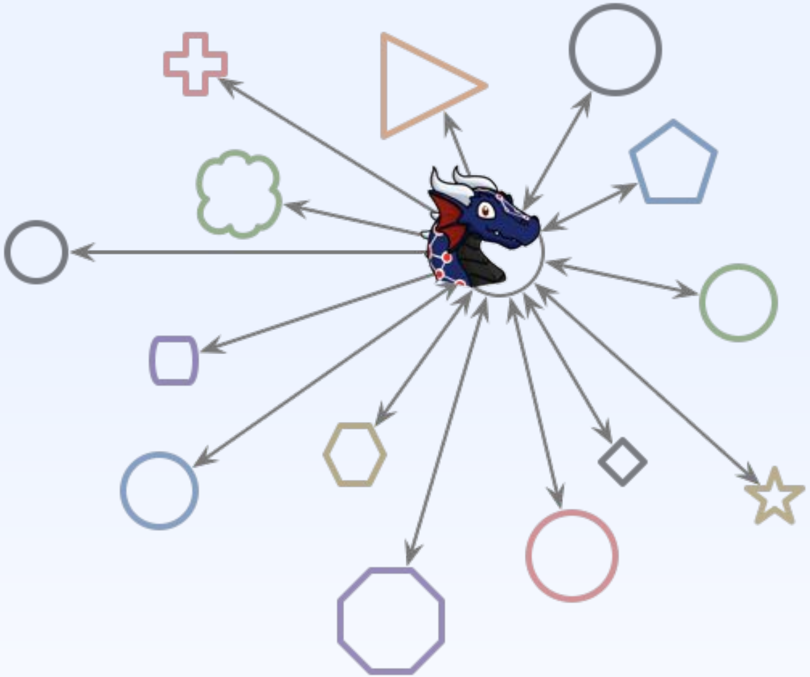
Anything-to-Graph

Knowledge Graph Conference

May 2021



Joshua Shinavier, PhD (🐦: joshsh)

Data @ Uber

KGC The Knowledge Graph Conference

Overview

- Building graphs
- Models
- Mappings
- Use cases
- TinkerPop 4

Building graphs

There are graphs, and graphs

- Domain-specific graphs are simplest
  - ETL a few data sources into a graph
  - Mappings can be written and maintained by hand
  - Off-the-shelf tools work well
- Difficulty increases with
  - **Complexity** of source schemas
  - Diversity of **ownership**, **quality**, and **governance** in data sources
  - Lack of **standardization** on languages and vocabulary
- Some challenges are organizational, others technical
  - **Organizational**: see Lessons From Reality (KGC 2019)
  - **Technical**: let's take a closer look

The challenges of heterogeneity

- Diverse data sources → need supporting **metadata**
  - E.g. see Uber's Databook
- Diverse data governance → need better data **isolation**
  - Typically, RDF triple stores do better than PG databases here
- Diverse domain data models → need **standardized schemas**
  - E.g. see Uber's Data Standardization
- Diverse schema and data languages → need well-defined **mappings**

Enter the Dragon

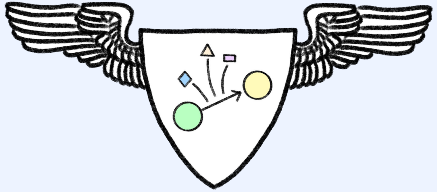
Prerequisites

- Data must conform to a **schema**
  - It is OK to have a mix of schemas, and schema languages
- **Unique identifiers** must be clearly distinguished, and typed
  - What is this UUID field? Does it identify a User, a Document, etc.?
- Some degree of **standardization** is needed
  - Are identifiers consistent across data sources?
  - Can *this* timestamp value be compared with *that* timestamp value? Etc.
- Need well-defined **mappings**
  - From each data language into a graph format
  - From each schema language into a graph schema language
  - ...without losing too much information
  - ...and while maintaining consistency between schema and data

