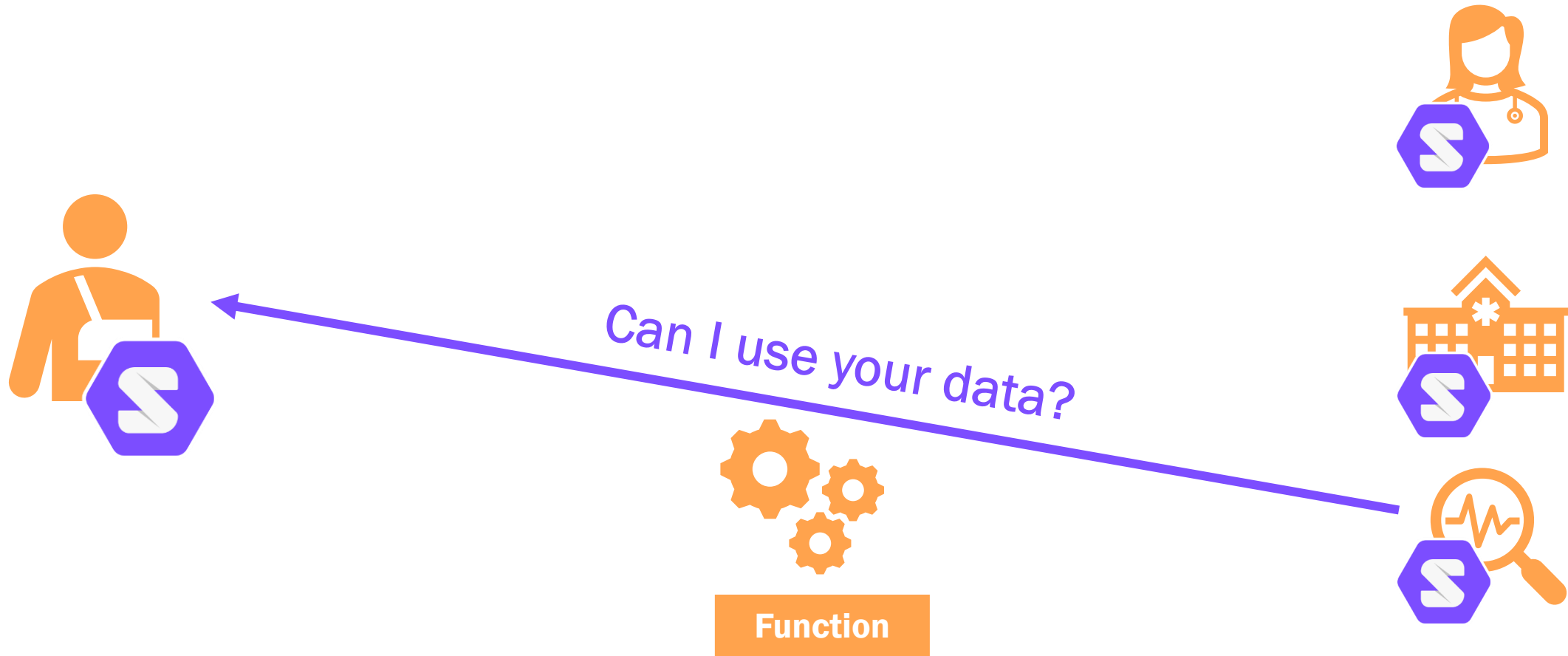
Provide **understanding**

-  Decoupling data flow from implementation framework using FnO
-  Automatically convert source code to FnO to not burden developers
-  Connect dataflow across implementation frameworks

