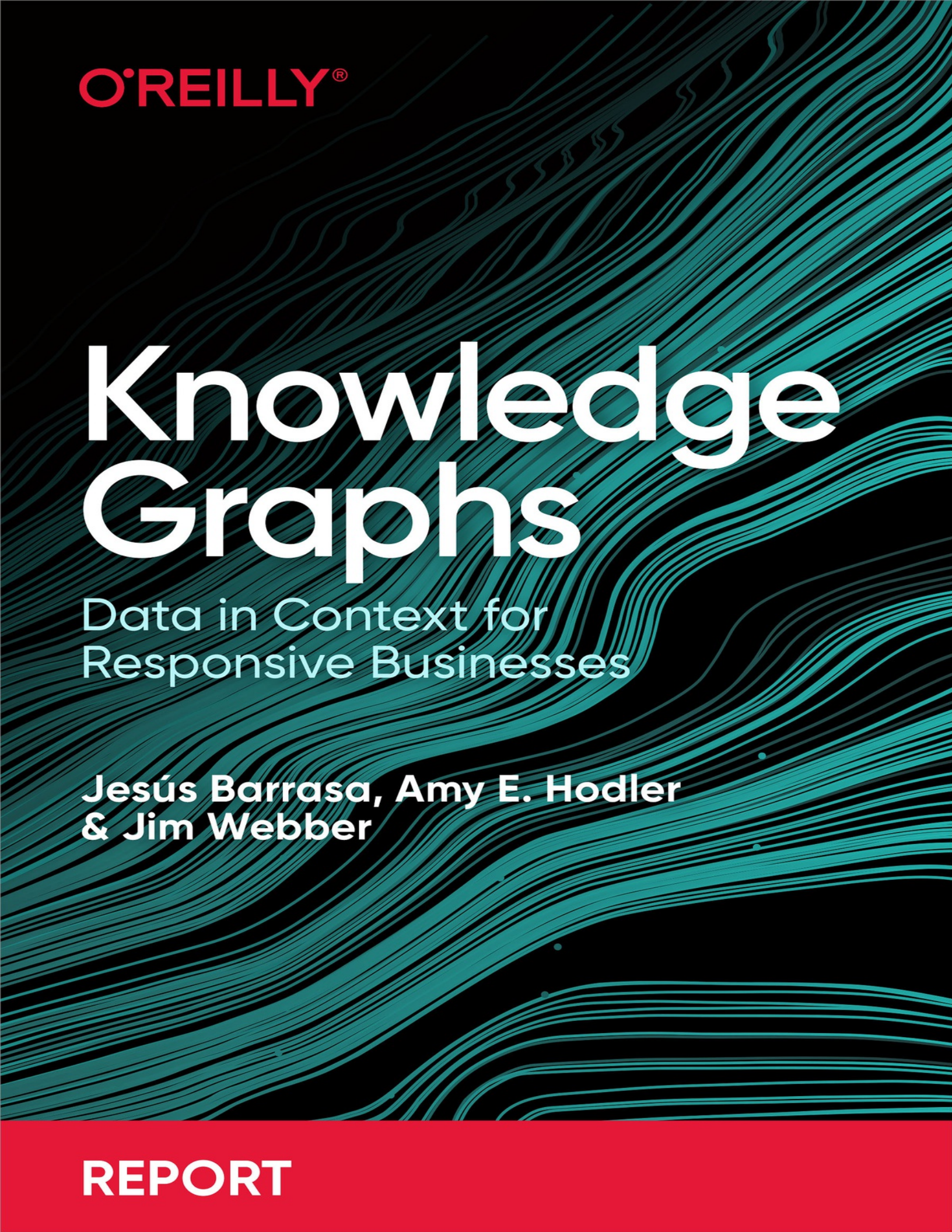
O'REILLY®
Knowledge Graphs
Data in Context for Responsive Businesses
Jesús Barrasa, Amy E. Hodler & Jim Webber
REPORT

Knowledge Graphs

Data in Context for Responsive Businesses

Jesús Barrasa, Amy E. Hodler, and Jim Webber

Knowledge Graphs

by Jesús Barrasa, Amy E. Hodler, and Jim Webber

Printed in the United States of America.

Published by O’Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O’Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (http://oreilly.com). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Rebecca Novack

Developmental Editor: Corbin Collins

Production Editor: Kristen Brown

Copyeditor: Shannon Turlington

Interior Designer: David Futato

Cover Designer: Randy Comer

Illustrator: Kate Dullea

July 2021: First Edition

Revision History for the First Edition

- 2021-07-26: First Release

978-1-098-10485-6

[LSI]

Foreword

The winners in the data industry know where the puck is going: making data smarter. This can be accomplished by integrating data with knowledge at scale, and this is where knowledge graphs come in. This book is a practical guide to understand what knowledge graphs are and why you should care. Importantly, it strikes the right balance between technical aspects and corresponding business value for an organization. If you need to make the business case for knowledge graphs, this is the book for you!

Let's first talk about the elephant in the room: RDF versus property graphs. Over the years, I've enjoyed my conversations with Jesús Barrasa on this topic. We have always been strong believers that these technologies will converge because at the end of the day, *it's all just a graph*! This book is evidence of this convergence: enriching the property graph model with taxonomies, ontologies, and semantics in order to create knowledge graphs. And don't forget that the conversation should focus on the business value and not just the technology.

How do you get started on your knowledge graph journey?

First, one of my mantras is *don't boil the ocean*. This means that your knowledge graph journey should start simple, be practical, and focus on the business return coming from the right amount of semantics. Second, I always say that you need to *crawl*, *walk*, and *run*. Crawling means to start by creating a knowledge graph of your metadata that catalogs the data within your organization. I'm thrilled to see that we are fully aligned: effective data integration solutions rely on *understanding the relationships* between data assets, which is at the heart of knowledge graphs. Furthermore, in the AI and ML era that we live in, understanding the quality and governance is key for effective decision-making.

Speaking of AI, knowledge graphs are changing AI by providing context. This leads to explainability, diversification, and improved processing. If AI is changing the future and knowledge graphs are changing AI, then by transitivity, knowledge graphs are also changing the future.

If you are still asking yourself "why knowledge graphs?," guess what...your

competitors aren’t! Don’t be that person! Jesús, Amy, and Jim have written this book just for you.

Juan Sequeda, PhD
Principal Scientist, data.world
July 2021

Chapter 1. Introduction

Graph data has become ubiquitous in the last decade. Graphs underpin everything from consumer-facing systems like navigation and social networks, to critical infrastructure like supply chains and policing. A consistent theme has emerged that applying knowledge in context is the single most powerful tool that most businesses have. Through research and experience, a set of patterns and practices called *knowledge graphs* has been developed to support extracting knowledge from data.

This report is for information technology professionals who are interested in managing and exploiting data for value. For the CIO or CDO, the report is brief yet thorough enough to provide an overview of the techniques and how they are delivered. For the data professional, data scientist, or software professional, this report provides an easy on-ramp to the world of knowledge graphs, and a language for discussing their implementation with peers and management.

Our fundamental tenet is that knowledge graphs are useful because they provide contextualized understanding of data. They achieve this by adding a layer of metadata that imposes rules for structure and interpretation. We'll illustrate how using knowledge graphs can help extract greater value from existing data, drive automation and process optimization, improve predictions, and enable an agile response to changing business environments.

This chapter explains the background and motivation behind knowledge graphs. To do so, we'll discuss graphs and graph data and show how we can build systems with smarter data using knowledge graph techniques.

What Are Graphs?

Knowledge graphs are a type of graph, so it's important to have a basic understanding of graphs before we go much further. Graphs are simple

structures where we use nodes (or vertices) connected by relationships (or edges) to create high-fidelity models of a domain. To avoid any confusion, the graphs we talk about in this book have nothing to do with visualizing data as histograms or plotting a function, which we consider to be *charts*, as shown in Figure 1-1.

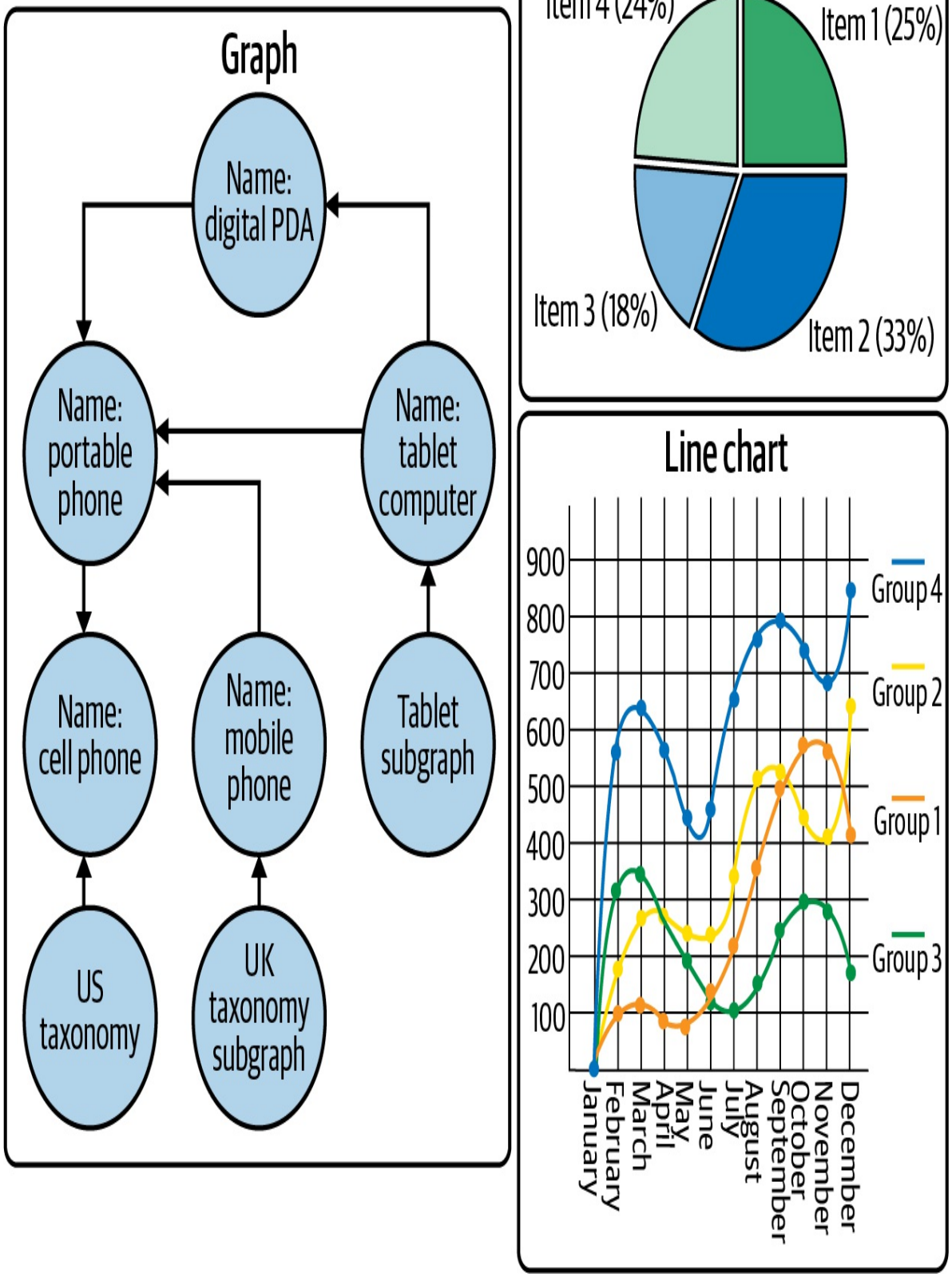
Graph
Name: digital PDA
Name: portable phone
Name: tablet computer
Name: cell phone
Name: mobile phone
Tablet subgraph
US taxonomy
UK taxonomy subgraph
Pie chart
Item 4 (24%)
Item 1 (25%)
Item 3 (18%)
Item 2 (33%)
Line chart
900
800
700
600
500
400
300
200
100
Group 4
Group 2
Group 1
Group 3
January
February
March
April
May
June
July
August
September
October
November
December

Figure 1-1. Graphs versus charts

The graphs we talk about in this book are sometimes referred to as *networks*. They are a simple but powerful way of describing how things connect.

Graphs are not new. In fact, *graph theory* was invented by the Swiss mathematician Leonhard Euler in the 18th century. It was created to help compute the minimum distance that the emperor of Prussia had to walk to see the town of Königsberg (modern-day Kaliningrad) by ensuring that each of its seven bridges was crossed only once, as shown in Figure 1-2.

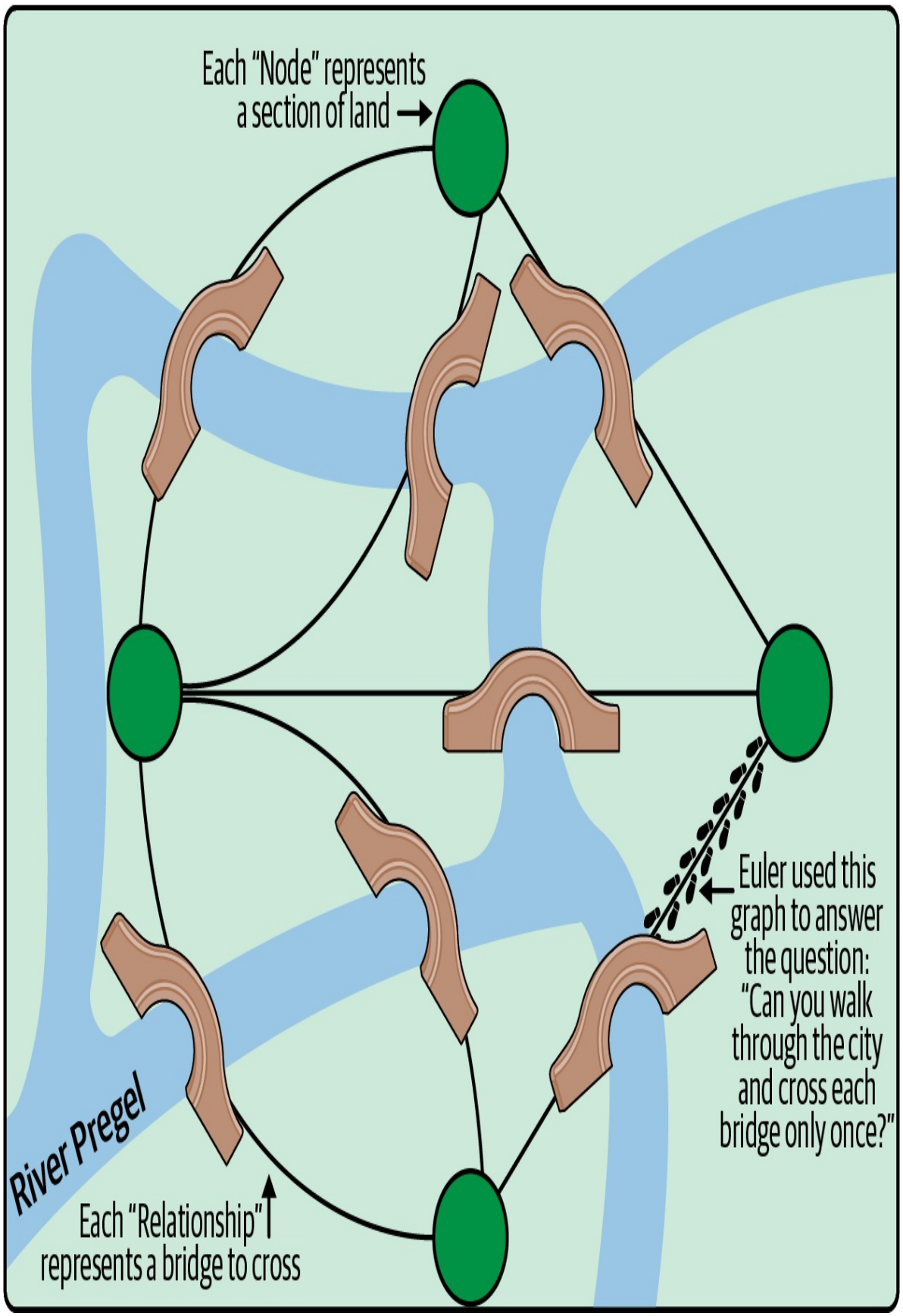
Each "Node" represents a section of land
Euler used this graph to answer the question: "Can you walk through the city and cross each bridge only once?"
River Pregel
Each "Relationship" represents a bridge to cross

Figure 1-2. A graphical representation of Königsberg and its seven bridges crossing the River Pregel

Euler's insight was that the problem shown in Figure 1-2 could be reduced to a logical form, stripping out all the noise of the real world and concentrating solely on how things are connected. He was able to demonstrate that the problem didn't need to involve bridges, islands, or emperors. He proved that in fact the physical geography of Königsberg was completely *irrelevant*.

Using the superimposed graph in Figure 1-2, you can try to figure out the shortest route for walking around Königsberg without having to put on your walking boots and try it for real. In fact, Euler proved that the emperor could *not* walk the whole town crossing each bridge only once, since there would have needed to be (at least) one island (node) with an even number of connecting bridges (relationships) from which the emperor could start his walk. No such island existed in Königsberg.

Building on Euler's work, mathematicians have studied various graph models, all variations on the theme of nodes connected by relationships. Some models allow relationships to be *directed*, where they have an explicit start and end node, while some have *undirected* relationships connecting nodes. Some models, like *hypergraphs*, allow relationships to connect more than one node.

Some graph models like the *property graph* model allow both nodes and relationships to contain *properties*. A property consists of a name (also called a key) and a value. Properties on a node can be used, for example, to give a name to a node representing a person or coordinates to a node representing a vehicle. Properties on relationships can be used to store distances between road junctions or the number of times an algorithm has processed a relationship.

THE PROPERTY GRAPH MODEL

The property graph model is the most popular model for modern graph databases, and by implication, a popular method for creating knowledge grah. It consists of the following:

Nodes representing entities in the domain

- Nodes can contain zero or more *properties*, which are key-value pairs representing entity data such as price or date of birth.
- Nodes can have zero or more *labels*, which declare the node's purpose in the graph, such as representing customers or products.

Relationships representing how entities interrelate

- Relationships have a *type*, such as bought or liked.
- Relationships have a *direction*, going *from* one node *to* another (or back to the same node).
- Relationships can contain zero or more properties, which are key-value pairs representing some characteristic of the link such as a timestamp or distance.
- Relationships never dangle—there is always a start and end node (which can be the same node).

These primitives—nodes, relationships, and properties—and rules can be used to assemble sophisticated, high-fidelity graph data models with relative ease.

Each of the graph models has its own quirks and benefits. In contemporary IT systems, enterprises have mostly settled on the property graph model. It's a model that is well suited to common data management problems and straightforward for software and data professionals to work with. To illustrate the property graph model, we've created a tiny social graph in Figure 1-3, but compared to the example in Figure 1-2, this graph is far richer.

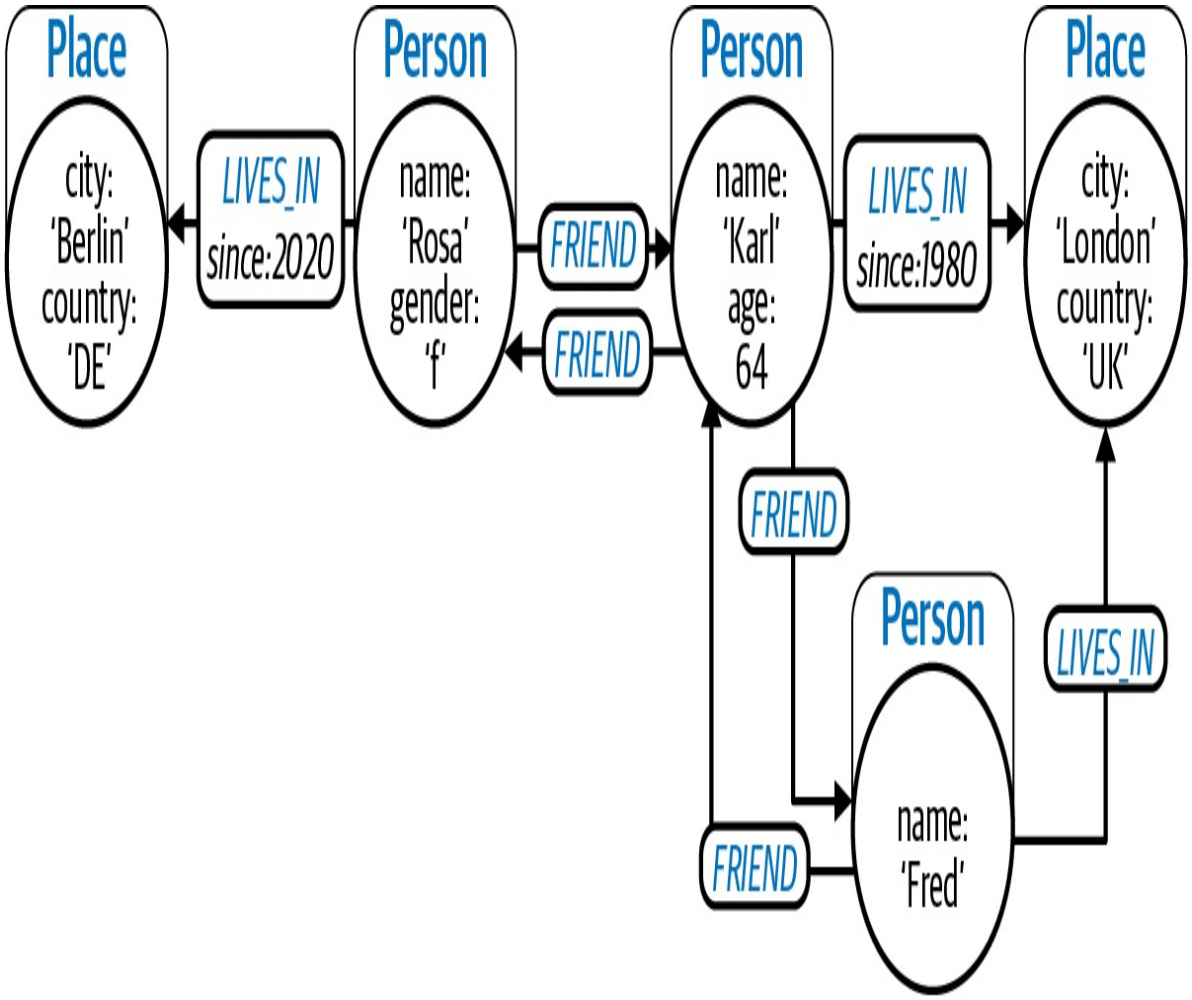

Figure 1-3. A graph representing people, their friendships, and their locations

In Figure 1-3 each node has a label that represents its role in the graph. Some nodes are labeled `Person` and some labeled `Place`, representing people and places respectively. Stored inside those nodes are properties. For example, one node has `name:'Rosa'` and `gender:'f'` that we can interpret as being a female person called Rosa. Note that the `Karl` and `Fred` nodes have slightly different properties on them, which is perfectly fine too. If we need to ensure that all `Person` nodes have the same property keys, we can apply *constraints* to the label to ensure those properties exist, are unique, and so on.

Between the nodes in Figure 1-3 we have relationships. The relationships are richer than in Figure 1-2, since they have a type, a direction, and can have optional properties on them. The `Person` node with the `name:'Rosa'`

property has an outgoing `LIVES_IN` relationship with property `since: 2020` to the `Place` node with `city:'Berlin'` property. We read this in slightly poor English as "Rosa lives in Berlin since 2020" and definitely not that Berlin lives in Rosa! We also see that `Fred` is a `FRIEND` of `Karl` and that `Karl` is a `FRIEND` of `FRED`. `Rosa` and `Karl` are also friends, but `Rosa` and `Fred` are not. Relationships in property graphs are not symmetric.[1] In most domains, relationships apply in one direction such that people own cars and cars do not own people.

In the property graph model, there are no limits on the number of nodes or the relationships that connect them. Some nodes are densely and intricately connected while others are sparsely connected, to match the problem domain. Similarly, some nodes have lots of properties, while some have few or none at all. Some relationships have lots of properties, but many tend to have none.

GRAPH DATABASES

Finding connections between data points is a natural and powerful way of making information discoveries. Graphs and graph theory are amazing tools in their own right for modeling and analyzing data. *Graph Databases, 2nd Edition* (O'Reilly, 2015) by Ian Robinson, Jim Webber, and Emil Eifrem can help you understand how to use a graph database to power your systems.

It's easy to see how the graph in Figure 1-3 can answer questions about friendships and who lives where. Extending the model to include other important data items like interests, publications, or jobs is also straightforward. Just keep adding nodes and relationships to match your problem domain. Creating large, complex graphs with many millions or billions of connections is not a problem for modern graph databases and graph-processing software, so building even very large knowledge graphs is possible.

Graph data models are uniquely able to represent complex, indirect relationships in a way that is both human readable, *and* machine friendly.

Data structures like graphs might seem computerish and off-putting, but in reality they are created from very simple primitives and patterns. The combination of a humane data model and ease of algorithmic processing to discover otherwise hidden patterns and characteristics is what has made graphs so popular. It's a combination we will exploit in our knowledge graphs.

Now that we're comfortable with graphs, we move forward to interpreting connected data as *knowledge*.

The Motivation for Knowledge Graphs

There has been a recent explosion of interest in knowledge graphs, with a myriad of research papers, solutions, analyst reports, groups, and conferences. Knowledge graphs have become so popular partly because graph technology has accelerated in recent years but also because there is strong demand to make sense of data.

