Structured Data as JSON-LD

Vanderbilt Linked Data Working Group

2018-09-24

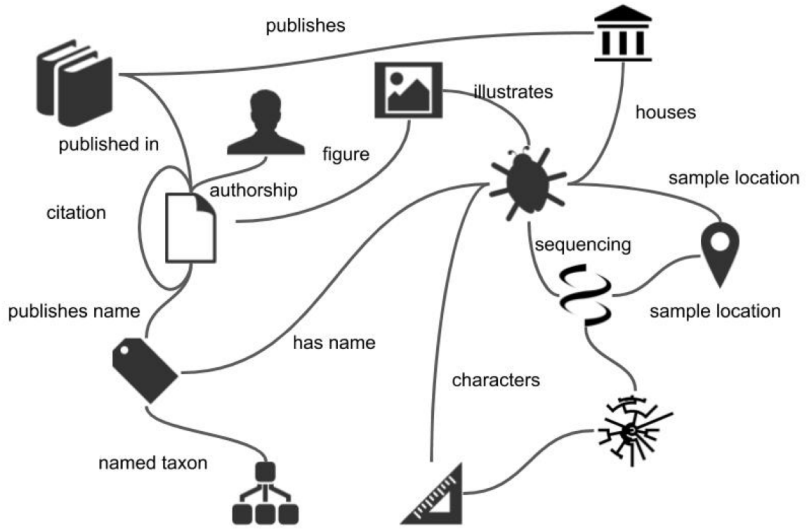
Rod Page @rdmpage

http://iphylo.blogspot.com

Recall from our first meeting:

Ozymandias: A biodiversity knowledge graph

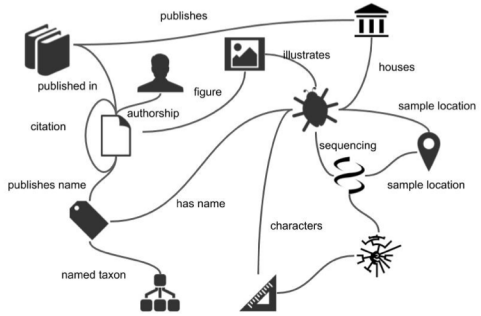
#knowledgegraph #semanticweb #linkeddata



Recall from our first meeting:

Obstacles to building knowledge graphs

- Need globally unique, persistent identifiers (**how to label the nodes of the graph**)
- Need to create and agree on vocabularies (**how to label the edges of the graph**)
- Need to agree how to transmit the graph
- Who stores the global graph?
- “Killer apps”

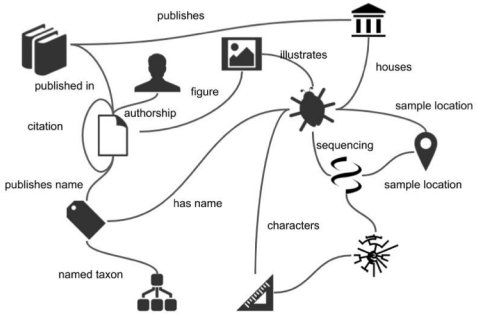


Recall from our first meeting:

A new hope…

- The **identifier wars** are (nearly) over (DOIs FTW)
- Lots of domain-specific vocabularies, **but schema.org is "good enough" for most things**
- XML becoming a bedtime story to frighten the children

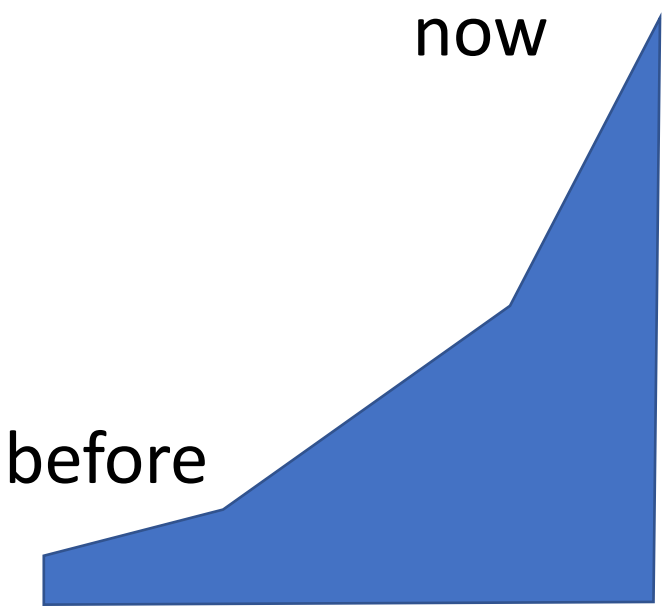
  JSON is everywhere (JSON-LD FTW).
- **Wikidata** as global knowledge graph ***(or is it the Google Knowledge Graph???)***
- apps? ***(Google search results are the ultimate app…)***



