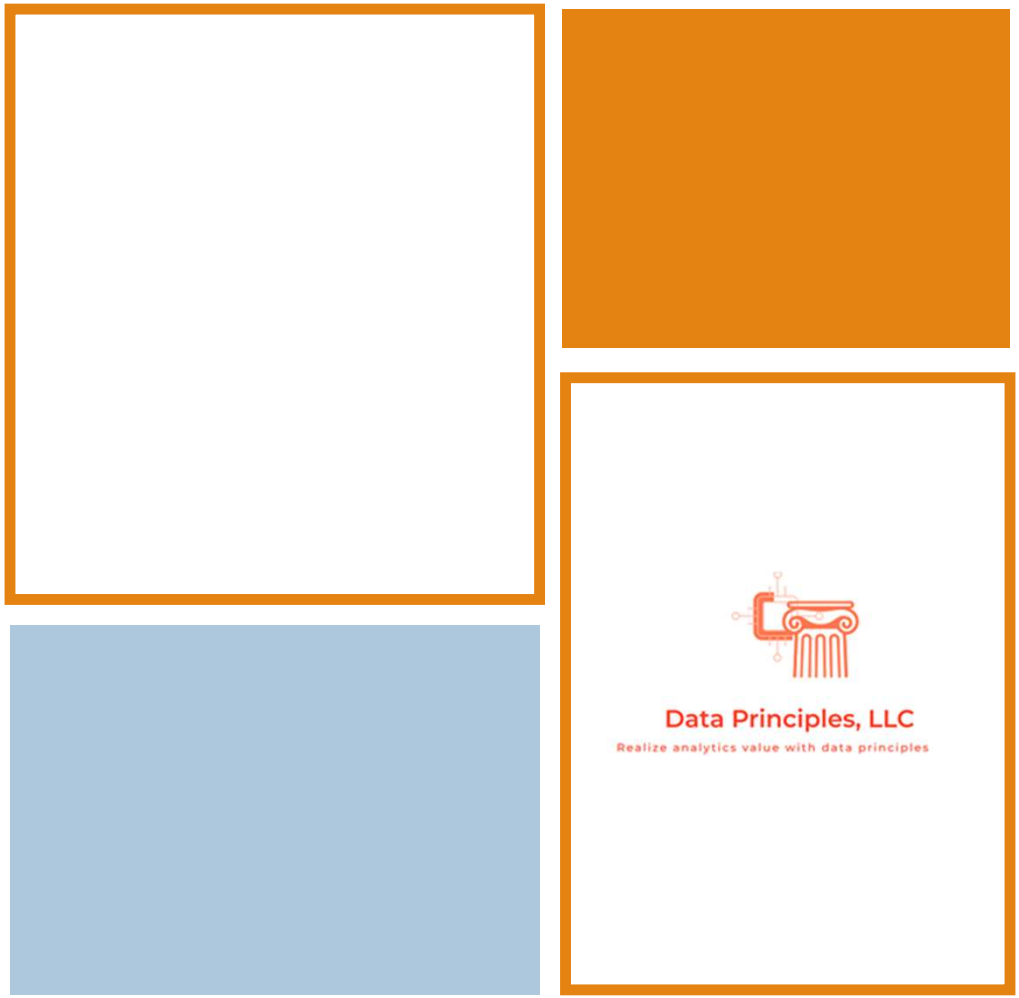


Data Principles, LLC

Realize analytics value with data principles

Specialized database types

Data Principles, LLC ™

Realize analytics value with data principles

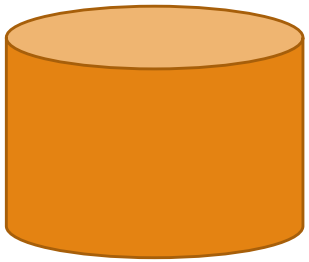
Data Principles is a data management and analytics consultancy, specializing in ensuring high-quality, timely, integrated data based on solid business and technical data models as the foundation for successful analytics. Data Principles is a Qlik partner.