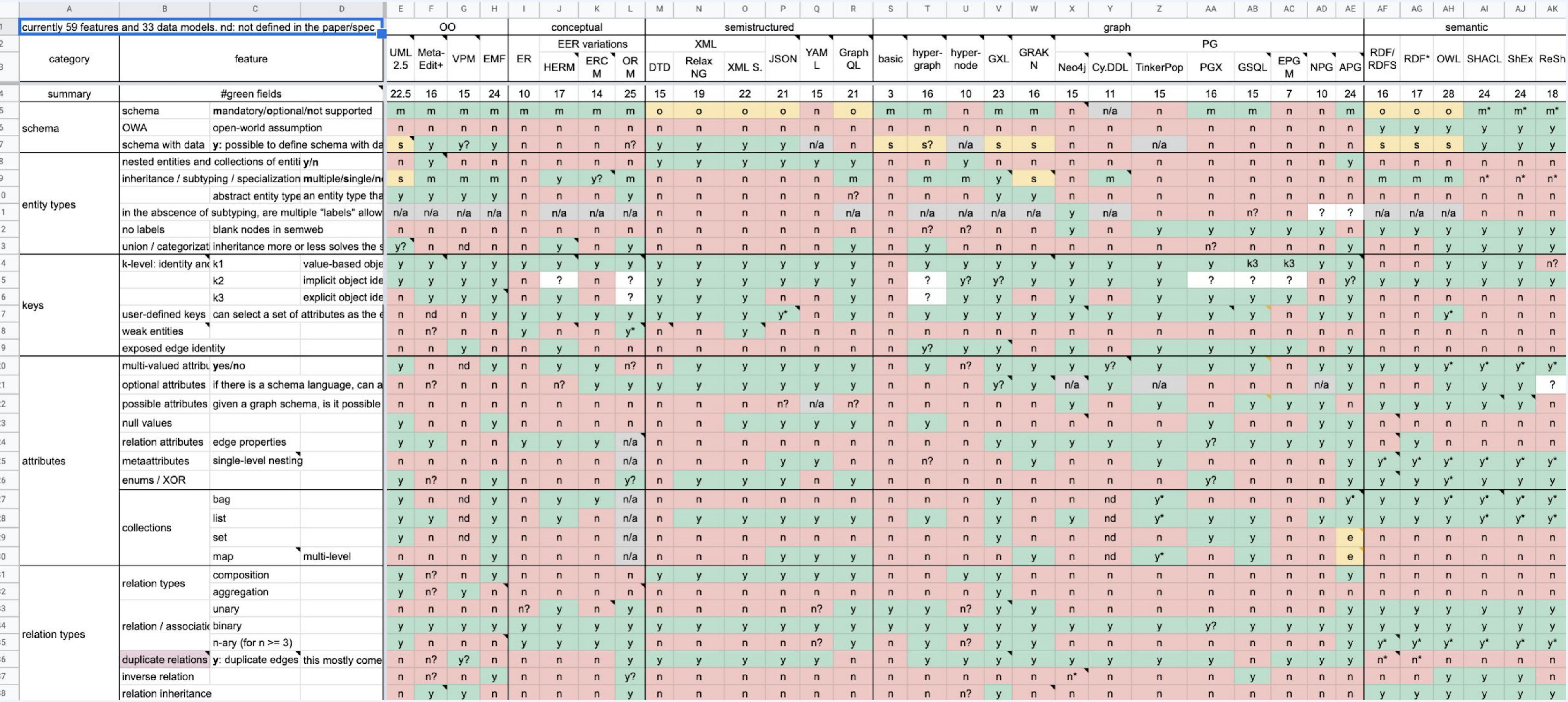
Models

Is there a “universal data model” for graphs?

- Desirable characteristics
  - **Centrality**: ease of alignment with other graph and non-graph models
  - **Flexibility**: captures a wide variety of graph structures
  - **Formality**: serves as a tool for reasoning about data models
  - **Practicality**: serves as a basis for inference, query optimization, etc.
  - **Intuitiveness**: a good graph schema captures our mental model
- Where can the right abstractions be found?
  - **Logics**? **Set theory**? **Algebras**? **Category theory**?
- Does the data model already exist?
  - What about **RDF**-based languages (RDFS, OWL, SHACL, ShEx, etc.)?
- Is that what **GQL** is going to be?