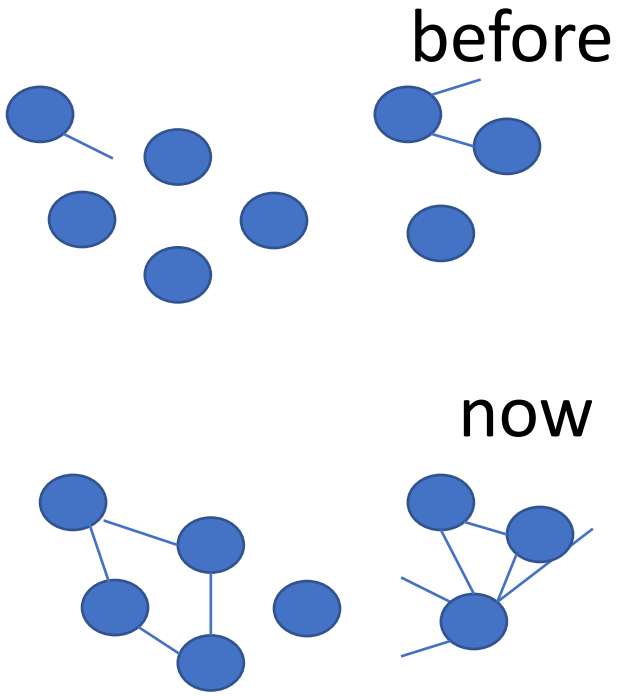
Recall from our first meeting:

How do we measure progress?

Linear growth (easy)

