Semiodesk

EVERYTHING CONNECTS.

Accessing a Semantic Graph Database with OpenUI5 via OData

by Andy Le, Sebastian Faubel, Moritz Eberl, Stefan Summesberger

Agenda

1. About the presenters
2. Introduction to Linked Data
3. COVID-19 Use Case
4. Live Demo
5. Demo: Technical Background

About the Presenters.

Andy Le

www.linkedin.com/in/andyle13

- Recently graduated from Sheffield Hallam University
  - Worked on an R&D project concerning IoT and the Semantic Web for indoor tracking as part of my internship for Airbus contractor Geometric Europe.
  - Investigated in Object-Mapping for Graph Databases and their usage in industrial use cases for business process automation for my Bachelor project and two corporate clients.
- How did I meet Semiodesk?
  - Contacted them during a research activity at my internship.
  - Gained their support on Trinity RDF throughout my Bachelor project.
- Strategy gamer, travel addict and meme enthusiast in real-life.

ABOUT THE PRESENTERS

Semiodesk

EVERYTHING CONNECTS.

Semiodesk provides know-how and efficient tools to speed up the development of enterprise knowledge graph applications

Moritz Eberl
Co-Founder

Sebastian Faubel
Co-Founder

Stefan Summesberger
Head of Business Development

Our Products

modom.io

Easily create well-documented ontologies and turn them into ready-to-use software packages.

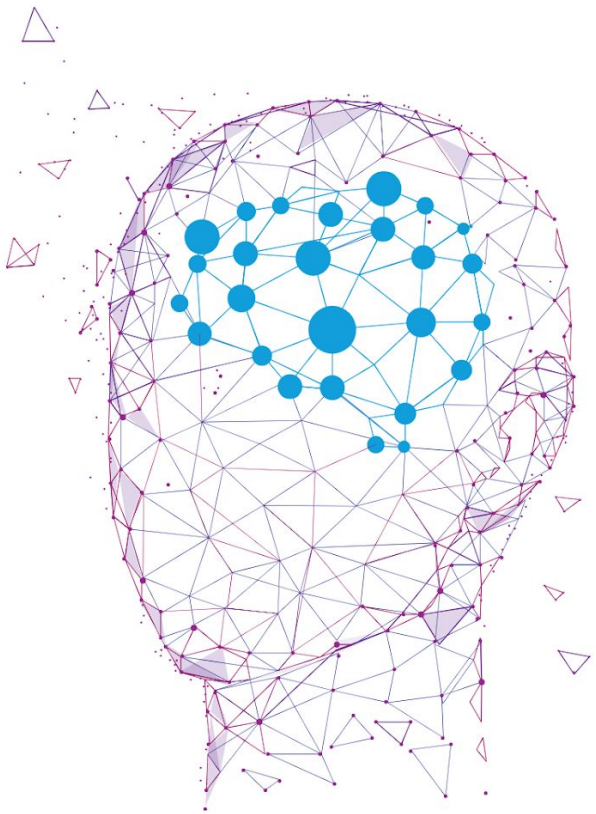
https://modom.io

Trinity RDF

Enterprise ready object mapper for developing RDF knowledge graph applications with .NET.

https://trinity-rdf.net

modom.io



Design Knowledge Graphs the Smart Way.

modom.io is an easy to use ontology management platform. Save precious time with the powerful, built-in knowledge base that helps you discover and re-use 20K+ existing terms while you type.

Easier

No ontology skills required for domain experts and software engineers to collaborate.

Faster

Assists with creating articles, diagrams and source code to support projects.

Smarter

Autofill-suggestions from 260+ ontologies turbo boost knowledge engineering.

https://modom.io

Trinity RDF

Entity Framework for Graph Databases.

Enterprise ready object mapper for developing RDF knowledge graph applications with .NET

Free & Easy

With Trinity data model packages, developers can handle large-scale RDF knowledge graphs without even knowing!

Smart

Many RDF databases support logical reasoning during application runtime to answer queries about facts which have not been explicitly saved.

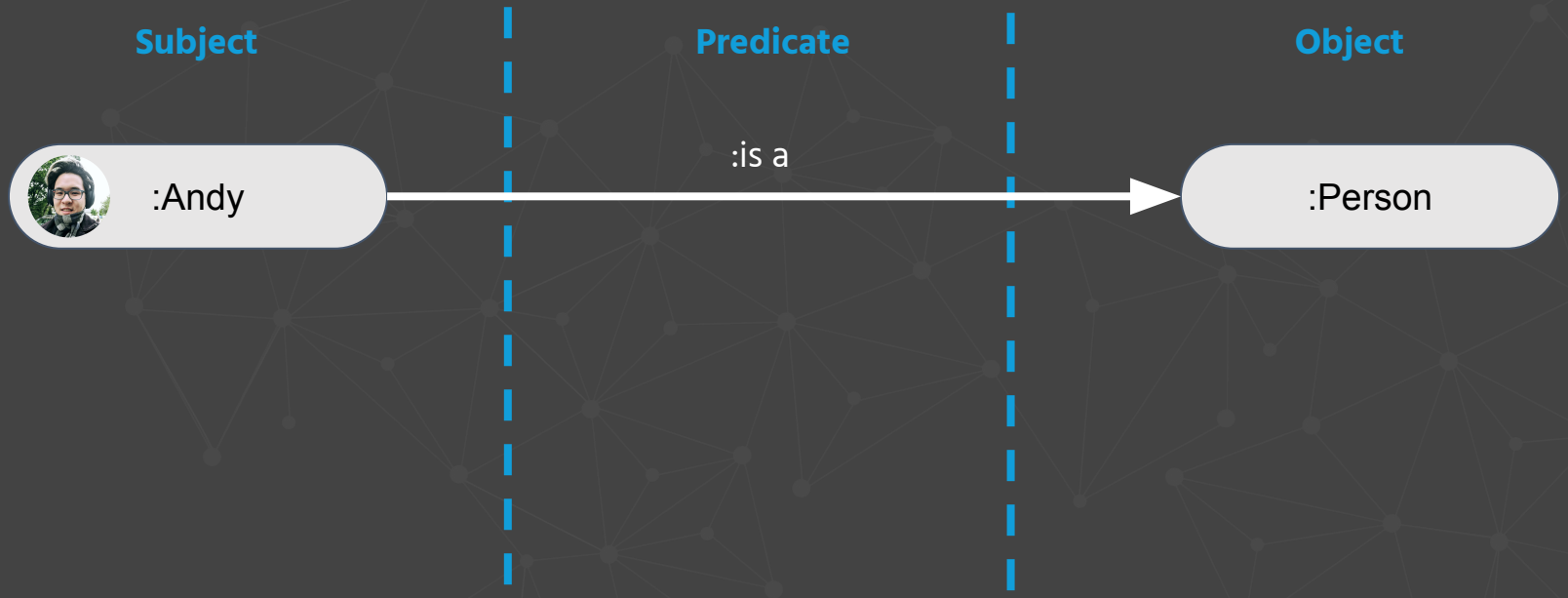
Sustainable

The Resource Description Framework linked data platform is an open standard maintained by the World Wide Web Consortium (W3C).

https://trinity-rdf.net

Introduction to Linked Data.

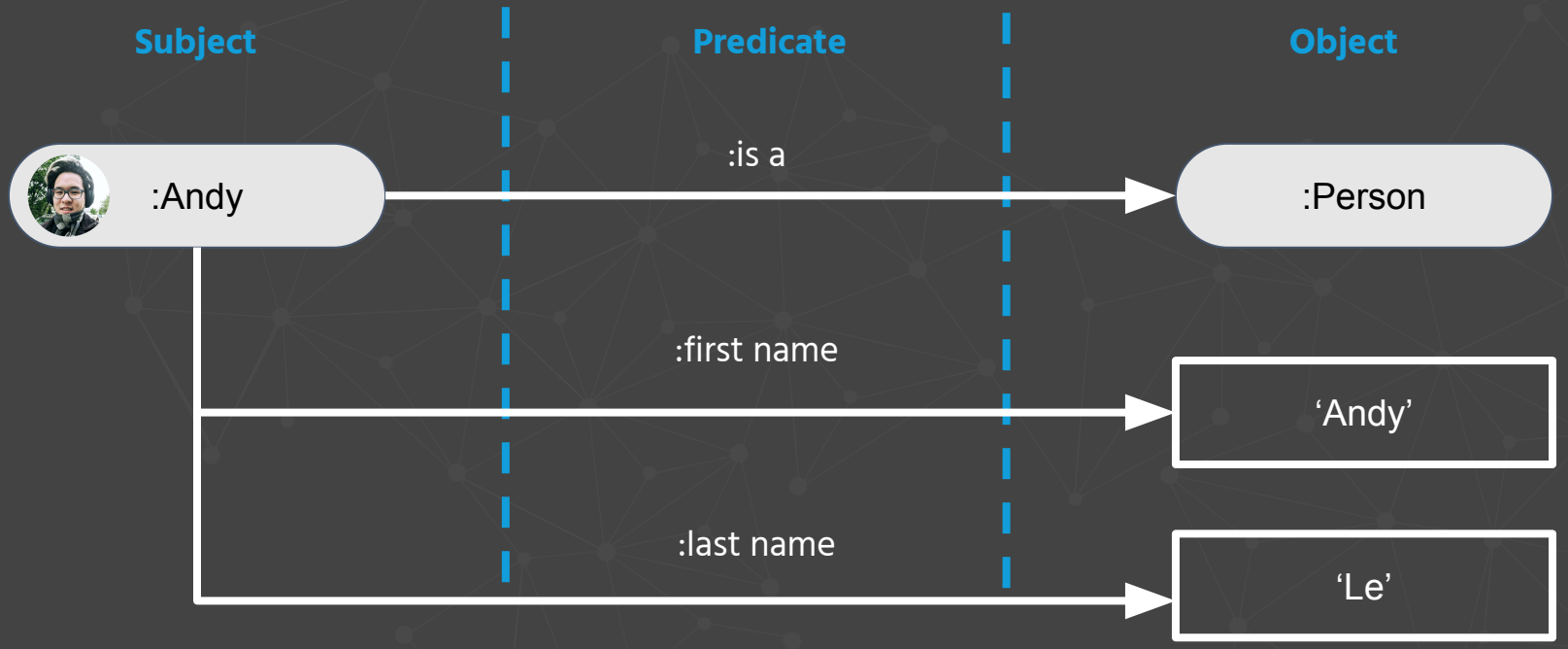
What is Linked Data?



At the heart of linked data is a sentence.

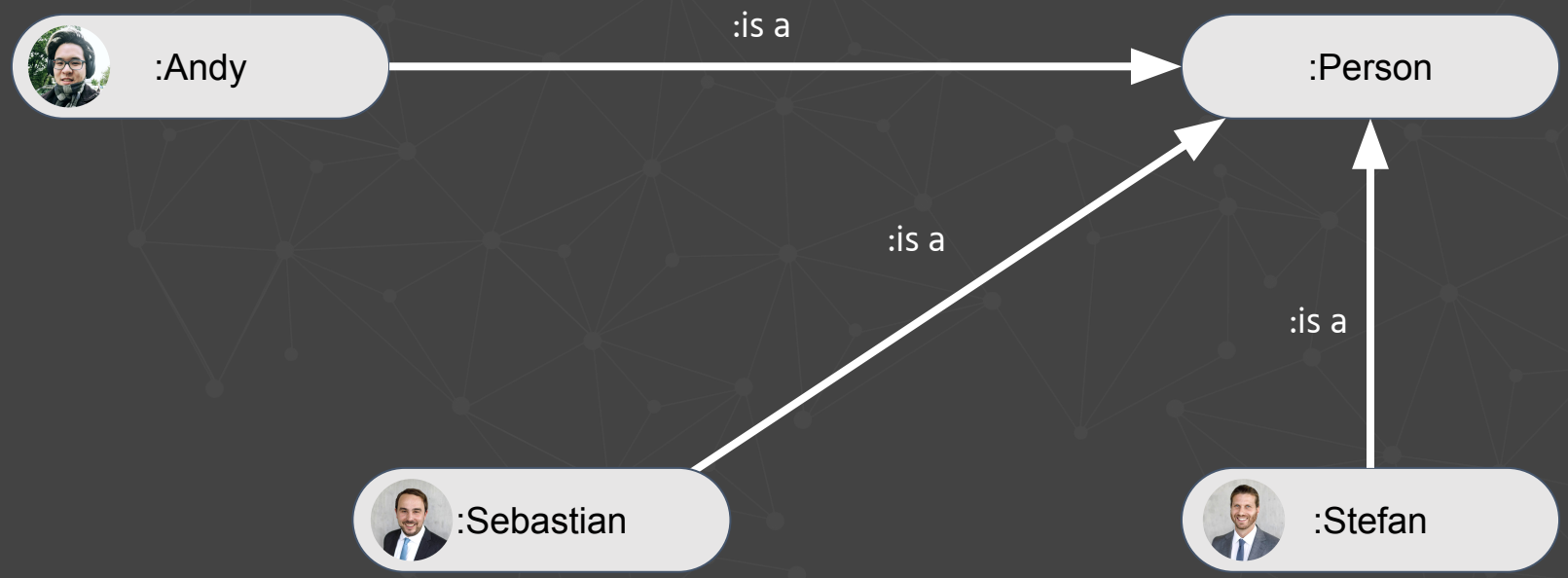
It's also called **Triple**.