Graph features are very diverse

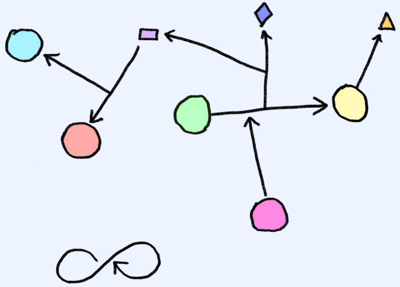


	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK
1	currently 59 features and 33 data models. nd: not defined in the paper/spec				OO				conceptual				semistructured						graph													semantic					
2	category	feature			UML 2.5	Meta-Edit+	VPM	EMF	ER	EER variations			XML			JSON	YAML	GraphQL	basic	hyper-graph	hyper-node	GXL	GRAKN	PG								RDF/RDFS	RDF*	OWL	SHACL	ShEx	ReSh
3										HERM	ERCM	ORM	DTD	Relax NG	XML S.									Neo4j	Cy.DDL	TinkerPop	PGX	GSQL	EPGM	NPG	APG						
4	summary	#green fields			22.5	16	15	24	10	17	14	25	15	19	22	21	15	21	3	16	10	23	16	15	11	15	16	15	7	10	24	16	17	28	24	24	18
5	schema	schema	**m**andatory/**o**ptional/**n**ot supported		m	m	m	m	m	m	m	m	o	o	o	o	n	o	m	m	n	m	m	n	n/a	n	m	m	n	n	m	o	o	o	m*	m*	m*
6		OWA	open-world assumption		n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	y	y	y	y	y	y
7		schema with data	**y**: possible to define schema with da		s	y	y?	y	n	n	n	n?	y	y	y	y	n/a	n	s	s?	n/a	s	s	n	n	n/a	n	n	n	n	n	s	s	s	y	y	y
8	entity types	nested entities and	collections of entiti	**y/n**	n	y	n	n	n	n	n	n	y	y	y	y	y	y	n	n	y	n	n	n	n	n	n	n	n	n	y	n	n	n	n	n	n
9		inheritance / subtyping / specialization		**m**ultiple/**s**ingle/**n**	s	m	m	m	n	y	y?	m	n	n	n	n	n	m	n	m	m	y	s	n	m	n	n	n	n	n	n	m	m	m	n*	n*	n*
10			abstract entity type	an entity type tha	y	y	y	y	n	n	n	y	n	n	n	n	n	n?	n	n	n	y	y	n	n	n	n	n	n	n	n	n	n	n	n	n	n
11		in the absence of subtyping, are multiple "labels" allow			n/a	n/a	n/a	n/a	n	n/a	n/a	n/a	n	n	n	n	n	n/a	n	n/a	n/a	n/a	n/a	y	n/a	n	n	n?	n	?	?	n/a	n/a	n/a	n	n	n
12		no labels	blank nodes in semweb		n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n?	n?	n	n	y	n	y	y	y	y	y	n	y	y	y	y	y	y
13		union / categorizat	inheritance more or less solves the s		y?	n	nd	n	n	y	n	y	n	n	n	n	n	y	n	y	n	n	n	n	n	n	n?	n	n	n	y	n	n	y	y	y	y
14	keys	k-level: identity anc	k1	value-based obje	y	y	y	y	y	y	y	y	y	y	y	y	y	y	n	y	y	y	y	y	y	y	y	k3	k3	y	y	n	n	y	y	y	n?
15			k2	implicit object ide	y	y	y	y	n	?	n	?	y	y	y	y	y	y	n	?	y?	y?	y	y	y	y	?	?	?	n	y?	y	y	y	y	y	y
16			k3	explicit object ide	n	y	y	y	n	y	n	?	y	y	y	n	n	y	n	?	y	y	n	y	n	y	y	y	y	n	y	n	n	n	n	n	n