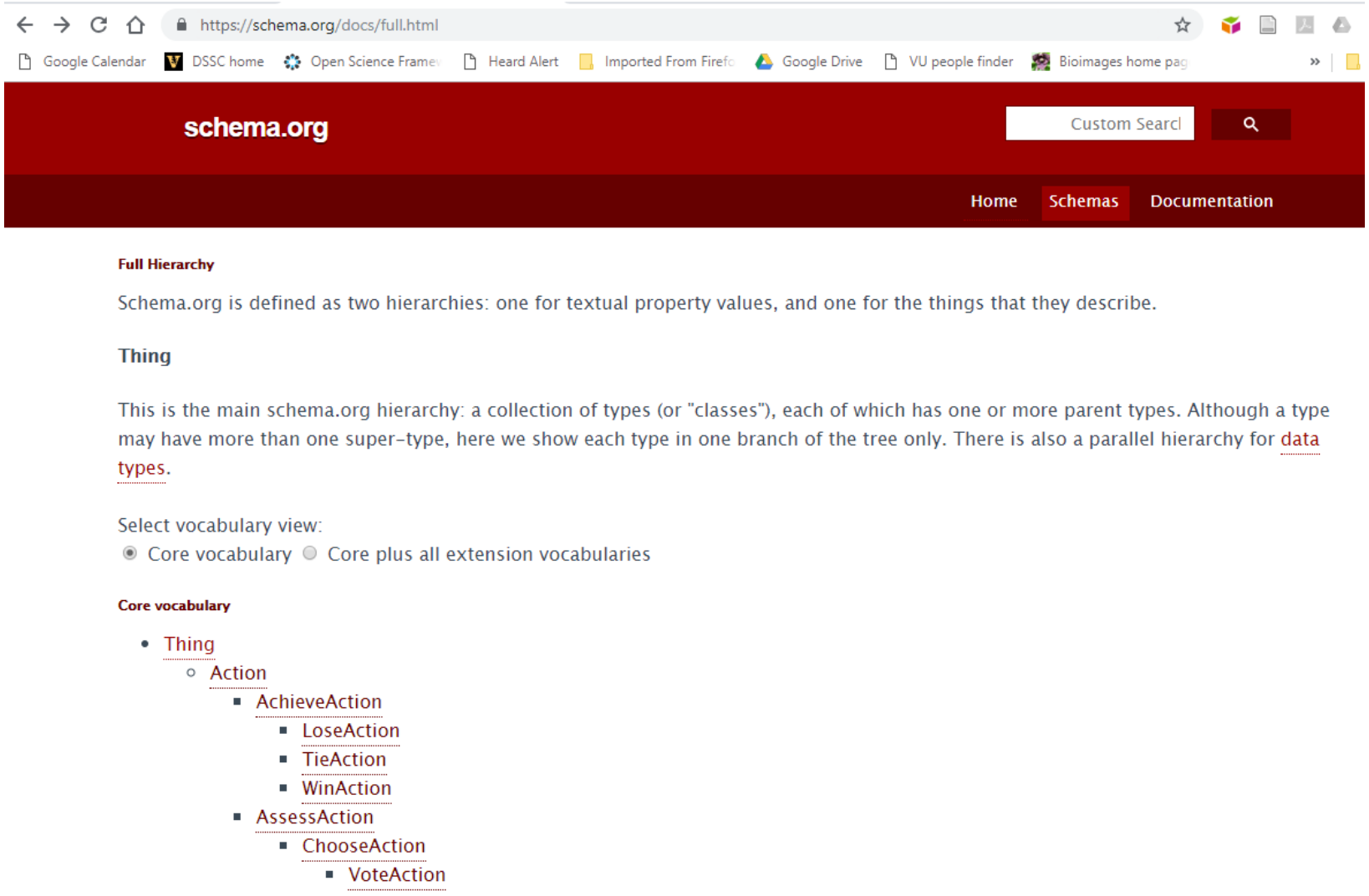


Connectivity (hard)



The schema.org vocabulary

https://schema.org/docs/full.html



There are many schema.org properties

Blog

Canonical URL: http://schema.org/Blog

Thing > CreativeWork > Blog

A blog.

Usage: Over 1,000,000 domains

[more...]

Property	Expected Type	Description
Properties from Blog		
blogPost	BlogPosting	A posting that is part of this blog. Supersedes blogPosts.
issn	Text	The International Standard Serial Number (ISSN) that identifies this serial publication. You can repeat this property to identify different formats of, or the linking ISSN (ISSN-L) for, this serial publication.
Properties from CreativeWork		
about	Thing	The subject matter of the content. Inverse property: subjectOf.
accessMode	Text	The human sensory perceptual system or cognitive faculty through which a person may process or perceive information. Expected values include: auditory, tactile, textual, visual, colorDependent, chartOnVisual, chemOnVisual, diagramOnVisual, mathOnVisual, musicOnVisual, textOnVisual.
accessModeSufficient	Text	A list of single or combined accessModes that are sufficient to understand all the intellectual content of a resource. Expected values include: auditory, tactile, textual, visual.
accessibilityAPI	Text	Indicates that the resource is compatible with the referenced accessibility API (WebSchemas wiki lists possible values).
accessibilityControl	Text	Identifies input methods that are sufficient to fully control the described resource (WebSchemas wiki lists possible values).
accessibilityFeature	Text	Content features of the resource, such as accessible media, alternatives and supported enhancements for accessibility (WebSchemas wiki lists possible values).

but which ones will Google actually use ???

Metadata on people

subject unique identifier (URI) – like the primary key of the record

columns = properties (predicates)

id	familyName	givenName	email	url
https://orcid.org/0000-0003-4365-3135	Baskauf	Steven	steve.baskauf@vanderbilt.edu	https://my.vanderbilt.edu/baskauf/
https://orcid.org/0000-0003-0328-0792	Anderson	Clifford	clifford.anderson@vanderbilt.edu	https://www.cliffordanderson.net/
https://orcid.org/0000-0002-8449-6263	Steffanick	Adam	adam.steffanick@vanderbilt.edu	https://www.steffanick.com/adam/

cells = values (objects)

Property/value pairs about a subject:

```
https://orcid.org/0000-0003-4365-3135

        familyName "Baskauf";
       givenName "Steven";
       email "steve.baskauf@vanderbilt.edu";
       url "https://my.vanderbilt.edu/baskauf/".
```

Making terms globally unique using namespaces

`https://schema.org/familyName` has a well-defined meaning.