External factors have undoubtedly accelerated knowledge graphs to greater prominence. Stresses from the COVID-19 pandemic have strained some organizations to the point of breaking. Decision making has needed to be rapid, but businesses have been hampered by the lack of timely and accurate insight.

Businesses are reconfiguring their operations and processes to be ready to flex rapidly. As historical knowledge ages faster and is invalidated by market dynamics, many organizations need new ways of capturing, analyzing, and learning from data. We need to fuel rapid insights and recommendations across the business, from customer experience and patient outcomes to product innovation, fraud detection, and automation: we need contextualized data to generate knowledge.

Knowledge Graphs: A Definition

We now have an understanding of graphs and the motivation for using

knowledge graphs. But clearly not all graphs are knowledge graphs. *Knowledge graphs* are a specific type of graph with an emphasis on contextual understanding. Knowledge graphs are interlinked sets of facts that describe real-world entities, events, or things and their interrelations in a human- and machine-understandable format.

Knowledge graphs use an *organizing principle* so that a user (or a computer system) can *reason* about the underlying data. The organizing principle gives us an additional layer of organizing data (metadata) that adds connected context to support reasoning and knowledge discovery. The organizing principle makes the data itself smarter, rather than locking away the tools to understand data inside application code. In turn this both simplifies systems and encourages broad reuse.

WHERE'S THE DATA?

A knowledge graph can be a self-contained unit living in a single graph data store, or it can involve several coordinated graph stores forming a federation of graphs. A knowledge graph can also be a logical layer providing structure and insight over multiple data sources of different kinds (graph or nongraph) so that data consumers get a contextualized and integrated view of the data. These are all different ways in which knowledge graphs can materialize, and we will discuss even more in the following chapters.

Knowledge graphs are agnostic on the physical storage of the underlying data and support different types of architectural approaches, from the more virtualized ones where the knowledge graph is a smart index over externally stored data to the fully materialized ones where the external data is fully replicated in a graph platform—and any hybrid approach in between the two.

Organizing principles, reasoning, and knowledge discovery might seem intimidating and complicated at first. But in reality, we can think of knowledge graphs as an index over data that provides curation like a skilled

librarian recommending pertinent books and journals to a researcher. An organizing principle acts as a contract between the provider and user of a knowledge graph, and Chapter 2 explores the options for organizing principles that we might use.

---

1 Human relationships such as *love* are often symmetric, but we express that symmetry with two relationships.

Chapter 2. Building Knowledge Graphs

Graphs are common in modern computer systems. They're a pleasant and flexible data model for supporting interactive queries, real-time analytics, and data science. But what transforms a graph into a knowledge graph is the application of an *organizing principle* that helps human and software users to understand it. Sometimes this is loftily called *semantics*, but we just think of it as *making the data smarter*.

Knowledge graphs are the result of decades of research in semantic computing but with modern graph technology, we can modernize and generalize that research so that it can be applied to contemporary real-world problems. In this chapter, we introduce common organizing principles for knowledge graphs—how to add metadata to a graph to make it smarter. Once you've finished reading this chapter, you will be able to choose from different organizing principles that best suit your predicament.

Organizing Principles of a Knowledge Graph

We like the notion that knowledge graphs help make data smarter. Rather than having to repeatedly encode smart behavior into applications, we encode it once, directly into the data. Smarter data benefits knowledge reuse and reduces duplication and discrepancies.

There are several different approaches to organizing data in a graph, each with its own benefits and quirks. We're free to pick and choose the ones that are best suited to our problem, and we're free to compose them together too. Starting with a basic (but useful) graph, we'll show how to add successive layers of organization, demonstrating how knowledge graphs can be used to solve increasingly sophisticated problems.

Plain Old Graphs

We use the term *graph* when we're thinking about graphs in general, and in particular those graphs that haven't had organizing principles applied to them. But as graph enthusiasts, we know that regular graphs are very useful. They underpin some very important systems. The difference is that the interpretation of the information they contain is encoded into the systems that use the graph. In other words, the organizing principle is "hidden" in the logic of the queries and programs that consume the data in the graph.

As a familiar example, think about the sales data of a typical online store. Sales data is typically large and dynamic, and it combines customer shopping information with product catalog, including a product descriptions, categories, and manufacturers.

Figure 2-1 shows a small fragment of a sales and product catalog graph.

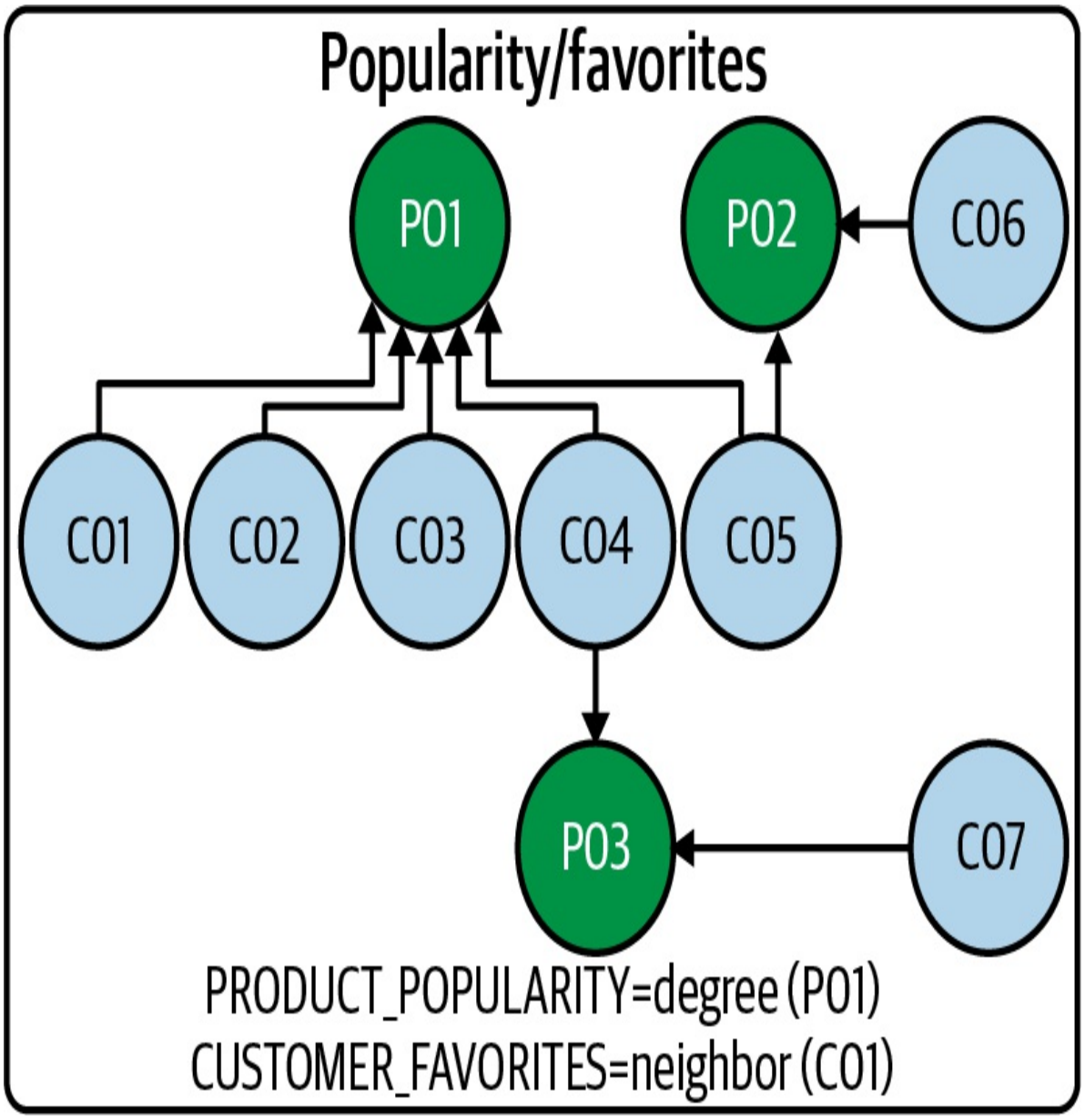

Figure 2-1. A snapshot of customers and their purchases as a plain old graph

You may not find it intuitive at first. But a program where an engineer has encoded the knowledge that the P nodes represent products and C nodes represent customers, and that connections between nodes represent purchases, will be able to answer straightforward questions like “What products did this customer buy?” and the reciprocal “Which customers bought this product?” which are valuable to the retailer.

Such a program would also be able to compute a product’s popularity by adding the number of incoming edges (calculating the degree of the node)

and see that one product is quite popular, the others less so.

A graph algorithm encoding the same interpretation of the information in the graph could even reveal customers' buying behavior. It could also segment them (roughly) into most buying a few popular products and the others spread among many less popular products.

There is no doubt that there is a lot of value in the graph, but what would happen if a data scientist with no previous knowledge of the domain wanted to try to run, for example, some basket analysis (what products are purchased together) to build a recommendation system? Well, someone would have to explain how to interpret the data in the graph because there is no organizing principle that can help them to make sense of it. The knowledge of how to interpret the data in the graph is encoded into the algorithms that perform the processing.

There's also a real problem if whoever created the graph has left the business. The data scientist now has to reverse engineer the code in the algorithms in order to interpret the graph. For many mature businesses, this scenario is a frequent occurrence.

Of course, once we understand the meaning of the data in the graph, we could continue to build new algorithms to process knowledge out of that graph, but a better solution is to make the data in the graph smarter by applying an organizing principle. The organizing principle surfaces the latent and implicit knowledge in a graph, turning it into a knowledge graph.

Richer Graph Models

As it happens, we have already seen a first organizing principle: the property graph model itself! Compared to the austere lines-and-circles graphs in the sales and product catalog graph in "Plain Old Graphs", the property graph model is richer and far more organized: it supports labeled nodes, type and direction for relationships, and properties (key-value pairs) on both nodes and relationships. Any software that understands the property graph model can process it in accordance with that simple organizing principle.

Figure 2-2 shows an enriched view of the sales and product catalog graph, including labels, properties, and named relationships.

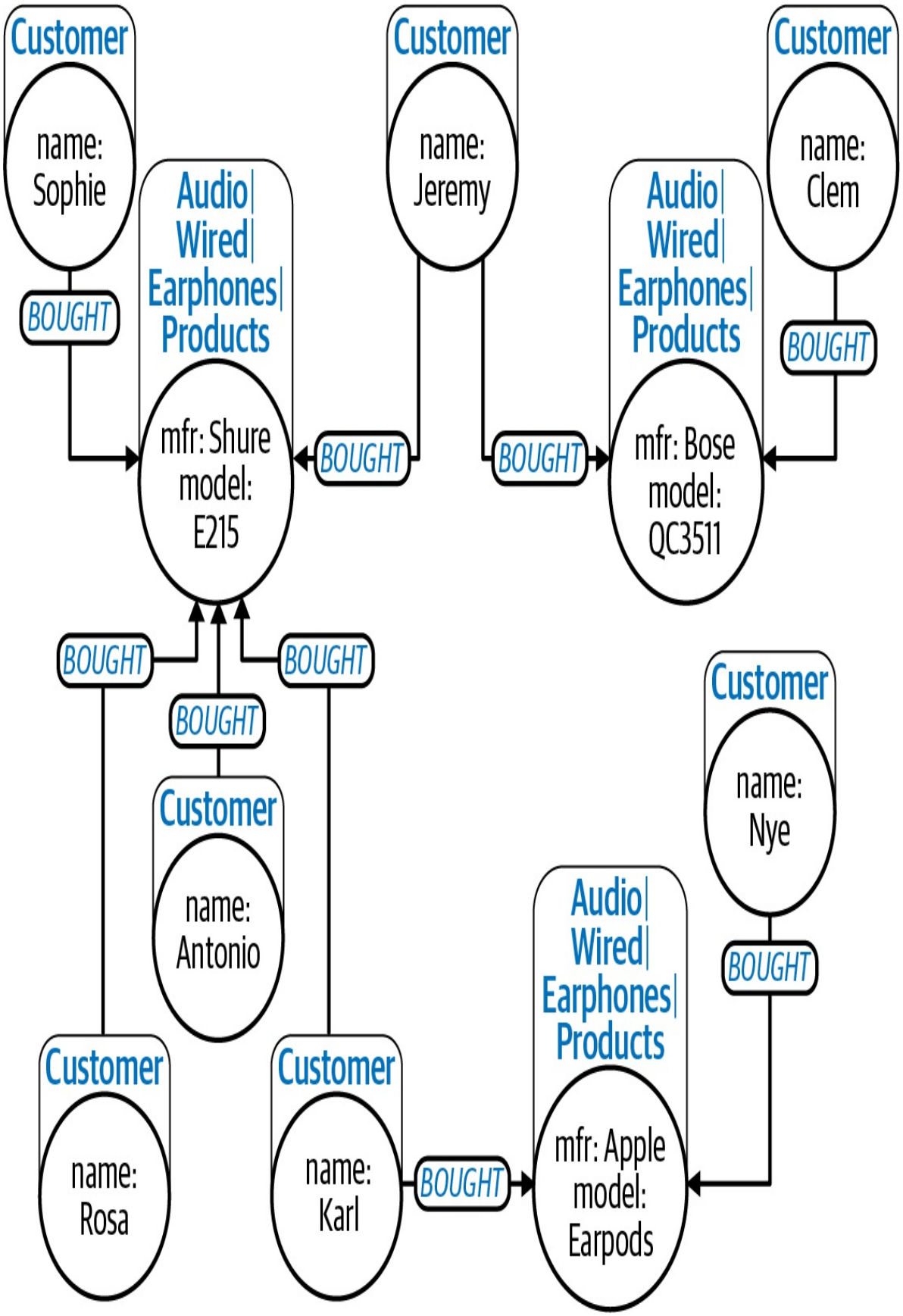
Customer
name: Sophie
BOUGHT
Audio| Wired| Earphones| Products
mfr: Shure model: E215
Customer
name: Jeremy
BOUGHT
BOUGHT
Audio| Wired| Earphones| Products
mfr: Bose model: QC3511
Customer
name: Clem
BOUGHT
BOUGHT
BOUGHT
BOUGHT
Customer
name: Antonio
Customer
name: Rosa
Customer
name: Karl
BOUGHT
Audio| Wired| Earphones| Products
mfr: Apple model: Earpods
Customer
name: Nye
BOUGHT

Figure 2-2. A snapshot of customers and their purchases as a property graph

A property graph gives human and machine agents a set of essential clues about the information it contains: node labels, relationship types and direction, and the types and names of property data stored on them. It is even possible to get a formal description of the shape of the graph (its schema) via introspection. Effectively, this organizing principle makes a graph self-describing to a certain extent and is a clear first step toward making data smarter. Just by using node labels, software can extract all similar types of entities from a graph.

And what about our data scientist? They would definitely have a much better (and more productive) time working on this graph compared to more austere data models.

Importantly, some processing can be done without knowledge of the domain, just by leveraging the features of the property graph model (the organizing principle). A common example is visualization. Figure 2-3 shows how nodes with the same labels will be displayed with similar visual style in two popular visualization tools, Linkurious and Bloom. No domain knowledge is needed for these tools to render the data in a helpful, visual manner, just an understanding of the organizing principle. Moreover, Bloom uses the metadata inherent in the organizing principle to provide a curated, domain-affined, Google-query style exploration interface atop the graph, again with no understanding of the domain.

To a consumer application, and a visualization tool is an excellent example, the organizing principle is a contract. It sets out that the tool can expect to find labeled nodes, connected by directed and typed relationships with properties on both. As long as the graph and the tools both honor this contract, the software can work with any graph.

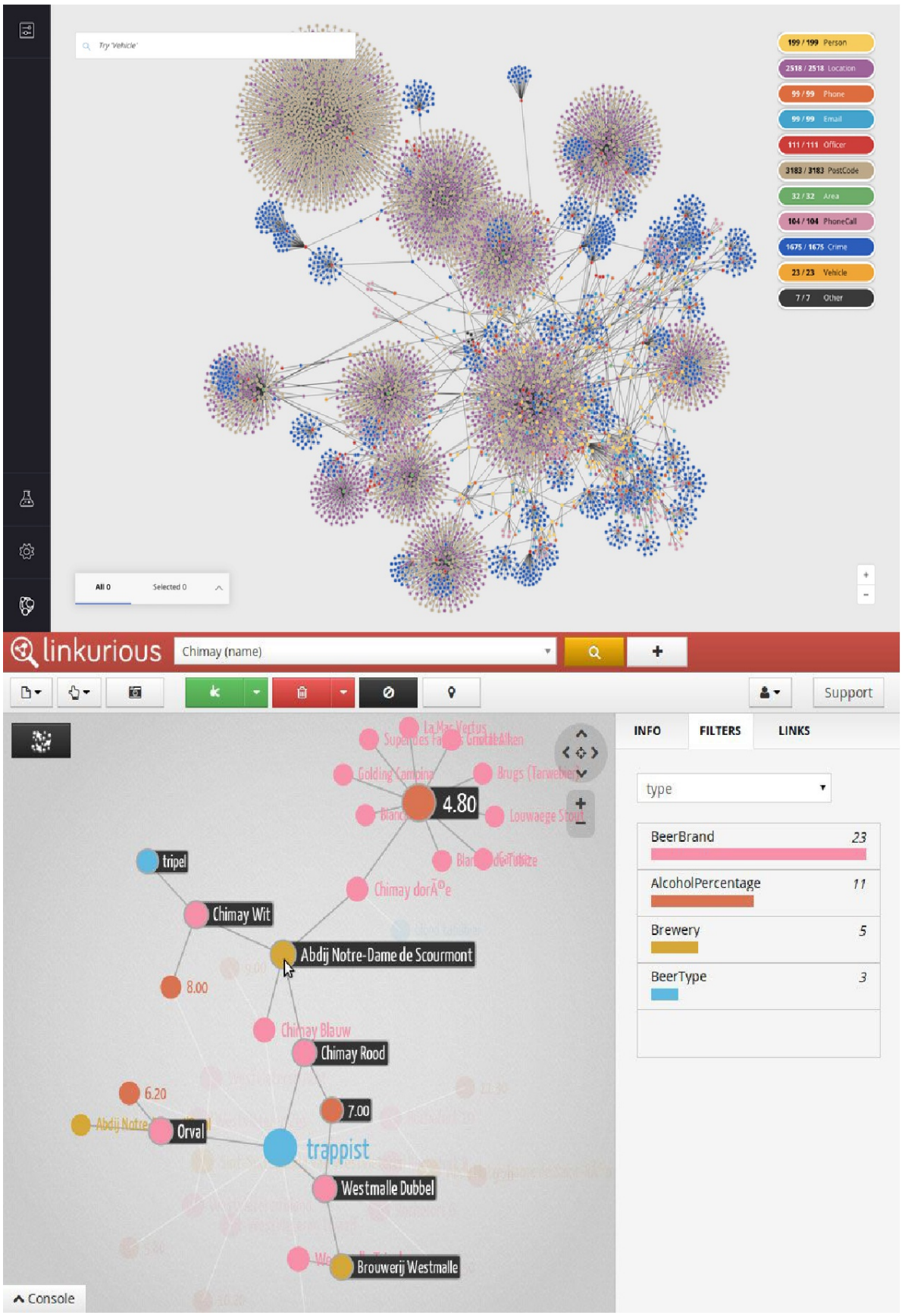
Try 'Vehicle'
199 / 199 Person
2518 / 2518 Location
99 / 99 Phone
99 / 99 Email
111 / 111 Officer
3183 / 3183 PostCode
32 / 32 Area
104 / 104 PhoneCall
1675 / 1675 Crime
23 / 23 Vehicle
7 / 7 Other
All 0
Selected 0
linkurious
Chimay (name)
Support
INFO
FILTERS
LINKS
type
BeerBrand 23
AlcoholPercentage 11
Brewery 5
BeerType 3
4.80
tripel
Chimay Wit
Abdij Notre-Dame de Scourmont
8.00
Chimay Blauw
Chimay Rood
6.20
7.00
Orval
trappist
Westmalle Dubbel
Brouwerij Westmalle
Console

Figure 2-3. Bloom (top) and Linkurious (bottom) visualize property graph data using the model's organizing principles

ORGANIZING PRINCIPLES ARE CONTRACTS

The organizing principle provides a contract between the graph and its users. But this doesn't only apply to human developers or users—it also applies to software agents.

With the property graph model, consuming software can expect to find labeled nodes and directed, labeled relationships in the data. The property graph model is a *contract*, and software tools that uphold this contract can process, visualize, transform, and transmit the data in a manner that is consistent with its intent.

Although it's commonplace and powerful, the property graph model is a relatively low-level organizing principle. The property graph model becomes even more useful if it's composed with other higher-order organizing principles like the taxonomies and ontologies we shall discuss later in the chapter.

In Figure 2-2 we've used the organizing principle from the property graph model to showcase various product features. A competent user could extract all *wireless headphones* from the data as easily as they could extract all *audio* products. With only slightly more effort, aggregate sales data per product or per product type could also be computed. But the way the data is set up does not provide much scope for reasoning: labels don't provide enough information to know that, for example, one product is substitutable for another. For that we need a stronger organizing principle.

Knowledge Graph Using Taxonomies for Hierarchy

Creating categories of nodes using labels is clearly useful. In Figure 2-2, we saw that the connection between customers and the products they've bought is explicit in the graph in the form of a relationship. The same applies to the labels that categorize products. But the associativity *between* labels is

missing.

Labels don’t tell us, for example, that one category is broader than another one, or that certain products are compatible with one another or even substitutable for one another based on the categories to which they belong. At a business level, that means lost sales opportunities. Better product catalogs mean better shopping experiences for the customer and more revenue for the retailer, so a richer way of categorizing products is a worthwhile investment.

As savvy buyers, we know that headphones and earphones are both part of the more general category of personal audio. We also know that there are specific styles of *wireless* and *wired* headphones and earphones, and we know that some personal audio can be both wired and wireless. In order to meet customer expectations, the retailer needs to be able to use this associative information. It might also be useful to mix product information with historic customer preferences, stock levels, profit margins, and so on in order to provide an even richer set of data to guide customers’ buying decisions.

In the example of a product catalog, those questions aren’t limited to buying specific headphones but also include higher-order questions like “What’s good equipment for me to listen to classical music on the move?” These kinds of questions cannot be answered with algorithms alone. We need smarter data.

A good place to start is to enrich the way products are classified with a higher-order organizing principle so that we can better reason about products’ substitutability. That is, if the retailer is out of stock of a specific item, it might still win a sale if it is able to offer similar items.

To support *x is a kind of y* reasoning, we need a more hierarchical view called a *taxonomy*. A taxonomy is a classification scheme that organizes categories in a broader-narrower hierarchy. Items that share similar qualities are grouped into the same category, and the taxonomy provides a global organization by relating categories to one another. This kind of hierarchical organization places more specific things like products (which are numerous) toward the bottom of the hierarchy while more general things like brands or

product families (which are less numerous) are placed toward the top of the hierarchy.

The hierarchy is constructed with *category* nodes connected by *subcategory_of* relationships.[1] Subsequently, products can be connected to the appropriate part of the taxonomy to classify them as ready for sale. This is shown in Figure 2-4, where you can see how customers can be better served by offering alternative products in the same category *and* in nearby categories.