About the Speaker

- Pete Stiglich - Founder, Data Principles, LLC
- Over 25 years of experience in Data Management, Data Architecture, and Analytics
- Holds CDMP (Certified Data Management Professional) and CBIP (Certified Business Intelligence Professional) certifications at the mastery level
- Past-President, VP of Programs for DAMA Phoenix
- Consultant, Trainer, Writer

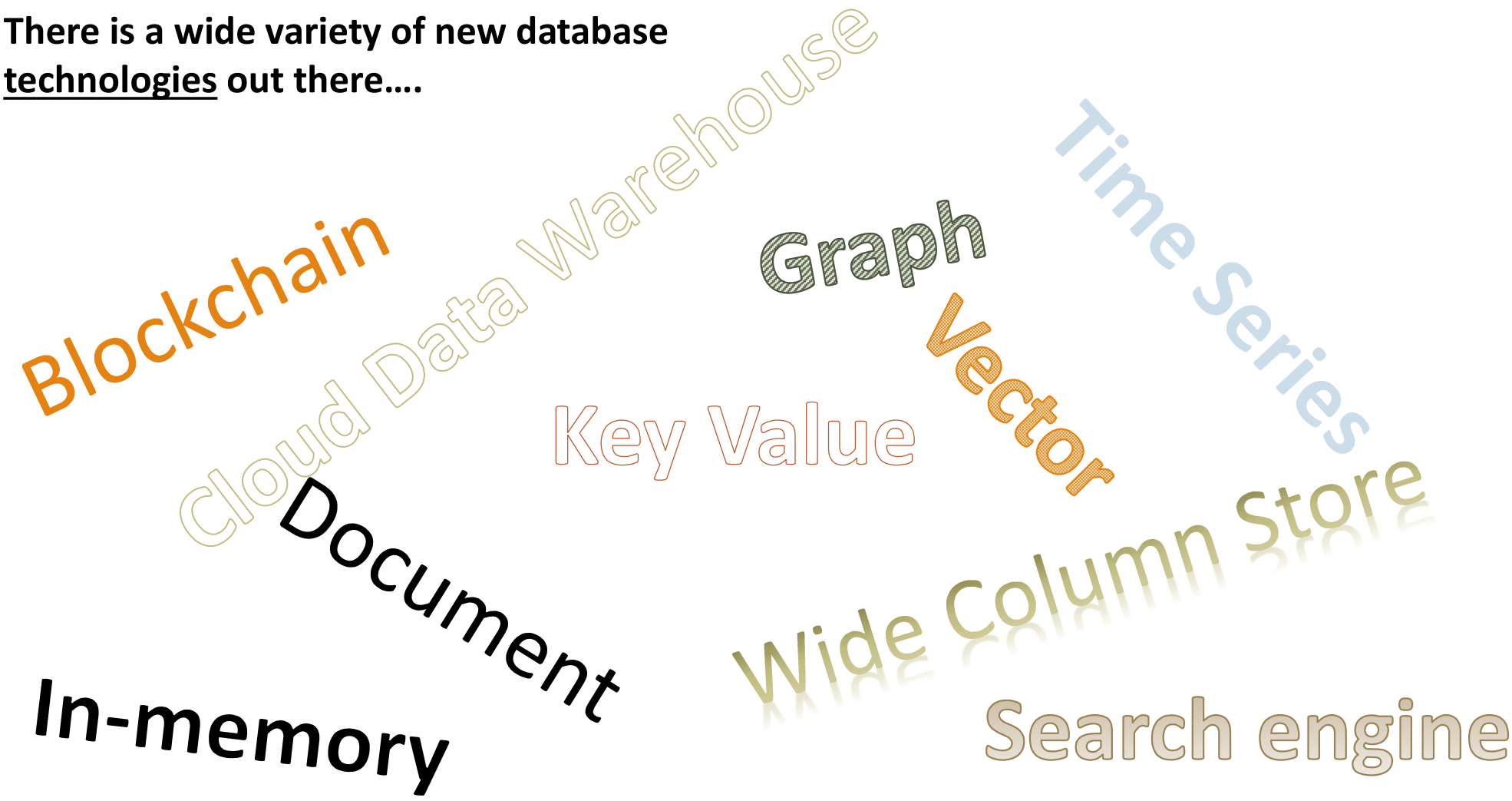
Pstiglich@data-principles.com
https://www.data-principles.com/
https://www.linkedin.com/in/petestiglich/