Let the namespace abbreviation `schema:` stand for `https://schema.org/`

We can then abbreviate `https://schema.org/familyName` as `schema:familyName`

Our property/value example becomes

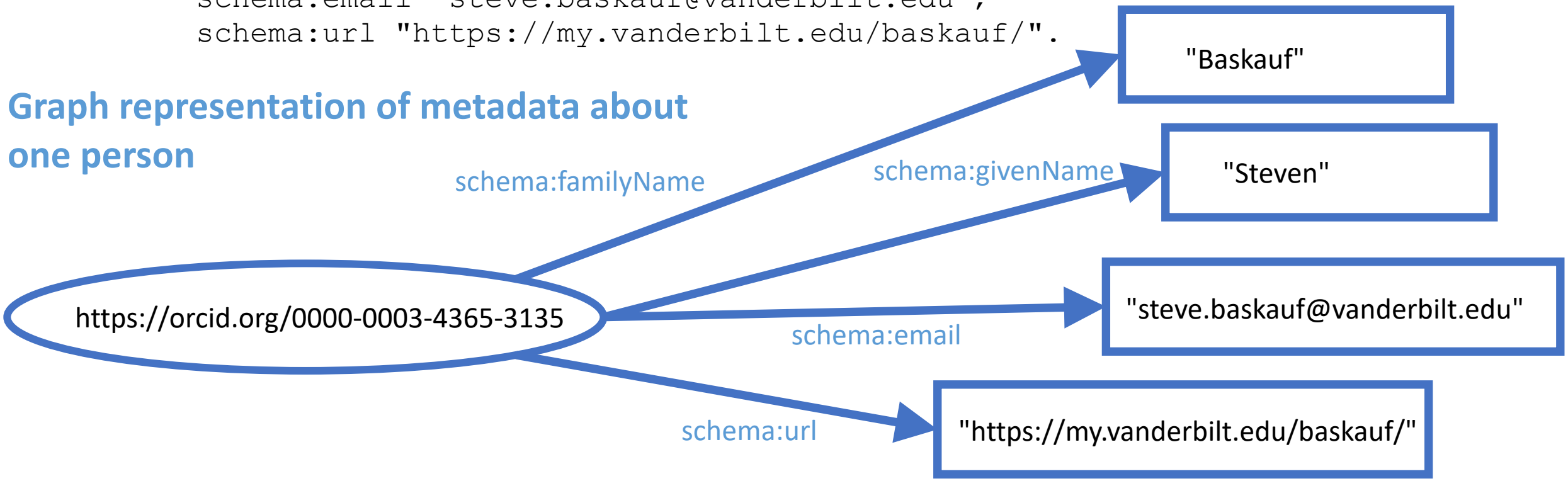
```
<https://orcid.org/0000-0003-4365-3135>
       schema:familyName "Baskauf";
       schema:givenName "Steven";
       schema:email "steve.baskauf@vanderbilt.edu";
       schema:url "https://my.vanderbilt.edu/baskauf/".
```

(Putting a URI in angle brackets indicates that we are using it as a unique identifier for a thing.)