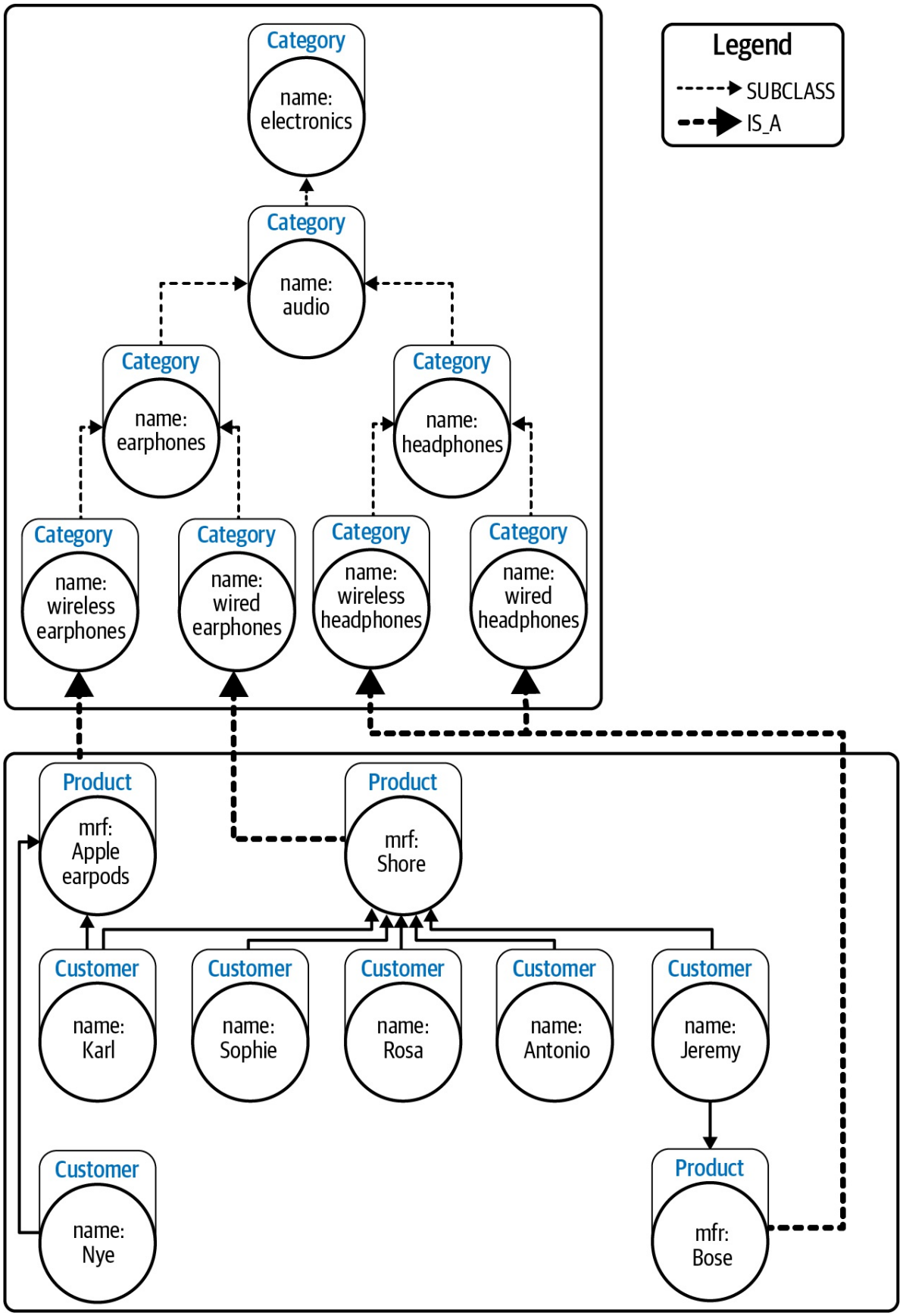
Legend
SUBCLASS
IS_A
Category
name: electronics
Category
name: audio
Category
name: earphones
Category
name: headphones
Category
name: wireless earphones
Category
name: wired earphones
Category
name: wireless headphones
Category
name: wired headphones
Product
mrf: Apple earpods
Product
mrf: Shore
Customer
name: Karl
Customer
name: Sophie
Customer
name: Rosa
Customer
name: Antonio
Customer
name: Jeremy
Customer
name: Nye
Product
mfr: Bose

Figure 2-4. A product catalog hierarchy layered on top of the customer and sales data depicted in Figure 2-2

But the beauty of knowledge graphs is that we can choose to use multiple hierarchical organizations simultaneously to provide even more insight. In Figure 2-5 we see how different taxonomies can coexist in a single knowledge graph. Currently, the Bose QC35II headphones are classified under `Half Price Products`, a subcategory of `Black Friday Deals`, which is a seasonal promotion. They are also connected to both the `wired` and `wireless` categories and would potentially be presented to customers browsing in those categories too. The same pair of headphones would at other times of the year be classified differently, for example in a high-end audio category. This is possible because classification is totally dynamic in a knowledge graph. The new categories and their associativity as well as the linkage of products to the categories are just additional nodes and relationships in the knowledge graph, which gives us extreme flexibility.

Just as in the previous section we achieved a first level of domain independence by using the property graph as organizing principle, we are taking it one step further when using a taxonomy. Any application aware of the semantics of a taxonomical organization (the meaning of broader-narrower) could exploit the graph in a much richer way without knowledge of the domain, for example by using the taxonomy to compute semantic similarity between products applying standard taxonomy metrics like *path similarity*, *Leacock-Chodorow*, or *Wu & Palmer*.

Using multiple categories makes the data more expressive and allows for more sophisticated exploitation of the data in the graph. Even with this relatively modest organization of data using taxonomical hierarchies, we get immediate benefits in terms of the knowledge that can be extracted. But this is not the end of our options for organizing knowledge; there are still higher-order organizing principles we can use.

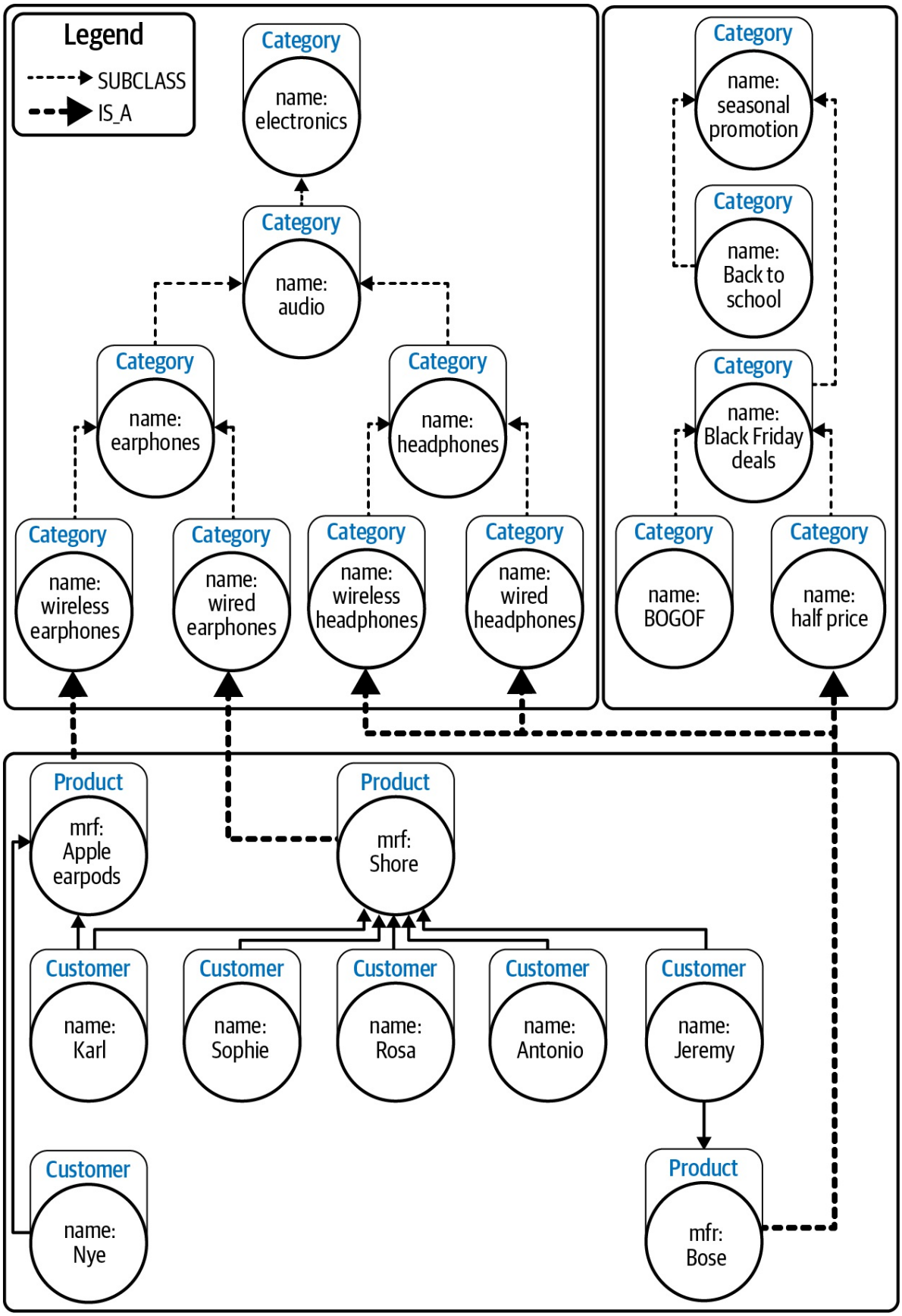
Legend
SUBCLASS
IS_A
Category
name: electronics
Category
name: audio
Category
name: earphones
Category
name: headphones
Category
name: wireless earphones
Category
name: wired earphones
Category
name: wireless headphones
Category
name: wired headphones
Category
name: seasonal promotion
Category
name: Back to school
Category
name: Black Friday deals
Category
name: BOGOF
Category
name: half price
Product
mrf: Apple earpods
Product
mrf: Shore
Customer
name: Karl
Customer
name: Sophie
Customer
name: Rosa
Customer
name: Antonio
Customer
name: Jeremy
Customer
name: Nye
Product
mfr: Bose

Figure 2-5. Multiple hierarchies can be dynamically layered on top of the customer and sales data; this diagram adds one more level to the contents of Figure 2-4

Knowledge Graph Using Ontologies for Multilevel Relationships

While taxonomies represent collections of topics with *subcategory_of* relationships between them, we might want to enrich our organizing principle with additional constructs using an *ontology*.[2] Ontologies are also classification schemes that describe the categories in a domain and the relationships between them. But ontologies are not restricted to just the hierarchical (broader-narrower) structures.

Ontologies allow for the definition of more complex types of relationships between categories, such as *part_of*, *compatible_with*, or *depends_on*. They also allow for the definition of hierarchies of relationships and for further characterization of relationships (transitive, symmetric, etc.).

Following the instructions in an ontology, we can explore the categories in a domain not just vertically (hierarchically) but also horizontally, where we can address cross-cutting concerns. For example, we could reason that an iPhone 12 is a valid search result for a customer looking for a mobile phone because it is an iOS device and the taxonomy states that iOS is a subcategory of mobile phone. From the semantics of the `UPSELL` relationship defined in the ontology, we can reason that an iPhone 12 Pro should be recommended to customers who own an iPhone 12. Figure 2-6 shows how an ontology supports precisely this upsell feature using `UPSELL` relationships to help guide a user of the product catalog toward better buying decisions.

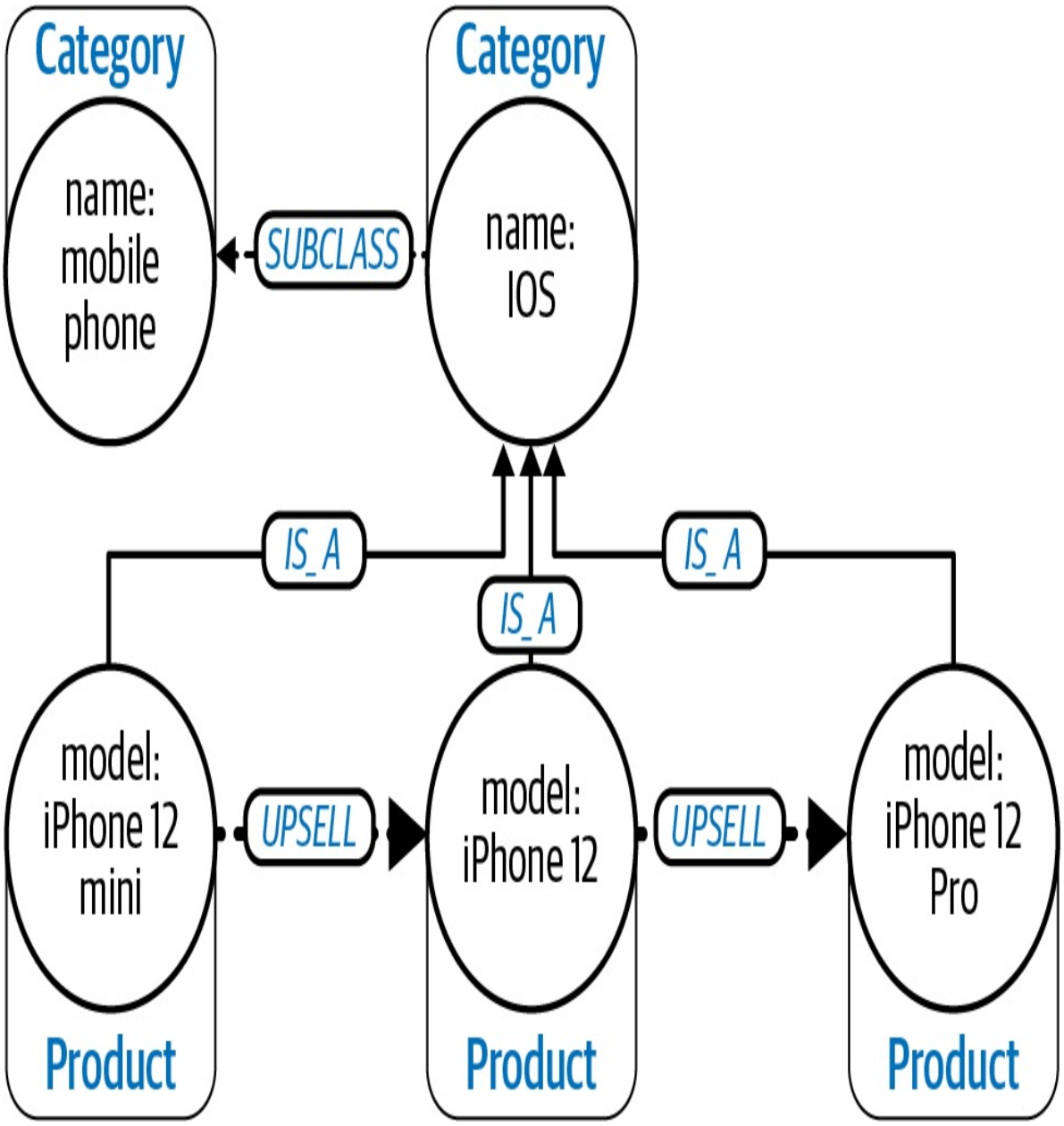

Figure 2-6. An ontology showing upgrade paths (upsell opportunities) for products in the catalog

In larger systems spanning departments, we might even use an ontology to act as a bridge between distinct departmental taxonomies to facilitate semantically aligned integration. By deploying an ontology that defines cross-taxonomy equivalence (links between the same concepts in distinct topologies), we can traverse the entirety of the business domain.[3]

Ontologies can be built in a modular fashion to make them composable. Figure 2-7 shows a sophisticated use of *layered* ontologies to bridge between

several underlying taxonomies/ontologies. Each layer in the graph can be queried independently, but when the layers are brought together, they provide an ability to reason *across* those domains.

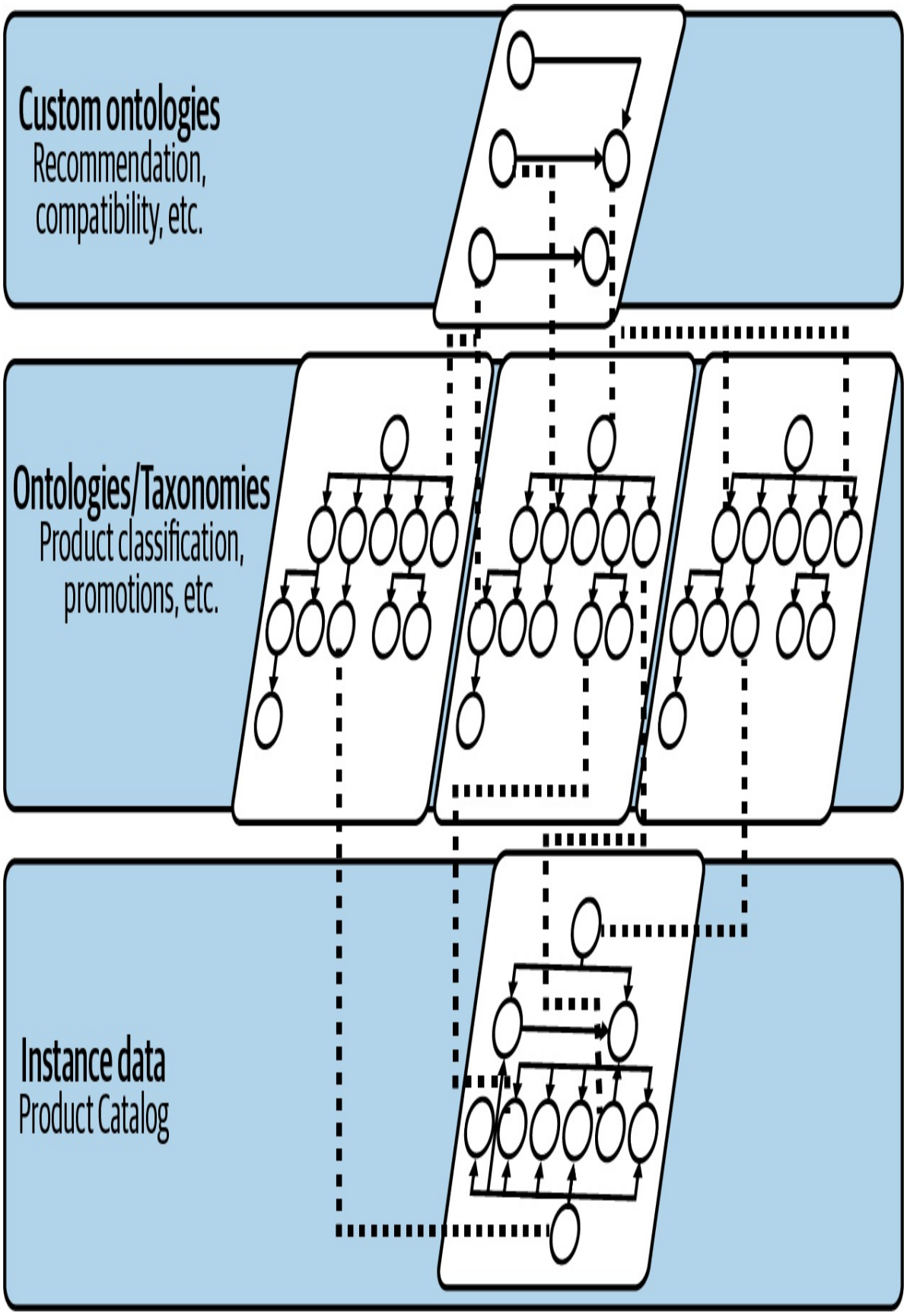
Custom ontologies
Recommendation, compatibility, etc.
Ontologies/Taxonomies
Product classification, promotions, etc.
Instance data
Product Catalog

Figure 2-7. A sophisticated layering of ontologies and taxonomies over the same data set

Where originally we were able to reason about electronics that complement one another through a single ontology, when we use a cross-domain overarching ontology, we are able to reason about more wide-ranging needs, including a bigger range of products, recommendations, special offers and promotions, and more. This is a distinct shift from "tell me what electronics complement each other" to "help me get the things I need (at the best price)."

Ontologies make knowledge actionable. They enable human or software agents to carry out sophisticated tasks. For example, if our e-commerce retailer ties its product hierarchy into stock control data through another ontological layer, its has a way of offering other good choices to the user when the current item is out of stock or to recommend products that have better margins. All of this comes at the modest cost of writing down how the business works as a machine-readable ontology.

Which Is the Best Organizing Principle for Your Knowledge Graph?

The type of organizing principle of a knowledge graph should always be driven by its intended use. There is little value in building rich and expressive features into the organizing principle if there is no associated process or agent (human or software) that makes use of them. It is a very common mistake to aim for an overly ambitious organizing principle up front. A prematurely detailed organizing principle will be an expensive effort in both time and resources at a point where the utility of the work is least well understood. It also risks being out of date by the time it's finished.

A good general principle for building organizing principles for knowledge graphs is *just enough semantics*. Use the simplest techniques you can find to address the needs you have today. Tomorrow you can always compose more advanced organizing principles into your system when future requirements drive out the need. Why build a complex ontology when a taxonomy or just a property graph is enough to drive the capabilities required by a use case?

Building only what you need plays well with iterative systems-delivery practices. Iterative construction of knowledge graphs helps to avoid the common trap of ontological perfectionism and keeps knowledge graphs fit for purpose for the long haul, delivers value early, and reduces overall risk.

Organizing Principles: Standards Versus Custom

There are several standard, or at least widely used, ontologies in existence that service a variety of domains, such as SNOMED CT for clinical documentation and reporting, the Library of Congress Classification (LCC) for library classification, Financial Industry Business Ontology (FIBO) for finance and business, and Schema.org and Dublin Core for general-purpose web resource annotation, to name just a few of the most popular ones. Whether for interoperability reasons or to reuse an existing public model, if you happen to work in a domain for which such a standard exists, it can be a good idea to consider adopting that model wholesale.

If your driver is interoperability, probably the consumers of your knowledge graph are aware of Schema.org, SNOMED, or FIBO, in which case speaking the same common language in your knowledge graph will obviously simplify communication. Interoperability can be crucial when the communicating parties are separate entities, such as independent participants in a supply chain. In some cases, it may even be mandatory to use a specific standard for things like regulatory reporting. From the perspective of knowledge reuse, you are treading a path that others have trodden before and can benefit from their efforts.

Where there is no standard for your domain, only a partial fit, or where you really do need fine-grained control over the organizing principle, then you will need to write your own from scratch or use a public standard as a starting point for your own organizing principle. When faced with the task of creating an organizing principle, you could take different approaches. One is to use natural language to describe the semantics of your organizing principle, as we did earlier in the chapter where we described property graphs using English

prose. This approach has a low barrier to entry but has the downside that it's not machine readable.

Another way to create an organizing principle would be to define it formally using one of the standard languages available. The most widely used ones are RDF Schema and Web Ontology Language (OWL) for ontologies and Simple Knowledge Organization System (SKOS) for taxonomical classification schemes. Each allows for different expressivity levels: from the basic definition of categories and relationships to taxonomies and more sophisticated constructs like complex classes through combination and operators for existence and universality.

One good thing about using a standard ontology language to describe your organizing principle is that you get good support from software in the creation process. Visual ontology editors help knowledge engineers in this task since writing ontologies by hand is difficult and error prone. Nonetheless, becoming fluent in one (or more) of these languages has a cost associated, and that cost needs to be weighed against the benefits.

It's not only during the creation of the organizing principle that you can benefit from standards-aware software, though. Once the organizing principle has been applied to your knowledge graph, software based on standards can run automated tasks on your data. For example, suppose you use OWL as the framework for the organizing principle of a knowledge graph. In that case, a program capable of running OWL-based reasoning would be able to derive new facts from your data *without it having to understand that data*. That's very appealing, of course, but it is essential always to keep an objective view. What are the kind of things that I can express with these standards? What is the nature of the inferences that can be run on them? And the most important question: how do they align with your business needs?

We observe that organizations that have been successful with knowledge graphs tend to leverage standards to a certain extent while ensuring they can easily layer on additional meaning when needed and move fast to keep up with business changes. We find that a mix of standards and adaptive customization is most aligned with the reality of modern business.

DATA EXCHANGE AND RDF

Resource Description Framework (RDF) is often presented as a key differentiator between different knowledge graph implementations or as a way to prescribe a particular technology stack. This is a mistake. RDF is a model for data exchange and does not mandate how data should be stored or manipulated or what architecture should be in place. RDF cares about how data is serialized and exchanged, not how it is stored and organized.[4]

The techniques described in this chapter are completely independent from implementation decisions. Choosing an organizing principle is an independent act from sharing data in a common format, as is the storage and querying of that data. You don't need an RDF triple store to build knowledge graphs.

Use RDF to share knowledge, not to manage data.

Essential Capabilities of a Knowledge Graph

Before we move on to real-world examples of knowledge graphs, we will briefly revisit the foundations. First, we don't believe that any specific graph technology is required for knowledge graphs, despite the historical association with so-called semantic triple stores. We prefer the property graph model since it's the most popular method today, and it's a performant way to manage knowledge graphs but is not mandatory. In fact, with a general graph model we can readily design new (or reuse existing) organizing principles.

A good knowledge graph stores and unifies underlying data so that it can be reasoned about. It does not necessarily seek to change the underlying data but instead provides guidance on how that data can be understood, no matter its source technology.

> **TIP**
>
> Good knowledge graphs are flexible and easy to maintain. Avoid technologies that bog you down in heavyweight processes because while you're wading through that process, the business will have moved on.

Finally, a good knowledge graph will be performant. This is not a throwaway point, far from it. Teams will work around pain points, and slow systems are painful. A fast, up-to-date knowledge graph empowers users. A slow, hard-to-change one is a hurdle for users.

People and systems *using* a knowledge graph *enhance* the knowledge graph. Do everything you can to encourage use, and your knowledge graph will flourish.

At this point we have a good definition of a knowledge graph and the organizing principles that enable us to better derive meaning from a graph. Next, we'll turn to the applications of knowledge graphs to illustrate how businesses generate value from knowledge extracted from their smarter data.

---

1 Depending on the framework/vocabulary used, these could also be called *narrower_than* or *subclass_of*.

2 Strictly speaking, taxonomies are also ontologies, but they only use the basic constructs of categories and hierarchical relationships.

3 Not all the data will necessarily be in a single database. Generally, we require some systems integration work so that the data can be accessed over the network from the myriad of databases where it is stored.

4 Interestingly enough, the vast majority of the public RDF data in the web is not hosted in triple stores but embedded in web pages in the form of JSON-LD snippets, further proving this point.

Chapter 3. Data Management for Actionable Knowledge

When the data team of the International Consortium of Investigative Journalists (ICIJ) received a data dump that came to be known as the Panama Papers, they probably thought they'd be looking for the digital equivalent of a needle in a haystack.[1]

With such a large and complex set of data, it may have seemed an insurmountable task for a small team of knowledge workers, but their decision to build a knowledge graph with the data fundamentally changed that situation. The knowledge graph the ICIJ built gave context and connections to the data. The complex, multiyear, multimedia data in the knowledge graph was linked in a way that investigators just needed to follow the connections to uncover often scandalous stories that we have been reading in the news since 2015.

In unraveling the Panama Papers, the ICIJ demonstrated that explicit connections in data are transformative for consumers, whether they are human or software agents. Amongst the in excess of 214,000 entities and 140 prominent individuals discovered in the leak, the files also showed how major banks played a significant role in helping to move wealth offshore.

But that was not the end of the story. The work of the ICIJ is ongoing, and so the data team at ICIJ kept improving the data pipeline for populating the knowledge graph. They delivered a freshly updated graph to the journalists every day, despite ongoing changes like new code releases, additional data sources, and new team members.

In many respects, the ICIJ's journey is not so different from most other data owners, chief data officers (CDOs), or chief information officers (CIOs). A CDO or CIO has to ensure that data with the right quality (accuracy, completeness, timeliness) is accessible to the right people and processes and

is used a way that complies with rules and regulations.

