Graph Abstractions Matter

(a Personal View)

Dr. Ora Lassila

Principal Technologist

Amazon Neptune

Game plan:

1. Some graph history
2. Graph abstractions, queries and access patterns
3. Graph as an abstraction: *"One graph to rule them all"*
4. The way forward

aws

A brief history of graphs and ontologies

3rd Century BCE: Categories & logic (Aristotle)

1730s: Graph theory (Euler)

1950s and onwards: Graphs as the essential underpinning of computer science

1960s: Social networks, “small-world experiment”, Erdős number (Milgram et al)

1960s-1970s: Network databases (CODASYL), semantic networks (Quillian et al)

1730s: Taxonomical classification of plants and animals (Linnaeus)

1870s: Library classification (Dewey)

1900: Semantics, ontology and logic (Husserl)

1970s-1990s: Predicate logic as the foundation of Knowledge Representation (Hayes et al)

1997 and onwards: The Semantic Web, RDF, OWL, etc. (Lassila et al)

Today: Modern knowledge graphs and graph databases

aws

A brief history of graphs and ontologies

3rd Century BCE: Categories & logic (Aristotle)

1730s: Graph theory (Euler)

1730s: Taxonomical classification of plants and animals (Linnaeus)

1950s and onwards: Graphs as the essential underpinning of computer science

1870s: Library classification (Dewey)

1960s: Social networks, "small-world experiment", Erdős number (Milgram et al)

1900: Semantics, ontology and logic (Husserl)

1960s-1970s: Network databases (CODASYL), semantic networks (Quillian et al)

1970s-1990s: Predicate logic as the foundation of Knowledge Representation (Hayes et al)

Despite all this history, programming with graphs and ontologies is still cumbersome

1997 and onwards: The Semantic Web, RDF, OWL, etc. (Lassila et al)

Today: Modern knowledge graphs and graph databases

aws

RDF

Features:

- vertices and edge types: identifiers
- graphs decompose into “triples”
- scalar data types from XSD, values are special vertices
- no composite data types; objects use graph structure

Query languages:

- SPARQL (declarative)

Labeled property graphs

Features:

- vertices and edges: objects
- no common identifier scheme
- scalar and composite data types borrowed from implementation languages

Query languages:

- Gremlin (imperative)
- Cypher (declarative)

aws

Graph abstractions

Triples are a very low-level abstraction of graphs

- akin to assembly language

Mapping graphs to trees does not necessarily work

- metamodel mismatch
- RDF/XML and JSON-LD as warning examples
- (also: syntax does not matter!)

“Object-graph mapping”…?

- how to determine what constitutes an object?
- in RDF, blank nodes can be difficult to handle

aws

Graph queries (or: *“how do I get there from here?”*)

Traversal is an excellent graph access paradigm

- *“where can I go from here given this path pattern?”*
- *“are these two nodes connected, and how?”*
- using path patterns for traversal is akin to pattern matching [Lassila 2007]

SPARQL is a "very big hammer" for many access problems

We need simple APIs:

- search, faceted search (filtering)
- *“give me everything you know about this node”* (cf. object-graph mapping)
- traversal

aws

Models and ontologies

Problems:

1. LPGs (by and large) do not have schema languages
2. RDF has too many schema languages (RDFS, OWL, SHACL)
3. how to map models to code?

What does mapping your data to a shared ontology “buy” you?

Can we support ontologies with predefined libraries?

- ”ontology engines”

What about reasoning?

Symbolic reasoning:

- generative reasoning can be easily hidden from the application/client
- restrictions as a means of data validation (SHACL rather than OWL)

Non-symbolic reasoning:

- ML techniques can be used much like symbolic reasoning

We need an expanded view of reasoning, and we need to rethink what an "application" is:

Logic should be associated with data and models, not with an application

aws

Graphs as a logical abstraction

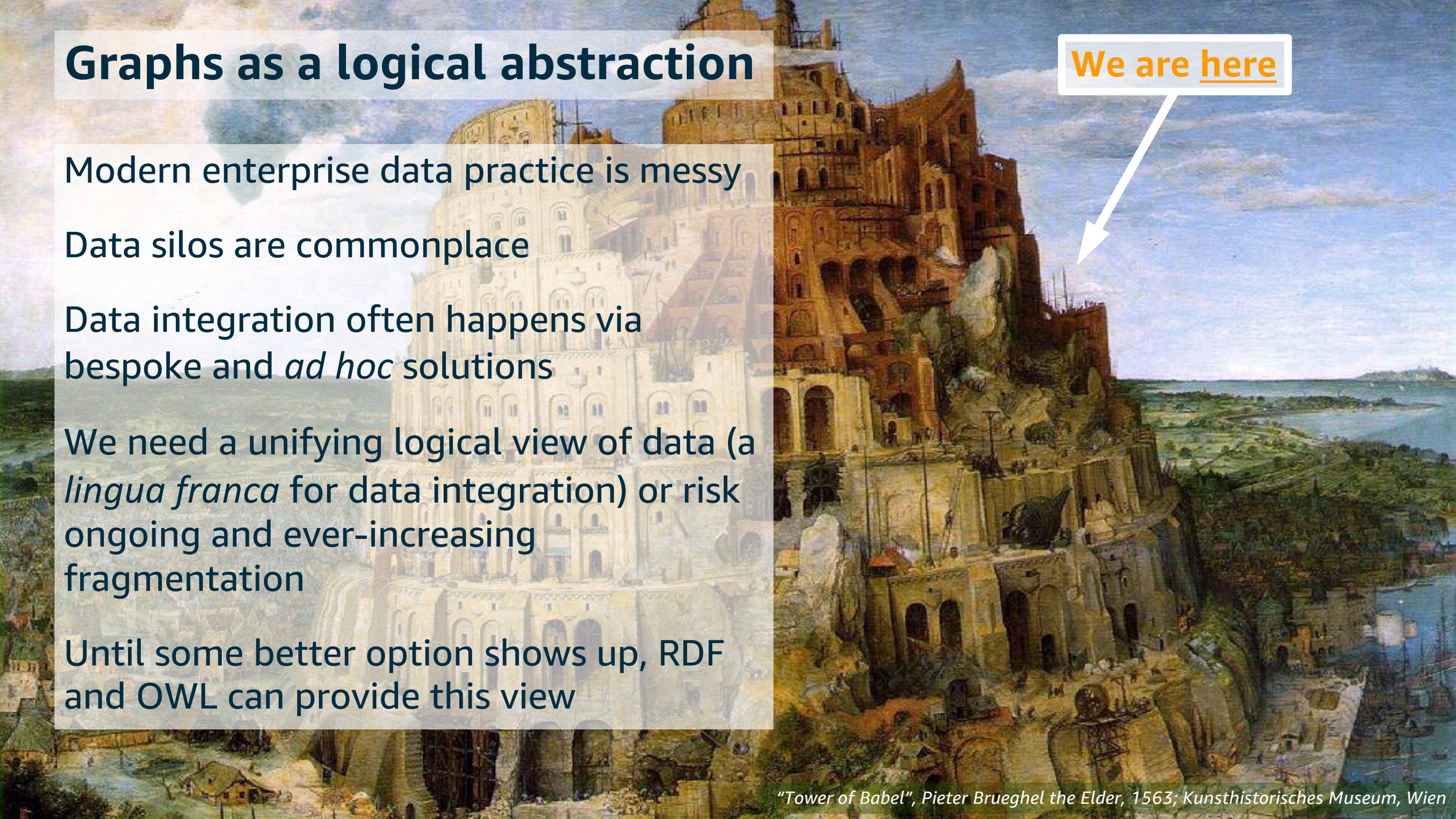
Modern enterprise data practice is messy

Data silos are commonplace

Data integration often happens via bespoke and *ad hoc* solutions

We need a unifying logical view of data (a *lingua franca* for data integration) or risk ongoing and ever-increasing fragmentation

Until some better option shows up, RDF and OWL can provide this view



"Tower of Babel", Pieter Brueghel the Elder, 1563; Kunsthistorisches Museum, Wien

RDF vs. labeled property graphs…?

Confuses users (when given the choice)

Should not matter

The real issue may in fact be

Graph as a logical representation vs. graph as a data structure

The practical problem is still that we have two separate ecosystems

aws

RDF

Pros:

- W3C standard, well established
- easy to use external data sources
- schema language(s)
- formal semantics for graph merging
- supports reasoning

Cons:

- often considered “too academic”
- no easy “on-ramp”

Labeled property graphs

Pros:

- intuitive for software developers
- integrates well with programming languages (esp. Gremlin)

Cons:

- no standard, many divergent (proprietary) implementations
- no schema language
- no formal semantics

aws

RDF

Pros:

- W3C standard, well established
- easy to use external data sources
- schema language(s)
- formal semantics for graph merging
- supports reasoning

Labeled property graphs

Pros:

- intuitive for software developers
- integrates well with programming languages (esp. Gremlin)

We want both!

aws

Graph as an abstraction

What we really want is a unifying metamodel that covers both RDF and LPGs

- helps adoption → important for the graph industry
- gives users more choices (e.g., Gremlin over RDF)

We are working on this (codename “One Graph”)

This is not an easy undertaking, however…

aws

Some challenges in unifying RDF and LPGs

RDF and RDF-star semantics

- triples are unique, but LPG edges are not
- (“old style” reification does not have this problem)

Lack of formal semantics for LPGs

- difficult to determine “correct” semantics
- also: diversity of scalar datatypes

Graph partitioning

- no named graphs in current LPGs
- in RDF vertices are just identifiers (no structure), but in LPGs vertices are structured objects

Update semantics

- deleting non-unique triples: what happens?
- cascading delete for edge properties?

aws

The way forward (or: what are the real pain points?)

How do I write software that leverages a knowledge graph?

- bridging the "**graph ⟷ code -gap**"
- low-level interfaces (lots of boilerplate code, etc.)

New interfaces and abstractions

I mapped my data to an ontology… now what? (or: what is "ontology support"…?)

- role of reasoning
- "hiding" reasoning works well
- transitive taxonomies & `owl:sameAs` (including inverse functional properties)

Libraries, "ontology engine(s)"…?

aws

The way forward (or: what are the real pain points?)

What about property graphs, Gremlin, …?

- choosing between RDF and property graph models is hard and leads to confusion
- RDF-star to the rescue?

“One Graph” unified metamodel

*Lassila et al: “**Graph? Yes! Which one? Help!**”, SCG2021 @ Semantics 2021*

(arXiv:2110.13348)

Is there a way out of the silos that does not involve never-ending bespoke integrations?

- modern tools do not offer a single unified view of all data

Graphs as a unifying logical view

aws

Questions?

Contact: ora@amazon.com

More info: https://aws.amazon.com/neptune/knowledge-graphs-on-aws/

https://arxiv.org/abs/2110.13348

aws