Diagramming property/value pairs about a person as a Linked Data graph

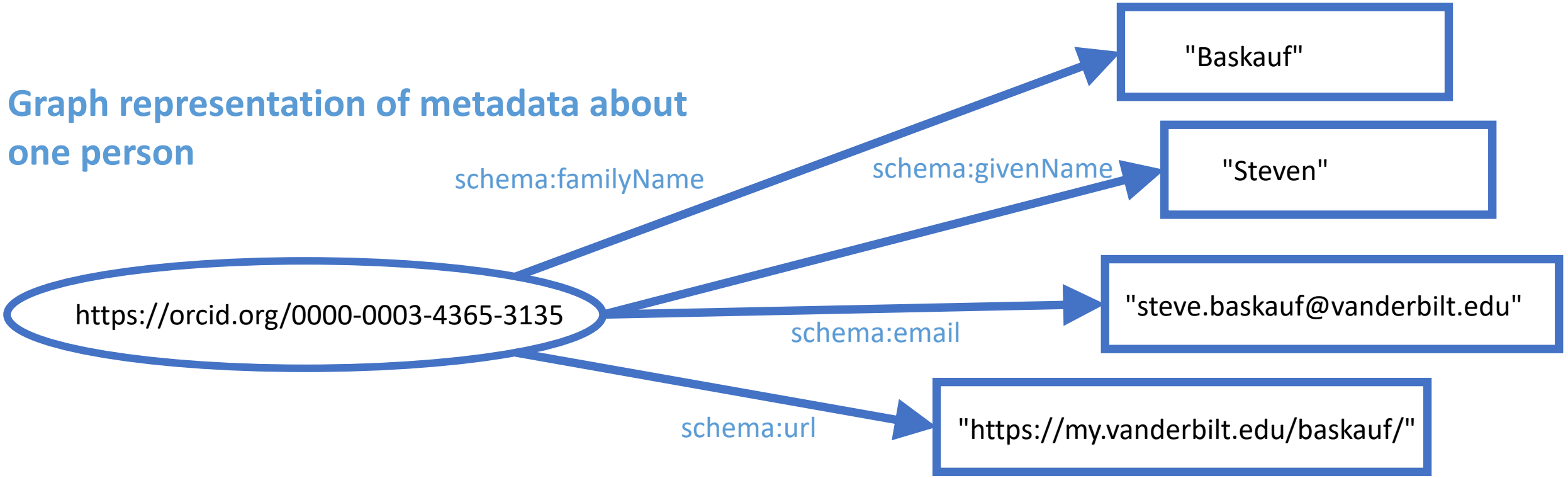
```
<https://orcid.org/0000-0003-4365-3135>
        schema:familyName "Baskauf";
        schema:givenName "Steven";
        schema:email "steve.baskauf@vanderbilt.edu";
        schema:url "https://my.vanderbilt.edu/baskauf/".
```

Graph representation of metadata about one person



Resource Description Framework (RDF) Triples

Graph representation of metadata about one person



Each arrow on the graph can be represented as one triple (subject/predicate/object):

Subject	predicate	object
<https://orcid.org/0000-0003-4365-3135>	schema:familyName	"Baskauf".
<https://orcid.org/0000-0003-4365-3135>	schema:givenName	"Steven".
<https://orcid.org/0000-0003-4365-3135>	schema:email	"steve.baskauf@vanderbilt.edu".
<https://orcid.org/0000-0003-4365-3135>	schema:url	"https://my.vanderbilt.edu/baskauf/".

Writing RDF triples in Turtle syntax

Subject	predicate	object
<https://orcid.org/0000-0003-4365-3135>	schema:familyName	"Baskauf".
<https://orcid.org/0000-0003-4365-3135>	schema:givenName	"Steven".
<https://orcid.org/0000-0003-4365-3135>	schema:email	"steve.baskauf@vanderbilt.edu".
<https://orcid.org/0000-0003-4365-3135>	schema:url	"https://my.vanderbilt.edu/baskauf/".

Each statement reads like a sentence and ends in a period.

```
@prefix schema: <http://schema.org/>.
<https://orcid.org/0000-0003-4365-3135>
       schema:familyName "Baskauf";
       schema:givenName "Steven";
       schema:email "steve.baskauf@vanderbilt.edu";
       schema:url "https://my.vanderbilt.edu/baskauf/".
```

Turtle reads like a more complicated sentence where predicates and objects that share the same subject are separated by semicolons.

For more information:

- Some basics about Linked Data https://github.com/HeardLibrary/semantic-web/blob/master/2016-fall/linked-data-ch1.pdf
- RDF primer video: https://youtu.be/XAGifYBiXMY

What is JSON?

- The basic unit of JSON is a **key:value pair**. For example:

  `"name":"Steve"` (strings must be in quotes)

  `"fingers":10` (numbers don't need quotes)

- A **JSON object** is a list of key:value pairs inside curly brackets.

  `{"name":"Steve", "fingers":10, "street":"Keri Drive"}`

- Multiple values can be put in an **array** inside square brackets.

  `{"name":["Steve","Steven","Esteban"],"street":"Keri Drive"}`

- Note: a key:value pair is basically the same thing as a property/value pair.

Whitespace

- Whitespace is not important – it can be used to make the JSON structure clearer. The following mean exactly the same thing:

```
{"name":["Steve","Steven","Esteban"], "fingers":10, "street":"Keri
Drive"}
```

```
{"name":["Steve","Steven","Esteban"],
 "fingers":10,
 "street":"Keri Drive"}
```

```
{
  "name":
          [
          "Steve",
          "Steven",
          "Esteban"
          ],
  "fingers":10,
  "street":"Keri Drive"
}
```

What is JSON-LD?