17		user-defined keys	can select a set of attributes as the e		n	nd	n	y	y	y	y	y	y	y	y	y*	n	y	n	y	y	y	y	y	y	y	y	y	n	y	y	n	n	y*	n	n	n
18		weak entities			n	n?	n	n	y	n	n	y*	n	n	y	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n
19		exposed edge identity			n	n	y	n	n	y	n	n	n	n	n	n	n	n	n	y?	y	y	n	y	n	y	y	y	y	n	y	n	n	n	n	n	n
20	attributes	multi-valued attribu	**yes/no**		y	n	nd	y	n	y	y	n?	n	y	y	y	y	y	n	y	n?	y	y	y	y?	y	y	y	n	y	y	y	y	y*	y*	y*	y*
21		optional attributes	if there is a schema language, can a		n	n?	n	n	n	n?	y	y	y	y	y	y	y	y	n	n	n	y?	y	n/a	y	n/a	n	n	n	n/a	y	n	n	y	y	y	?
22		possible attributes	given a graph schema, is it possible		n	n	n	n	n	n	n	n	n	n	n	n?	n/a	n?	n	n	n	n	n	y	n	y	n	y	y	y	n	y	y	y	y	y	n
23		null values			y	n	n	y	n	n	n	n	n	n	y	y	y	y	n	y	n	n	n	n	n	n	y	n	n	y	y	n	n	n	n	n	n
24		relation attributes	edge properties		y	y	n	n	y	y	y	n/a	n	n	n	n	n	n	n	n	n	y	y	y	y	y	y?	y	y	y	y	n	y	n	n	n	n
25		metaattributes	single-level nesting		n	n	n	n	n	n	n	n/a	n	n	n	y	y	n	n	n?	n	n	y	n	n	y	n	n	n	n	y	y*	y*	y*	y*	y*	y*
26		enums / XOR			y	n?	n	y	n	n	n	y?	n	y	y	y	n	y	n	n	n	n	n	n	n	n	y?	n	n	n	y	y	y	y*	y	y	y
27		collections	bag		y	n	nd	y	n	y	y	n/a	n	n	n	n	n	n	n	n	n	y	n	n	nd	y*	n	n	n	n	y*	y	y	y*	y*	y*	y*
28			list		y	y	nd	y	n	y	n	n/a	n	y	y	y	y	y	n	y	n	y	n	y	nd	y*	y	y	n	y	y	y	y	y	y*	y*	y*
29			set		y	n	nd	y	n	n	n	n/a	n	n	n	n	n	n	n	n	n	y	n	n	nd	n	y	y	n	n	e	n	n	n	n	n	n
30			map	multi-level	n	n	n	y	n	n	n	n/a	n	n	n	y	y	y	n	n	n	n	y	n	nd	y*	n	y	n	n	e	n	n	n	n	n	n
31	relation types	relation types	composition		y	n?	n	y	n	n	n	n	y	y	y	y	y	y	n	n	y	y	n	n	n	n	n	n	n	n	y	n	n	n	n	n	n
32			aggregation		y	n?	y	n	n	n	n	n	n	n	n	n	n	n	n	n	n	y	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n
33		relation / associatic	unary		n	n	n	n	n?	y	n	y	n	n	n	n	n?	y	y	y	n?	y	y	n	n	n	n	n	n	n	y	y	y	y	y	y	y
34			binary		y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y?	y	y	y	y	y	y	y	y	y	y
35			n-ary (for n >= 3)		y	n	n	n	y	y	y	y	n	n	n	n	n?	y	n	y	n?	y	y	n	n	n	n	n	n	n	y	y*	y*	y*	y*	y*	y*
36		duplicate relations	**y**: duplicate edges	this mostly come	n	n?	y?	n	n	n	n	y	y	y	y	y	y	n	n	y	y	y	y	y	y	y	y	n	y	y	y	n*	n*	n	n	n	n
37		inverse relation			n	n?	n	y	n	n	n	y?	n	n	n	n	n	n	n	n	n	n	n	n*	n	n	n	y	n	n	n	n	n	y	y	y	n
38		relation inheritance			n	y	y	n	n	n	n	y	n	n	n	n	n	n	n	n	n?	y	n	n	n	n	n	n	n	n	n	y	y	y	y	y	y