This chapter explains the foundations for *actioning knowledge graphs*, which are used to drive decisions or actions based on data. We discuss knowledge graphs that support holistic understanding and linking of multiple data sources across the enterprise and how we can incorporate provenance metadata into the mixture. Throughout, we provide several use cases and an end-to-end example to illustrate approaches a CDO or CIO could take.

Relationships and Metadata Make Knowledge Actionable

We use relationships to describe how entities interrelate. For example, it is possible for a graph to state that a customer subscribes to a service by linking the customer node and the service node through a `SUBSCRIBES_TO` relationship. But relationships can also be used to connect data with metadata, and this is a very powerful combination.

Figure 3-1 shows a two-layered graph. On the bottom layer, we see the data pertaining to customers and the services to which they subscribe. On the top layer, we have some metadata that describes the provenance of the customer and subscription data. This metadata shows how customer information is part of a dataset sourced from an external system, which is curated by a particular human data steward.

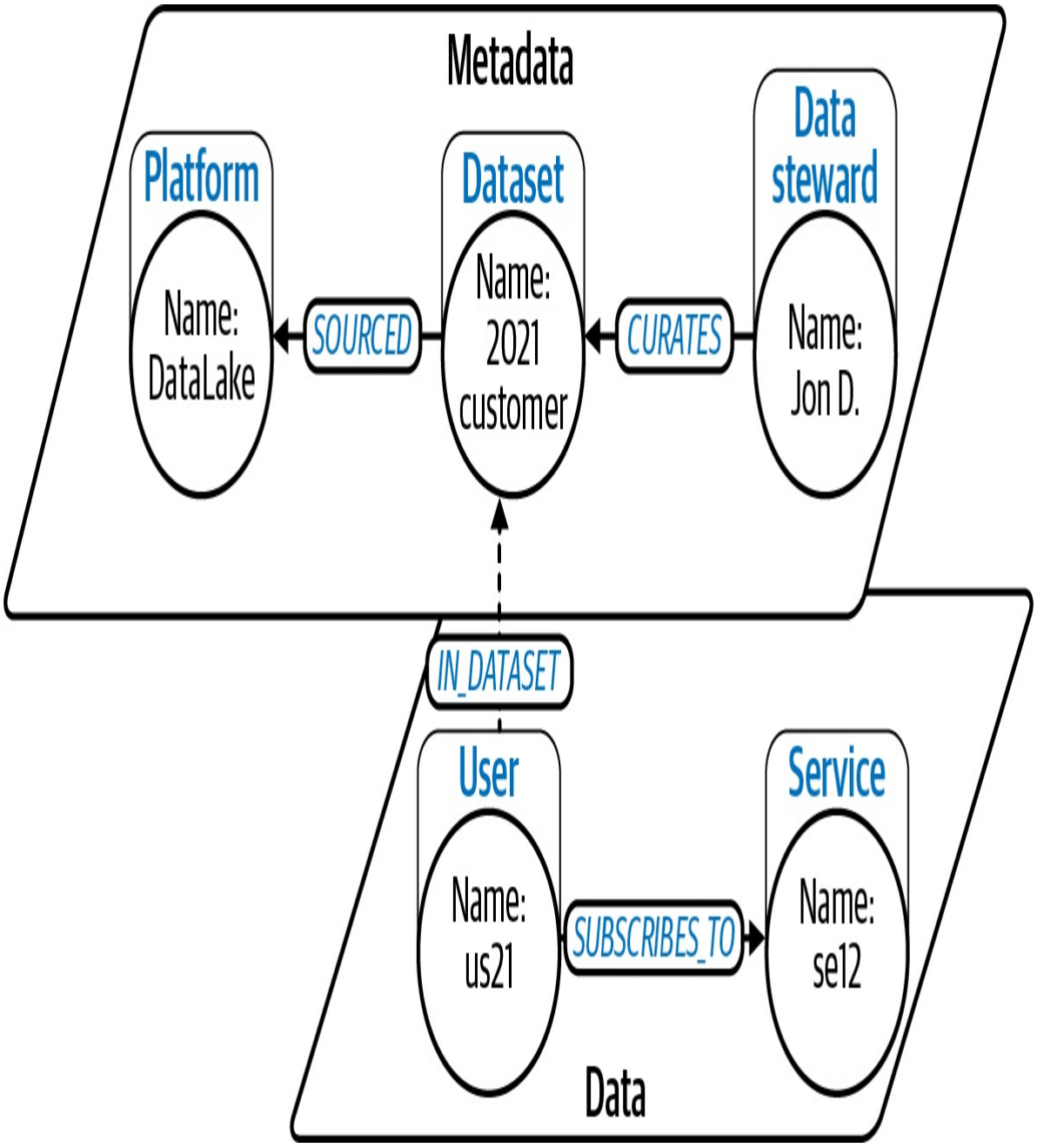

Figure 3-1. Relationships connect the data and metadata layers

Knowledge graphs like the one in Figure 3-1 can answer domain questions like “What customers subscribe to service X?” but can also provide confidence in the answer with provenance and governance metadata. Importantly, data architects can implement this technique in a noninvasive manner with respect to the source systems containing customer data, by building it as a layer above those systems. A popular example of this is to

build a knowledge graph of metadata that describes data residing separately in an otherwise murky data lake.

A globally linked view of data unlocks many significant use cases. Figure 3-1 shows the powerful combination of the data and metadata planes in the graph, but it's easy to see how we can construct a complete view of a customer by linking disparate partial views of customers across different systems. We could enrich the original customer view with the customer's behavior over their lifetime, detect patterns for good and bad customers, and adapt processes to suit their behavior. Similarly, on the metadata plane, we've started by showing how to offer a catalog of richly described datasets connected to the data sources where they are hosted, but we can also semantically enrich them by linking them to vocabularies, taxonomies, and ontologies to enable interoperability. Or we can add ownership/stewardship information to enhance governance and map where data came from as well as which systems and individuals have handled it.

The Actioning Knowledge Graph

An actioning knowledge graph provides a way of understanding good courses of action. Whether that's recommending a product with a risky or unreliable supply chain or optimizing routes for mail delivery, the actioning graph's responsibility is to guide someone to a better outcome.

To achieve this, an actioning knowledge graph requires integrated, contextualized, quality-assured, and well-governed data. Actioning knowledge graphs often integrate data from multiple information silos, providing a unified, deduplicated view of data. Such unified views are normally built around key (master) business entities; that's why they are often referred to as *Master Data Management (MDM)* graphs and power systems with names like Customer 360, Product 360, or Single View of Patient.

NOTE

Some organizations build graphs integrating all of their customers' touchpoints over time to provide a complete and cohesive view of the customers' journeys. Customer journey graphs are invaluable for detecting behavioral patterns that can help to predict churn or recommend the next best action in a highly personalized manner.

The Data Fabric Architecture

Actioning knowledge graphs can be used to provide integrated access to multiple data domains in what's called a *data fabric architecture*. Data fabrics are general-purpose, organization-wide data access interfaces that offer a connected view of the integrated domains by combining data stored in a local graph with data retrieved on demand from third-party systems. Their job is to provide a sophisticated index and integration points so that they can curate data across silos, offering consistent capabilities regardless of the underlying store (which might or might not be graph based), as shown in Figure 3-2.

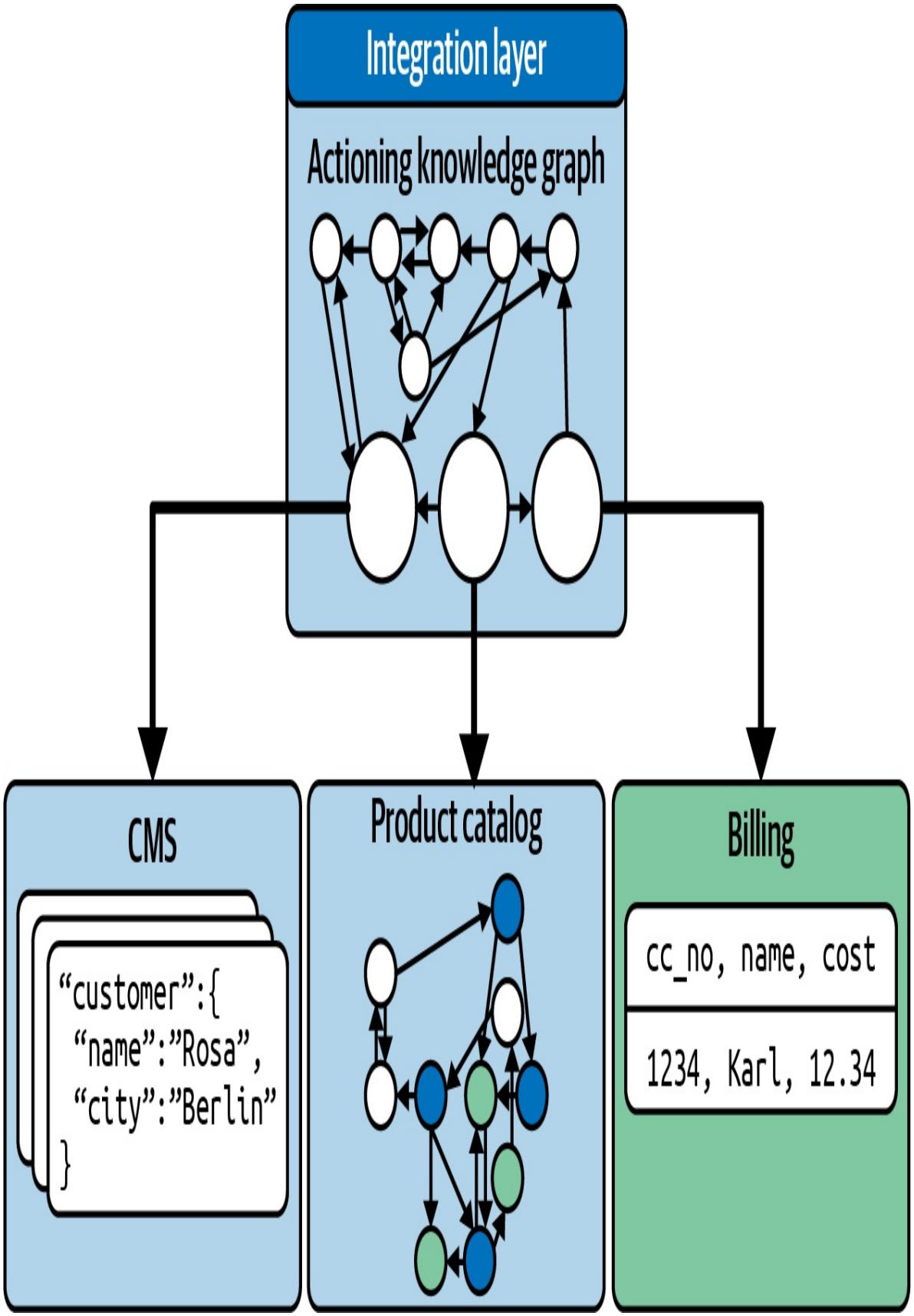
Integration layer
Actioning knowledge graph
CMS
"customer":{
"name":"Rosa",
"city":"Berlin"
}
Product catalog
Billing
cc_no, name, cost
1234, Karl, 12.34

Figure 3-2. An actioning knowledge graph supported by a data fabric

The example in Figure 3-2 is typical of an enterprise data fabric. The actioning knowledge graph provides an index and perhaps some metadata management across several data systems (which are themselves typically unaware of the data fabric). It acts as an entry point so that when a client application or user requests, "Give me the customer John Smith from Seattle," it can follow the relationships in the graph to leaf nodes, which redirect to records in other systems. In Figure 3-2, customer records are kept in a document database, the product catalog is stored in another graph database, and billing information is stored in a relational database. All of these are accessed transparently through an integration layer as part of the data fabric.[2] The actioning knowledge graph brings back data from the systems and combines them into a holistic set of data for the client, without the client having any knowledge that multiple backend systems were used.

Data architects often turn to graphs because they are flexible enough to accommodate multiple heterogeneous representations of the same entities as described by each of the source systems. With a graph, it is possible to associate underlying records incrementally as data is discovered. There is no need for big, up-front design, which serves only to hamper business agility. This is important because data fabric integration is not a one-off effort and a graph model remains flexible over the lifetime of the data domains.

Most data integration solutions unfortunately ignore relationships (with a notable exception being the Web). Using relationships to form an actioning knowledge graph has significant benefits since characteristics like node degrees, neighborhood information, and centrality metrics can be leveraged to build sophisticated and effective data integrations. For example, we can define a matching rule as follows: two nodes represent the same product (and therefore can be deduplicated) if their names have a strong string similarity (say over 95%) and they have identical clustering coefficients. The result is that data owners get significantly improved matching rates when aligning multiple data sources using graph metadata.

When a data fabric overlays an organizing principle, such as an ontology,

taxonomy, or enterprise canonical model atop the integrated data, it is presented to consumers as an actioning knowledge graph, making it possible to validate across systems and check for inconsistencies or violations of the semantics of the data. This improves the quality and correctness of the data and increases interoperability across the enterprise.

Metadata Management

Patterns of data usage have changed profoundly in the enterprise. In organizations with mature data management practices, expertise and control over data are shifting from a centralized setup to a distributed setup where business units provide domain expertise and systems architecture is often designed as a set of loosely coupled cooperating services.

As a consequence, your organization probably has many data users, and you have probably seen explosive growth in the number of internal data resources: data tables, dashboards, reports, metrics definitions, and so on. On the positive side, this shows your investment in data-informed decision making, but how do you make sure your users effectively navigate this sea of data assets of varying quality, freshness, and trustworthiness? The solution to keeping data fresh is a metadata hub, which has an actioning knowledge graph at its core.

> **NOTE**
>
> The trend for knowledge graph–backed metadata hubs started in 2017 when Airbnb announced its Dataportal platform at their "Democratizing Data at Airbnb" talk at GraphConnect London. This were followed by Lyft with Amundsen and LinkedIn with Datahub. Commercial solutions have followed and now also offer metadata management platforms enhanced with knowledge graphs.

The actioning knowledge graph for a metadata management hub is built from metadata collected from all systems across an estate. It typically includes information like datasets, any schemas, transformations, mappings, code, and so forth, which manipulate data. It also often includes information on

governance and usage of the data assets, such as access patterns and usage stats, data ownership and stewardship, data quality ratings from consumers, and so on. In some cases, it might contain operational information about the systems themselves, such as their health or utilization. In the graph, the data coming from the systems can be semantically enriched with fragments of "business knowledge" in the form of glossaries, ontologies, descriptions of business processes, compliance information, and more—just link the graphs together.

The graph in Figure 3-3 shows a simple example of how the actioning knowledge graph for a metadata hub would look. A *data pipeline* can be traversed to provide detailed provenance reports and answer impact analysis queries such as "Data source X is unavailable and all dependent pipelines are disrupted. Which data owners need to be notified?" In fact, the graph is a representation of a data pipeline that reads a report, standardizes the values of a given field—for example, an *industry sector* field—and then calculates aggregates by standardized industry sector and stores the result as a file for further distribution. The graph also contains data governance and provenance information showing which individuals are responsible for (or own) data assets and what groups these individuals belong to—both of which can be reasoned about for correctness and reported on for regulatory compliance.

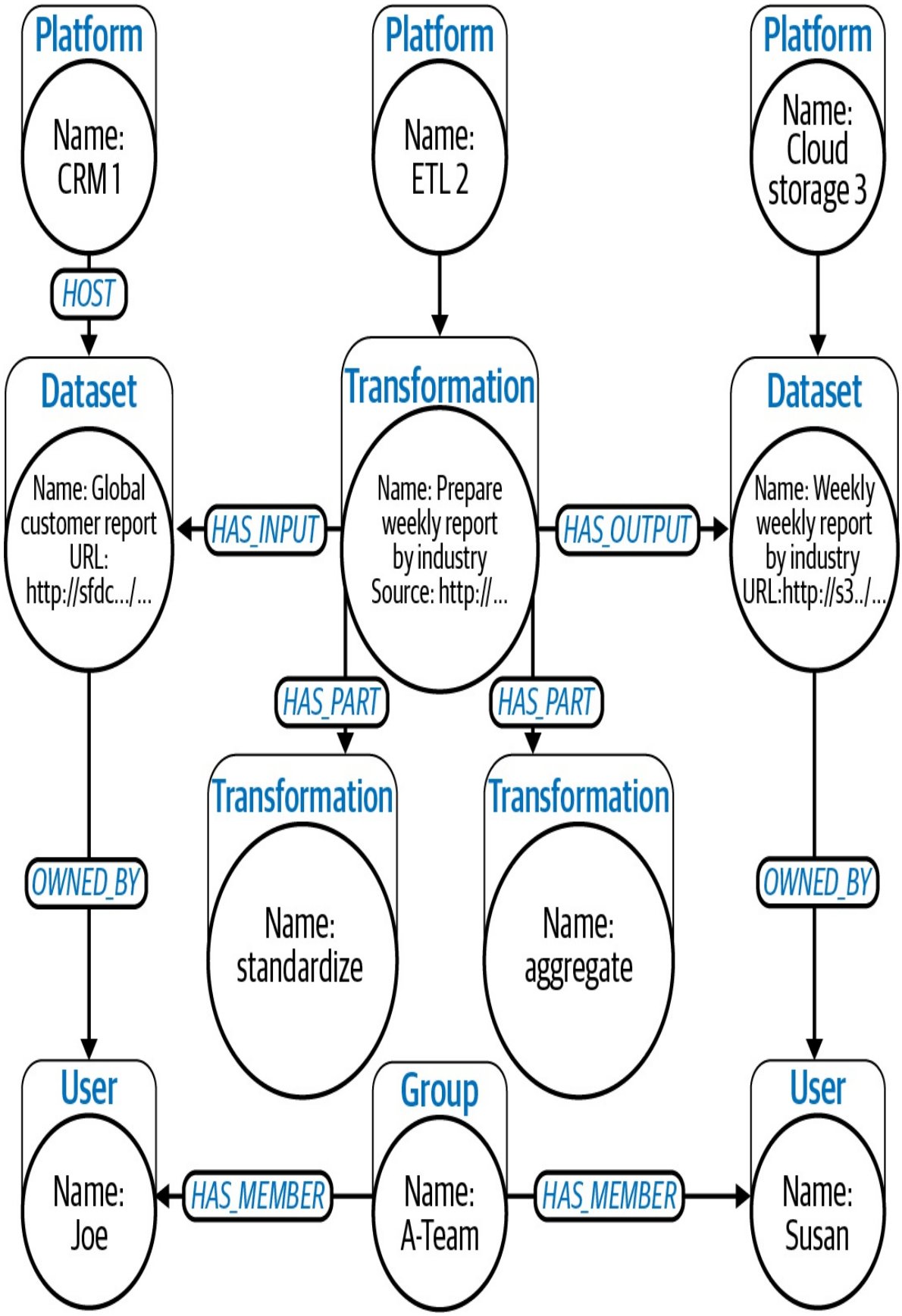
Platform
Name: CRM 1
Platform
Name: ETL 2
Platform
Name: Cloud storage 3
HOST
Dataset
Name: Global customer report URL: http://sfdc.../...
Transformation
Name: Prepare weekly report by industry Source: http://...
Dataset
Name: Weekly weekly report by industry URL:http://s3../...
HAS_INPUT
HAS_OUTPUT
HAS_PART
HAS_PART
Transformation
Name: standardize
Transformation
Name: aggregate
OWNED_BY
OWNED_BY
User
Name: Joe
Group
Name: A-Team
User
Name: Susan
HAS_MEMBER
HAS_MEMBER

Figure 3-3. A graph describing a simple data pipeline

Scaling up the example in Figure 3-3, we end up recording this information at a company-wide level. This brings together the connections of all data pipelines and processes and extends them with usage and data quality information to provide global visibility.

An actioning knowledge graph for metadata management builds trust in data and promotes self-service data consumption across the enterprise. It can also map data assets to concepts in the enterprise ontologies to make them discoverable and accessible and promote self-service data consumption. We believe it is the foundation for data management functions like data quality, data stewardship, and data governance.

TRUSTWORTHY AI NEEDS TRUSTED DATA

In an era of machine learning, where data is likely to be used to train AI, getting quality and governance under control is a business imperative. Failing to govern data surfaces problems late, often at the point closest to users (for example, by giving harmful guidance), and hinders explainability (garbage data in, machine-learned garbage out).

Deploying an actioning knowledge graph drives an increase in the productivity of data consumers (data analysts/data scientists) and brings high value insight into operations. In the words of data engineering at Airbnb: “A graph of the ecosystem has value far beyond tracking lineage and cross-functional information. Data is a proxy for the operations of a company. Analyzing the network helps to surface lines of communication and identify facets or disconnected information.”

Popular Use Cases for Actioning Knowledge Graphs

Here we summarize popular use cases for actioning knowledge graphs. This is by its nature incomplete, but the set of ideas and patterns is commonplace in contemporary adopters.

Data lineage

Traces all steps in data pipelines from data sources to data consumers to provide trust and high-fidelity provenance information.

Data catalog

Actionable inventory of all data assets with their detailed structure.

Impact analysis and root cause analysis

These are two similar *meta* use cases that materialize in many concrete examples like risk management, service assurance, ultimate beneficiary ownership (UBO), or fraud origination. Impact analysis is used to assess the direct or indirect effects of an event or a change in an ecosystem (the butterfly effect). Reciprocally, root cause analysis tries to map symptoms or consequences to originating causes. Both exploit the transitive dependencies modeled as relationships in the underlying graph.

Information search

Actioning knowledge graphs are also used to enhance information search. Documents, lessons learned, and knowledge in general can be indexed in a knowledge graph, making it possible to search for things instead of strings. In an actioning knowledge graph, documents are annotated according to one or many ontologies to enable semantic searches, document similarity computation, and generate recommendations that can anticipate the needs of an expert.

Single view of X

Also known as *X360* (X being the customer, the patient, the product, or any other key business entity), this provides a trusted and contextualized aggregation of all information relevant to X. Trusted views are usually built to drive contextual data-driven decisions around personalization, recommendation, and general next-best-action for activities related to X. The actioning knowledge graph provides a contextualized understanding of X and consequently provides the right data and context from which to

suggest actions.

THE UBS ACTIONING KNOWLEDGE GRAPH

UBS is a multinational investment bank and financial services company based in Switzerland. Founded in 1862, it maintains offices in more than 50 countries, employs more than 66,000 people, and reported total assets of more than $972 billion in 2019.

To comply with regulations, all banks need to provide transparency into the data flows that feed their risk reporting. This requires broad data governance and detailed data lineage. Using an actioning knowledge graph, UBS built its data lineage and data governance tool called the Group Data Dictionary (GDD). With the GDD, UBS can track information as it flows through the enterprise, monitor its quality, discover errors, and trace them to the source, minimizing damage and reducing data duplication.

UBS workflows and auditing capabilities are mostly built on relational technology, so synchronization of the knowledge graph with the underlying legacy systems is essential. GDD achieves this by constantly collecting metadata from the systems it tracks in order to compute a current view of them at any point. Some of the more modern systems added to GDD more recently are able to stream their updates into the graph by using a distributed publish-subscribe messaging system.

GDD’s actioning knowledge graph includes nodes representing the different data assets and their internal structures (attributes). It also includes relationships representing data flows connecting the data assets. The organizing principle uses transitive relationships generating dependency chains in the graph that they can traverse to analyze and expose lineage spanning dozens of levels of entities and dependencies.

Using an actioning knowledge graph, UBS is able to detect and resolve deep dependencies that would have been impossible to uncover in real time using traditional technologies.

Increased Trust and Radical Visibility

Few things are more damaging for organizations than mismanaging data, and few things are more potent than good data used well. As organizations grow, the complexity of their data ecosystems and the challenges to data management grow too. One of the most common challenges is that information and people tend to become siloed by tools or teams, leading to disconnected knowledge among staff. Using relationships in knowledge graphs is an excellent way to counter this entropic tendency inherent in a large or growing business.

Understanding the entire data ecosystem, from the production of a data point to its consumption in a dashboard or a visualization, provides the ability to invoke action, which is more valuable than the mere sum of its parts. In this chapter, we to have raised awareness of opportunities for your own business to map its data, data provenance, and data governance. We also hope that you'll be keen to take this further and drive actions from the comprehensive view of data assets. That being said, data discovery is a key element to this strategy, as we shall explain in Chapter 4.

---

1 According to the ICIJ, the Panama Papers was a "giant leak of more than 11.5 million financial and legal records exposes a system that enables crime, corruption and wrongdoing, hidden by secretive offshore companies."

2 For example, the Neo4j graph database provides user-defined procedures that can call out to other systems and combine the results from those systems with local data, which is all processed as if it were a local graph query.

Chapter 4. Data Processing for Driving Decisions

Graphs provide context to answer questions, improve predictions, and suggest best next actions. But uncovering insight from graph data is a necessary step toward unleashing value.

In the actioning knowledge graphs we saw in Chapter 3, an organizing principle was applied to an underlying graph in order to extract knowledge. We said this makes the data smarter. Deciding upon or discovering an organizing principle, or even just exploring the graph to find its general properties, is a useful activity in its own right.

In this chapter, we're going to explore *decisioning knowledge graphs*. A decisioning knowledge graph does not drive actions directly but surfaces trends in the data, which can be used in several ways such as to extract a view or subgraph for:

- Specific analyses (e.g., monopartite graphs like customer-bought-product) yielding actionable knowledge that can be written back into an actioning knowledge graph
- Human analysis (assisted by tooling) for data science exploration and experimentation, eventually possibly yielding insight that is written to the actioning knowledge graph or influences organizational structure
- Further processing by downstream systems (e.g., training machine-learning models)

Physically, our decisioning graph might or might not be the same graph as our actioning knowledge graph. Sometimes it's helpful to keep all the actionable data and decision making together (particularly when we want to enrich the actioning knowledge graph), and sometimes we want to physically

separate them (for data science workflows).