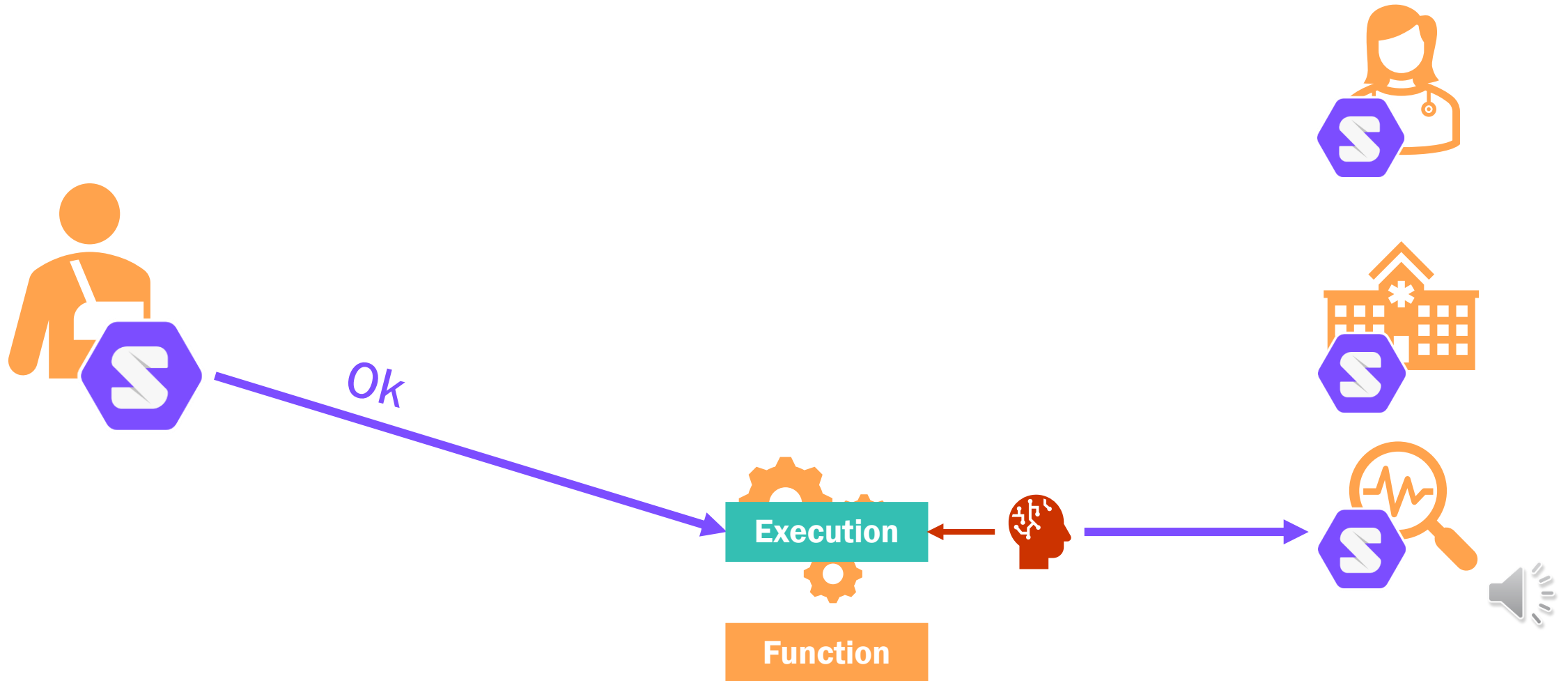
Ensure **transparency**

-  Detail how and when data is processed using PROV-O
-  Make FnO executable to allow provenance capture across frameworks