- JSON-LD is **valid JSON**.
- JSON-LD is structured so that **key:value pairs "mean something"**. They tell how the subject being described by the JSON-LD is related to other things represented by the values.
- The **subject** being described is identified by a special key "**@id**":

```
"@id":"https://orcid.org/0000-0003-4365-3135"
```

- The namespace **abbreviation** being used for keys is specified by "**@context**":

```
"@context": "https://schema.org/"
```

- The **class** (kind) of the thing is specified by "**@type**":

```
"@type": "Person"
```

- Other **property/value** (or predicate/object) **pairs** are given by key:value pairs:

```
{"name":["Steve","Steven","Esteban"],"street":"Keri Drive"}
```

Describing a person using JSON-LD and schema.org terms

- Here is the Turtle we had before (with the type schema:Person added):

```
@prefix schema: <http://schema.org/>.
<https://orcid.org/0000-0003-4365-3135>
       schema:familyName "Baskauf";
       schema:givenName "Steven";
       schema:email "steve.baskauf@vanderbilt.edu";
       schema:url "https://my.vanderbilt.edu/baskauf/"
       a schema:Person.
```

- Here's how the metadata from the Turtle would look as JSON-LD:

```
{
  "@context": "https://schema.org/",
  "@id": "https://orcid.org/0000-0003-4365-3135",
  "@type": "Person",
  "familyName": "Baskauf",
  "givenName": "Steve",
  "email": "steve.baskauf@vanderbilt.edu",
  "url": ""https://my.vanderbilt.edu/baskauf/"
}
```

- Notice that once we define the default context, we don't have to use a namespace with the keys

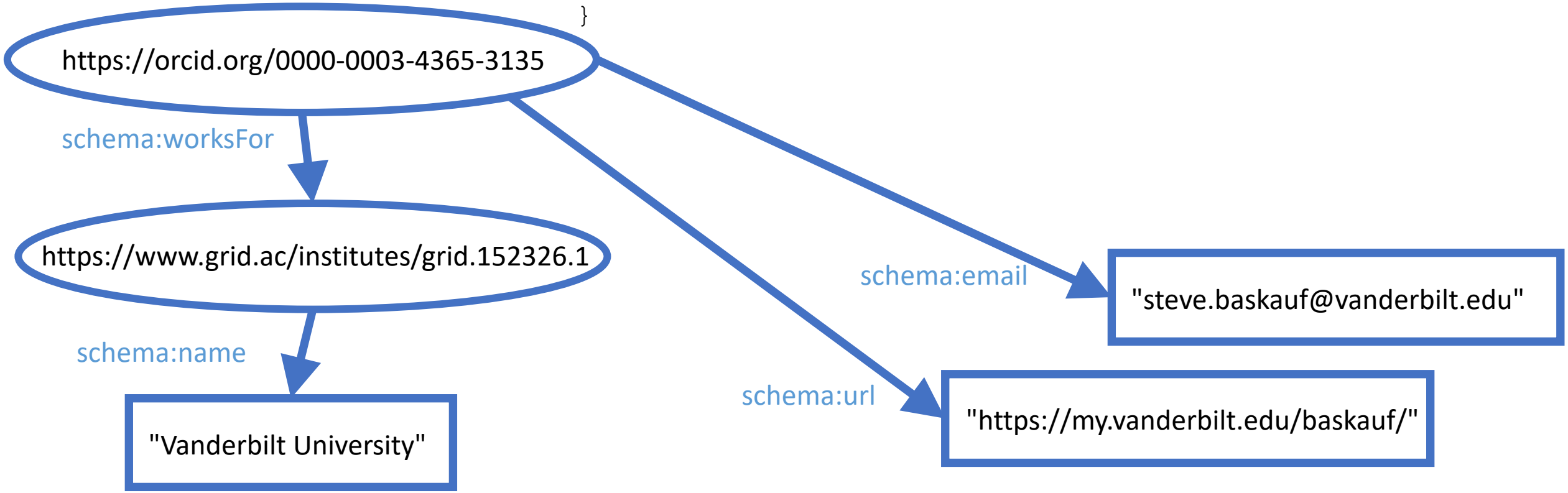
Repeating values using arrays

- In JSON-LD it does not work to repeat properties with the same subject resource. Instead, use an **array** for the list of values

```
{
  "@context": "https://schema.org/",
  "@id": "https://orcid.org/0000-0003-4365-3135",
  "@type": "Person",
  "familyName": "Baskauf",
  "givenName": "Steve",
  "name": [
      "Steven J. Baskauf",
      "Steve Baskauf",
      "Steven James Baskauf"
    ],
  "email": "steve.baskauf@vanderbilt.edu",
  "url": ""https://my.vanderbilt.edu/baskauf/"
}
```