Specialized database types

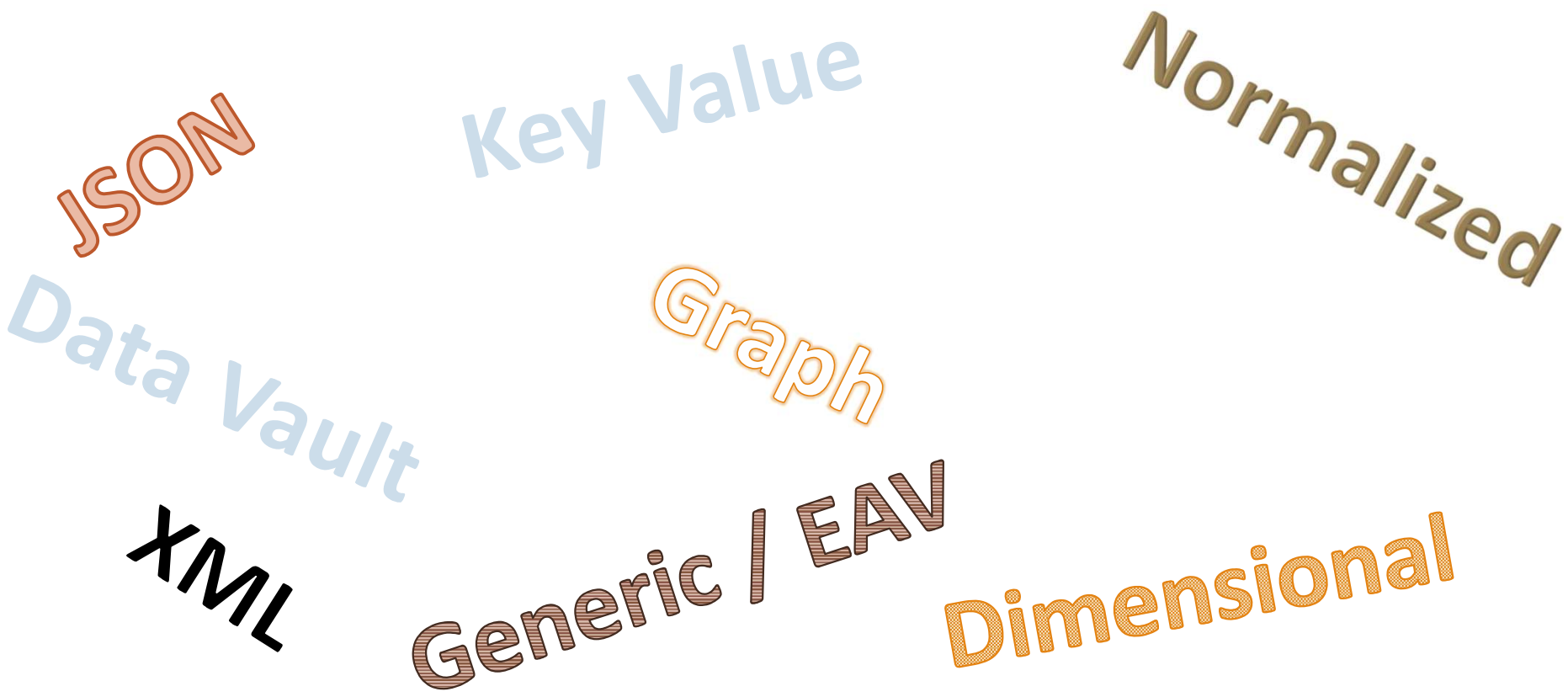


- New(er) types of database technologies (DBMS's) and model types
- Use cases for new(er) DBMS's
- Impacts of new DBMS's on data governance, quality, data modeling, federation, etc.
- This will be an interactive session – be prepared to share your use cases, success stories, and lessons learned!!