However we physically arrange our infrastructure, our toolkit for these jobs consists of discovery, analytics, and data science. Separately, discovery, analytics, and data science are helpful. But together they become extremely powerful tools in our toolbox for turning decisions into useful actions.

GRAPH ANALYTICS, MACHINE LEARNING, AND DATA SCIENCE

Processing graph data typically uses techniques from *graph analytics*, *graph machine learning*, and *graph data science*. These methods excel at finding unobvious connections because they can surface patterns in our data even when we don't exactly know what to look for.

Graph analytics comes from graph theory and entails processing data based on relationships that connect it. It's something we do to answer specific questions using existing data (e.g., for social network analysis) or to discover how the connections in that data might evolve (e.g., for supply chain optimization). Graph queries, visualization, and algorithms are tools we apply (often together), with the results used directly for analysis.

Graph machine learning uses graph data and graph analytics output to train machine learning (ML) models. Both graph data and graph metrics (degree, centrality, even topology) can be used to generate features to be learned by a ML model.

Graph data science is a multidisciplinary approach that incorporates graph analytics and graph machine learning as a holistic way of gaining insights from data.

This chapter covers the advantages of bringing discovery, graph data analytics, and graph data science into the mix. It explores the capabilities of decisioning knowledge graphs, to drive better actions and outcomes and highlights a number of enterprise-ready use cases.

Data Discovery and Exploration

The first step in any analysis is to find the data we need. For example, in criminal investigations, a suspect, individual, or organization is identified because connections in the data point to it in unusual ways. In a telecoms network, a device that is the root cause of a failure is identified because a surrounding constellation of working devices that depend directly or indirectly on that devices themselves report degradation in service. A group of fraudsters collaborating to create synthetic identities can often be identified because their shared means of identification forms rings in ways that would be otherwise highly unlikely.

Knowledge graphs provide the organizing principles to connect disparate datasets and a contextual platform for reasoning over linked information. Prime examples of this are POLE (Persons, Objects, Locations, and Events) databases often applied to governmental/law enforcement use cases or in IT systems management where failures can be predicted or retrospectively analyzed using a knowledge graph.

Leveraging the connections in data is transformative when sifting through large volumes of information. In Chapter 3 for example, we explained how the ICIJ makes sense of terabytes of leaked data. Similarly, NASA enables semantic searches over millions of documents to shave years and many millions of dollars off projects in its space program. Government agencies all over the world process countless phone records, financial transactions, fingerprints, DNA, and court records to fight crime and prevent terrorism. Financial institutions are able to use data discovery to improve fiscal responsibility and fight money laundering at scale.

The common thread among all these examples is that useful properties and patterns in the data first have to be discovered. Some intuition and thought have to go into the design of an organizing principle (such as a taxonomy), and from there the data can be explored to discover its useful properties. When useful patterns are discovered, they can be analyzed, used to train ML models, be written back to an actioning knowledge graph, or sent downstream to other systems.

The Predictive Power of Relationships

It's worth noting at this point that beyond helping with discovery and exploration, relationships are highly predictive of behavior. In fact, researchers have found that even without demographic information like age, location, and socioeconomic status, they can be highly accurate in predicting who will vote, smoke, or suffer obesity based on one thing: social relationships. It's not surprising that if we have many friends who vote, we're more likely to vote, or that if we're friends with smokers, we'd be more likely to smoke.

However, it is remarkable that a researcher can make this prediction even more accurately based on our friends-of-friends behavior, not one but *two* hops away from us. That is, the behavior of our friends-of-friends, whom we may not know that well or at all, is more predictive of our behavior than information that pertains only to us.

> **TIP**
>
> For more information on the science underlying social graphs, see *Connected* by James Fowler and Nicholas Christakis (Little, Brown and Company, 2009).

Despite their predictive power, most analytics and data science practices ignore relationships because it has been historically challenging to process them at scale. Consider trying to find similar customers or products in a three-hop radius of an account. With nongraph technology, you might be able to process this data, even if it is slower than querying a knowledge graph. But what if you need to scale such processing over a large graph of your customer base, then distill useful information (e.g., for every pair of accounts in this radius, calculate the number of accounts in common), and finally transform the results into a format required for machine processing? It's just not practical in a nongraph system. This explosion of complexity quickly overwhelms the ability to perform processing and hinders the use of "graphy" data for predictions to the ultimate detriment of decision makers.

Instead of ignoring relationships, knowledge graphs incorporate them into analytics and ML workflows. Graph analytics excels at finding the unobvious because it can process patterns even when we don't exactly know what to look for, while graph-based ML can predict how a graph might evolve. This is precisely what most data scientists are trying to achieve!

A performant knowledge graph makes it practical to incorporate connections and network structures into data analytics and from there to enrich ML models. For the business, this means better predictions and better decisions *using the data we already have*.

The Decisioning Knowledge Graph

We call a knowledge graph used for analytics, ML, or data science a *decisioning* knowledge graph because the aim is ultimately to improve decisions made by human or software agents. A decisioning knowledge graph must support analytics and data science workflows from simple queries to ML as well as provide graph visualizations.

Figure 4-1 illustrates the capabilities of a decisioning knowledge graph. These capabilities may be used alone or combined with one another, often in a pipeline.

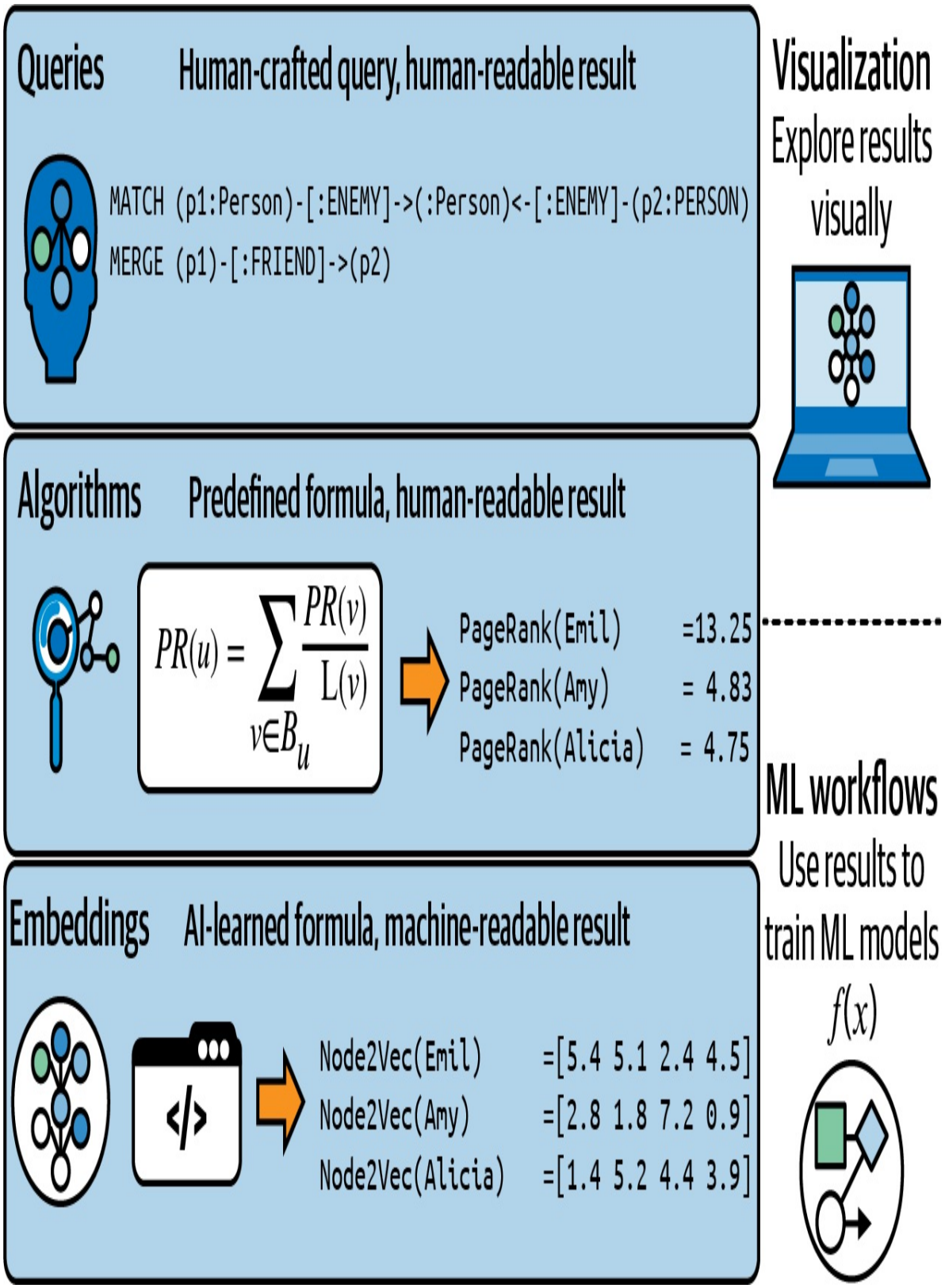

Figure 4-1. Unique capabilities combined in a decisioning knowledge graph

Queries

These are written by humans during an investigation and typically produce human-readable results.

Algorithms

While algorithms also produce human-readable results, they are coded in advance of any particular investigation and are based on well-understood principles from graph theory.

Embeddings

These are also defined in advance and use machine-learned formulas to produce machine-readable results.

Once we have results from queries, algorithms, and embeddings, we can put them to further use. As we saw in Chapter 2, graph-based visualization tools are helpful for exploring connections in graph data, but we can also use these outputs as training data for ML models.

Graph Queries

Most analysts start down the path of graph analytics with *graph queries*, which are (usually) human crafted and human readable. They're typically used for real-time pattern matching when we know the shape of the data we're interested in. For example, in Figure 4-2 we're looking for potential allies in a graph of enemies on the basis of the concept, "the enemy of my enemy is my friend." Once a potential ally has been located, we create a `FRIEND` relationship. Unlike data discovery, where we're asking a specific question for investigation, here we use the query results to feed subsequent analyses.

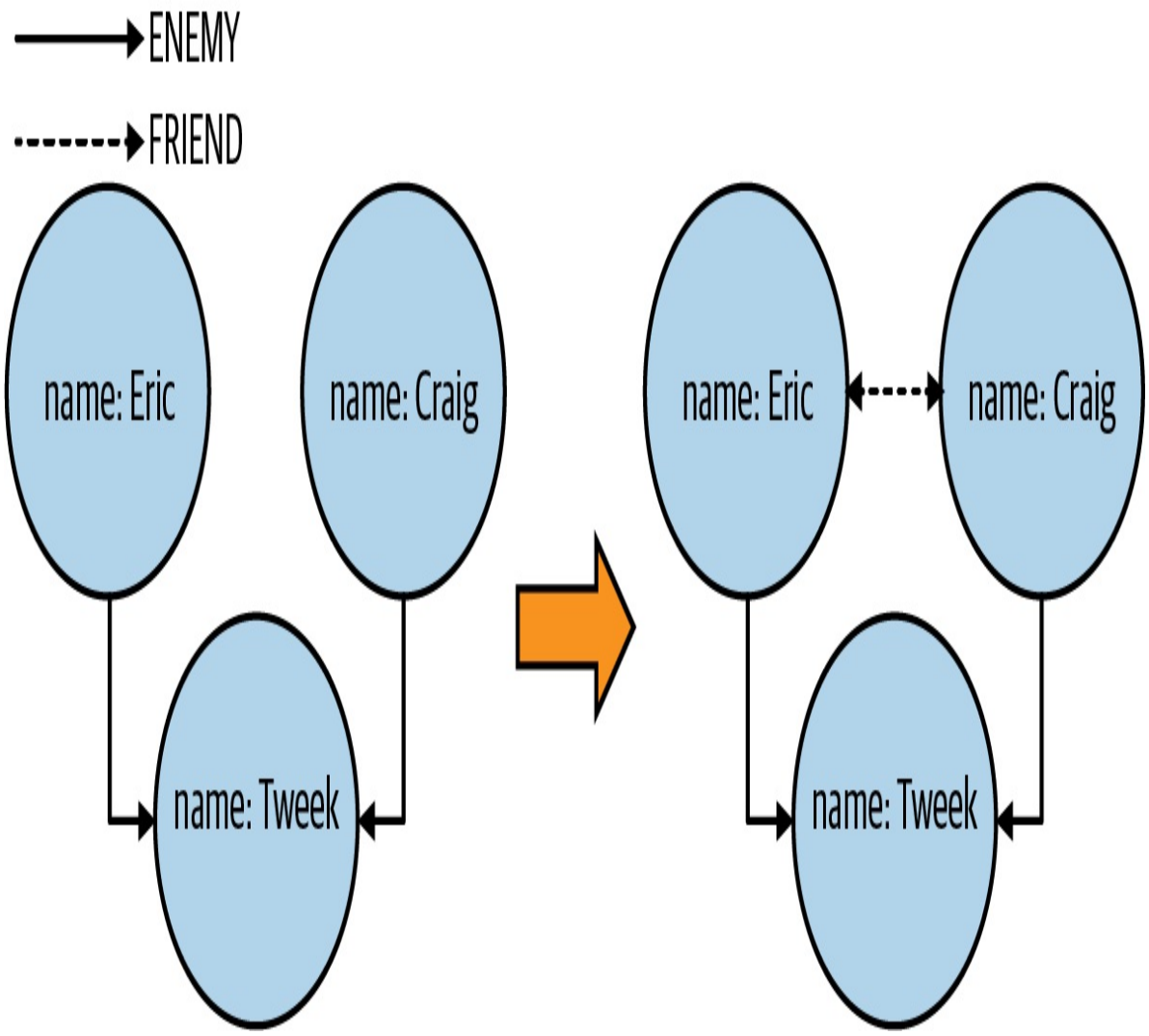

Figure 4-2. Making friends with the enemy of my enemy.

With a graph database and a graph query language, these kinds of *graph-local* patterns are computationally cheap and straightforward to express.

Graph Algorithms

But what if we don't know where to start the query or want to find patterns anywhere in the graph? We call these operations *graph-global*, and querying is not always the right way to tackle these challenges. A graph-global problem is often an indication that we should instead consider a graph *algorithm*.

For more comprehensive analysis, where we need to consider the entire graph (or substantial parts of it), graph algorithms provide an efficient and scalable way to produce results. We use them to achieve a particular goal, like looking for popular paths or influential nodes. For example, if we're looking for the most influential person in a social graph, we'd use the PageRank algorithm,[1] which measures the importance of a node in the graph relative to the others.

In Figure 4-3 we see a snapshot of part of a graph. Visually, we can see that the node representing `Rosa` is the most connected, but that's an imprecise definition.

If we run the PageRank algorithm over the data in Figure 4-3, we can see that `Rosa` has the highest PageRank *score*, indicating that she's more influential than other nodes in the data. We can use this metadata in knowledge graphs by incorporating it as part of the organizing principle, just like any other data item, to drive users toward good decisions.

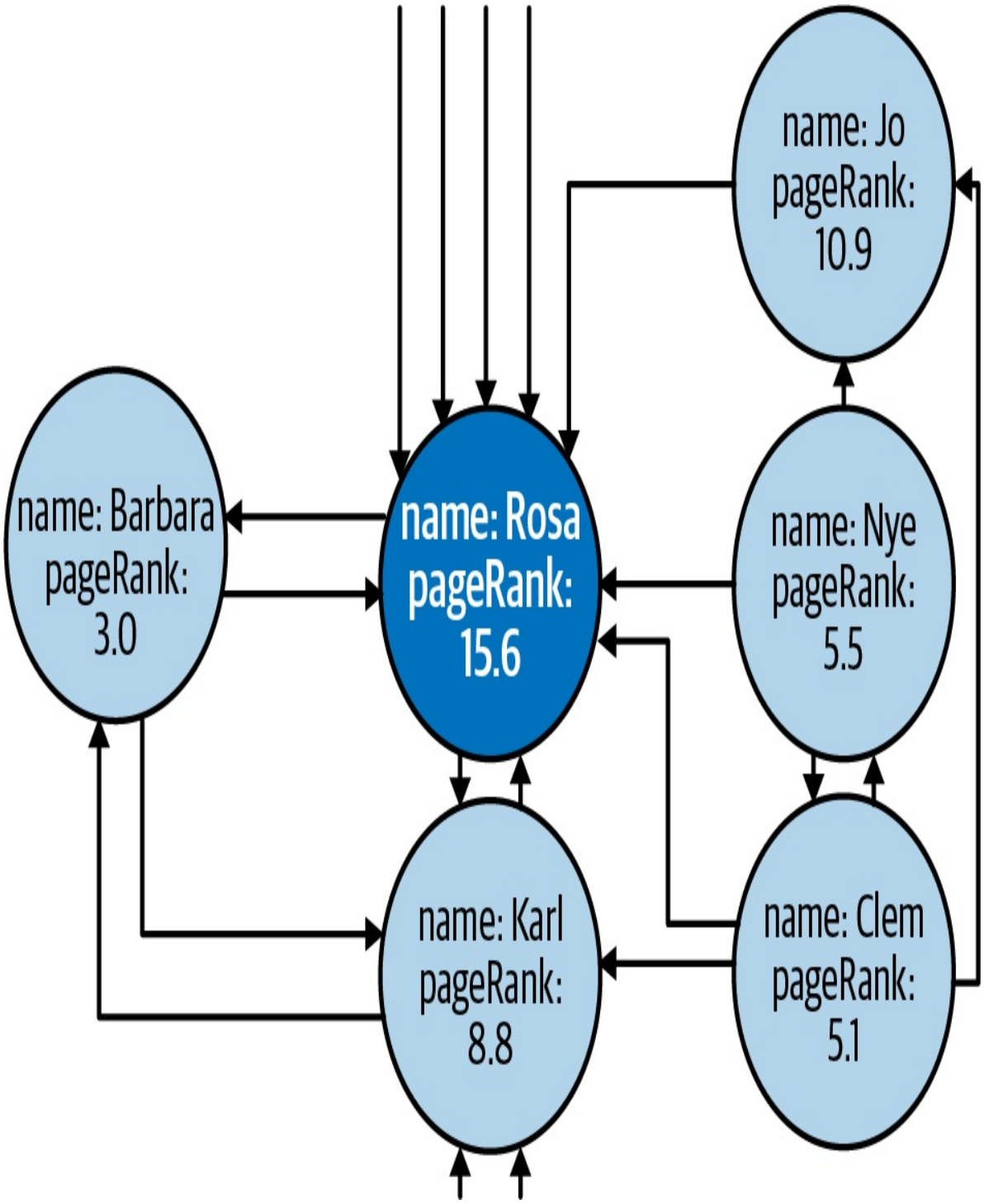

Figure 4-3. Node importance via the PageRank algorithm

Graph algorithms excel at finding global patterns and trends, but we'll want to choose and tune the algorithms to suit our specific questions. A decisioning knowledge graph should support a variety of algorithms and allow us to customize for future growth.

DIVING DEEPER INTO GRAPH ALGORITHMS

The most well-known graph algorithms fall into five classic categories:

- Community detection for finding clusters or likely partitions
- Centrality for determining the importance of distinct nodes in a network
- Similarity for evaluating how alike nodes are
- Heuristic link prediction for estimating the likelihood of nodes forming a relationship
- Pathfinding for evaluating optimal paths and determining route quality and availability

Graph Algorithms (O'Reilly, 2019) by Mark Needham and Amy E. Hodler can help you understand how to use graph algorithms in your domain.

Graph Embeddings

Beyond just understanding data better, graph queries and algorithm results can be used to train ML models, but what if you don't know what to query or which algorithm to use? Do you know if PageRank or a different type of algorithm would be more or less predictive? You could try them all and compare the results, but that would be tedious.

Graph embeddings are a special type of algorithm that encodes the topology of a graph (its nodes and relationships) into a structure suitable for consumption by ML processes. We use these when we know important data exists in the graph, but it's unclear which patterns to look for and we'd like the ML pipeline to do the heavy lifting of discovering patterns. Graph embeddings can be used in conjunction with graph queries and algorithms to enrich ML input data to provide additional features.

Embeddings encode a representation of what's significant in our graph for our specific problem and then translate that into a vectorized format (as seen at the bottom of Figure 4-1). We humans can't readily understand the list of numbers it creates, but it's precisely in the format we can use to train ML models.

> **TIP**
>
> Some types of graph embeddings also learn a function to represent our graph. We can apply that function to new incoming data and predict where it fits in the graph topology.

Graph embeddings are very useful because rather than running multiple algorithms to describe specific aspects of our graph topology, we can use graph structure itself as a predictor. Graph embeddings expand our predictive capabilities, but they typically take longer to run and have more parameters to tune than other graph algorithms. If we know what elements are predictive, we use queries and algorithms for feature engineering in ML. If we *don't* know what is predictive, we use graph embeddings. Both are good ways to improve decisioning graphs.

ML Workflows for Graphs

Analyzing the output of graph queries and algorithms and using them to improve ML is great, but we can also write results back to enrich the graph. In doing so, those results become queryable and processable in a virtuous cycle that feeds the next round of algorithmic or ML analysis. Creating a closed loop within our decisioning knowledge graph means we can start with a graph, learn what's significant, and then predict things about new data coming in, such as classifying someone as a probable fraudster, and write it back into the graph. The knowledge graph is enriched by the cycle shown in Figure 4-4.

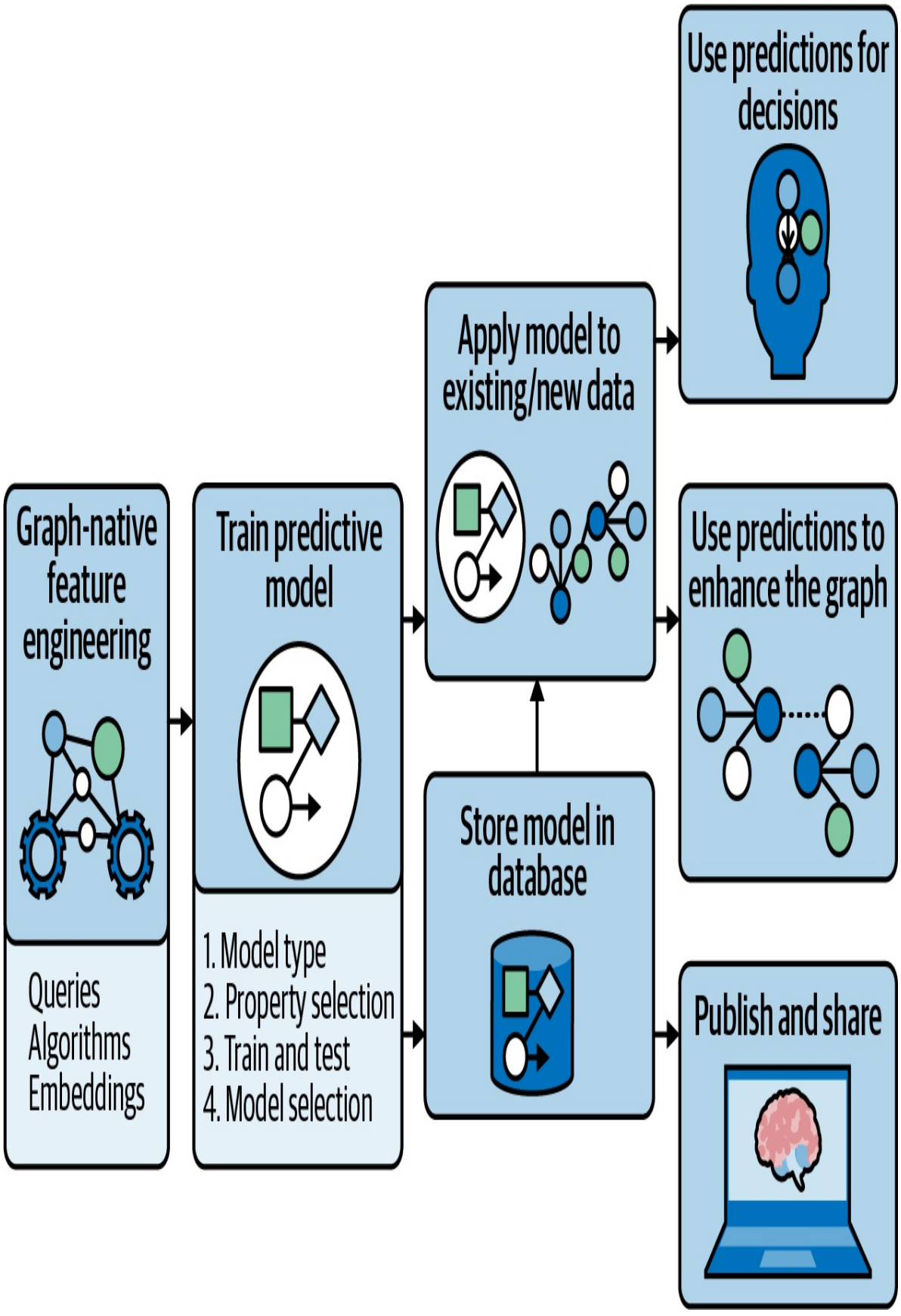
Use predictions for decisions
Apply model to existing/new data
Graph-native feature engineering
Train predictive model
Use predictions to enhance the graph
Store model in database
Queries
Algorithms
Embeddings
1. Model type
2. Property selection
3. Train and test
4. Model selection
Publish and share

Figure 4-4. A closed-loop graph ML workflow

Graph machine learning is often used for knowledge graph completion to predict missing data and relationships. *In-graph ML* keeps ML training inside the graph, which enables us to incorporate its ML workflows into our knowledge graph for continuous updates as new data is added.