What if the value (i.e. object) is an entity, not a string?

```
{
"@context": "https://schema.org/",
"@id": "https://orcid.org/0000-0003-4365-3135",
"email": "steve.baskauf@vanderbilt.edu",
"worksFor": {
    "name": "Vanderbilt University",
    "@id": "https://www.grid.ac/institutes/grid.152326.1"
    },
"url": ""https://my.vanderbilt.edu/baskauf/"
}
```



The final product:


```
{
  "@context": "https://schema.org/",
  "@id": "https://orcid.org/0000-0003-4365-3135",
  "@type": "Person",
  "name": [
      "Steven J. Baskauf",
      "Steve Baskauf",
      "Steven James Baskauf"
    ],
  "familyName": "Baskauf",
  "givenName": "Steve",
  "email": "steve.baskauf@vanderbilt.edu",
  "worksFor": {
      "name": "Vanderbilt University",
      "@id": "https://www.grid.ac/institutes/grid.152326.1",
      "@type": "Organization"
    },
  "image": "https://avatars3.githubusercontent.com/u/5765781?s=400&u=ce07e6c5611a3f4d0abb82593736e2c4eb56b095&v=4",
  "url": "https://my.vanderbilt.edu/baskauf/",
  "sameAs": [
      "http://viaf.org/viaf/63557389",
      "http://www.wikidata.org/entity/Q40670042"
    ]
}
```


This is available as a Gist at https://bit.ly/2Oe8hhQ (https://gist.github.com/baskaufs)

Test the JSON-LD

- Go to the JSON-LD playground (https://json-ld.org/playground/)
- Copy and paste the Gist from https://bit.ly/2Oe8hhQ into the JSON-LD Input box. If it's valid there will be no error message below the box.
- Click on the N-Quads and Table output tabs to see the RDF triples.
- Click on the Visualized output tab to see a diagram. Click on the blue and yellow dots to expand the diagram.

Find out what Google can learn from the JSON-LD structured data

- Go to the Google Structured Data Testing Tool: https://search.google.com/structured-data/testing-tool
- Click on the Code Snippet tab, then paste the code.
- Click the Run Test button and check whether the output on the right makes sense.

Embedding the JSON-LD structured data in an HTML document

- Enclose the JSON-LD inside a script tag:

```
<script type="application/ld+json">
  {
 (JSON-LD goes here)
  }
</script>
```

- Place the script element in the head of the HTML document (the BODY is also OK)

```
<html>
  <head>
     <script type="application/ld+json">
       {
         (JSON-LD goes here)
       }
     </script>
  </head>
  <body>
     (page content here)
  </body>
</html>
```

Google's take on structured data

- For a good introduction, see Google's "Introduction to Structured Data" https://developers.google.com/search/docs/guides/intro-structured-data
- JSON-LD is now the preferred method for exposing structured data.
- Google doesn't promise to actually use your structured data
  - You can improve the likelihood of it being used
  - It's possible required properties for the content type aren't available
  - Likelihood depends on conformance.

Will Google use it? Five important factors

- Do your structured data comply with technical guidelines? (valid JSON-LD, using schema.org terms, proper structure)
- Are you following quality guidelines? (structured data consistent with page content, not "spammy", up-to-date)
- Do your data conform to one of the content types? See the sidebar at https://developers.google.com/search/docs/data-types/article
- More recommended features = more likely to be used
- Do your pages follow the AMP HTML specification? See https://www.ampproject.org/docs/fundamentals/spec

Hackable examples for content types

- Article (including blog post and videos; see also specific type for video):
  - guidelines: https://developers.google.com/search/docs/data-types/article
  - click on "see markup" button to hack a template using the Structured Data Testing Tool https://search.google.com/structured-data/testing-tool
- Dataset (in pilot) https://developers.google.com/search/docs/data-types/dataset
- Event https://developers.google.com/search/docs/data-types/event
- Podcast https://developers.google.com/search/docs/data-types/podcast