There is a wide variety of new database technologies out there….



And a wide variety of data model paradigms….



And there are a wide variety of requirement types (functional and non-functional)…

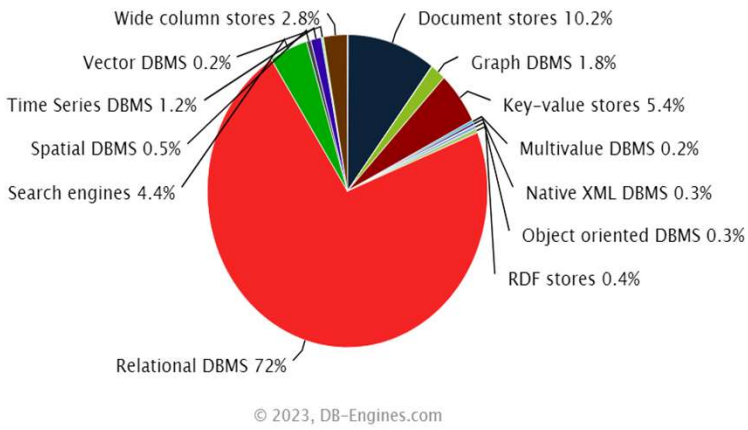
There's a lot of DB technologies out there..

420 systems in ranking, June 2023

Rank Jun 2023	Rank May 2023	Rank Jun 2022	DBMS	Database Model	Score Jun 2023	Score May 2023	Score Jun 2022
1.	1.	1.	Oracle	Relational, Multi-model	1231.48	-1.16	-56.27
2.	2.	2.	MySQL	Relational, Multi-model	1163.94	-8.52	-25.27
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	930.06	+9.97	-3.76
4.	4.	4.	PostgreSQL	Relational, Multi-model	612.82	-5.08	-8.02
5.	5.	5.	MongoDB	Document, Multi-model	425.36	-11.25	-55.36
6.	6.	6.	Redis	Key-value, Multi-model	167.35	-0.78	-7.96
7.	7.	7.	IBM Db2	Relational, Multi-model	144.89	+1.87	-14.30
8.	8.	8.	Elasticsearch	Search engine, Multi-model	143.75	+2.11	-12.25
9.	↑ 10.	9.	Microsoft Access	Relational	134.45	+3.28	-7.36
10.	↓ 9.	10.	SQLite	Relational	131.21	-2.65	-4.22
11.	11.	↑ 13.	Snowflake	Relational	114.13	+2.41	+17.71
12.	12.	↓ 11.	Cassandra	Wide column	108.55	-2.58	-6.90
13.	13.	↓ 12.	MariaDB	Relational, Multi-model	97.31	+0.44	-14.27
14.	14.	14.	Splunk	Search engine	89.45	+2.81	-6.11
15.	15.	↑ 16.	Amazon DynamoDB	Multi-model	79.90	-1.20	-3.98
16.	16.	↓ 15.	Microsoft Azure SQL Database	Relational, Multi-model	78.96	-0.23	-7.05
17.	17.	17.	Hive	Relational	75.52	+1.91	-6.06

https://db-engines.com/en/ranking

Ranking scores per category in percent, July 2023



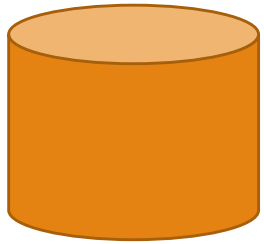
Over 400 database technologies!!