What is Linked Data?



The object of a triple is either **a literal value** such as a name..

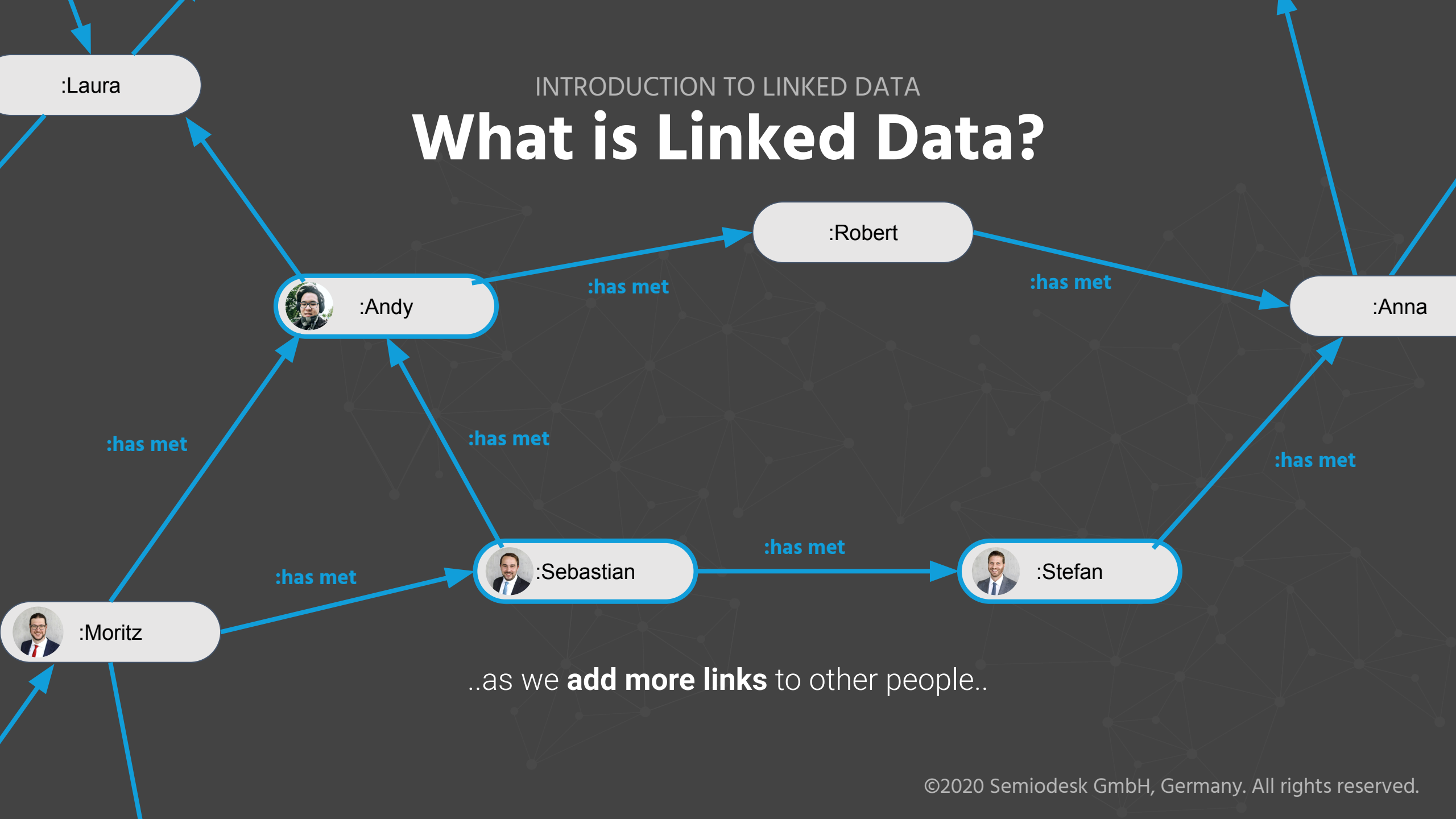
What is Linked Data?



..or another **subject**, such as another person.

INTRODUCTION TO LINKED DATA

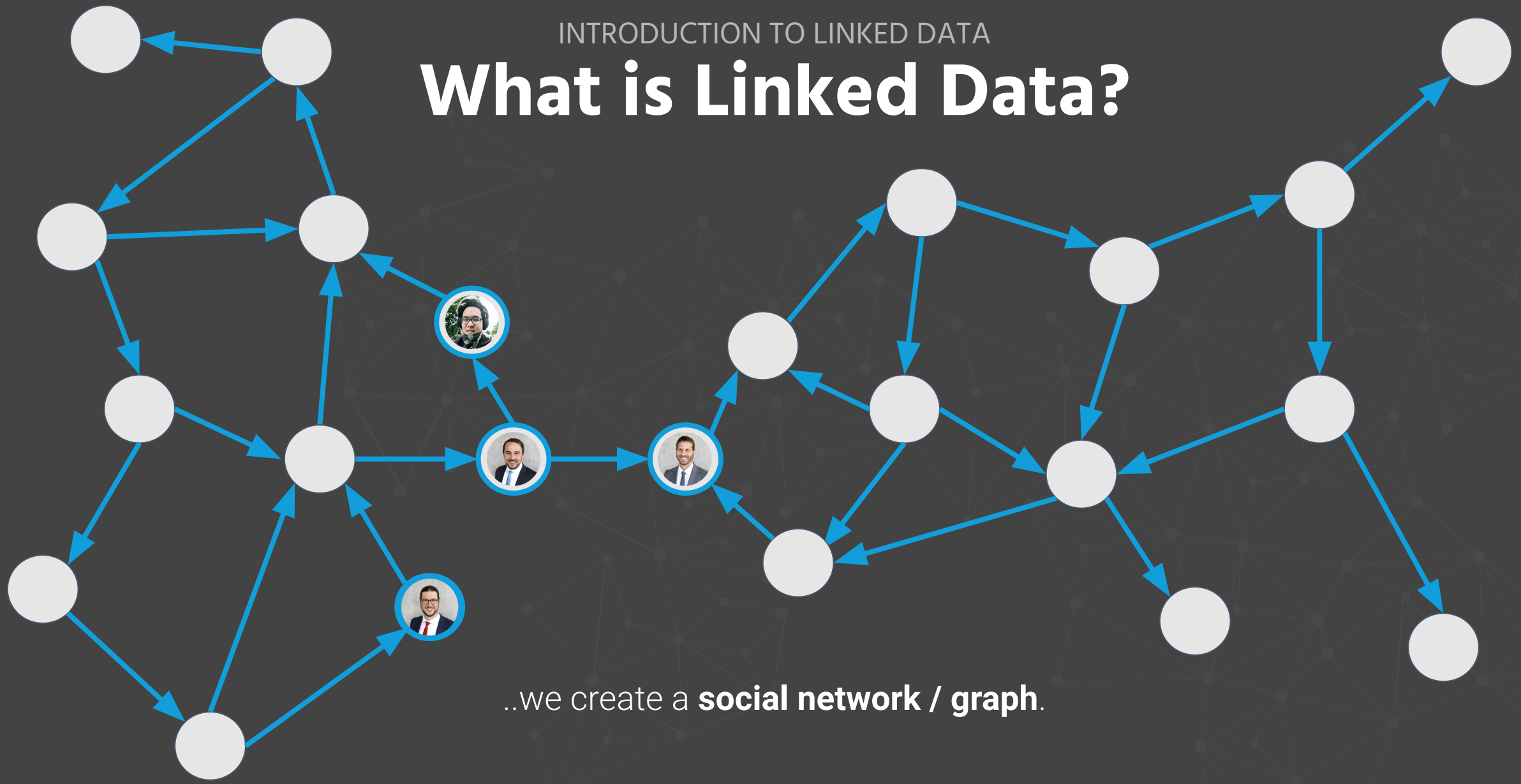
What is Linked Data?



..as we **add more links** to other people..

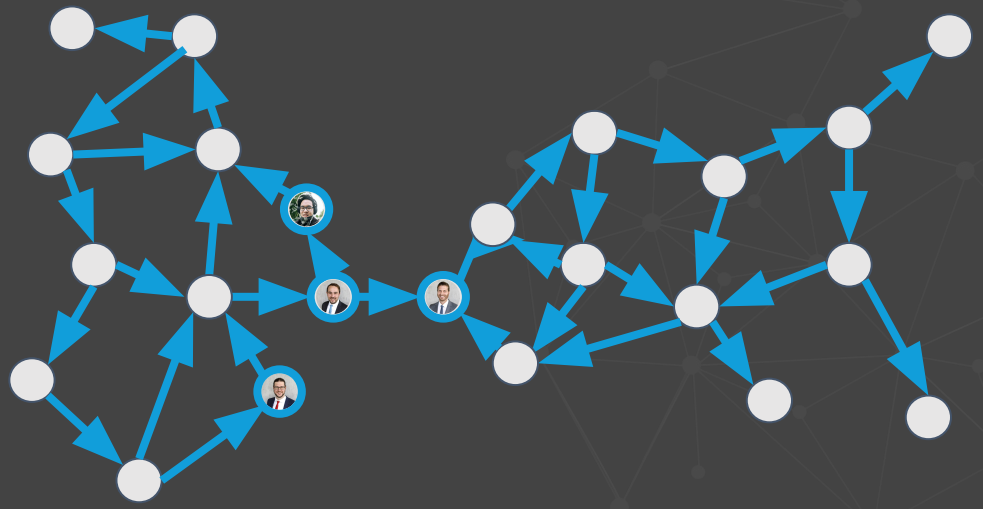
INTRODUCTION TO LINKED DATA

What is Linked Data?



..we create a **social network / graph**.

What is a Graph Database?



A database which is optimized for:

- **Storing and querying linked data**
- **Finding interconnections in data**
- **Scalability**

W3C Graph Database Standards:

- **RDF / RDF* Linked Data Format**
- **SPARQL Query Language**

Who is using Graph Databases?