It also avoids the need to create data integration pipelines and move data between various systems when making graph-specific predictions. For example, using a graph-centered approach we can avoid having to export similarity scores and embeddings from our knowledge graph into toolkits like TensorFlow[2] to predict node categories and then write back the results to update our knowledge graph—all at high latency and systemic complexity. Instead, doing this entirely within our knowledge graphs significantly streamlines and accelerates the process.

> **TIP**
>
> We recommend automating some of the trickier tasks like transforming data into a graph format, test/train splitting of graph data, multiple model building, and accuracy evaluation. If you have graph and ML expertise, you might be able to build a decisioning knowledge graph that includes and automates ML model training for graphs. If not, consider a vendor or open-source solution with in-graph ML capabilities.

Graph Visualization

Visualizing data, especially relationships, allows domain experts to better explore connections and infer meaning. For knowledge graphs, we need tools like those shown in Figure 2-3 to help visually inspect raw data, understand analytics results, and share findings with others.

Walking through relationships, expanding scenes, filtering views, and following specific paths are natural ways to investigate graphs. A visual representation of a graph needs to be dynamic and user customizable to support interactive exploration. In addition to direct query support, graph visualizations need low-code/no-code options and assistive search features,

like type-ahead suggestions, to empower a broader range of users.

Data scientists also benefit from visualizing algorithm and ML results. With the level of abstraction raised visually, a data scientist can focus on the necessary complexity of an experiment and not become bogged down in accidental complexity. For example, our tools can visualize PageRank scores as node sizes, node classifications as icons, traversal cost as line thickness, and community groups as colors. With these visual abstractions, the important aspects of the underlying data are immediately apparent, where they would be hidden in raw data. Once a data scientist is satisfied with their results, a graph visualization tool enables them to quickly prototype and communicate findings with others, even where those others are not experts in graph technology.

Decisioning Knowledge Graph Use Cases

There are many use cases for analytics, data science, and ML with decisioning knowledge graphs. Here are a few:

- *Finding and preventing fraud* based on detecting communities of like behavior, unusual transactions, or suspicious commonalities. Results are typically written into an actioning knowledge graph to support online fraud detection or passed downstream into a ML workflow such that better predictive models can be built.
- *Improving customer experience and patient outcomes* by surfacing complex sequences of activities for journey analysis. Typically, results are interpreted by domain experts who understand the journey of the user and can spot anomalous data, usually with help from visualization tools.
- *Preventing churn* by combining individual data with community behavior and influential individuals in that network. Results are often written back into an actioning knowledge graph so that risky customers can be identified at points of contact. Results are also used to improve ML models so that holistic churn prediction across

the customer base can be improved.

- *Forecasting supply chain needs* using a holistic view of dependencies and identified bottlenecks. Results are written into the actioning knowledge graph that underpins the supply chain system, enriching it so that the supply chain is more robust.
- *Recommending products* based on customer history, seasonality, warehouse stock, and product influence on sales of other items. Results are written into the actioning knowledge graph that supports the product catalog.
- *Eliminating duplicates* and ambiguous entities in data based on highly correlated attributes and actions. The output is typically sent to downstream ML systems (e.g., graph neural networks) for predictive analysis around whether claims to identity should be linked. Output may also be used to train those downstream systems.
- *Finding missing data* using an existing data structure to predict undocumented relationships and attributes. Often computed data is written back into the actioning knowledge graph or sent downstream to ML for further processing before being written back into an actioning knowledge graph.
- *What-if scenario planning* and "next best action" recommendations using alternative pathways and similarities, typically consumed by experts in the first instance and ultimately written back to an actioning knowledge graph so that online systems can have better "next best actions."

REAL-TIME DECISIONING

Real-time decisioning solutions require immediate analysis of current options before immediately matching them to the most appropriate choice (i.e., making a recommendation). For instance, in Chapter 2 we show how retail recommendations can be made in real time based on a knowledge graph.

> It's important to understand that interactive speeds prohibit online use of global algorithms and ML training. The computation cost and latency to run large graph algorithms are simply too high to run on a per-request basis. Instead, graph algorithms, data science, and ML often operate on a different cadence to real-time queries. The more expensive processing runs in the background, continuously enriching the actioning knowledge graph, while real-time queries get better results over time as the underlying knowledge graph improves.

Each of these use cases is valuable, and it's possible that more than one may apply to your business. What's nice about these use cases is that tooling already exists that can implement them. We don't need to invest in building such tools; we just need to get our data in a format so that the tools can conveniently process it. Often this can be as simple as storing it in a graph database. From here, we can incorporate the knowledge graph into our business processes to help to make better decisions.

Boston Scientific's Decisioning Graph

Boston Scientific is a global medical device company that develops and manufactures a wide range of innovative diagnostic and treatment medical products, including pacemakers and artificial heart valves. Health care practitioners have helped more than 30 million patients around the world using the company's products.

Boston Scientific has an integrated supply chain from raw materials to complex devices that includes development, design, manufacturing, and sales. Predicting and preventing device failures early in the process is crucial. However, the company had difficulty pinpointing the root cause of defects, limiting its ability to prevent future problems.

Using a decisioning graph, Boston Scientific was able to apply graph analytics to their supply chain and consequently improve the reliability of their products. It started with a knowledge graph that included parts, finished products, and failures, as seen in Figure 4-5. The chosen organizing

principles used an ontology to define a hierarchy of parts as `ISSUED` other parts to create assemblies that lead to finished products. Then they add relationships of finished products to events that `RESULTS_IN` failures. Using graph queries, Boston Scientific is able to quickly reveal subcomponents' complex relationships and trace any failures to relevant parts.

The company was able to identify previously unknown vulnerabilities by adding graph algorithms to rank parts based on their proximity to failures and match other components based on similarity. Since results can be automatically written back to their decisioning knowledge graph, Boston Scientific continues to enhance its data and improve product reliability across multiple collaborating teams.

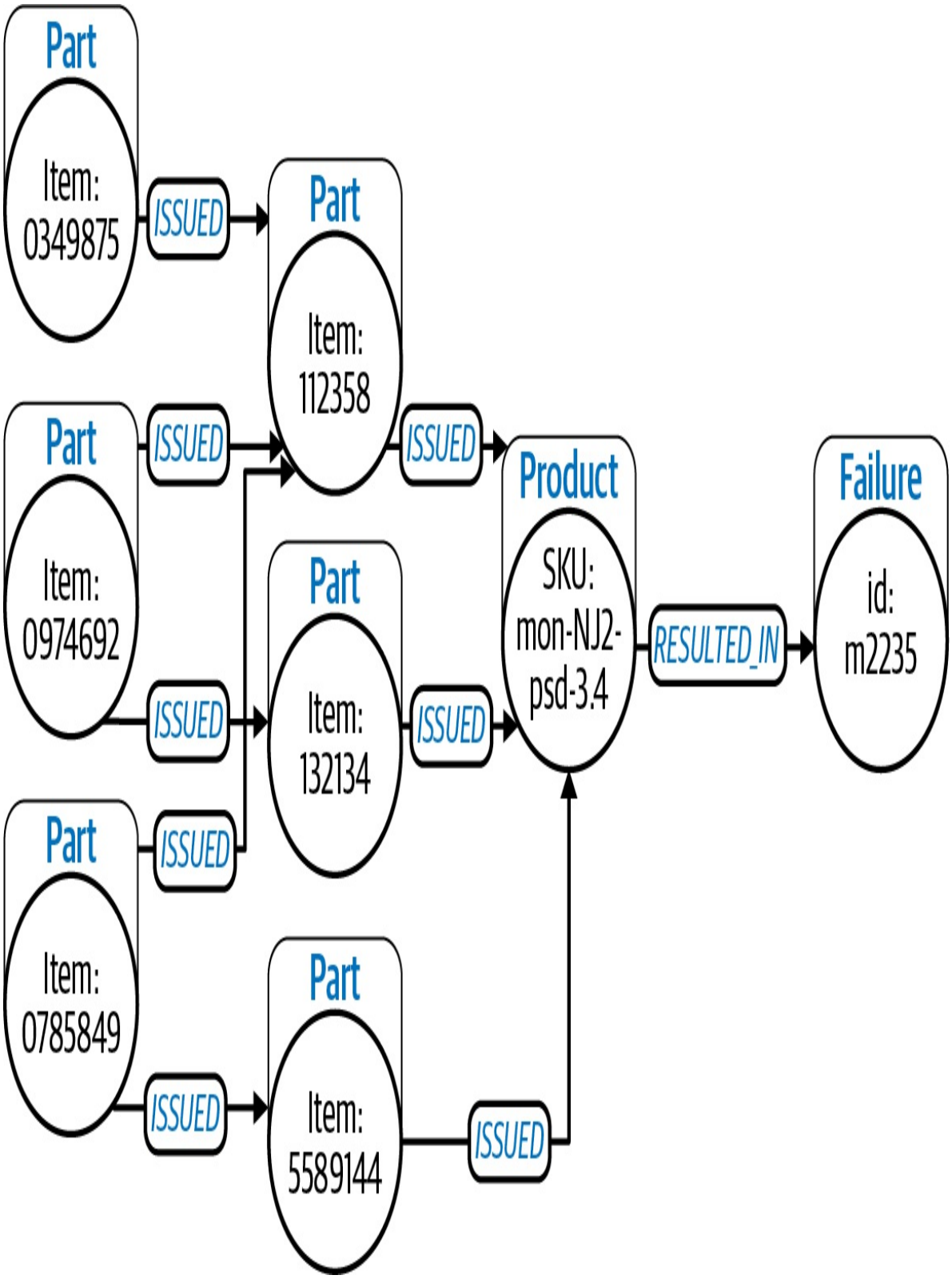

Figure 4-5. Illustrative example of a simple graph similar to Boston Scientific's data model

Better Predictions and More Breakthroughs

With a decisioning knowledge graph, we can answer otherwise difficult questions based on nuanced relationships from large graphs. Graph algorithms and in-graph ML make it possible to predict outcomes based on the connections between data points rather than raw data alone. Combining both approaches can substantially improve the quality of results obtained.

A decisioning knowledge graph is not used directly by online systems but powers those online systems by enriching their underlying knowledge graphs, either directly or as the result of a longer analytics and ML workflow. The tools and patterns around a decisioning knowledge graph open up a new possibilities for gaining insight from data that has until recently only been accessible to researchers and a very few advanced technology companies. A decisioning knowledge graph commoditizes and democratizes a powerful set of tools for widespread business use.

---

1 Although PageRank was developed by Google to understand the relative ranking of Web pages, it's actually named after its inventor, Larry Page.

2 TensorFlow is a free, open-source software library for ML.

Chapter 5. Contextual AI

The last few years have demonstrated impressive improvements in AI predictive capabilities but with narrow application and sometimes disturbing results. For AI to reach its full potential, we believe it must incorporate wider contextual information from knowledge graphs.

We think of *context* as the network surrounding a data point of interest that is relevant to a specific AI system. Using knowledge graphs with AI systems is the most effective way to achieve *contextual AI*, which incorporates neighboring information, is adaptive to different situations, and is explainable to its users.

This chapter explains why AI needs the connected context of knowledge graphs and its benefits for more trustworthy data, higher accuracy, and better reasoning. We explain how this can be achieved with some straightforward knowledge graph patterns and showcase some successful systems where graphs and AI have been combined.

Why AI Needs Context

AI is intended to create systems for making probabilistic decisions, similar to the way humans make decisions. Humans make thousands of decisions every day, often without conscious thought, by matching observed patterns against contextualized experiences.

A person who says, “You saw her duck,” may be asking you directly whether you saw a woman get out of the way of a flying object. But perhaps they are stating that you or someone named *Yu* (于 in Chinese) saw a friend’s pet bird or are telling them to help butcher poultry for dinner. It’s hard to know the meaning of this text without context. Likewise, if we had to meticulously review the inputs in isolation to interpret the phrase in Figure 5-1, let alone all the decision paths we make every day, the complexity would paralyze us.

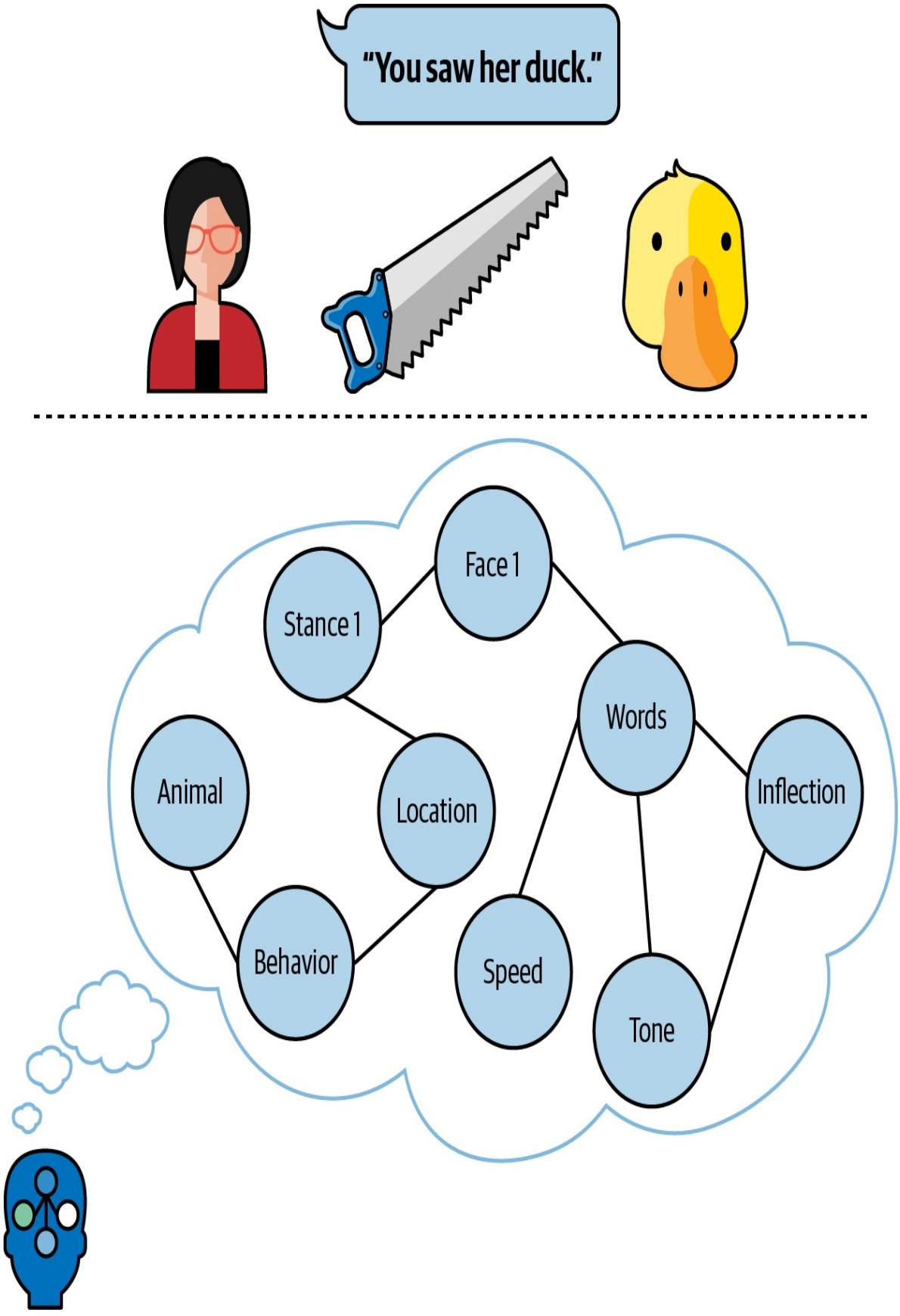
"You saw her duck."
Face 1
Stance 1
Words
Animal
Location
Inflection
Behavior
Speed
Tone

Figure 5-1. Linked context used to interpret the phrase "you saw her duck"

Instead, humans deal with ambiguities by using context to determine what's significant in a situation and extend that learning across domains to understand new conditions. Once we learn a complicated or nuanced task like driving a car, we can apply that to other scenarios, such as different vehicle types. We are masters of abstraction and of recycling lessons learned.

Today's AI is not very able to generalize. Instead, it is effective for specific, well-defined tasks. It struggles with ambiguity and mostly lacks transfer learning that humans take for granted. For AI to make humanlike decisions that are more situationally appropriate, it needs to incorporate context.

EXPLAINABILITY

Predictions made by AI must be interpretable by experts and ultimately explainable to the public if AI systems are to expand their utility into more sectors. In the absence of understanding how decisions were reached, citizens may reject recommendations or outcomes that are counterintuitive. In systems where human safety is paramount, such as medical imaging, explainability becomes a critical aspect of running a system that will not harm people. Explainability isn't a nice-to-have—it is a required component of AI, and being context driven improves explainability.

Graph technology is the best way to maintain the context for explainability. It offers a human-friendly way to evaluate connected data, enabling human overseers to better map and visualize AI decision paths. By better understanding the lineage of data (context of where it came from, cleansing methods used, and so forth), we can better evaluate and explain its influence on predictions made by the AI model.

Contextual information from a knowledge graph helps an AI solution flex by enabling it to learn and abstract guiding principles across scenarios. For example, we might train a semiautonomous car to slow down in rainy weather or near-freezing temperatures. We can use pretrained knowledge

about this situation, including context, and then transfer that learning, which reduces the amount of data and training required. For example, in Figure 5-2 we want the AI to apply contextual information such as an approaching bridge or unusual driver behavior and decide that its response should be similar to wet weather.

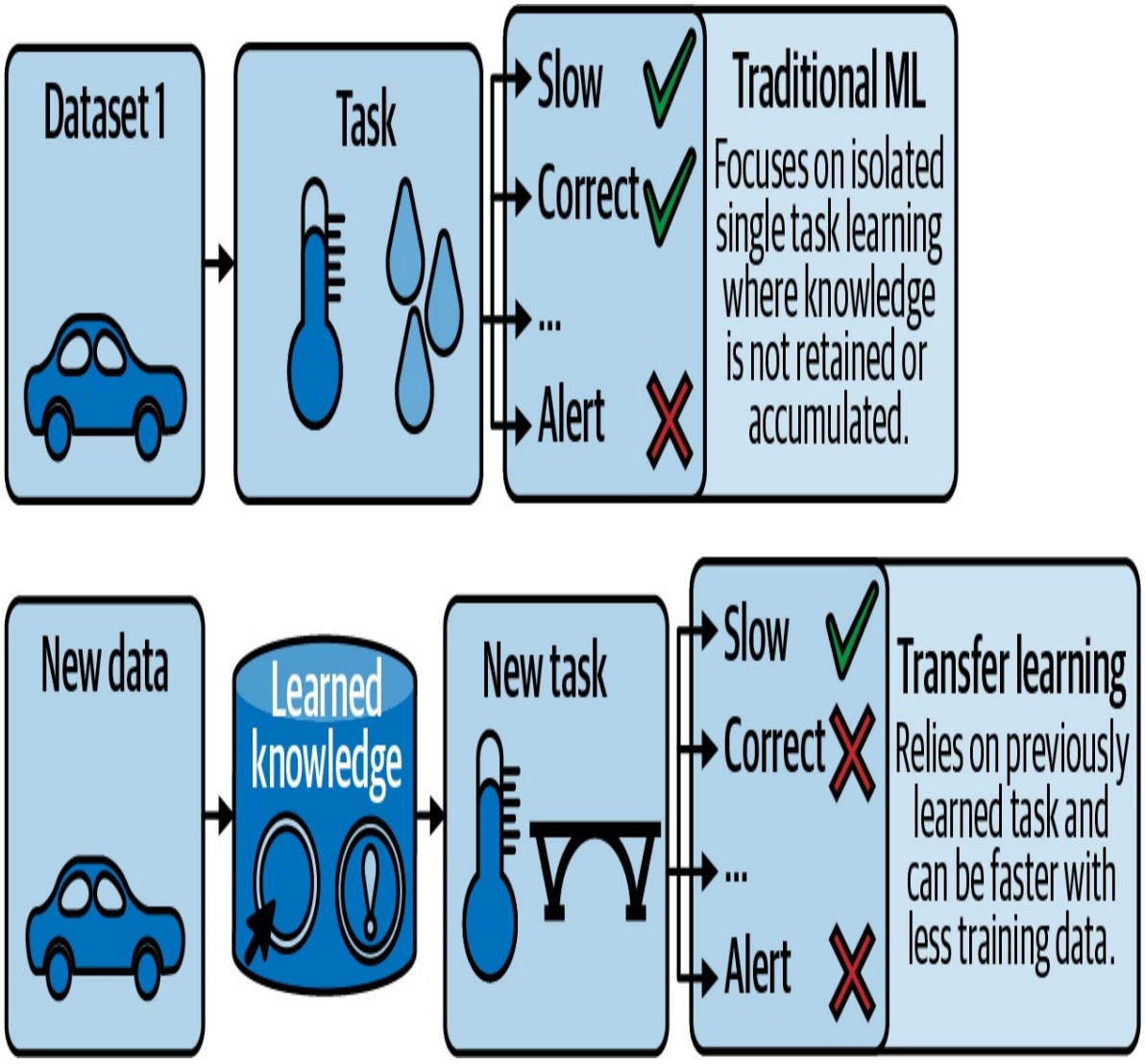

Figure 5-2. Example of traditional ML versus transfer learning for a semiautonomous vehicle

AI systems with many-layered and interacting relationships, like smart homes, also need a considerable amount of context to respond appropriately to different situations. Even a simple request like “turn off my daughter’s lights” requires a system to understand contextual information, as shown in Figure 5-3. Specifically, it requires knowledge of who’s speaking, which

person is your daughter, which lights *belong* to her, whether the lights are on now, and so on.

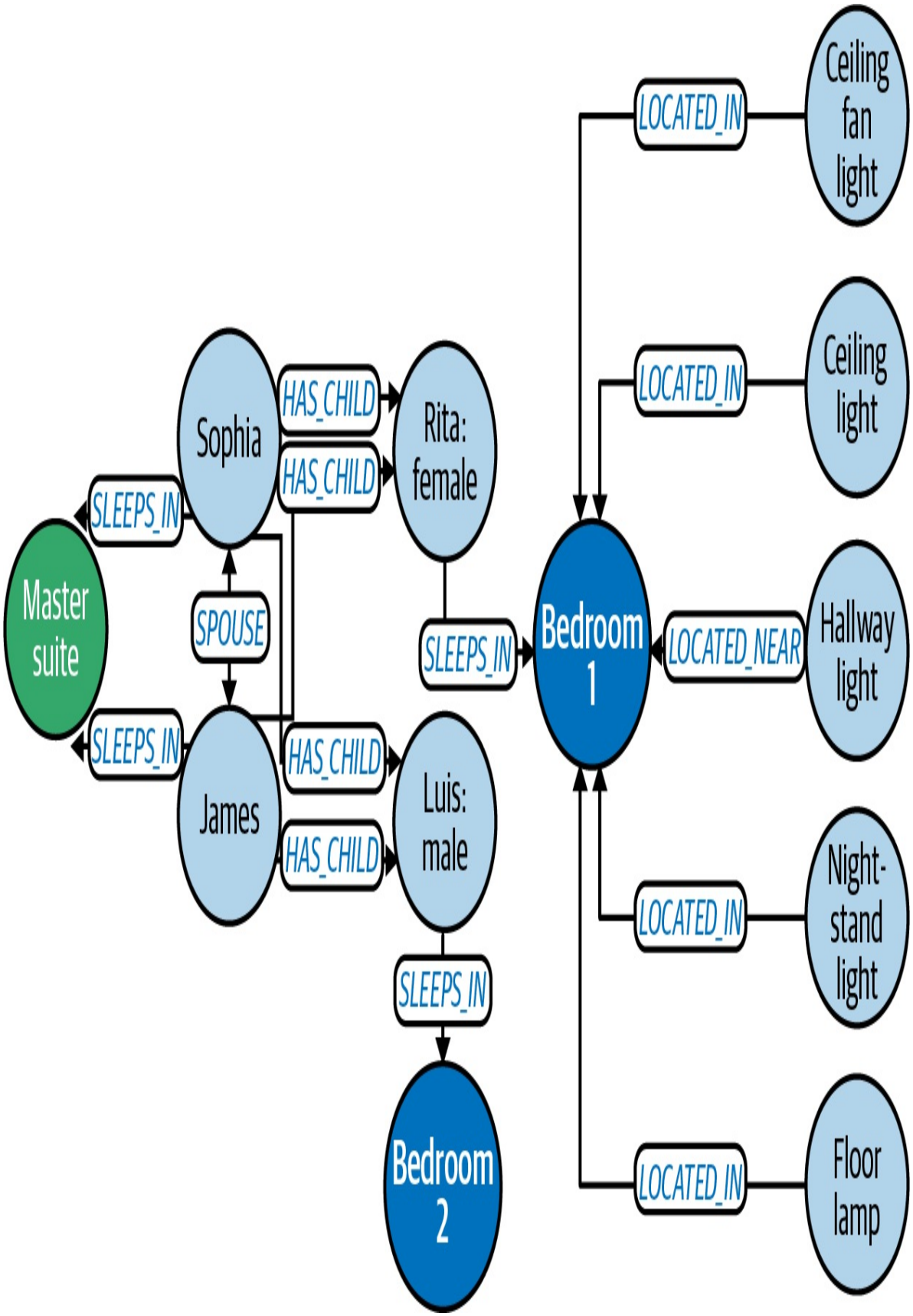
Ceiling fan light
LOCATED_IN
Ceiling light
LOCATED_IN
Hallway light
LOCATED_NEAR
Night-stand light
LOCATED_IN
Floor lamp
LOCATED_IN
Bedroom 1
SLEEPS_IN
Rita: female
HAS_CHILD
HAS_CHILD
Sophia
SLEEPS_IN
Master suite
SPOUSE
SLEEPS_IN
James
HAS_CHILD
HAS_CHILD
Luis: male
SLEEPS_IN
Bedroom 2

Figure 5-3. Simplified graph data model for lights in one area of a smart home

The point is that AI benefits greatly from context to enable probabilistic decision making for real-time answers, handle adjacent scenarios for broader applicability, and be maximally relevant to a given situation. But all systems, including AI, are only as good as their inputs. Garbage in means garbage out, even if AI produces that garbage. To ensure that we use good-quality data to build AI systems, we need to understand that data's provenance.