Spreadsheet by Gábor Szárnyas

Complexity is not the answer

- No one language will ever satisfy all graph use cases
- Implementing the model must be simple, or developers won't
  - Who remembers CODASYL? Why aren't we using it?
  - Go for **minimalism + extensibility**
- Dimensions being explored for GQL
  - Mathematical **foundations**
  - **Nominal** vs. **structural** typing
  - **Inheritance** vs. **composition**
  - Graph **element types** vs. **property types**
  - **Descriptive** vs. **prescriptive** schemas



A little algebra goes a long way

- **Algebraic Property Graphs**[1]
    - Pragmatic and opinionated graph data model
- Schemas are vertex/edge/etc. labels bound to **algebraic data types**
- Formally defined using **category theory**
- Effectively a subset of SHACL
- Ideally, GQL will also be a superset

$$\begin{tikzcd}
G(\mathcal{E}) \arrow[r, "G(\lambda)"] \arrow[d, "G(\upsilon)"'] & G(\mathcal{L}) \arrow[d, "G(\sigma)"] \\
G(\mathcal{V}) \arrow[r, "G(\tau)"] & G(\mathcal{T})
\end{tikzcd}$$

[1] https://arxiv.org/abs/1909.04881

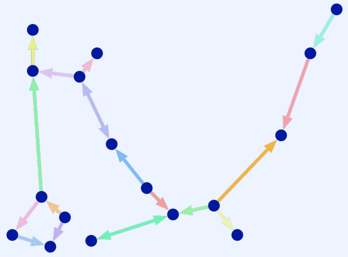
Mappings

Desirable properties

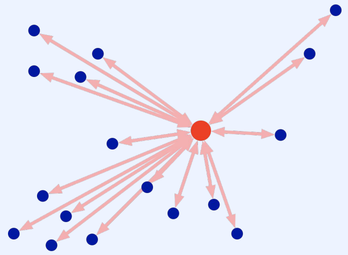
- **Composability**
  - If we have f : A ⟶ B and g : B ⟶ C, we should also have f;g : A ⟶ C
- **Bidirectionality**
  - If we have f : A ⟶ B, we should also have f^{-1} : B ⟶ A, with $f;f^{-1} \cong f^{-1};f \cong id$
- Data : schema **consistency**
  - Mappings should be defined for data and schemas languages in parallel
  - If data d conforms to schema s, then we need f(d) to conform to f(s)

Topology of mappings

- Arbitrary / ad-hoc topology
  - Pros: seems easiest; just use the pairwise mappings you have
  - Cons: more indirection, and mappings may not compose well



- **Star topology**
  - Pros: mappings compose well, and all paths are short
  - Cons: need to define/identify a central data model (the “dragon”)



Transforming schemas and data

- **Schema**-level mappings
  - Transformations on data types
- **Data**-level mappings
  - Transformations on instances of data types
- Schema and data-level mappings must be **consistent**
  - a :: A ⇒ F(a) :: F(A)
- Use common intermediate representations

$$\begin{tikzcd}
G_1(\mathcal{E}) \arrow[r, "G_1(\lambda)"'] \arrow[d, "G_1(\upsilon)"'] \arrow[rrr, bend left, "h_{\mathcal{E}}"] & G_1(\mathcal{L}) \arrow[d, "G_1(\sigma)"] \arrow[rrr, bend left, "h_{\mathcal{L}}"] & & G_2(\mathcal{E}) \arrow[r, "G_2(\lambda)"'] \arrow[d, "G_2(\upsilon)"'] & G_2(\mathcal{L}) \arrow[d, "G_2(\sigma)"] \\
G_1(\mathcal{V}) \arrow[r, "G_1(\tau)"] \arrow[rrr, bend right, "h_{\mathcal{V}}"'] & G_1(\mathcal{T}) \arrow[rrr, bend right, "h_{\mathcal{T}}"'] & & G_2(\mathcal{V}) \arrow[r, "G_2(\tau)"] & G_2(\mathcal{T})
\end{tikzcd}$$

Dragon

- Framework for data and schema **migration**
  - Developed for Uber’s Data Standardization
- Dragon data model
  - Extension of Algebraic Property Graphs
  - Covers the majority of schemas at Uber
- Sources and targets
  - **RPC** / interface languages: Protocol Buffers, Apache Thrift, Apache Avro
  - **RDF**-based languages: SHACL, OWL
  - “Schemaless” formats: YAML, JSON
  - Programming languages: Haskell, Java, Scala
- Implementations in Haskell and Java
  - Dragon **generates** over half of its own source code (from YAML)

```
$ dragon transform -i Protobuf -o SHACL $SCHEMAS -d maps /tmp/shacl
               ______.    \______/    .______
Dragon 0.3.4     ----___   O .. O   ___----
---------------------vvv----vvvv----vvv---------------------
Loading configuration from dragon.yaml
Reading from Protobuf starting at maps in base directory /Users/joshsh/projects/uber/example/idl
Found 3 *.proto sources in /Users/joshsh/projects/uber/example/idl/maps
Visiting 3 files (total of 0 so far):
    /Users/joshsh/projects/uber/example/idl/maps/coordinates.proto
    /Users/joshsh/projects/uber/example/idl/maps/geometry.proto
    /Users/joshsh/projects/uber/example/idl/maps/position.proto
Visiting 3 files (total of 3 so far):
    /Users/joshsh/projects/uber/example/idl/physics/units.proto
    /Users/joshsh/projects/uber/example/idl/time/duration.proto
    /Users/joshsh/projects/uber/example/idl/time/epoch.proto
Instantiated a graph of 12 schemas
Note 1 alert:
    info: unsigned integers not supported for RDF targets. Using signed integers
Writing 12 SHACL artifacts to /tmp/shacl
```

Use cases

JSON-to-Graph

- Graph data formats are used nowhere in the enterprise
  - RDF serialization formats, GraphML, etc. are a hard sell
- **JSON** and **YAML** are used everywhere
- Give the JSON or YAML a schema, then treat it like a serialized graph
- E.g. Databook[1] snapshots to RDF