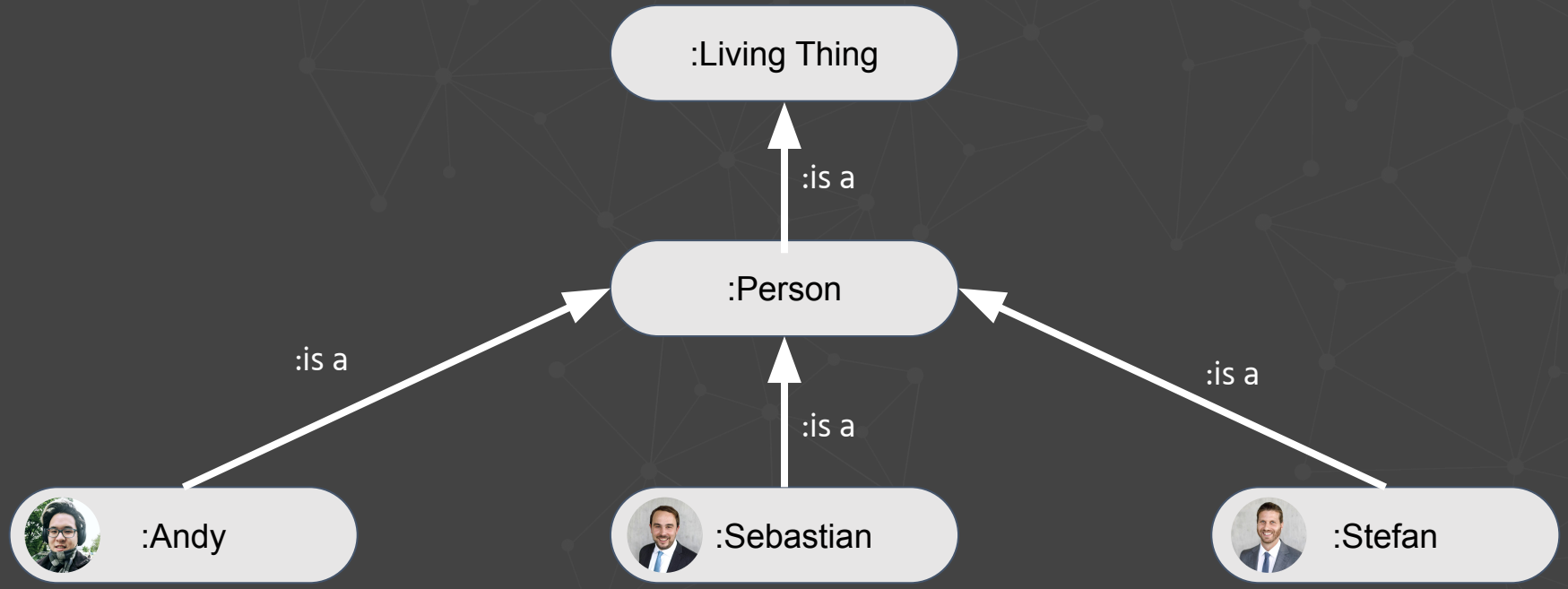
Known Knowledge Graph Builders

	Company name	Location	Industry	Market Cap Change 2009 - 2018 ($bn)
1	Apple	United States	Technology	757
2	Amazon.Com	United States	Consumer Services	670
3	Alphabet	United States	Technology	609
4	Microsoft Corp	United States	Technology	540
5	Tencent Holdings	China	Technology	483
6	Facebook	United States	Technology	383
7	Berkshire Hathaway	United States	Financials	358
8	Alibaba	China	Consumer Services	302
9	JPMorgan Chase	United States	Financials	275
10	Bank of America	United States	Financials	263

Source: 'Collapsing the IT Stack' (2018) by Allan Morrison, PwC

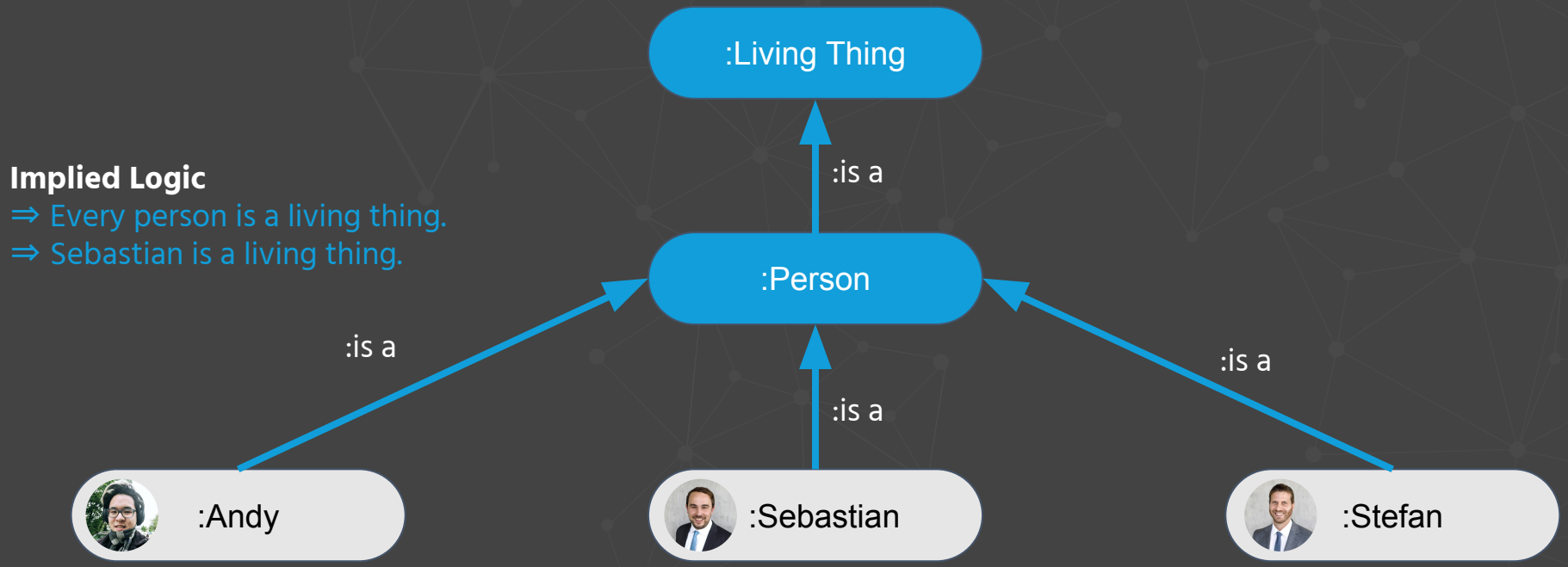
What is a Knowledge Graph?

A ***Knowledge Graph*** is a graph that incorporates knowledge in machine understandable form.



What is a Knowledge Graph?

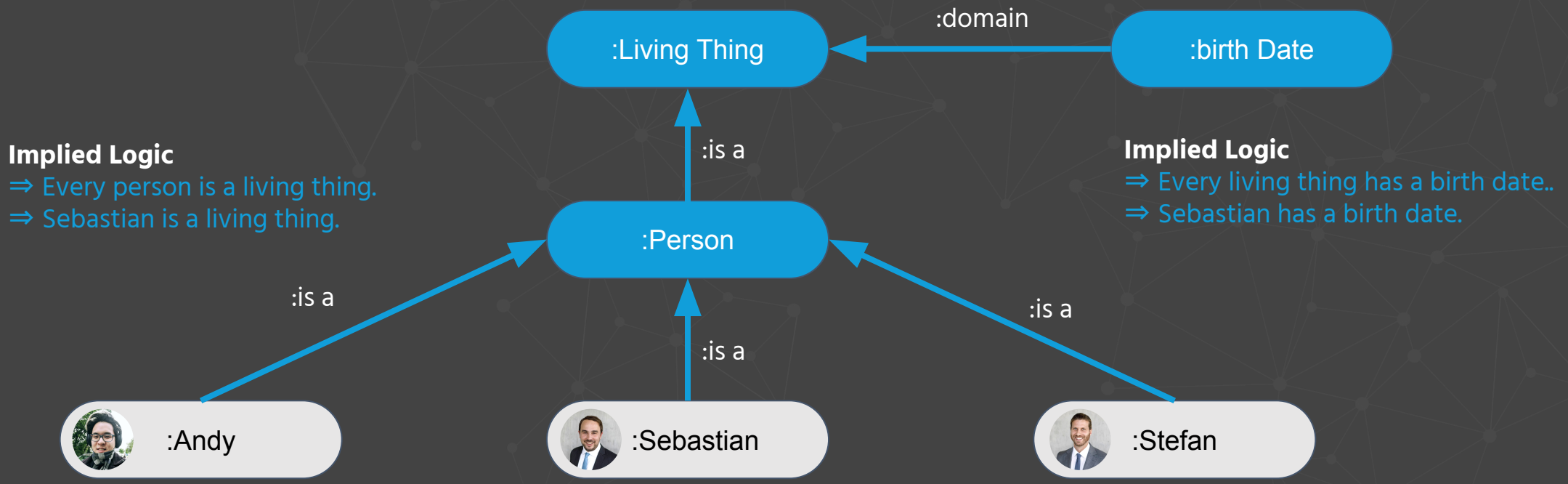
Type hierarchies categorize individuals into groups with common properties.



Implied Logic
⇒ Every person is a living thing.
⇒ Sebastian is a living thing.

What is a Knowledge Graph?

Properties describe possible attributes or relations of a type.

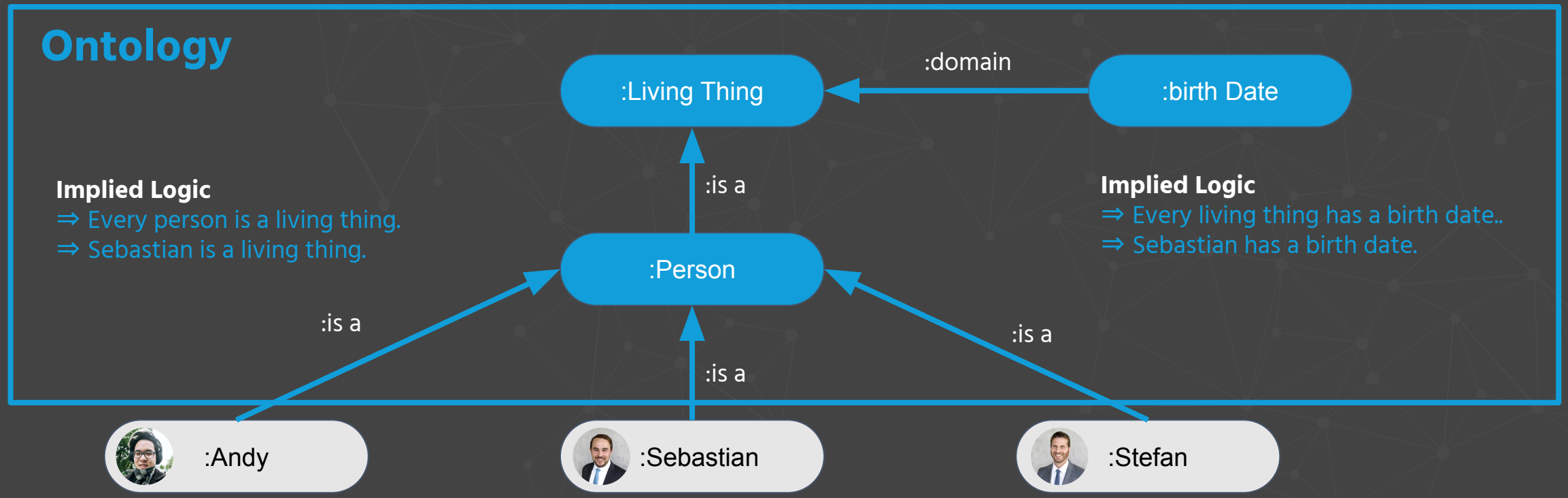


Implied Logic
⇒ Every person is a living thing.
⇒ Sebastian is a living thing.

Implied Logic
⇒ Every living thing has a birth date..
⇒ Sebastian has a birth date.

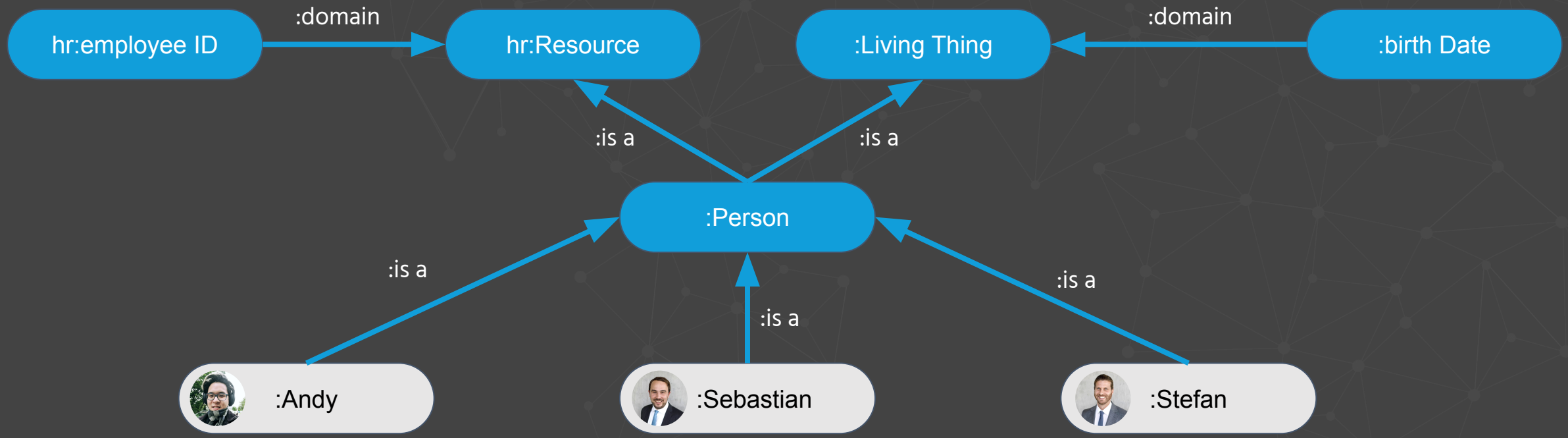
What is a Knowledge Graph?