Data Provenance and Tracking for AI Systems

Knowing the provenance of data is essential for trustworthy AI. Although this seems like a straightforward data lineage task as discussed in Chapter 3, we also need to track adjacent information like who collected the data, how it was transformed, who had access to it, and so on.

> **WARNING**
>
> As a cautionary example, some predictive policing models have used decades-old arrest and prosecution data despite inherent racism in the data. Simulations have illustrated the tendency of such approaches to promote a cycle of increased policing and arrests.[1] Understanding our data's backstory requires tracking context.

Knowledge graphs are well suited for bringing together information and tracking how information has changed as it passes from system to system and person to person. In the course of their duties, data engineers may extract data used in AI from various systems, more data may be purchased from third-party providers, and all of it may be transformed and mixed together by multiple processes even before it comes to rest in a knowledge graph. Each stage of acquiring, cleansing, and curating data may inject errors and biases into the system.

Tracking data lineage with even a simple model as in Figure 5-4 gives data engineers another lens into the suitability and quality of the data for the problem at hand. Automatically tracking such provenance metadata (as a

graph, of course) allows us to efficiently monitor the system to ensure it is operating within acceptable parameters. For example, we can use the graph to discover that the team cleansing the data was diverse or otherwise relatively free from bias. It also allows us to understand the assumptions they made when balancing data for statistical representation.

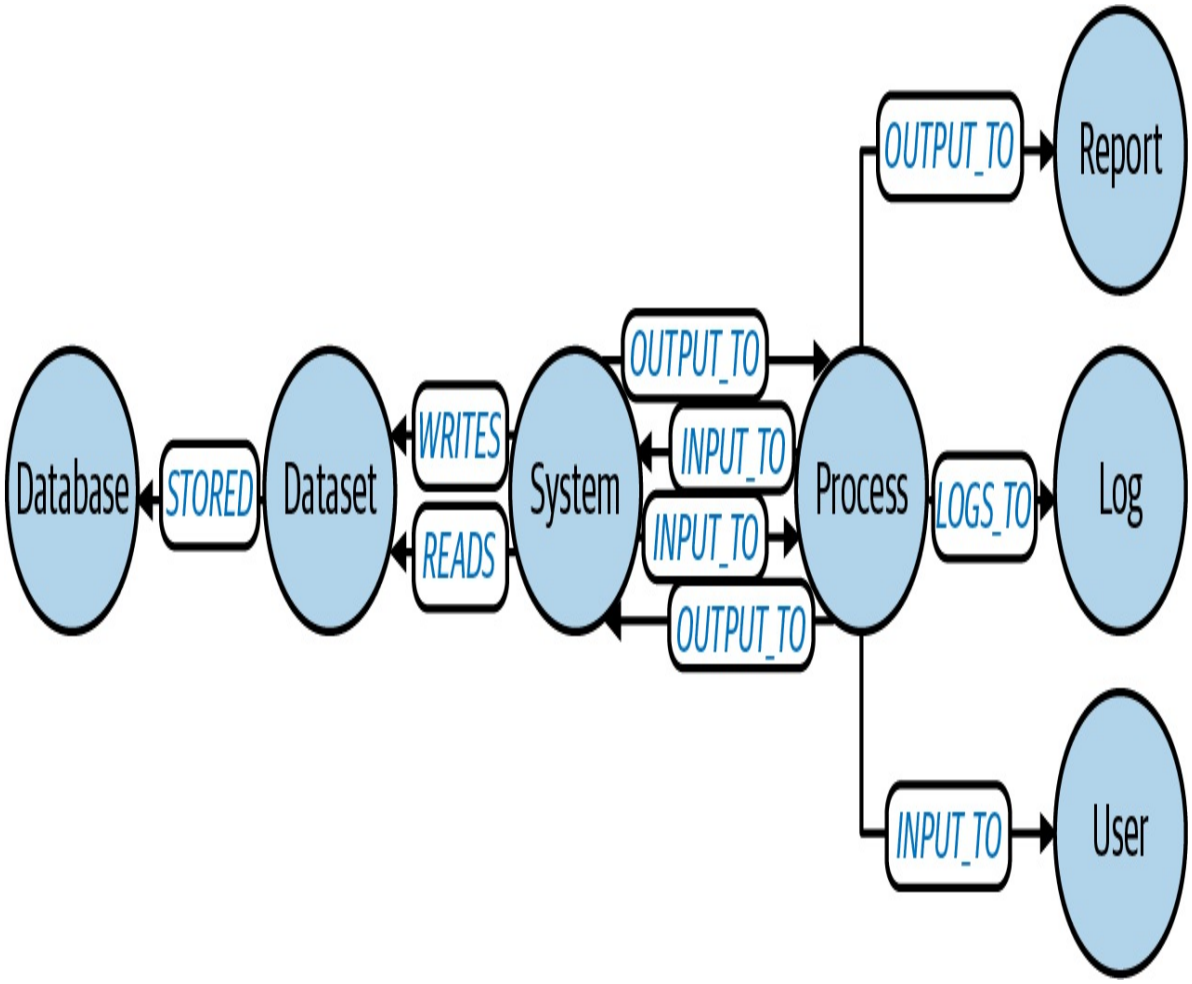

Figure 5-4. Simple data lineage model for tracking changes

Tracking knowledge graphs helps with audit trail and existing compliance requirements too. Although AI norms and regulations are still evolving, we need to have systems to manage an increasing amount of this complex peripheral information.

> **TIP**
>
> As an important case in point, recent judgments against using biometric data without

permission have made it very clear that the industry needs to verify contextual information in order to operate some AI systems. We need to be ready for provenance to be a prerequisite for owning and operating compliant AI systems. See, for example, this article from The Verge.

We believe that transparent data sourcing and provenance are essential for trustworthy AI. That starts with internal business guidelines and processes but also requires technology to support better data lineage. Knowledge graphs allow us to employ organizing principles based on linking where the data came from and how data has changed. An AI system that lacks such traceability has no way of verifying any claims it makes of its behavior and, by definition, cannot be trusted. To earn that trust, we must be able to trace.

But having reliable AI results doesn't end with the data. We also have to understand the ML techniques and processes themselves.

Diversifying ML Data

Knowledge graphs enable us to employ context to help solve some of the most difficult issues in training data: small data sizes and the lack of data variety. Since training data is often scarce, graphs can help squeeze out every possible feature, including relationships and topological information. This might require a change in mindset to diversify the type of data regularly included in ML.

Suppose you're not using a knowledge graph. In that case, the tendency is to demand more training data, typically of the same type of information you're already using. Additional data isn't always readily available or affordable. Even if we can obtain more data, we need to be careful of diminishing returns and *overfitting*. Overfitting involves heavily training predictive models on specific data or features, resulting in increased accuracy scores for that training data but a decline when used on new data. Broadening the data types used in ML, like adding relationships, is a simple tactic to guard against overfitting and expands results for broader scenarios.

Contextual information describes how entities relate to one another and

events instead of describing static elements about individual entities, and context is a fundamentally different type of information. When building AI systems, we can extract context from graphs for more variety in training and improve predictions.

Imagine we're trying to improve patient outcomes for a complex disease like diabetes. We can train a predictive model on things like current medications, test results, and patient demographics. Perhaps our model tells us which medications by age and gender predict good versus poor outcomes, and it's 98% accurate according to our training and testing data.

However, in practice we find these predictions have a high false-positive rate. What factors could be missing? We could try to boost our results by simply broadening the number of features used in the model training, but that can lead to overfitting and still leaves out the context in which this disease occurs. Since complex diseases develop over time, the sequence of diagnosis and medications might be influential. Or perhaps contextual information like lifestyle might have a bigger impact than we expected.

Using graphs, we can capture the path of events over time down to each doctor visit, test result, diagnosis, and medication change. Figure 5-5 illustrates a patient journey for diabetes, a very complex disease, considering only a 90-day window around the diagnosis. The paths in this relatively small journey quickly become unwieldy. It's clear that we need more advanced tools when we add multidimensional context. To incorporate context into an AI system, we can use graph embedding to codify the topology of the journey paths and make predictions based on all the surrounding information we have *in context*. We can also look at various predictive aspects, like the different professionals involved, and see how that might have a strong influence on outcomes.

Example patient-diagnosis journey 90 days after initial condition

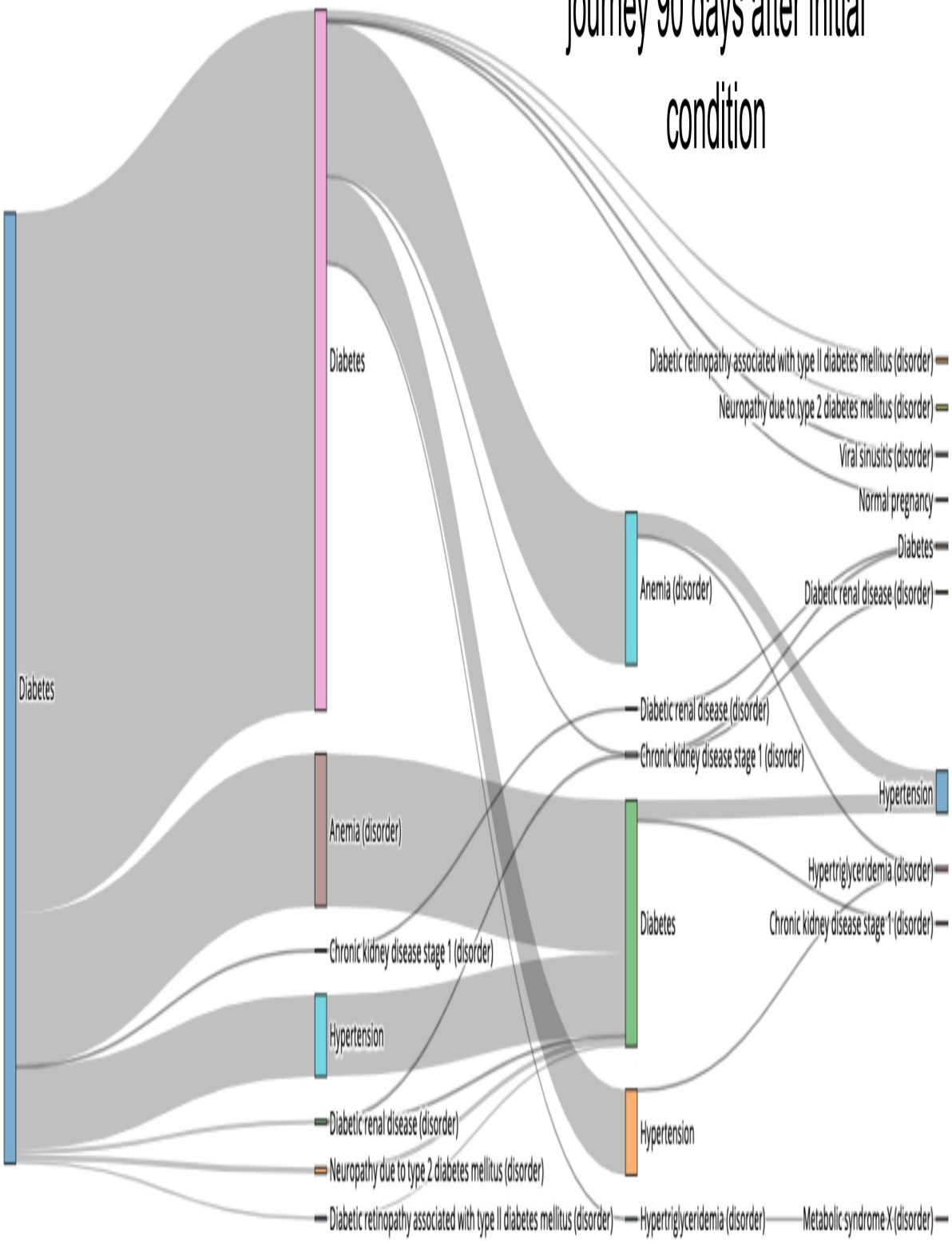

Figure 5-5. Example patient journey of diagnosis using a graph to capture paths (source: Neo4j demonstration software)

Adding relationship and structural information to ML is one of the best ways to add contextual information and diversify the type of data used in AI systems. Once we have added context to the foundation of AI systems for data assurance and better ML results, we should turn to how we can further leverage context to improve ML processes and operations.

Better ML Processes

All of these graph-AI systems produce value only when they're in production. The practice of *MLOps* (*machine learning plus operations*) has grown up around managing the life cycle of AI systems, but emphasizes the tasks of taking an ML model into production. MLOps uses high levels of automation to ensure the quality of production ML and business and regulatory compliance.

It's fitting that since graphs are a natural underlay for ML, they are also a natural underlay for MLOps. In fact, a knowledge graph makes an excellent *AI system-of-truth*, which gives a complete picture of the AI data. We can use a knowledge graph to collect and dynamically track all the building blocks of an AI system: train/test data, ML models, test results, production instances, and so on. With this system-of-truth, a data scientist can use an existing predictive model as a template for building a new model, quickly identify the building blocks to change, and automate sourcing the building blocks they intend to keep.

FEATURE TRACKING FOR AI SYSTEMS

Data scientists can use a system-of-truth graph to manage even granular building elements such as where and what kind of predictive features are in different ML models. Tracking feature usage becomes extremely important if a particular data element is present in multiple forms or datasets used for ML. For example, if our feature engineering uses zip

code and county data to create two predictive features, we now have heavily weighted the importance of location in our ML model. In this case, we'd easily be able to identify that two *location* features were used in the same model by referencing our system-of-truth that tracked feature usage and type.

Another area of concern for MLOps is how quickly predictions degrade over time, called *predictive* or *concept drift*. Data scientists train ML models on historical data to make predictions on new, unseen data. Since there is always new data coming in, the correlation of the new data to your predictions tends to become less accurate as time passes (or there is a major event like a pandemic). Interestingly, graph-native predictive models, discussed in Chapter 4, appear to have less predictive drift over time than traditional ML models.[2]

This characteristic is likely due to the use of structural information (relationships). Data structures don't typically change as quickly as discrete data points do. Consequently, these predictive models appear to have a longer shelf life with less retraining required, meaning that operations teams can use their decisioning knowledge graphs to create accurate models that stay more accurate longer.

Finally, we need to consider how and where we should keep the *human in the loop*, allowing a human user to change the outcome of an event or process. Although there are many ways to incorporate human interaction, knowledge graphs can help early in the process when an ML model has been developed but not yet put into production. At this point in the process, we want to understand if the model is making the correct predictions. An expert might test hypotheses by exploring similar communities in the knowledge graph or debug unexpected results by drilling into hierarchies and dependencies with an organized graph. In this context, graphs provide contextual information for investigations and counterfactual analysis by domain experts.

Improving AI Reasoning

Organizations are using knowledge graphs to improve the reasoning skills of AI itself by adding context to the decision process. Many AI systems employ *heuristic decision making*, which uses a strategy to find the most likely correct decision to avoid the high cost (time) of processing lots of information. We can think of those heuristics as shortcuts or rules of thumb that we would use to make fast decisions. For example, our hunter-gatherer ancestors would not have deliberated long about a rustle in the bushes. Instead, they would have quickly reacted to the situation as either a threat or opportunity based on heuristics like time of day and location.

Knowledge graphs help AI make better decisions by adding contextual information to their heuristic strategies. In a chatbot example, our AI might have natural language understanding of words and phrases. But to provide the best experience, we need to infer intent based on short interactions with the user and update assumptions as a dialogue progresses. In Figure 5-6 we can see how a chat layer, ML models, and a knowledge layer can work together for continual updates in a recommended architecture.

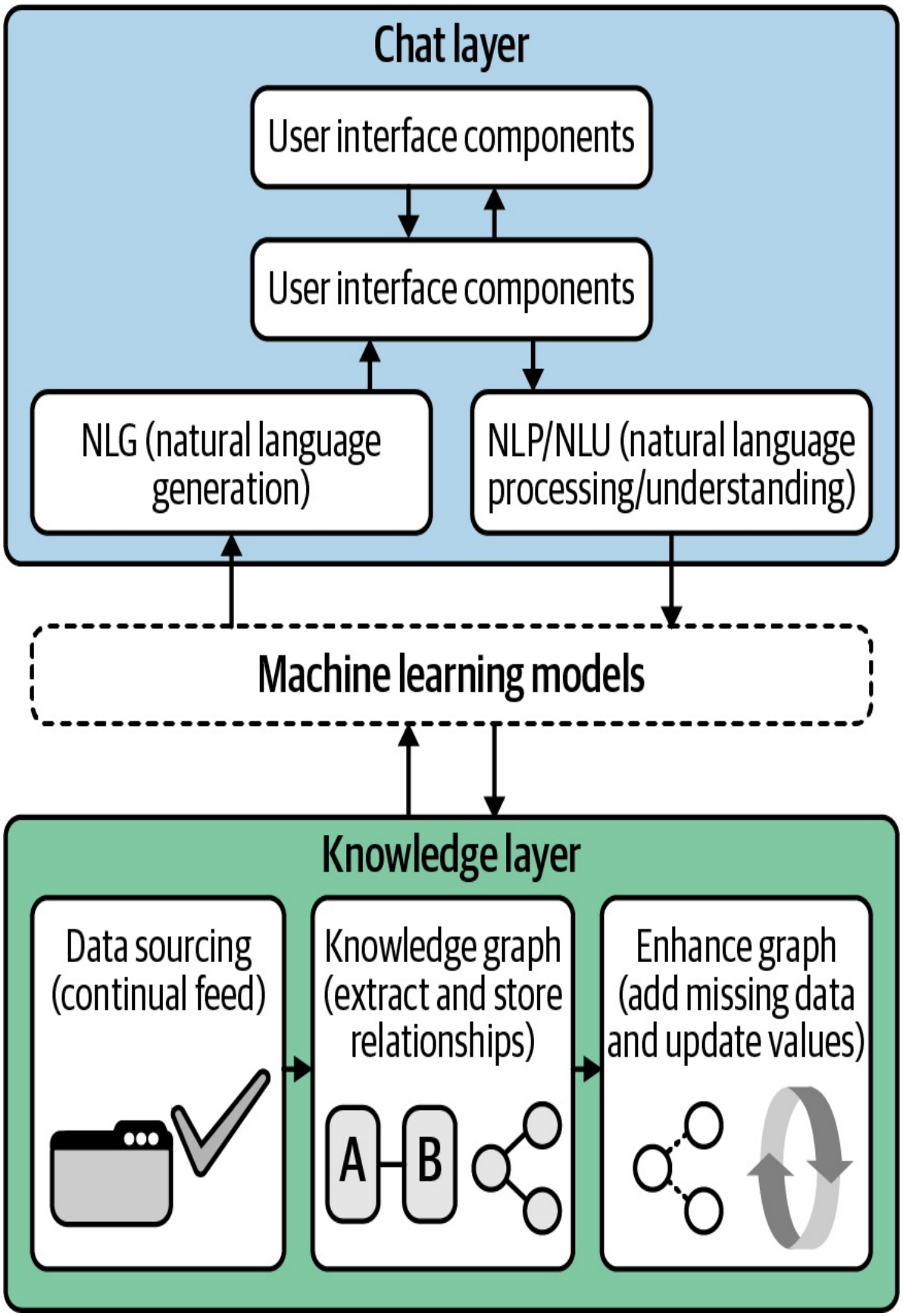
Chat layer
User interface components
User interface components
NLG (natural language generation)
NLP/NLU (natural language processing/understanding)
Machine learning models
Knowledge layer
Data sourcing (continual feed)
Knowledge graph (extract and store relationships)
A
B
Enhance graph (add missing data and update values)

Figure 5-6. Chatbot system architecture proposed by researchers[3]

Meandering chatbot dialogue is frustrating for end users. To address the need for progressive understanding, LPL Financial used a knowledge graph to power its financial chatbot based on a model simliar to Figure 5-7. After training its model to identify keywords in documentation, the company created metatags in its knowledge graph as its organizing principle, effectively providing conversational shortcuts through the graph. The company uses graph analytics to look for similar keywords and clusters of terminology to infer missing relationships and hierarchies. With this information, LPL Financial created a dynamic subgraph to better represent the relationships between questions and answers.

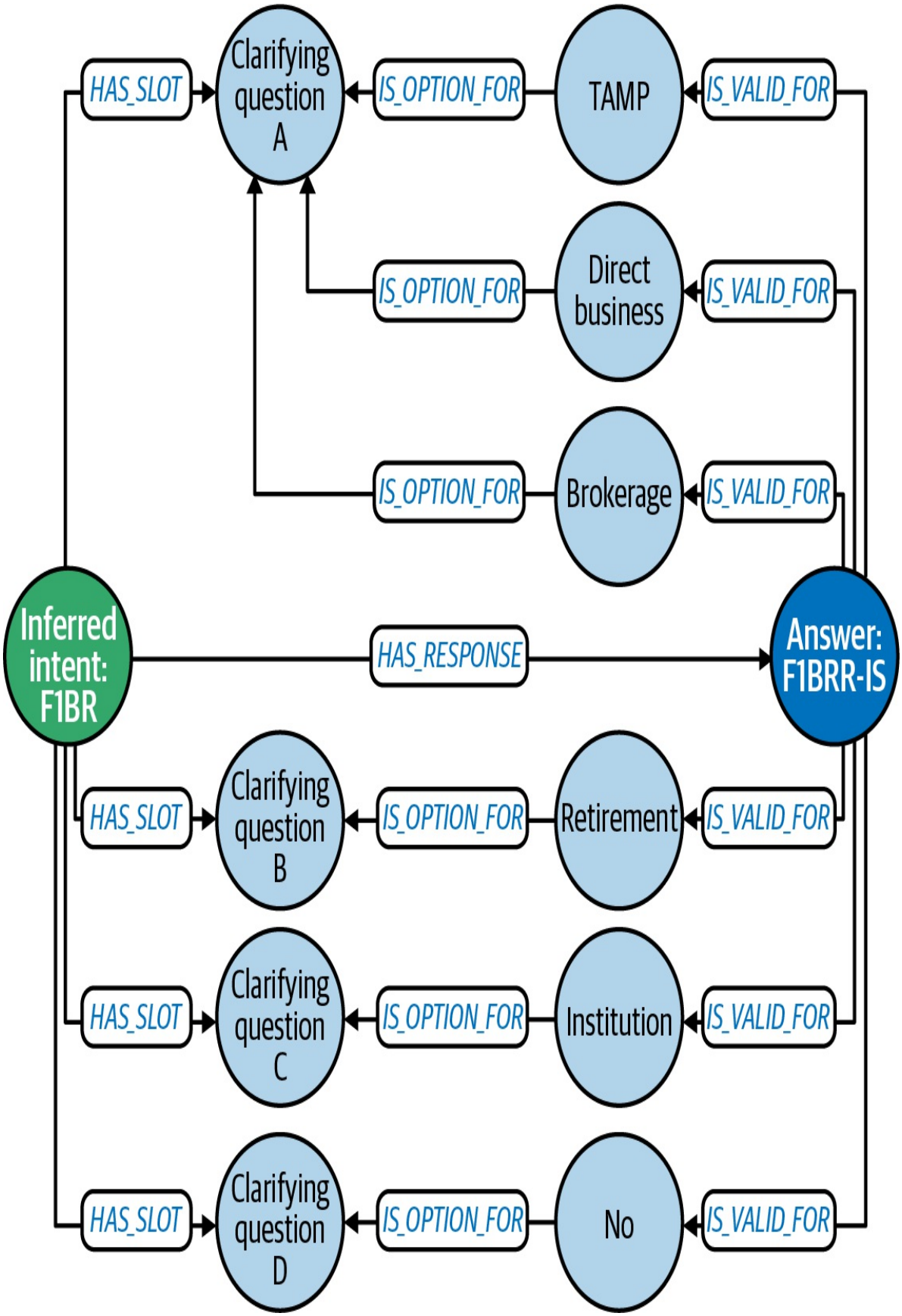
Inferred intent: F1BR
HAS_RESPONSE
Answer: F1BRR-IS
HAS_SLOT
Clarifying question A
IS_OPTION_FOR
TAMP
IS_VALID_FOR
IS_OPTION_FOR
Direct business
IS_VALID_FOR
IS_OPTION_FOR
Brokerage
IS_VALID_FOR
HAS_SLOT
Clarifying question B
IS_OPTION_FOR
Retirement
IS_VALID_FOR
HAS_SLOT
Clarifying question C
IS_OPTION_FOR
Institution
IS_VALID_FOR
HAS_SLOT
Clarifying question D
IS_OPTION_FOR
No
IS_VALID_FOR

Figure 5-7. Representation of the LPL Financial subgraph used for a financial chatbot

Figure 5-7 illustrates the graph data model with the addition of clarification steps that map AI questions to categories of options that are valid for particular answers. LPL Financial reduced the amount of detail needed in each step of the dialogue as well as paired down the number of options the chatbot had as it progressed. As the AI system was used, the actioning knowledge graph was further updated by weighting relationships based on how frequently they led to correct responses. We can see how using a knowledge graph to boost chatbots means faster answers and a more natural experience.