Picking the right DBMS technology

- Can be a very difficult task – but usually the best approach is to look at core functionality requirements first and then consider the non-functional requirements

- Unless there is a significant business case not to - usually a Relational DBMS (RDBMS) is the default choice given its maturity, capabilities for many use cases, ease of finding experienced resources / reduced learning curve

- Of course, there are some use cases that RDBMS's don't handle well, e.g., social media, unstructured data, messaging, data lake / massive volumes



Picking the right DBMS technology

Processing Types		Data Representation								
		RDBMS	Multidimensional	Key-Value	Wide Column	Graph	Document	Object	XML	In-Memory DB (relational or other)
Operational	ACID Compliant Transaction Processing	X						X		X
	Eventual Consistency Transaction Processing				X		X			
	Streaming / Complex Event Processing			X						X
	Messaging / SOA								X	
	Web Operations	X		X	X	X	X	X	X	X
	Content Management	X					X		X	
	Social Media			X	X	X				
	Geospatial					X				
Analytical	Reporting	X								
	OLAP	X	X							
	Unstructured			X						
	Data Mining	X		X						
	Near Real Time	X		X						X
	Machine Learning					X				
	Federation	X				X				

https://blogs.perficient.com/2013/02/04/perficients-big-data-stack-application-tier/

ACID vs BASE

- Not every application needs to be ACID compliant e.g., social media post vs. accounting
- Data volumes, unstructured data, and greater flexibility in scaling, cost, design, processing, DR gave rise to BASE

Atomicity – All operations in the transaction performed or none of them

Consistency – On completion of a transaction, the database is structurally sound

Isolation – Each transaction is independent unto itself

Durability – Once complete, the transaction cannot be undone

Basically Available – some level of availability to the data even when there are node failures

Soft State - Data is in constant state of flux; data not guaranteed to be current

Eventual Consistency – The data will eventually be consistent through all nodes, but not every transaction will be consistent at every moment

From DAMA DMBOK (partially)

ACID vs BASE

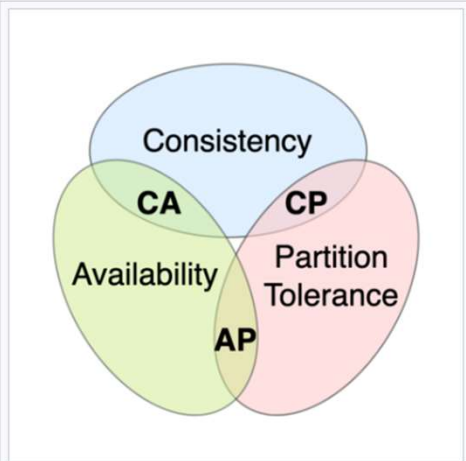
Item	ACID	BASE
Casting (data structure)	Schema must exist	Dynamic
	Table structure exists	Adjust on the fly
	Columns data typed	Store dissimilar data
Consistency	Strong Consistency Available	Strong, Eventual, or None
Processing Focus	Transactional	Key-value stores
Processing Focus	Row/Column	Wide-column stores
History	1970s application storage	2000s unstructured storage
Scaling	Product Dependent	Automatically spreads data across commodity servers
Origin	Mixture	Open-source
Transaction	Yes	Possible

DAMA DMBOK

CAP Theorem

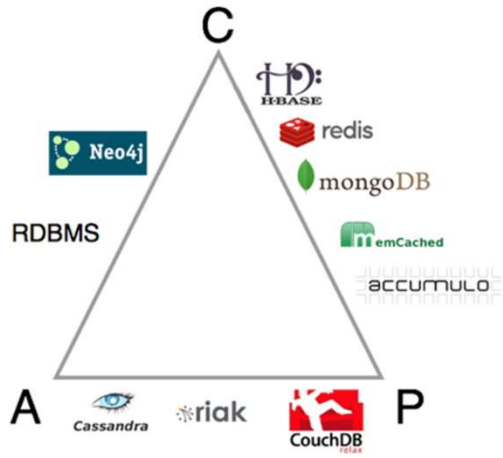
States that any distributed database can provide only 2 of the following guarantees

- **Consistency** - Every read receives the most recent write
- **Availability** – Every request receives a non-error response, without the guarantee that it contains the most recent write
- **Partition tolerance** – System continues to operate even with network issues between nodes