Ontologies provide logics to answer queries more flexibly.



What is a Knowledge Graph?

Ontologies can easily be extended and integrated.



COVID-19 Tracker Use Case.

COVID-19 TRACKER USE CASE

Disclaimer

- The solution presented here is just a simulation and does not remotely propose a competitive alternative to existing Contact Tracing apps.
- We encourage everyone to use the official Corona Tracing apps available in their country.

COVID-19 TRACKER USE CASE

Introduction

- Contact Tracing application for mitigating infection chains.
- Existing solutions require smartphones for exchanging ID codes with other users via Bluetooth if in close proximity.
- If tested positive from a COVID test, the user receives a code from their doctor which they need to enter on their app to notify the people they have recently encountered.

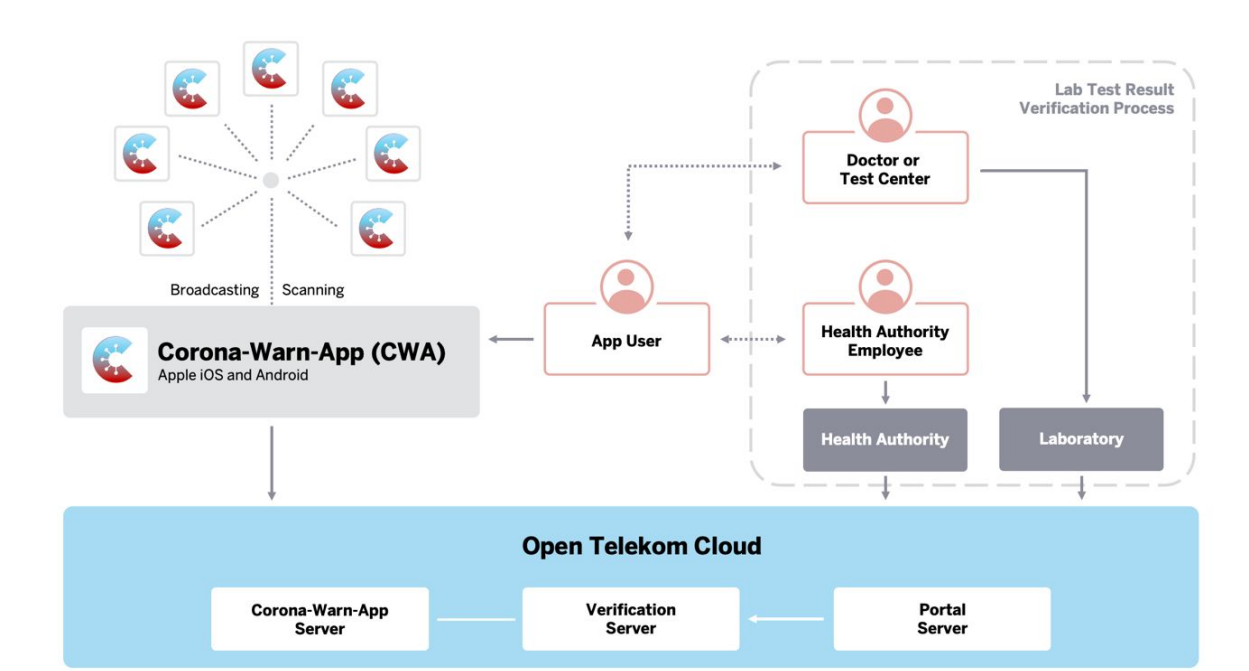
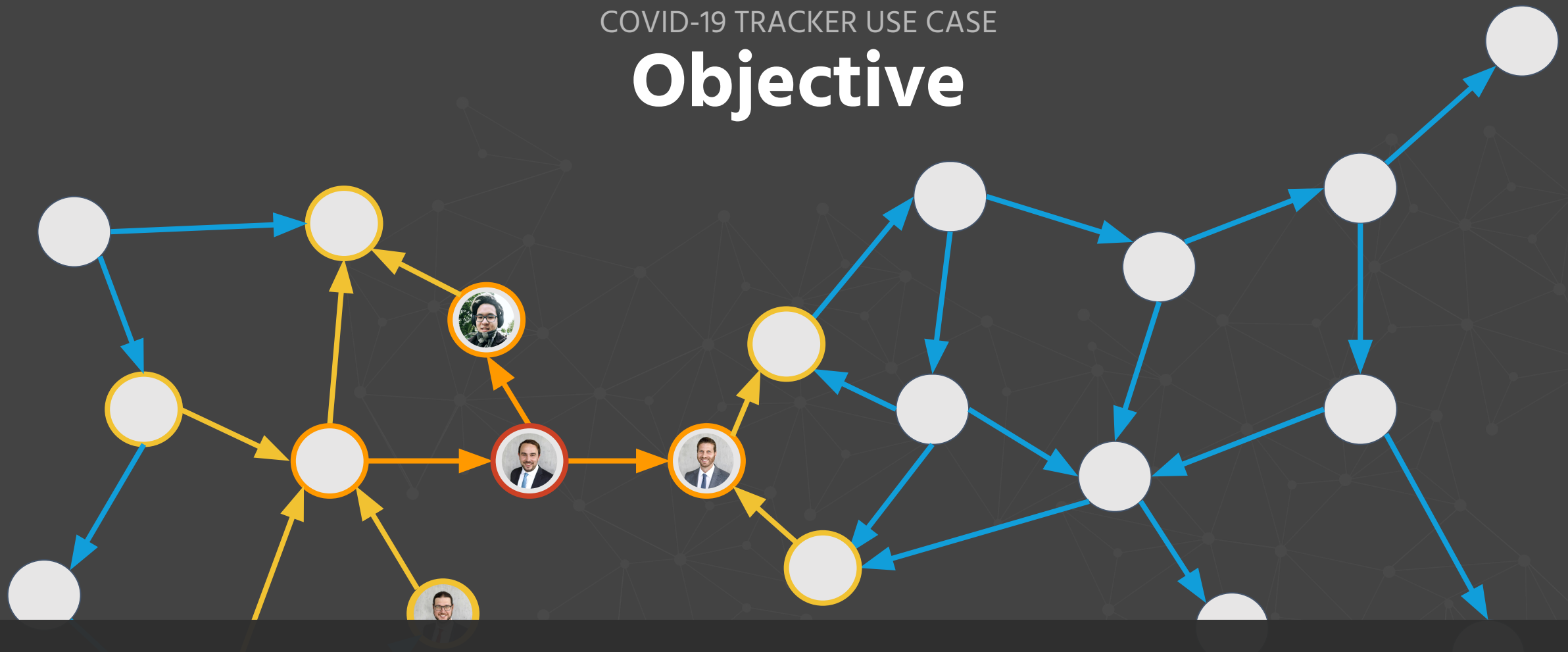


Image Source: SAP

COVID-19 TRACKER USE CASE

Objective

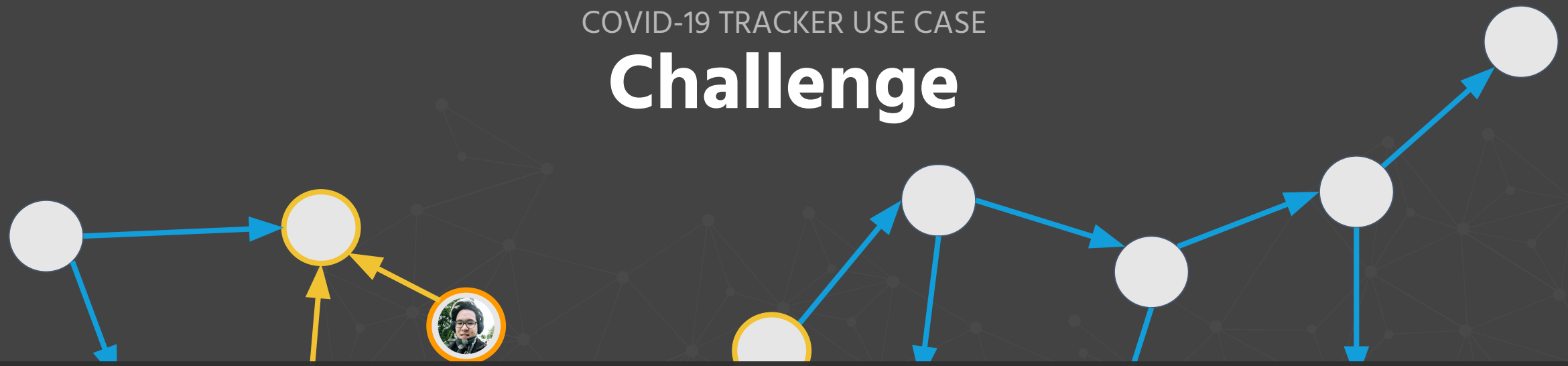


Track person encounters and determine infection risk for each person.

- Known Infection
- High Risk
- Medium Risk

COVID-19 TRACKER USE CASE

Challenge

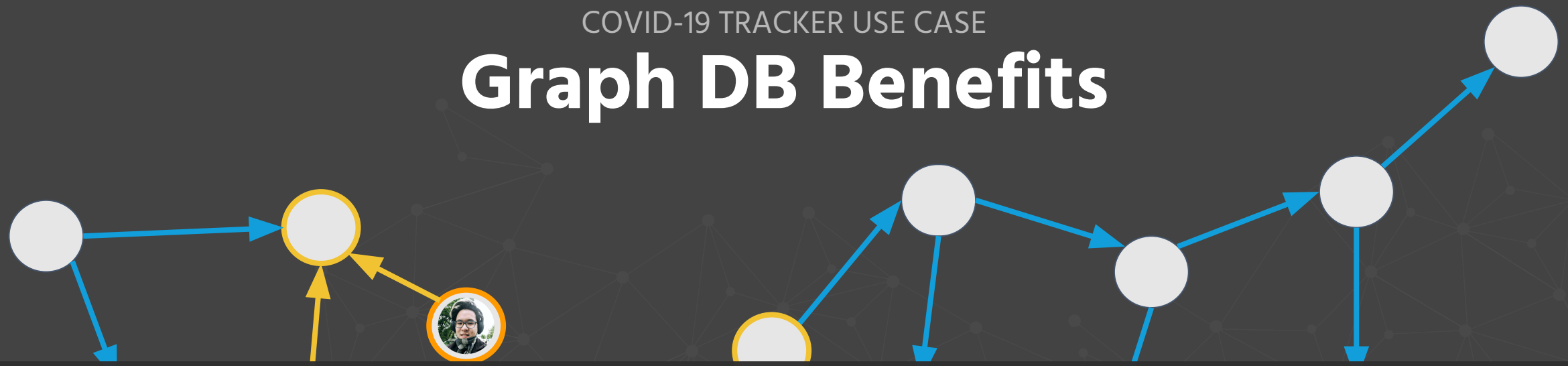


Can Graph Databases assist Contact Tracing?

- **Contact Tracing is the search for events when people meet**
  - People are interconnected through encounters like a social network.
- **SPARQL Path Search Capabilities**
  - Calculates potential risk encounters in a single SPARQL query.
- **High scalability**
  - Optimized for processing relations in large networks of data.