NEAR-FUTURE BREAKTHROUGHS

In a paper from DeepMind, Google Brain, MIT, and the University of Edinburgh, "Relational inductive biases, deep learning, and graph networks" (2018), researchers advocate for using a *graph network*, which "generalizes and extends various approaches for neural networks that operate on graphs, and provides a straightforward interface for manipulating structured knowledge and producing structured behaviors." These researchers believe that graphs have an excellent ability to generalize about data structures, which broadens the applicability and sophistication of AI systems.

The graph-network approach takes a graph as an input, performs learning computations while preserving transient states, and then returns a graph. This process allows the domain expert to review and validate the learning path that leads to more explainable predictions.

Using graph networks also enables whole-graph learning and multitask predictions that reduce data requirements and automate the identification of predictive features. This means richer and more accurate predictions that use fewer data and training cycles.

The research concluded that graphs offer the next major advancement in ML, and we're starting to see the groundwork of this research in production. In fact, the graph embeddings and graph-native ML covered

in Chapter 4 are required first steps to build such graph networks.

The Big Picture

Both AI and knowledge graphs are driving the next wave of breakthroughs and competitive advantage for organizations. But the combination of AI and knowledge graphs is where we are heading as a industry. Those companies that use them together successfully—to reduce the risk of fraud, improve patient outcomes, make better investment decisions, or increase employee productivity—will prosper in a world where good data directly influences good systems.

---

1 The Royal Statistical Society published an Oakland, California, simulation analysis of a ML approach often used for predictive policing and found, "…that rather than correcting for the apparent biases in the police data, the model reinforces these biases."

2 Using graphs to reduce predictive drift has been demonstrated in Jiaoyan Chen et al., "Knowledge Graph Embeddings for Dealing with Concept Drift in Machine Learning," *Journal of Web Semantics* 67 (February 2021).

3 Image is adapted from SoYeop Yoo and OkRan Jeong, "An Intelligent Chatbot Utilizing BERT Model and Knowledge Graph", *Journal of Society for e-Business Studies* 24, no. 3 (2019).

Chapter 6. Business Digital Twin

A *digital twin* is a software version of a real world artifact or process. The concept comes from manufacturing where a digitized version of some component or machine can be subjected to a barrage of virtual testing that would be too hazardous, destructive, expensive, or impractical to perform on the real physical entity. Digital twins are high-fidelity models of the real world and are kept synchronized with their counterparts as new data becomes available.

In the modern enterprise, digital twins are no longer just facsimiles of physical artifacts. Contemporary digital twins can include organizations, processes, and even people. They still consume data from the physical environments of their real world counterparts, whether that's environmental data from Internet of Things (IoT) sensors, packet loss statistics from a network router, or performance and key performance indicator (KPI) data for projects or departments. That data helps the digital twin maintain accuracy over time.

Like its manufacturing counterpart, the enterprise digital twin allows us to reason about the real world and perform all manner of investigations, hypotheses, and abuses to understand how a real system might react. Digital twins can also uncover variances between the as-designed processes and as-implemented processes, surfacing any divergence quickly.

With an enterprise digital twin we can model IT network operations (to look for points of failure or insecurity), optimize heating and cooling for facilities (often to drive down cost), and ensure that logistics for raw materials and processed goods are harmonious. But we can also support business-process optimization to improve the bottom line, look for weaknesses in supply chains to test the resilience of the business when subjected to external shocks, perform succession planning for staffing roles to ensure business continuity, and perform a myriad of other tasks that perhaps only exist informally today.

The enterprise digital twin provides key values:

- A map of the business or a smaller business unit (departmental) or layer (such as IT) of the business
- Real-time understanding of the business
- A way of exposing the model to pressures to see how it reacts

With a digital twin acting as a smart map, it is possible to create a rich virtual view of the business that closely matches reality. Our organizing principle follows accordingly: the elements in the real world and how they interrelate are captured in high fidelity as a property graph, and the constraints and rules that govern the real world become constraints and queries for that property graph.

Digital Twins for Secure Systems

In an increasingly digitized world, the dependability of IT systems is critical to the execution of most businesses. But IT is a complicated field. Even when things are going well, upgrades and changes to software can ripple through a system, causing unintended consequences. But systems can fail especially badly when servers, networks, power supplies, or even data center air conditioning fails.

Such entropic failures are an everyday part of a modern enterprise operating environment, but they are not the only kinds of failures. IT systems are targets for cybercriminals who would like to steal data, disrupt commerce, or hold systems hostage. These malicious failures are as much part of the modern IT landscape as failures due to entropy.

A digital twin can help us understand and manage both scenarios, as you can see in Figure 6-1.

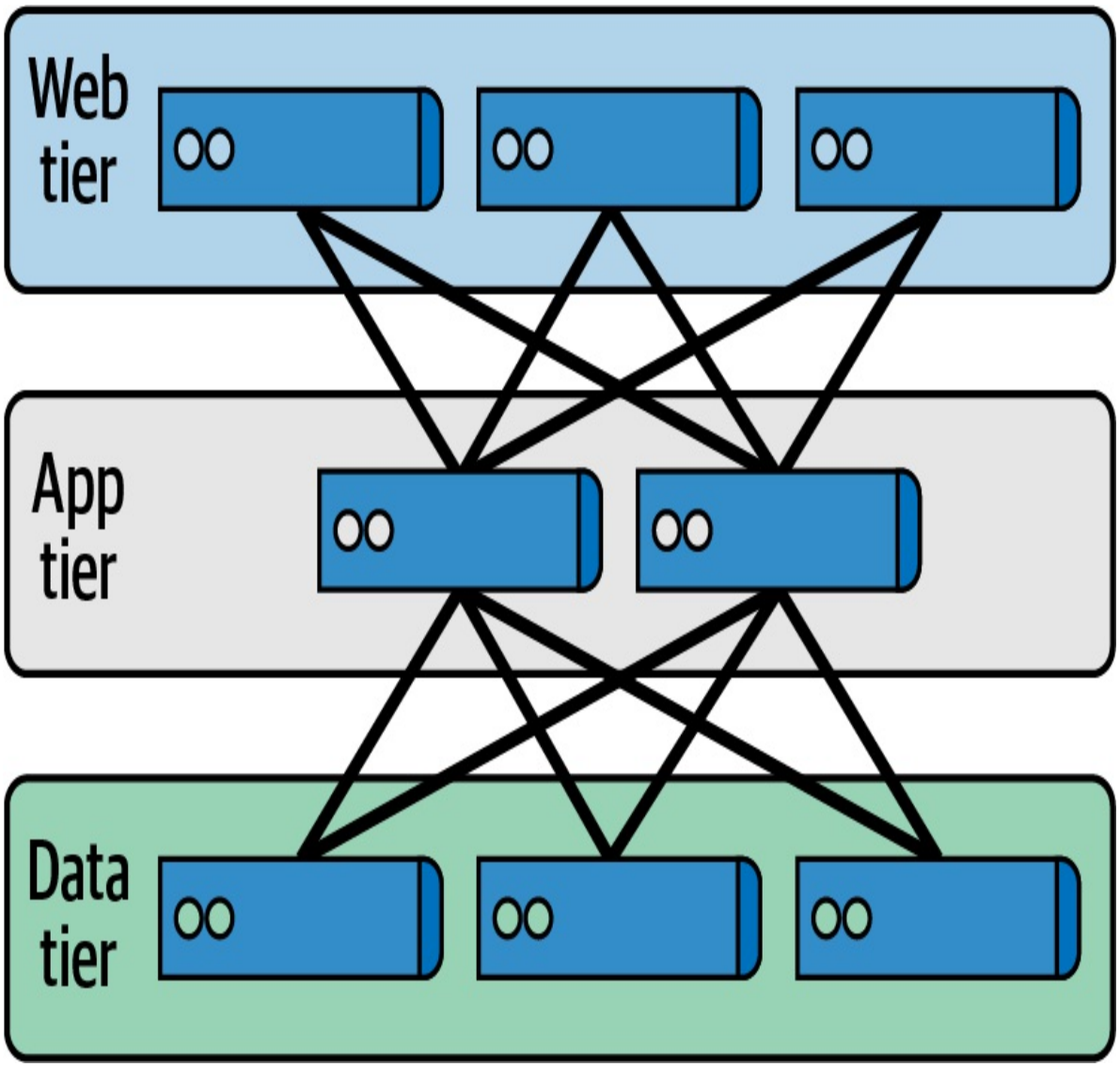

Figure 6-1. A simple IT network

In Figure 6-1 we see a computer network arranged in a classic tiered model with public-facing servers providing some API or Web service, communicating (via a firewall) with some middle-tier servers providing application logic, which in turn communicates (again via a firewall) with servers in the data tier. The connections between the servers form a graph (of course!), and the links show legal connections between machines supporting some application. Strictly, the intent is that a server in one tier should only talk to a server in an adjoining tier using the prescribed protocols, such as TCP or HTTP. Any deviation, like skipping a tier or intratier communications, might indicate a security breach.

The organizing principle for this graph is based on the property graph model, but where a path for a given interaction is only permitted to form between a single Web, App, and Data node, with the correct protocol. Any deviation from this, as shown in Figure 6-2, should be considered suspicious and must be raised as an issue rapidly.

In Figure 6-2 we see one server in the Web tier communicate directly with another host in the Data tier, which contravenes our organizing principle. Discovering this violation is trivial when we capture metrics from the system as a knowledge graph. Our organizing principle insists on a path length of four from Web tier through App tier into the Data tier and then back up the stack. Any connectivity that violates this constraint is suspect and can be quickly flagged.

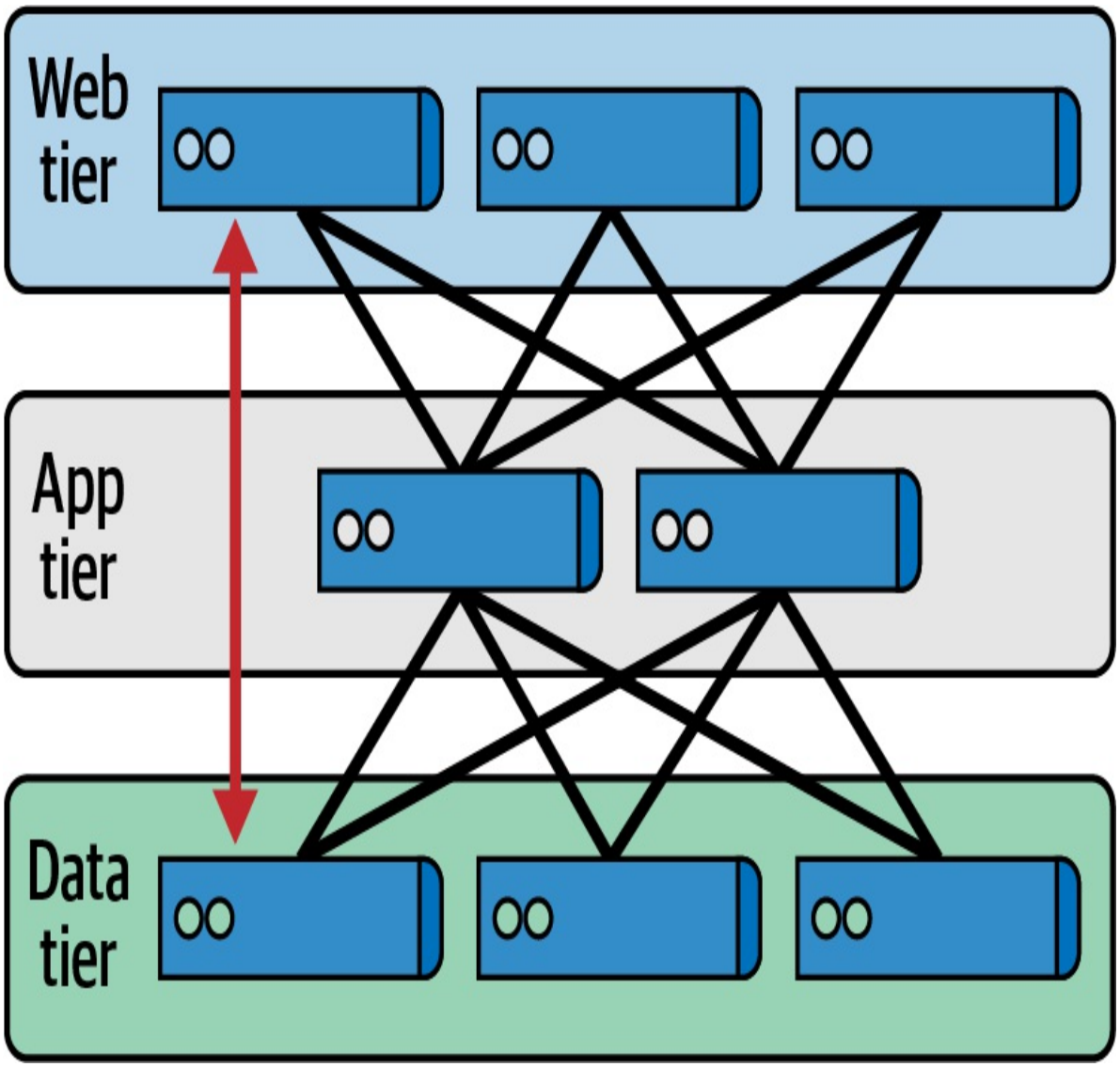

Figure 6-2. An obvious illegal path in the network, and a possible intrusion

ADVANCED PATTERNS

There are many ways a system can be compromised and, accordingly, there are many paths through the system we'd like to prohibit. In a real system, we would likely map all ports and protocols between systems, comparing them to our architecture as intended (the organizing principle). We might also add in human operators and their social patterns so that we can understand the extended attack surface (compromised employees included) and rapidly know who's best placed to respond with either human intervention or automation. By rapidly discovering illegal paths

and deviations in the graph representing the system, we uncover patterns of possible misbehavior and can address them quickly.

In reality, the world isn't so separated into such neat layers. The computer servers that run this system are probably located near one another,[1] in the same data centers and possibly even in the same rack, or as virtual machines on the same physical host. This means they have things in common like switches, routers, cabinets, power supplies, and cooling, to name but a few. Knowing this information, we understand that a brownout causing an air conditioner to halt can also cause disruption for a user thousands of miles away running an app on their phone. And if we understand it, perhaps a malicious actor understands it too. Therefore, that's the best part of our system to attack rather than the usually well-defended servers!

Of course this is speculation, and you might speculate about many other ways that this system could thrive or fail. That spark of curiosity drives experiments, but the engine for those experiments is the knowledge graph that underpins the digital twin, all driven by data from the systems themselves.

Bernard Marr writing in *Forbes Magazine* quotes NASA's John Vickers as saying, "The ultimate vision for the digital twin is to create, test and build our equipment in a virtual environment." The quote continues: "Only when we get it to where it performs to our requirements do we physically manufacture it. We then want that physical build to tie back to its digital twin through sensors so that the digital twin contains all the information that we could have by inspecting the physical build."

Fortunately, in the case of digital twins for IT systems, bringing together systems metrics is a straightforward task, as Figure 6-3 shows. Most computer systems already export operational metrics and alerts, which can be conveniently integrated into the digital twin. LendingClub's MacGyver platform for managing hundreds of microservices and infrastructure is a good example of integrating system metrics into a real-time graph for a digital twin. Done well, the digital twin can be an up-to-date source of activity, a historical view on how the system has developed, and a place where

hypotheses can be robustly tested.

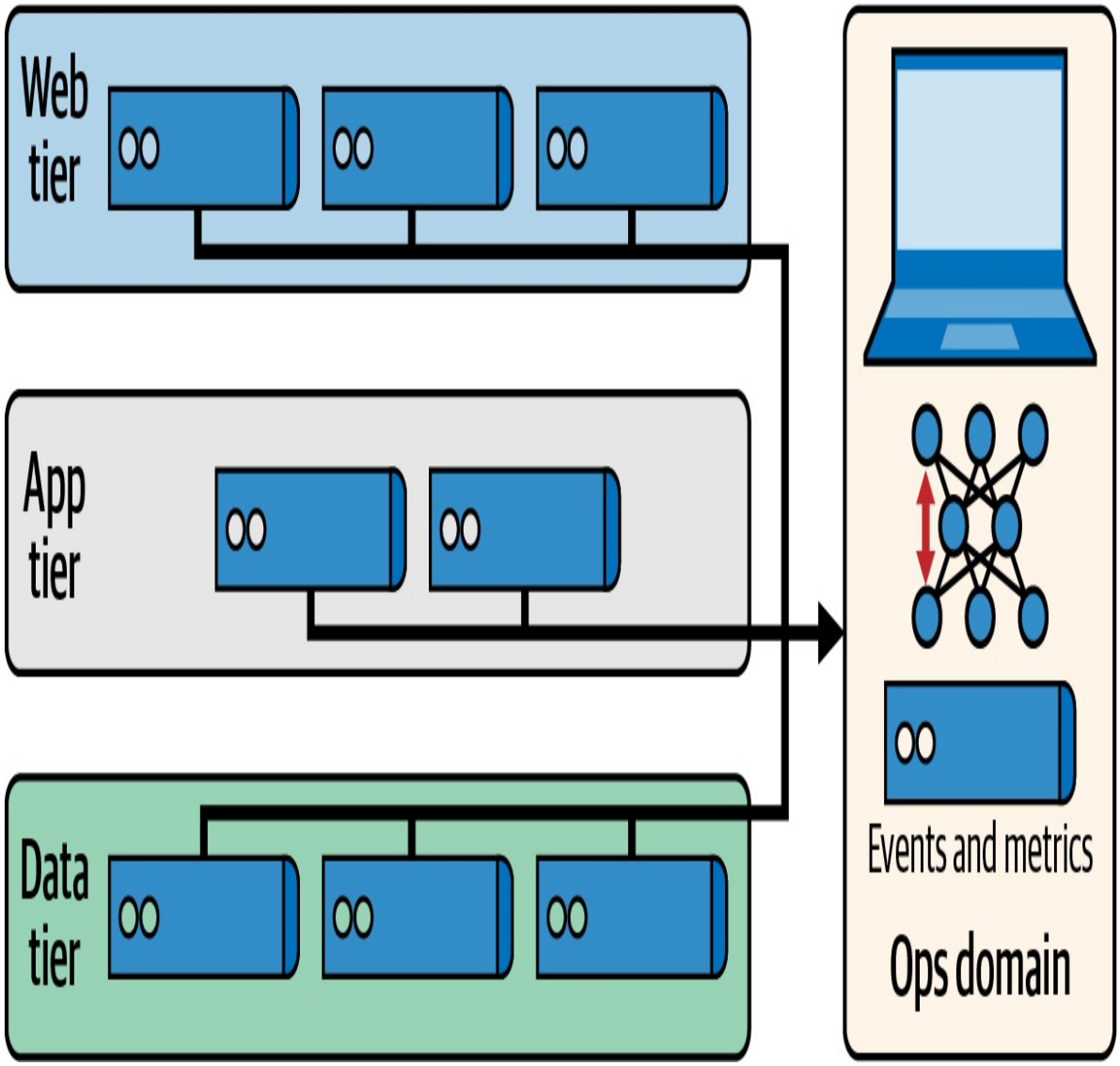

Figure 6-3. Creating the digital twin from data

But intrusions aren't the only use case for this kind of knowledge graph. Using the digital twin, a system operator can locate possibly faulty equipment or configurations by finding where paths intersect. This is useful because it ties together disparate users' error reports and enables them to be placed in context so they can be traced to a common fault, such as a flaky network switch or failing power supply. Furthermore, system architects can run what-if scenarios to see how failures will propagate through the network and use the digital twin to reduce the blast radius of any failures as well as any single

points of failure, especially the ones that are not obvious to human eyes. Security professionals can probe the model to see how an attacker might compromise it, with a view to creating a minimal spanning tree of a subgraph (the shortest or cheapest path encompassing all of the target graph) to determine the logical cost for an attacker to compromise it. Finally, we can also apply the analytics techniques presented in Chapter 4, so software can surface new or unusual patterns of interest to operators, giving the defenders an automated boost in the arms race against attackers.

Digital Twin for the Win!

The examples we've shown in this chapter come from just one corner of a modern enterprise. IT is often an easy candidate for digital twinning. The IT department has the expertise to implement a digital twin easily, and IT systems produce lots of metrics that are straightforward to gather. Moreover, the IT systems themselves are often complex and sophisticated enough for the return on investment in digital twins to be high.

But digital twins aren't just for IT. We've seen plenty of examples of digital twins being used across disparate business domains:

Wind turbine predictive maintenance

> By understanding the ebbs and flows of the power network, the seasonality and daily weather conditions, and the electrical and financial ripple effects of shutting down parts of the network (which may be continental scale), engineers are able to cost-effectively stop wind turbines for maintenance while keeping the grid balanced and ensuring minimal revenue losses.

Heating and cooling maintenance

> By understanding the mechanical heating and cooling systems for a building or campus and the prevailing weather conditions, engineers can select times for preventative maintenance with low impact on the facilities. Moreover, the digital twin can highlight opportunities for

preventative maintenance by aggregating signals from the network that might be suggestive of future failure that can be fixed more cheaply in the present.

Supply chain performance management

Businesses are complex entities with intricate, often international, supply chains. To understand how a typhoon in the Philippines affects the share price of a NASDAQ-listed technology company, analysts follow the network of how that company and its suppliers are interconnected. If a fabrication plant in typhoon-hit Manila is closed for weeks, then production of the final electronic consumer goods will be hit, and the lower volumes may lead to lower stock prices. Good businesses use their digital twins to not only find scenarios like this with poor outcomes but also evolve their systems and plans to try to avoid such impacts. A similar pattern can be used with power networks, incorporating long-distance transmission, renewables, and pricing.

Transport networks

Integrated transport planning has many variables: people, cars, trains, buses, the weather, delays, and more. Planners model the network as a digital twin to both respond to real-time operations and determine the best times to replace rails, dig up roads, or allow special events to occur. The digital twin is so useful in this case because common sense isn't sufficient. In such a large and diverse network, unexpected consequences can multiply and chaotically ripple through the system. A digital twin provides the tool for reasoning about and managing such complexity, especially when it is supported by graph algorithms and graph machine learning ahead of implementing any real world changes.

Digital twins aren't for everyone. If your business is small or the number of network permutations is manageable, then a digital twin might be overkill. But as businesses grow, the technique becomes increasingly valuable. If you often find yourself blindsided by change, that could be an indication that a

digital twin would help you.

---

1 We ignore the multi-data-center or multiregion deployment for clarity.

Chapter 7. The Way Forward

Data is one of a modern business' best assets. The irony is that as data volumes have grown, wrangling with—let alone capitalizing upon—data has become a struggle for many. Knowledge graphs are a tool that we can use to restore sanity to data by imposing an organizing principle to make data smarter. Through the organizing principle, businesses can reason about their data and bring together silos of disjointed information to form a consistent basis for business activities.

For each of us in business, there's a role to play. Business leaders must champion their data as one of their most strategic assets and boost efforts to increase its volume and usefulness, while data professionals carry the bulk of the workload in turning strategy into reality.

Governance and compliance teams will seek to understand the provenance and logistics of data through the enterprise and ensure that it is used compliantly. They will drive automation over the knowledge graph to achieve this goal. Analytics teams will find new ways to extract insights from data. They will become adept at building and using analytics tools and ML techniques to make the smart data in knowledge graphs work as hard as possible. Digital teams will create new business and revenue opportunities bottom-up by using knowledge graphs to support myriad higher-level services.

Throughout this book, we've shown how knowledge graphs help businesses harness connected data to drive business value. We've shown how this can be done without high-cost, high-risk, rip-and-replace technical projects. Instead, we've shown that knowledge graphs can be deployed as a nondisruptive technology sitting alongside existing systems while simultaneously enhancing their utility and value.

The construction of knowledge graphs is more bazaar than cathedral. You don't need grand designs to get started. In fact, we believe that a disciplined,

iterative approach is superior in many regards. So start small, and solve a single important problem first.

MORE RESOURCES FOR GETTING STARTED

- Neo4j's knowledge graph web page with whitepapers and case studies to help you understand how other enterprises have thrived with knowledge graphs
- Turing Institute's knowledge graph interest group facilitates research discussions with an emphasis on semantics and ontological reasoning
- Claudio Gutierrez and Juan Sequeda's article on the history of knowledge graphs
- Jésus Barrasa's guide to ontologies, inferencing, and integration
- A complete knowledge graph system, with source data, from the COVID-19 community's biomedical knowledge graph to help combat COVID-19

When you've shown value with your first project, move onto the next and be prepared to refactor some of the data. Knowledge graphs are flexible, so such refactorings are *not* to be feared—they are part and parcel of working with a living, valuable system. As you find you need more tooling (e.g., support for ontologies or ontological languages and tooling), bring it into scope but only as much as you need to solve the immediate problems.

Despite the years of academic research that underpin the approach, knowledge graphs are new to many of us. We recommend that you make best use of communities of practice, vendors, and subject matter experts to get off the ground. Even though you may not need it, your confidence will be bolstered in those critical early days.

About the Authors

Dr. Jesús Barrasa is a Director of Sales Engineering at Neo4j.

Amy E. Hodler is the Director for Graph Analytics and AI Programs at Neo4j and author of *Graph Algorithms* (O'Reilly, 2019).

Dr. Jim Webber is Chief Scientist at Neo4j and author of *Graph Databases* (O'Reilly, 2015).

Acknowledgments

It’s been a pleasure working on this material, and we thank all those who assisted us. We’re incredibly grateful to Dr. Maya Natarajan for her guidance and insights throughout the process, Gordon Campbell and Dr. Juan Sequeda for their invaluable feedback.