[1] https://eng.uber.com/metadata-insights-databook

gRPC-to-Graph

- Don’t let developers write individual edges / triples to the graph
  - Schema constraints are harder to enforce, errors are harder to understand
- Transform graph schemas into developer-facing schemas
  - Use familiar schema languages and protocols like Protobuf + gRPC
- Transform payload messages into graph data
  - Schema and data transformations guarantee **constraint satisfaction**

Enriching domain schemas

- Need a little more than “just Protobuf” / “just Thrift” to build a big graph
  - E.g. entity / identifier types need to be **widely re-used**
- Start with a **graph schema** (e.g. in YAML)
- Propagate logical data types into **domain schema languages**
  - At Uber: Protobuf, Thrift, Avro
- Re-use the **logical types** in many domain schemas
  - Incentivize re-use, and use schema transformations for metrics
- Now, graph-friendly types are everywhere

Toward TinkerPop 4

Dozens of graph systems

- **Alibaba** Graph Database
- Amazon **Neptune**
- **Arango**DB
- **Bitsy**
- **Blaze**graph
- **Cosmos**DB
- **Chrono**Graph
- **DSE**Graph
- **GRAKN**.AI
- **Hadoop** (Spark)
- **HGraph**DB
- **Huawei** Graph Engine Service
- **IBM** Graph
- **Janus**Graph
- **Neo**4j
- **neo**4j-gremlin-**bolt**
- **Orient**DB
- Apache **S2G**raph
- **Sqlg**
- **Stardog**
- **Tinker**Graph
- **Titan**
- Titan + **Tupl**
- **Uni**pop

Dozens of programming languages

- **Clojure**: ogre
- **Cypher**: cypher-for-gremlin

- **Elixir**: gremlex
- **Go**: grammes, gremgo

- **Haskell**: greskell, gremlin-haskell

- **Java**: Ferma, gremlin-objects, Peapod, spring-data-gremlin, gremlin-driver

- **JavaScript**: gremlin-javascript, gremlin-orm, gremlin-template-string

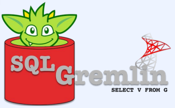
GRM

- **Kotlin**: kotlin-gremlin-ogm
- **.NET**: Gremlin.Net, Gremlinq

- **PHP**: gremlin-php

- **Python**: Goblin, gremlin-python, gremlin-py, ipython-gremlin, gremlinclient, gremlin-python, JUGRI, gremlinrestclient, python-gremlin-rest

- **Ruby**: gremlin_client
- **Rust**: gremlin-rs
- **Scala**: gremlin-scala, reactive-gremlin, scalajs-gremlin-client
- **SPARQL**: sparql-gremlin
- **SQL**: sql-gremlin
- **Typescript**: ts-tinkerpop



Interoperability via mappings

- We need **parity** across languages and systems
- Make it easier to create new **Gremlin language variants** in a consistent way
- “Escape from the JVM”
- **Code generation** can help
  - Generate graph APIs consistently into many programming languages
  - Generate grammars for parsing/validation

A unifying schema language

- A few vendor-specific schema languages exist
  - Weak and disconnected from each other
  - Also note TinkerPop's Graph.Features
- Need a real, vendor-neutral schema language
  - Facilitate **composability** of data and queries
  - Enable **inference** for type safety and optimizations
  - Propagate logical schemas into multiple graph back-ends
  - Build vendor-agnostic tools for data migration

Serialization formats for graphs

- Currently serialization options are limited
  - GraphML (XML), GraphSON (JSON), GraphBinary, Gryo (Kryo)
- Generic **JSON** and **YAML** are straightforward
- What about common **RPC formats**?
  - Protobuf, Thrift, Avro, etc.
- What about **RDF serialization** formats?
  - N-Triples, Turtle, JSON-LD, etc.
- Others?
  - Parquet? CBOR? FlatBuffers? MessagePack? Etc.

Stay tuned

- “How to Build a Dragon”
  - https://www.meetup.com/Category-Theory
- Release Dragon!
  - http://bit.ly/release_dragon
- Gremlin-users
  - https://groups.google.com/g/gremlin-users

@KGConference @joshsh

linkedin.com/company/the-knowldge-graph-conference/

youtube.com/playlist?list=PLAiy7NYe9U2Gjg-600CTV1HGypiF95d_D

KGC The Knowledge Graph Conference

Uber