COVID-19 TRACKER USE CASE

Graph DB Benefits

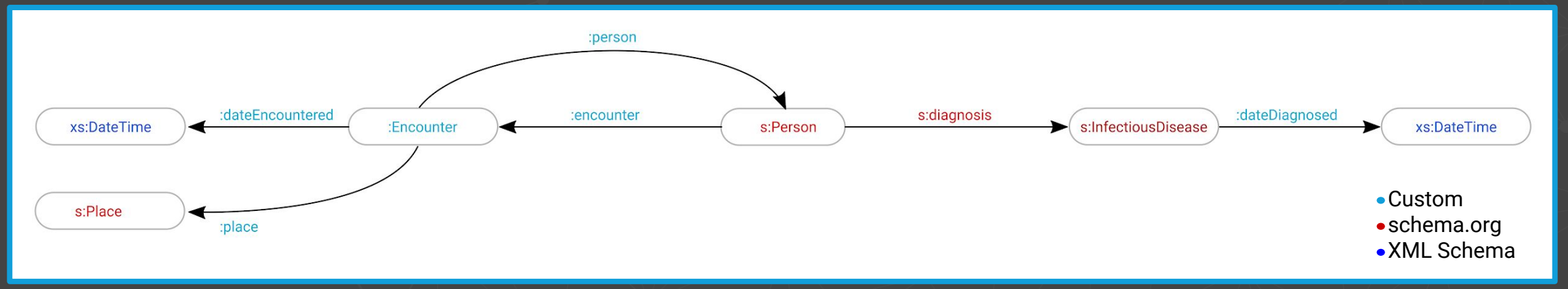


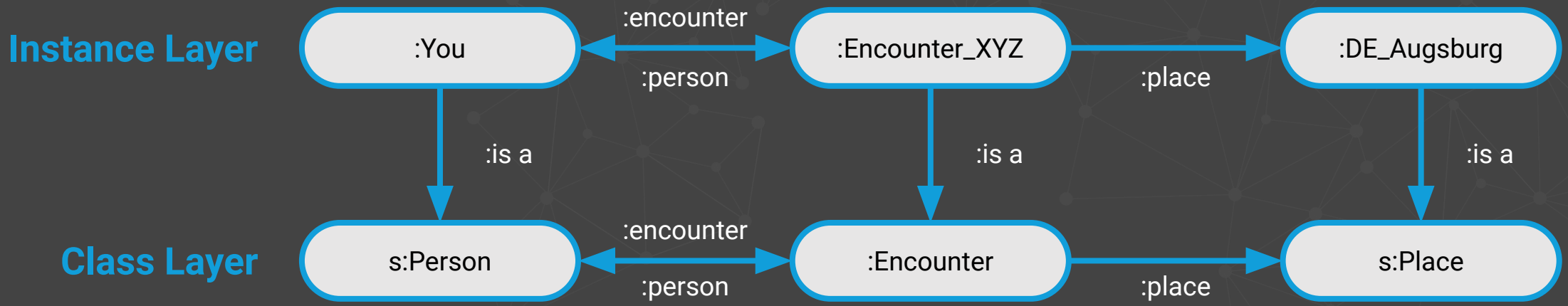
Graph Databases Save Development Time!

- **Built-In Arbitrary Depth Path Search**
  - Allows to easily search for connections between people who have met
- **Openly Available Data Models**
  - Schema.org has people, places and events already modelled
- **Easily Extendable**
  - Flexibility allows to cover other diseases with different parameters as well

COVID-19 TRACKER USE CASE

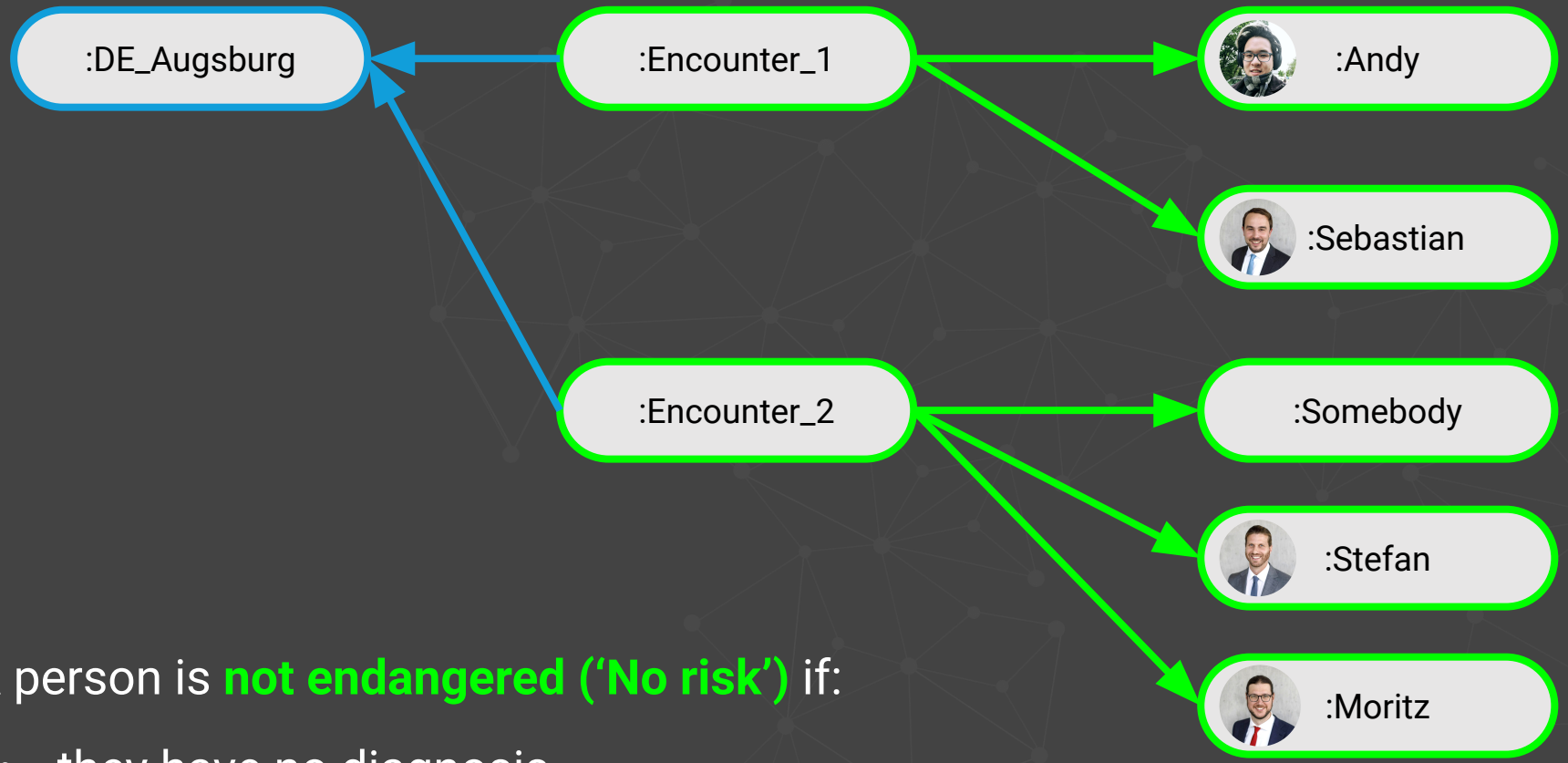
RDF Data Model





COVID-19 TRACKER USE CASE

Identifying Infection Chains

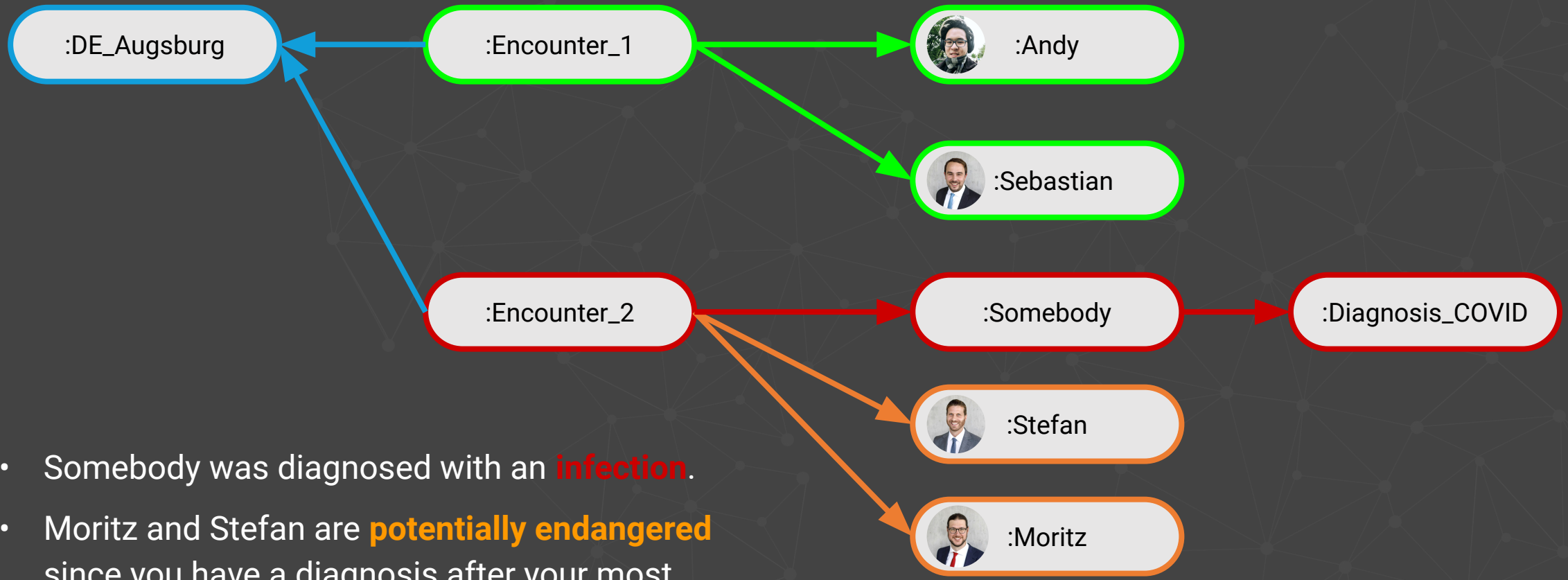


A person is **not endangered ('No risk')** if:

- they have no diagnosis
- have not met anyone who is endangered

COVID-19 TRACKER USE CASE

Identifying Infection Chains



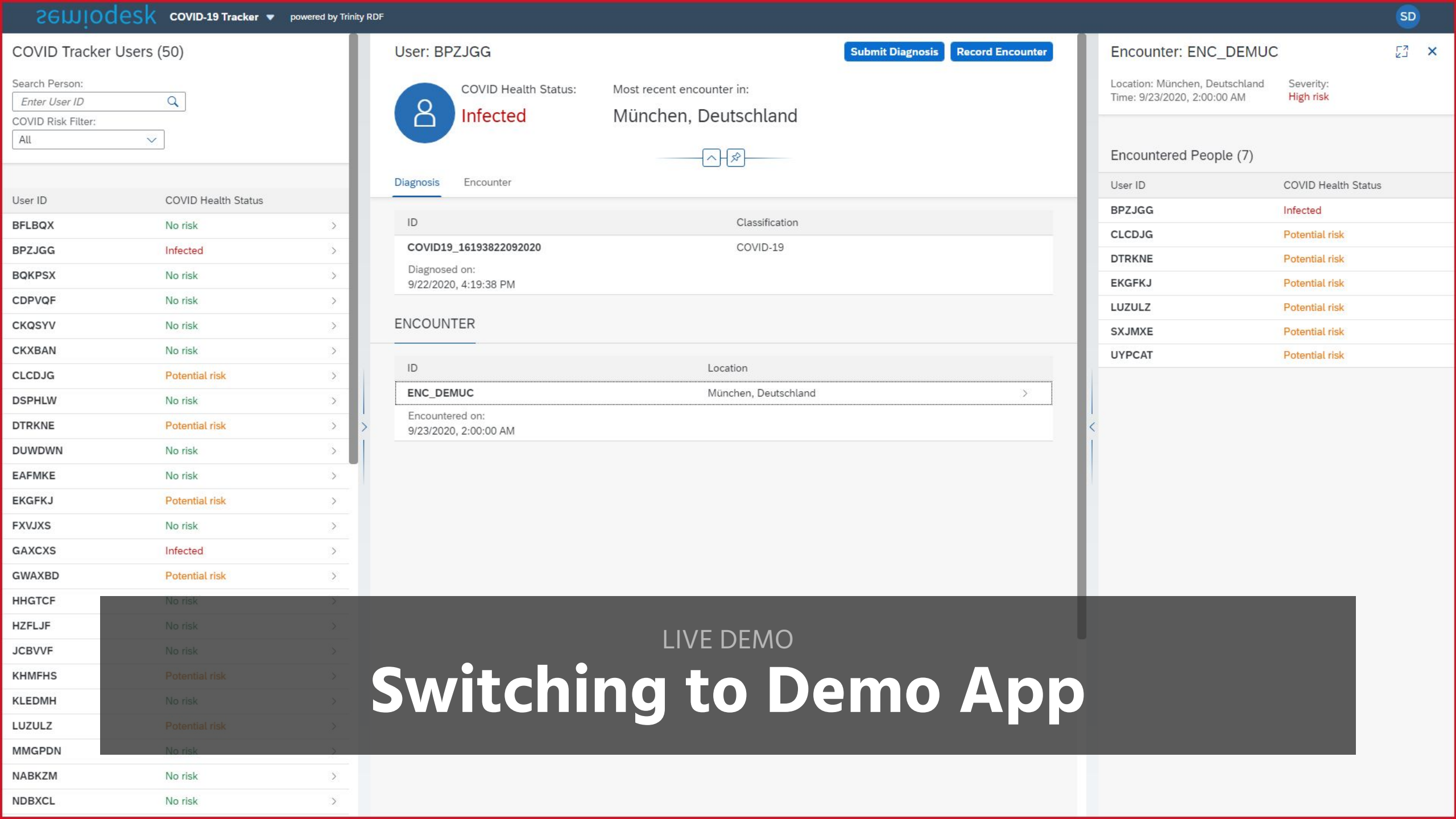
- Somebody was diagnosed with an **infection**.
- Moritz and Stefan are **potentially endangered** since you have a diagnosis after your most recent encounter with them.

COVID-19 TRACKER USE CASE

SPARQL Query for Risk Calculation

```
SELECT ?person ?hasDiagnosis MAX(?hasPotential) AS ?riskPotential
WHERE
{
   ?person a crt:Person ; crt:encounter / crt:person ?other .

   FILTER(?other != ?person)

   BIND(EXISTS { ?person crt:diagnosis ?diagnosis. } AS ?hasDiagnosis)
   BIND(EXISTS { ?other crt:diagnosis ?diagnosis. } AS ?hasPotential)
}
GROUP BY ?person ?hasDiagnosis
```

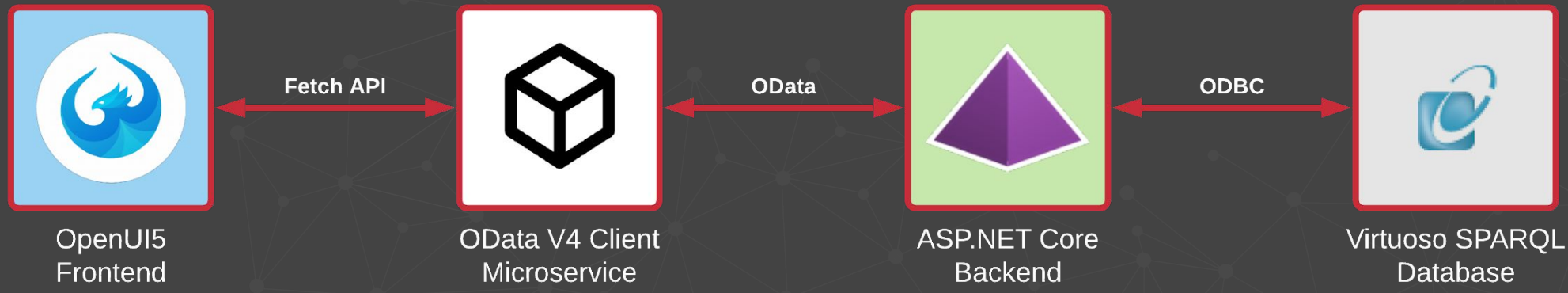
Live Demo.


COVID-19 Tracker
powered by Trinity RDF
SD
COVID Tracker Users (50)
Search Person:
Enter User ID
COVID Risk Filter:
All
User ID
COVID Health Status
BFLBQX No risk
BPZJGG Infected
BQKPSX No risk
CDPVQF No risk
CKQSYV No risk
CKXBAN No risk
CLCDJG Potential risk
DSPHLW No risk
DTRKNE Potential risk
DUWDWN No risk
EAFMKE No risk
EKGFKJ Potential risk
FXVJXS No risk
GAXCXS Infected
GWAXBD Potential risk
HHGTCF No risk
HZFLJF No risk
JCBVVF No risk
KHMFHS Potential risk
KLEDMH No risk
LUZULZ Potential risk
MMGPDN No risk
NABKZM No risk
NDBXCL No risk
User: BPZJGG
Submit Diagnosis
Record Encounter
COVID Health Status:
Infected
Most recent encounter in:
München, Deutschland
Diagnosis
Encounter
ID
Classification
COVID19_16193822092020
COVID-19
Diagnosed on:
9/22/2020, 4:19:38 PM
ENCOUNTER
ID
Location
ENC_DEMUC
München, Deutschland
Encountered on:
9/23/2020, 2:00:00 AM
Encounter: ENC_DEMUC
Location: München, Deutschland
Time: 9/23/2020, 2:00:00 AM
Severity:
High risk
Encountered People (7)
User ID
COVID Health Status
BPZJGG Infected
CLCDJG Potential risk
DTRKNE Potential risk
EKGFKJ Potential risk
LUZULZ Potential risk
SXJMXE Potential risk
UYPCAT Potential risk
LIVE DEMO
Switching to Demo App

Technical Background.

TECHNICAL BACKGROUND

System Architecture



Component	Technical Challenge	Architectural Solution
OpenUI5 Frontend - OData	Limited OData v4 Support	Fetch API to OData Microservice
OpenUI5 Frontend - ODataModel	No direct usage of OData v4	JSONModel for storing fetched entities
OData v4 Client Microservice	Means to process OData queries	o.js library
ASP.NET Core Backend	Harmonizing Trinity RDF and OData	Encapsulating Trinity RDF

TECHNICAL BACKGROUND

UI5: OData Request Interface

- To replicate some behavior of OData, a CRUD-based interface must be provided.
- It uses an *ODataGateway* instance for providing the Fetch API abstraction which takes a JSON payload as parameter.
- The JSON payload is isomorphic and adapts to the [required parameters in o.js](#).

```
1  sap.ui.define([
2      "sap/ui/base/Object",
3      "semiodesk/ui5/covidtracker/service/ODataGateway"
4  ], (Object, ODataGateway) => {
5      "use strict";
6      const client = new ODataGateway();
7
8      return Object.extend("semiodesk.ui5.covidtracker.service.ODataHandler", {
9          Create: (entityType, entityBody) => {
10             return client.IssueRequestToClient({
11                 "requestType": "POST",
12                 "entityType": entityType,
13                 "entityBody": entityBody
14             });
15         },
16         Read: (entityType, query = {}) => {
17             return client.IssueRequestToClient({
18                 "requestType": "GET",
19                 "entityType": entityType,
20                 "query": query
21             });
22         },
23 >       Update: (entityType, entityBody, entityID) => {···
30         },
31 >       Delete: (entityType, entityID) => {···
37         }
38     })
39 });
```

TECHNICAL BACKGROUND

UI5: From OData to JSONModel

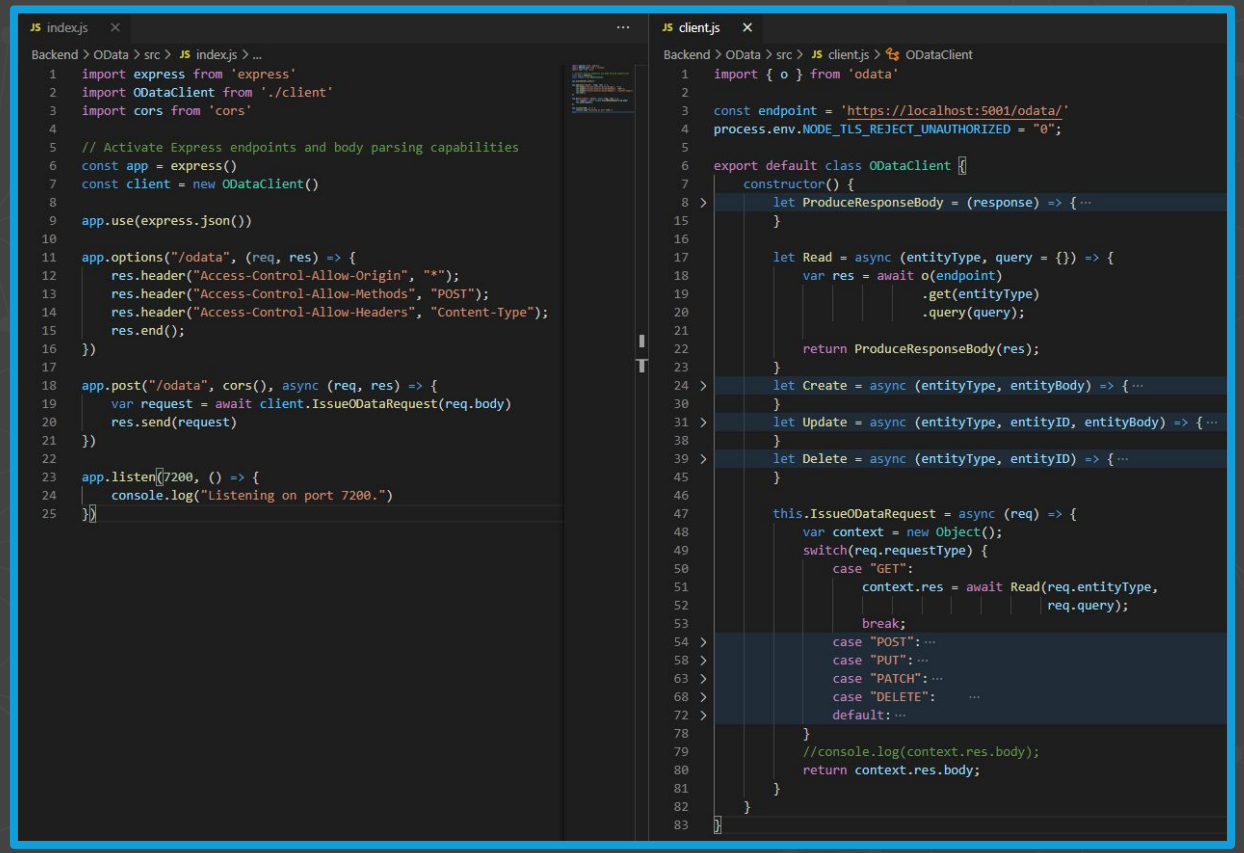
- The OData requests invoked by the *BaseController* have to be resolved for retrieving the entity.
- In this case, the entity retrieved is stored in UI5's *JSONModel*.
- The functionality of storing data in *JSONModel* classes is abstracted in this controller.

```
sap.ui.define([
    "sap/ui/core/mvc/Controller",
    "sap/ui/model/json/JSONModel",
    "sap/ui/core/UIComponent",
    "semiodesk/ui5/covidtracker/service/ODataHandler",
], function (Controller, JSONModel, UIComponent, ODataHandler) {
    "use strict";

    return Controller.extend("semiodesk.ui5.covidtracker.controller.FactoryController", {
        getClient: function () {
            return new ODataHandler();
        },
        getModel: function (sName) {···
        },
        setModel: function (oParams = {oModel, sModelAlias}) {
            if(oParams.oModel != null)
                oParams.oModel = new JSONModel(oParams.oModel);

            this.getOwnerComponent().setModel(oParams.oModel, oParams.sModelAlias);
        },
        read: function (oParams = {sEntityType, sModelAlias, oODataQuery: {}}) {
            // Trigger GET request
            let pRequest = this.getClient().Read(oParams.sEntityType, oParams.oODataQuery);

            // Resolve request
            pRequest.then((oData) => {
                this.setModel({
                    oModel: oData,
                    sModelAlias: oParams.sModelAlias
                });
            });
        },
        create: function (oParams = {sEntityType, sModelAlias, oPayload, oCallback, sODataAction: undefined}) {···
        },
```