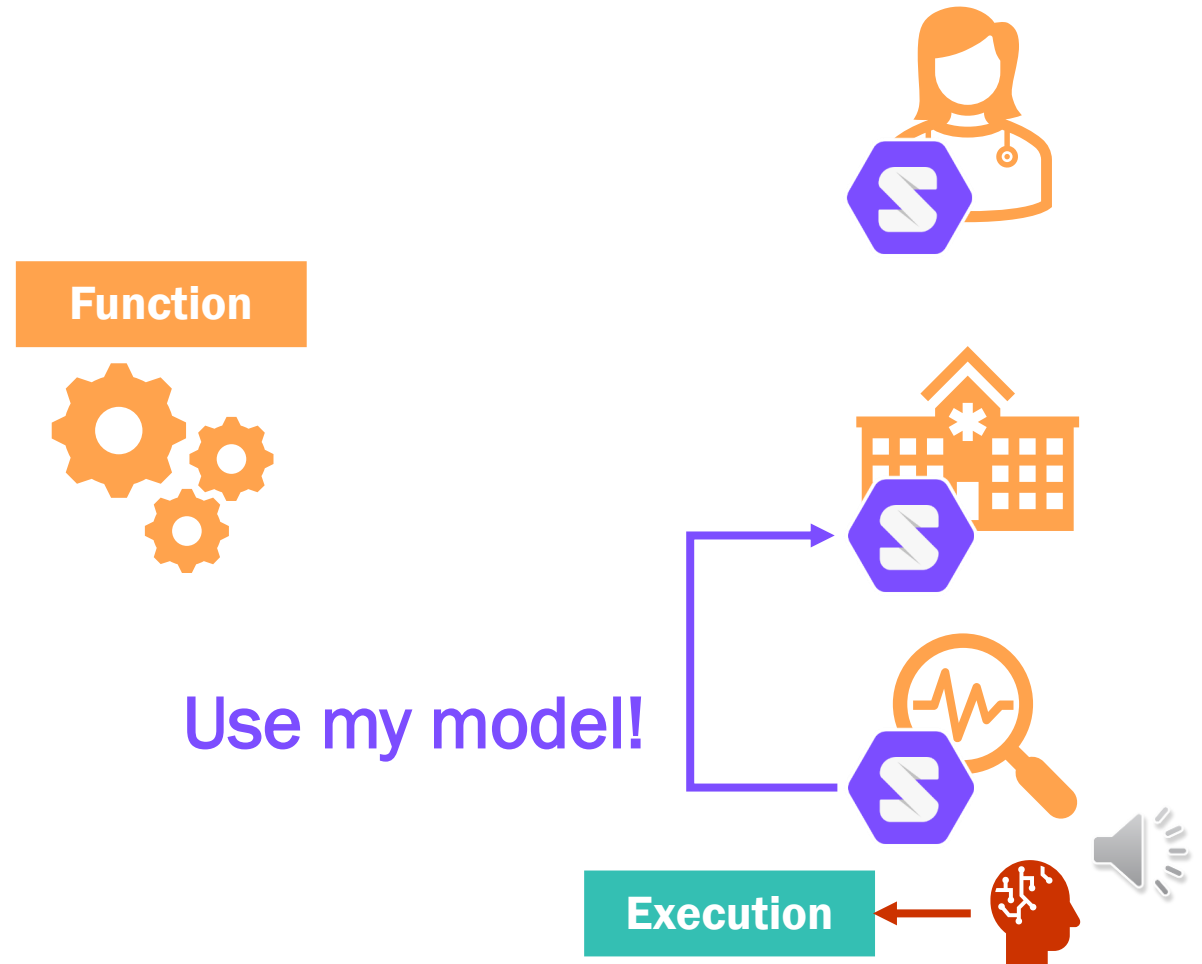
Example



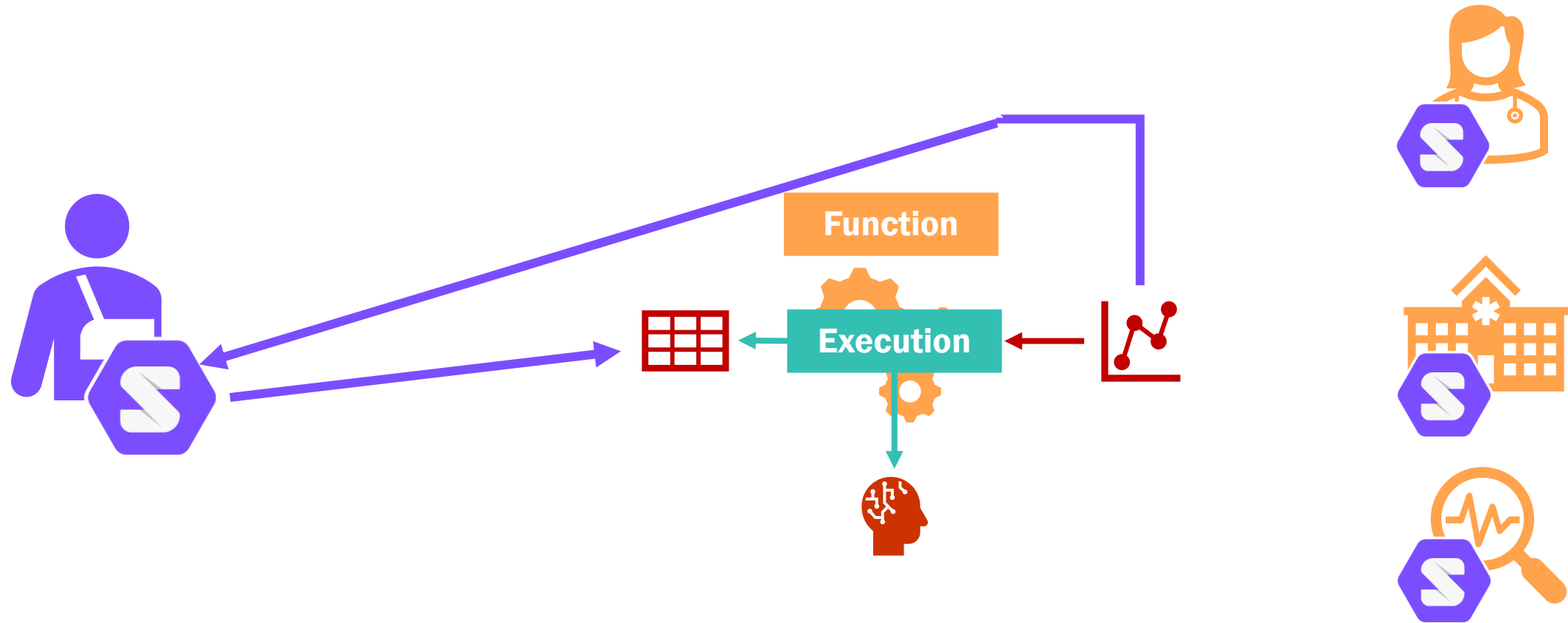