TECHNICAL BACKGROUND

OData V4 Client with o.js

- This microservice serves as an OData V4 Client which is based on Express.js and o.js.
- It takes some JSON payload as parameter and maps the request to the targeted OData service.
- In other words, it takes the request from UI5 and forwards it to ASP.NET Core which in turn returns the result as a *Promise*.



index.js

```
import express from 'express'
import ODataClient from './client'
import cors from 'cors'

// Activate Express endpoints and body parsing capabilities
const app = express()
const client = new ODataClient()

app.use(express.json())

app.options("/odata", (req, res) => {
    res.header("Access-Control-Allow-Origin", "*");
    res.header("Access-Control-Allow-Methods", "POST");
    res.header("Access-Control-Allow-Headers", "Content-Type");
    res.end();
})

app.post("/odata", cors(), async (req, res) => {
    var request = await client.IssueODataRequest(req.body)
    res.send(request)
})

app.listen(7200, () => {
    console.log("Listening on port 7200.")
})
```

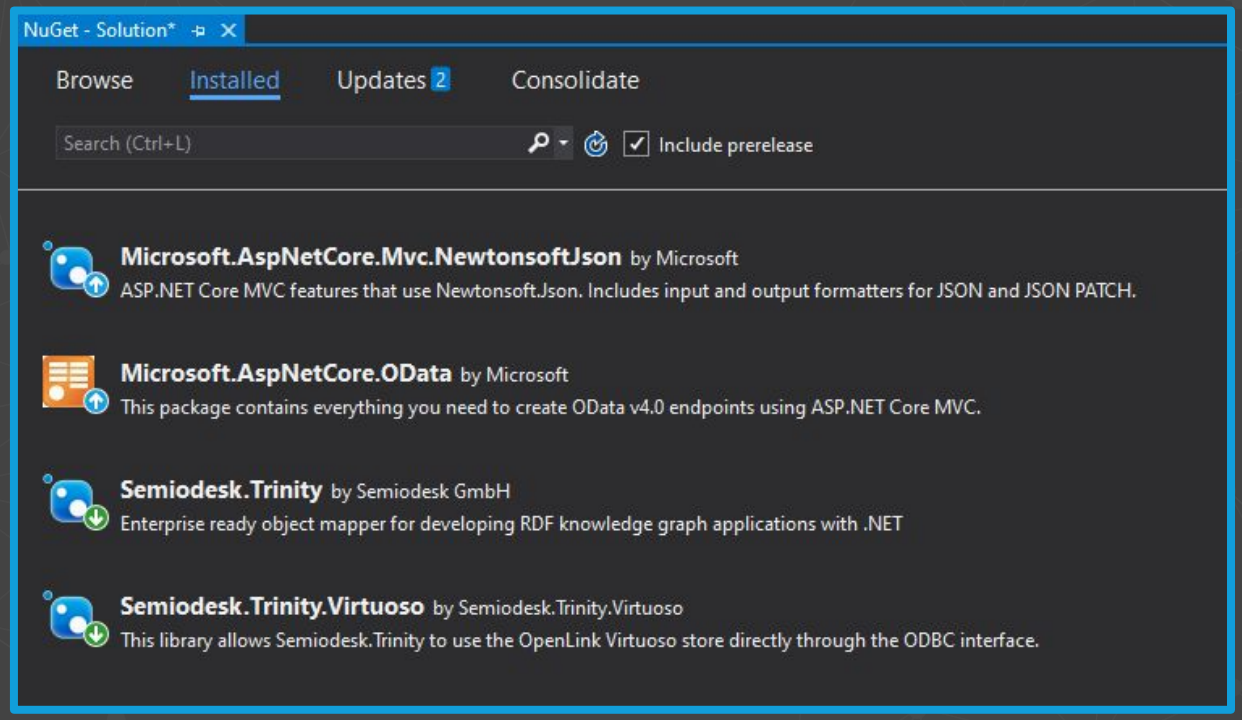
client.js

```
import { o } from 'odata'

const endpoint = 'https://localhost:5001/odata/'
process.env.NODE_TLS_REJECT_UNAUTHORIZED = "0";

export default class ODataClient {
    constructor() {
        let ProduceResponseBody = (response) => { ···
        }

        let Read = async (entityType, query = {}) => {
            var res = await o(endpoint)
                            .get(entityType)
                            .query(query);

            return ProduceResponseBody(res);
        }
        let Create = async (entityType, entityBody) => { ···
        }
        let Update = async (entityType, entityID, entityBody) => { ···
        }
        let Delete = async (entityType, entityID) => { ···
        }

        this.IssueODataRequest = async (req) => {
            var context = new Object();
            switch(req.requestType) {
                case "GET":
                    context.res = await Read(req.entityType,
                                             req.query);
                    break;
                case "POST": ···
                case "PUT": ···
                case "PATCH": ···
                case "DELETE": ···
                default: ···
            }
            //console.log(context.res.body);
            return context.res.body;
        }
    }
}
```

TECHNICAL BACKGROUND

C#: External Dependencies

- To enable Trinity RDF and OData for Knowledge Engineering and Object-Mapping purposes, the following dependencies need to be installed via NuGet Package Manager.



TECHNICAL BACKGROUND

C#: Dependency Injection

- The *Startup* class is an important element in an ASP.NET Core application as it is responsible for binding external dependencies to it.
- Inside this class, Trinity RDF, OData and Newtonsoft.Json need to be added.

```
2 references
public class Startup
{
    0 references
    public Startup(IConfiguration configuration)...

    1 reference
    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this method to add services to the container.
    0 references
    public void ConfigureServices(IServiceCollection services)
    {
        // Adds the relevant dependencies such as Newtonsoft.Json, Trinity and OData to the container.
        services.AddControllers().AddNewtonsoftJson();
        services.AddOData();

        services.AddScoped<DbContext>();
        DbContext virtuoso = new DbContext();
        virtuoso.Initialise();
    }

    // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
    0 references
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())...

        app.UseHttpsRedirection();
        app.UseRouting();
        app.UseAuthorization();

        // The following lines enable OData and its batching capabilities for HTTP routing.
        app.UseODataBatching();
        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllers();
            endpoints.Select().Expand().Filter().OrderBy().MaxTop(10).Count();
            endpoints.MapODataRoute("odata", "odata", ODataConfig.GetEdmModel(), new DefaultODataBatchHandler());
        });
    }
}
```

TECHNICAL BACKGROUND

C#: Trinity RDF Configuration

- The *Startup* class uses the *DbContext* class for initializing Trinity RDF.
- It contains the base configuration as to how Trinity RDF should be launched.
- Other databases such as Stardog can be used with Trinity RDF.

```
7 references
public class DbContext
{
    private string connectionString = "provider=virtuoso;host=localhost;port=1111;uid=dba;pw=dba;rule=urn:example/ruleset";
    private Uri defaultModelUri = CRT.Namespace;
    3 references
    public IStore Store { get => StoreFactory.CreateStore(connectionString); }

    4 references
    public IModel DefaultModel
    {
        get
        {
            Store.Log = (query) => Console.WriteLine(query);
            return Store.GetModel(defaultModelUri);
        }
    }

    1 reference
    public void Initialise()
    {
        StoreFactory.LoadProvider<VirtuosoStoreProvider>();

        // UNCOMMENT THE FOLLOWING LINE FOR PREVENTING THE ONTOLOGY TO BE OVERWRITTEN BY ITS DEFAULT VERSION
        //if (Store.GetModel(defaultModelUri).IsEmpty)
            Store.InitializeFromConfiguration(Path.Combine(Environment.CurrentDirectory, "ontologies.config"));

        OntologyDiscovery.AddAssembly(Assembly.GetExecutingAssembly());
        MappingDiscovery.RegisterCallingAssembly();
    }
}
```

TECHNICAL BACKGROUND

C#: Referencing Ontologies

- For enabling the Object-Mapping capabilities of Trinity RDF, an *ontologies.config* file must be created.
- It references the file location of every Ontology inside the *Ontologies* folder which must be located in the *project root*.
- Trinity RDF automatically creates the vocabulary from every Ontology as *Ontologies.g.cs* which can be reused across the entire application.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <ontologies namespace="ContactTracingGraph">

    <!--http://www.w3.org/1999/02/22-rdf-syntax-ns#-->
    <ontology uri="http://www.w3.org/1999/02/22-rdf-syntax-ns#" prefix="rdf">
      <filesource location="Ontologies/rdf.ttl"/>
    </ontology>

    <!--http://www.w3.org/2000/01/rdf-schema#-->
    <ontology uri="http://www.w3.org/2000/01/rdf-schema#" prefix="rdfs">
      <filesource location="Ontologies/rdfs.ttl"/>
    </ontology>

    <!--http://www.w3.org/2002/07/owl#-->
    <ontology uri="http://www.w3.org/2002/07/owl#" prefix="owl">
      <filesource location="Ontologies/owl.ttl"/>
    </ontology>

    <!--http://www.covid-rdf-tracker.com/graph#-->
    <ontology uri="http://www.covid-rdf-tracker.com/graph#" prefix="crt">
      <filesource location="Ontologies/crt.ttl"/>
    </ontology>

  </ontologies>
  <stores>
    <store type="virtuoso">
      <data>
        <rulesets>
          <ruleset uri="urn:example/ruleset">
            <graph uri="http://www.w3.org/1999/02/22-rdf-syntax-ns#" />
            <graph uri="http://www.w3.org/2000/01/rdf-schema#" />
            <graph uri="http://www.w3.org/2002/07/owl#" />
            <graph uri="http://www.covid-rdf-tracker.com/graph#" />
          </ruleset>
        </rulesets>
      </data>
    </store>
  </stores>
</configuration>
```

TECHNICAL BACKGROUND

C#: OData Configuration

- The *Startup* class uses the *ODataConfig* class for enabling OData routing with MVC.
- *SetEntitySets()* binds Model classes to the *EDM (Entity Data Model)*.
- *ActivateODataMethods()* enables OData Functions and Actions.
- *SuppressTrinityRdf()* ensures that OData and Trinity RDF can harmonize together.

```
1 reference
public class ODataConfig
{
    1 reference
    public static IEdmModel GetEdmModel()
    {
        var builder = new ODataConventionModelBuilder();
        SetEntitySets(builder);

        var tnt = builder.EntityType<ODataResource>();
        SuppressTrinityRdf(tnt);
        ActivateODataMethods(tnt);
        return builder.GetEdmModel();
    }

    1 reference
    private static void SetEntitySets(ODataModelBuilder b)
    {
        b.EntitySet<Person>("Person");
        ...
    }

    1 reference
    private static void SuppressTrinityRdf<T>(EntityTypeConfiguration<T> entity) where T : ODataResource
    {
        entity.Ignore(v => v.IsDisposed);
        entity.Ignore(v => v.IsNew);
        entity.Ignore(v => v.IsBlank);
        entity.Ignore(v => v.IsSynchronized);
        entity.Ignore(v => v.Language);
        entity.Ignore(v => v.Model);
        entity.Ignore(v => v.Uri);
    }

    1 reference
    private static void ActivateODataMethods<T>(EntityTypeConfiguration<T> entity) where T : ODataResource
    {
        var aSubmitDiagnosis = entity.Collection.Action("SubmitDiagnosis");
        aSubmitDiagnosis.Parameter<string>("ID");
        aSubmitDiagnosis.ReturnsFromEntitySet<InfectiousDisease>("InfectiousDisease");
        ...
    }
}
```

TECHNICAL BACKGROUND

C#: Object-Mapping with OData

- In OData, each Entity must contain a [Key] annotated property, i.e. an ID.
- In Trinity RDF, each Entity must inherit from the *Resource* class.
- Since the *Resource* class has no ID but a URI only, an *ODataResource* class inheriting from it will be used where the URI's Fragment Identifier will be used as ID instead.

```
14 references
public class ODataResource : Resource
{
    4 references
    public ODataResource(Uri uri) : base(uri) { }

    4 references
    public ODataResource(string id, string classUri) : base(GenerateUri(id, classUri)) { }

    4 references
    public ODataResource(string classUri) : base(GenerateUri(null, classUri)) { }

    [Key]
    5 references
    public string ID {
        get => Uri.Fragment.Substring(1);
        set {} // set must be given because of OData
    }

    // Generates the URI based on the ID and the class resource name
    2 references
    private static Uri GenerateUri(string id, string type)...
}
```

TECHNICAL BACKGROUND

C#: Entity Modelling

- Any Entity must inherit from *ODataResource* for Trinity RDF.
- The [Rdf] annotations for class and properties allow Trinity RDF to map the data retrieved from the data store to the relevant class instance.
- Each annotation property takes the stringified URI of the data property as parameter. These can be referenced from the *Ontologies.g.cs* class.

```
[RdfClass(CRT.Person)]
18 references
public class Person : ODataResource
{
    [JsonConstructor]
    0 references
    public Person(string ID) : base(ID, CRT.Person) {}

    0 references
    public Person() : base(CRT.Person) {}

    0 references
    public Person(Uri uri) : base(uri) {}

    [RdfProperty(CRT.covidHealthLevel)]
    2 references
    public int CovidHealthLevel { get; set; }

    [RdfProperty(CRT.diagnosis)]
    1 reference
    public List<InfectiousDisease> Diagnosis { get; set; }

    [RdfProperty(CRT.encounter)]
    1 reference
    public List<Encounter> Encounter { get; set; }
}
```

TECHNICAL BACKGROUND

C#: Designing Entity Controller

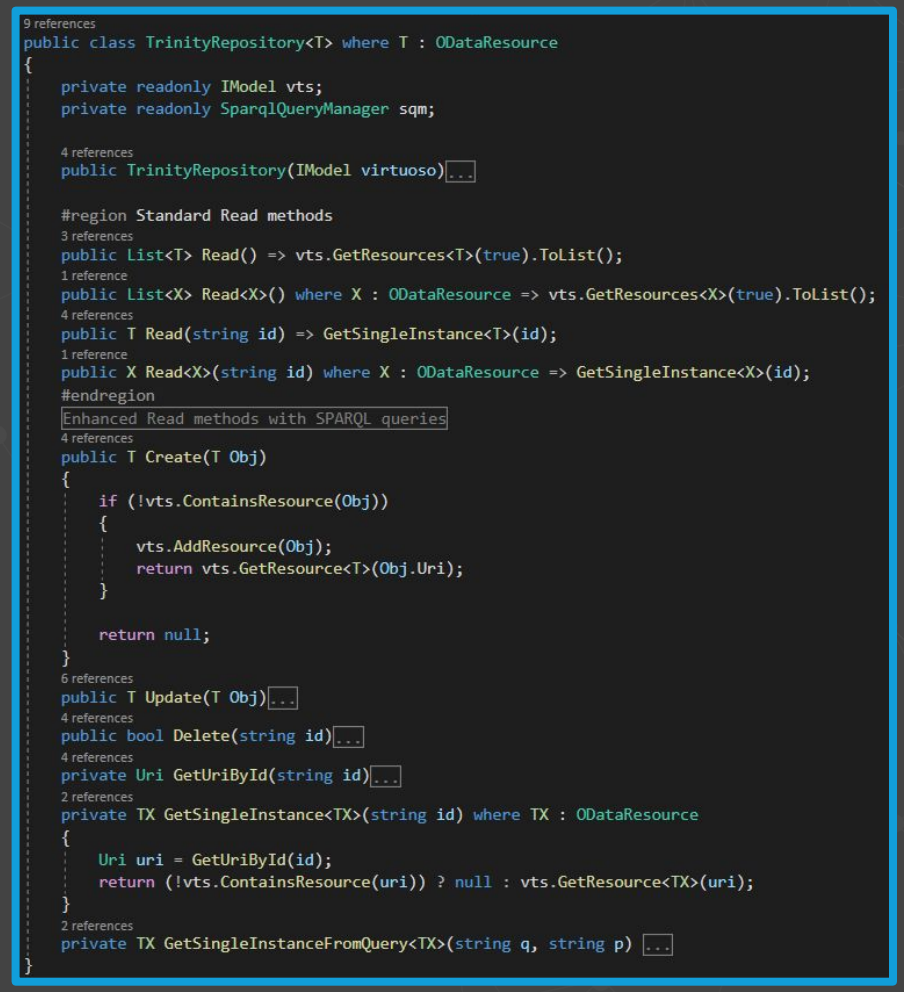
- To build APIs for some Entity in OData, their Controller classes must inherit from the *ODataController* class.
- Each *ODataController* is annotated with an [ODataRoutePrefix].
- The [EnableQuery] annotation enables querying with OData.
- This Controller encapsulates Trinity RDF entirely with the Repository Pattern.

```
[ODataRoutePrefix("Place")]
1 reference
public class PlaceController : ODataController
{
    private readonly TrinityRepository<Place> repo;

    0 references
    public PlaceController(DbContext trinity)
    {
        repo = new TrinityRepository<Place>(trinity.DefaultModel);
    }

    [EnableQuery]
    0 references
    public IActionResult Get()
    {
        return Ok(repo.Read());
    }

    [EnableQuery]
    0 references
    public IActionResult Get(string key)
    {
        Place Obj = repo.Read(key);
        return (Obj is null) ? (IActionResult) NotFound() : Ok(Obj);
    }

    0 references
    public IActionResult Post([FromBody]dynamic Obj)
    {
        Obj = repo.Create(Obj.ToObject<Place>());
        return (Obj is null) ? BadRequest() : Ok(Obj);
    }

    0 references
    public IActionResult Put([FromBody] dynamic Obj, [FromODataUri] string key)...

    0 references
    public IActionResult Delete([FromODataUri] string key)...
}
```

TECHNICAL BACKGROUND

C#: Repository Pattern

- The DB persistence in Trinity RDF is abstracted with the Repository Pattern in order to promote code reusability and single responsibility.
- This pattern ensures that the same code is not duplicated across all the Controller classes.
- Using Generics, it is possible to represent some Template classes with any *ODataResource* class.



```
9 references
public class TrinityRepository<T> where T : ODataResource
{
    private readonly IModel vts;
    private readonly SparqlQueryManager sqm;

    4 references
    public TrinityRepository(IModel virtuoso)...

    #region Standard Read methods
    3 references
    public List<T> Read() => vts.GetResources<T>(true).ToList();
    1 reference
    public List<X> Read<X>() where X : ODataResource => vts.GetResources<X>(true).ToList();
    4 references
    public T Read(string id) => GetSingleInstance<T>(id);
    1 reference
    public X Read<X>(string id) where X : ODataResource => GetSingleInstance<X>(id);
    #endregion
    Enhanced Read methods with SPARQL queries
    4 references
    public T Create(T Obj)
    {
        if (!vts.ContainsResource(Obj))
        {
            vts.AddResource(Obj);
            return vts.GetResource<T>(Obj.Uri);
        }

        return null;
    }
    6 references
    public T Update(T Obj)...
    4 references
    public bool Delete(string id)...
    4 references
    private Uri GetUriById(string id)...
    2 references
    private TX GetSingleInstance<TX>(string id) where TX : ODataResource
    {
        Uri uri = GetUriById(id);
        return (!vts.ContainsResource(uri)) ? null : vts.GetResource<TX>(uri);
    }
    2 references
    private TX GetSingleInstanceFromQuery<TX>(string q, string p) ...
}
```

Conclusion.

CONCLUSION

Takeaways

~1day

Implementation Time

- Implemented the app in record time
  - Even with OData V4 work-around
  - …
- Graph Database made backend implementation simple..
  - No need to implement graph search
  - …
- Re-Using open ontology saved time
  - No considerable data modelling efforts

CONCLUSION

Statistics

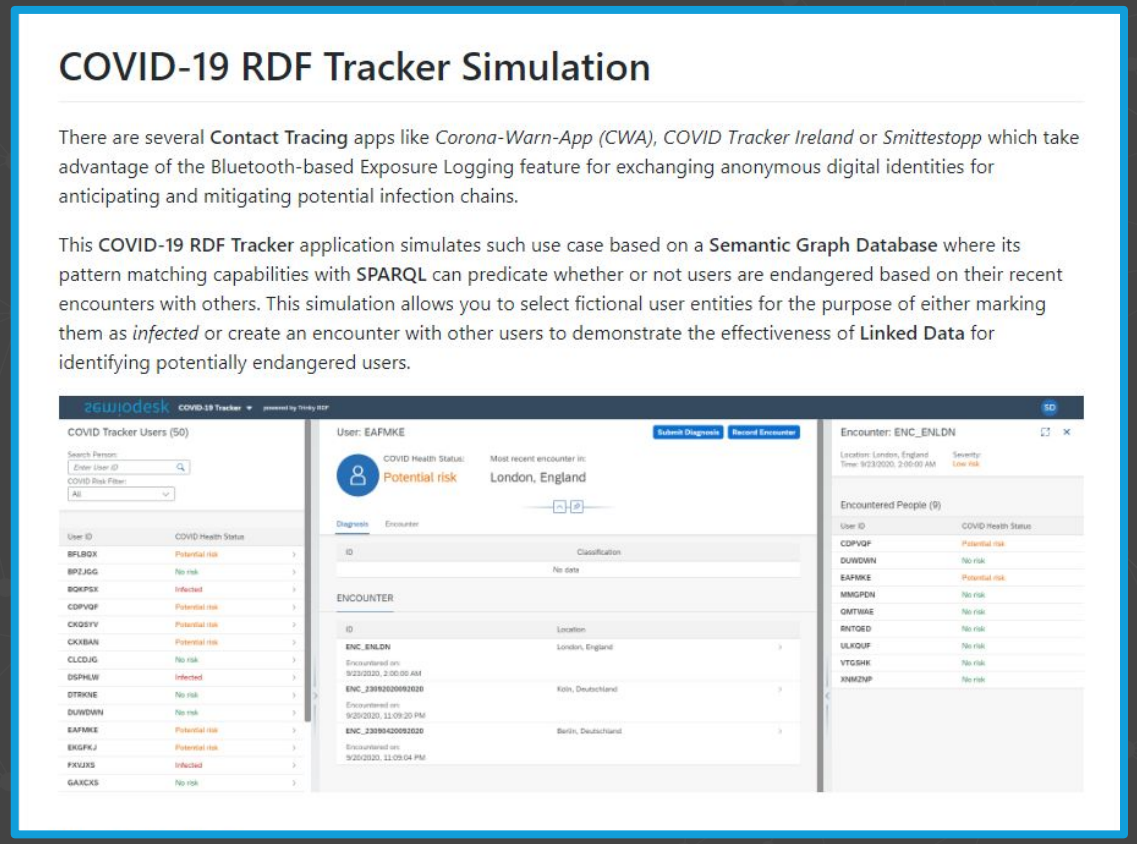
Component	Technology	Skill Level	Development Time (in min.)
Frontend - OpenUI5	OpenUI5 (JavaScript)	Entry	180
Backend - OData V4 Client	Node.js (JavaScript)	Advanced	20
Backend - Linked Data Server	ASP.NET Core (C#)	Advanced	60
		Total Development Time (in minutes)	260

- It took about 4 hours and 20 minutes to deliver the architected solution.
  - OpenUI5 was developed with VS Code instead of SAP Web IDE.
  - Breaks have not been considered for this measurement.
  - Skill level also determines pace of development.

CONCLUSION

Our server is your server!

- By reusing our provided solution, you don’t need to configure OData and Trinity RDF from scratch.
- You only need to plug in your ontology and design the *Entity* and their *Controller* classes and run the backend.
- You will save significant amount of development time on the Linked Data server. Even less than 20 minutes!
- The entire application can be downloaded from GitHub.



semiodesk

EVERYTHING CONNECTS.

Questions?

Download the demo app here:
https://github.com/andyle13/semiodesk-ui5-trinityrdf-contacttracing

Contact us to talk about your projects:

Stefan Summesberger
Head of Business Development

stefan@semiodesk.com

https://semiodesk.com

semiodesk

EVERYTHING CONNECTS.

Thank you!