Example



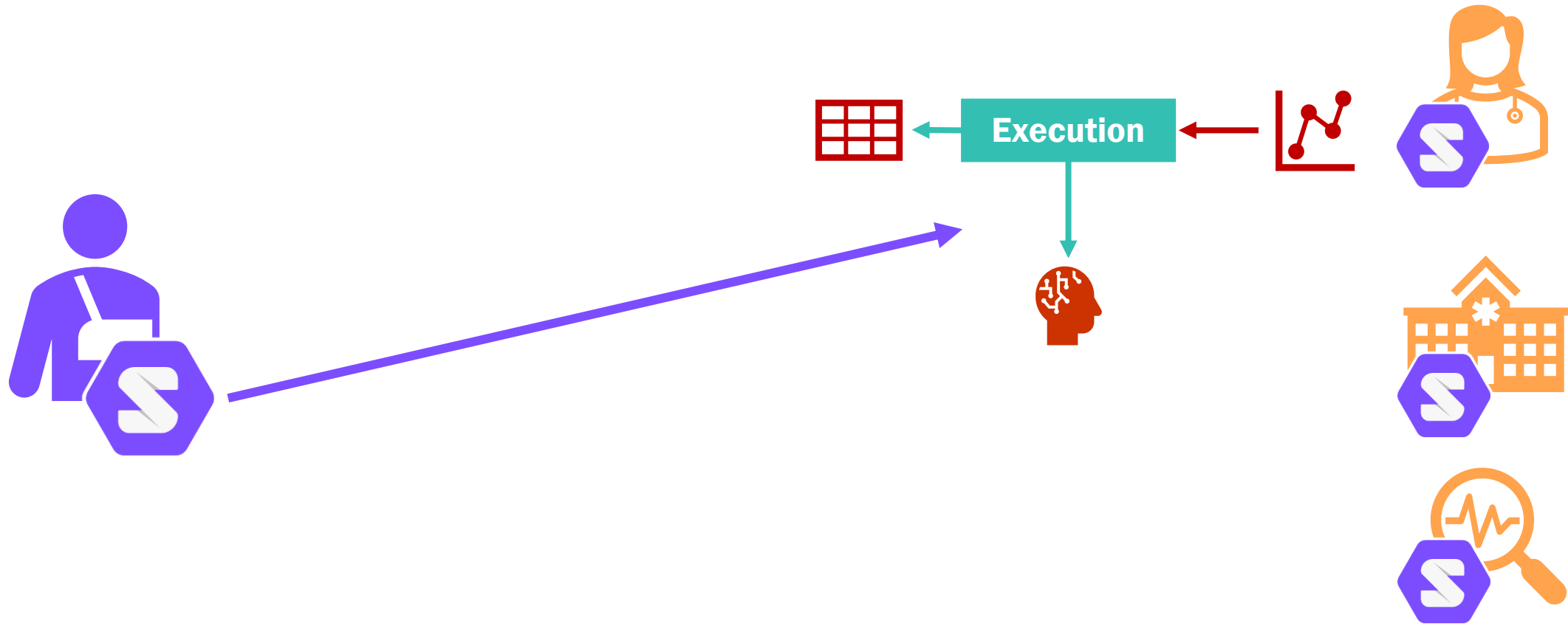
Example



Example



Example



Any questions?