CAP Theorem Venn Diagram

Wikipedia



https://polakowo.io/datadocs/docs/big-data/database-design

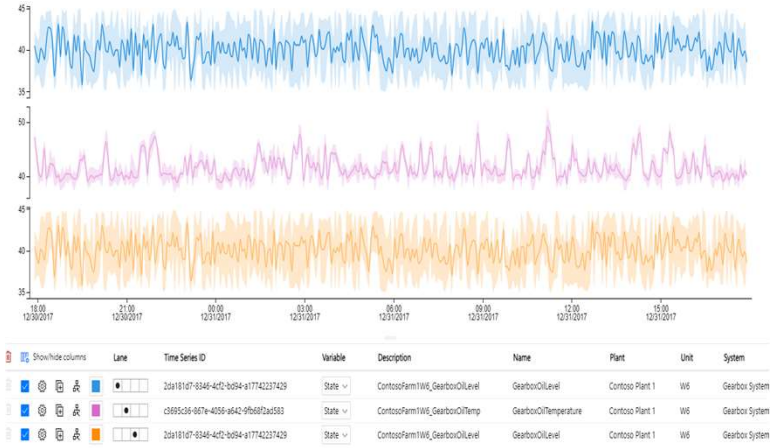
New Database Types

Time Series Database (TSDB)

- Time series data is just a set of datapoints that are associated with timestamps - time series data can be stored in RDBMS, NoSQL, or specialized time series databases (TSDB). Time series data usually immutable.
- Can have very huge volume & velocity e.g., intraday stock ticker, IoT, heart rate monitor
- Reasons for a TSDB – high performance (e.g., millisecond query times over months of data), fast querying of time windows, ease of eviction / summarization of old data, compression.
- May use both in-memory and disk storage, hot and cold storage
- Vendors/tools: InfluxDB, eXtremeDB, kdb+, AWS Timestream, Azure Data Explorer, Timescale, Riak TS, etc
- Query language(s): InfluxQL, PromQL, OpenTSDB Query Language

Time Series Database (TSDB)

- Data model for a TSDB varies by vendor but typically contains a measurement name, timestamp, measurement value, and tags
- Typically has limited dimensionality / joins to maintain query performance.
- Usually stored in time-ordered sequence
- Measurements can be metrics (regular interval) or events (irregular interval)



Insights.timeseries.azure.com/samples

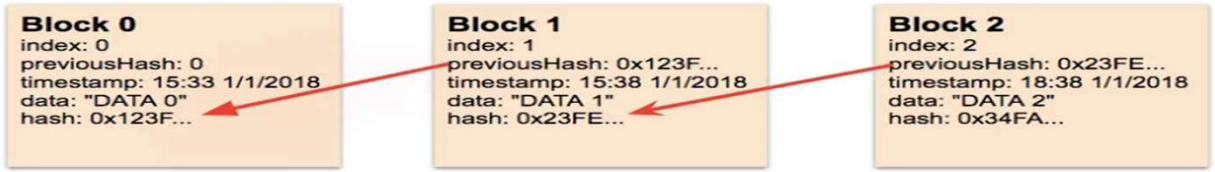
Time Series Database (TSDB)

- Is anyone here using a TSDB? If not, what are you using for storing / analyzing time series data?
- What are some of the issues / advantages of your current solution?
- What ancillary software used e.g., Kafka, Kinesis, Grafana?

Blockchain

- "A distributed database or ledger that is distributed among the nodes of a peer-to-peer network" Investopedia.com
- Decentralized – no single owner / authority
- Multiple copies of a blockchain are stored on many machines (nodes) and must match for it to be valid
- A "block" contains an immutable* transaction "appends only" – data format for the transaction can be JSON, XML, CSV, XLS, JPEG, etc.
- Ordered back-linked – a block points to the previous block to form a chain. *After 6 back links typically considered immutable.

Block Height 0 / genesis block

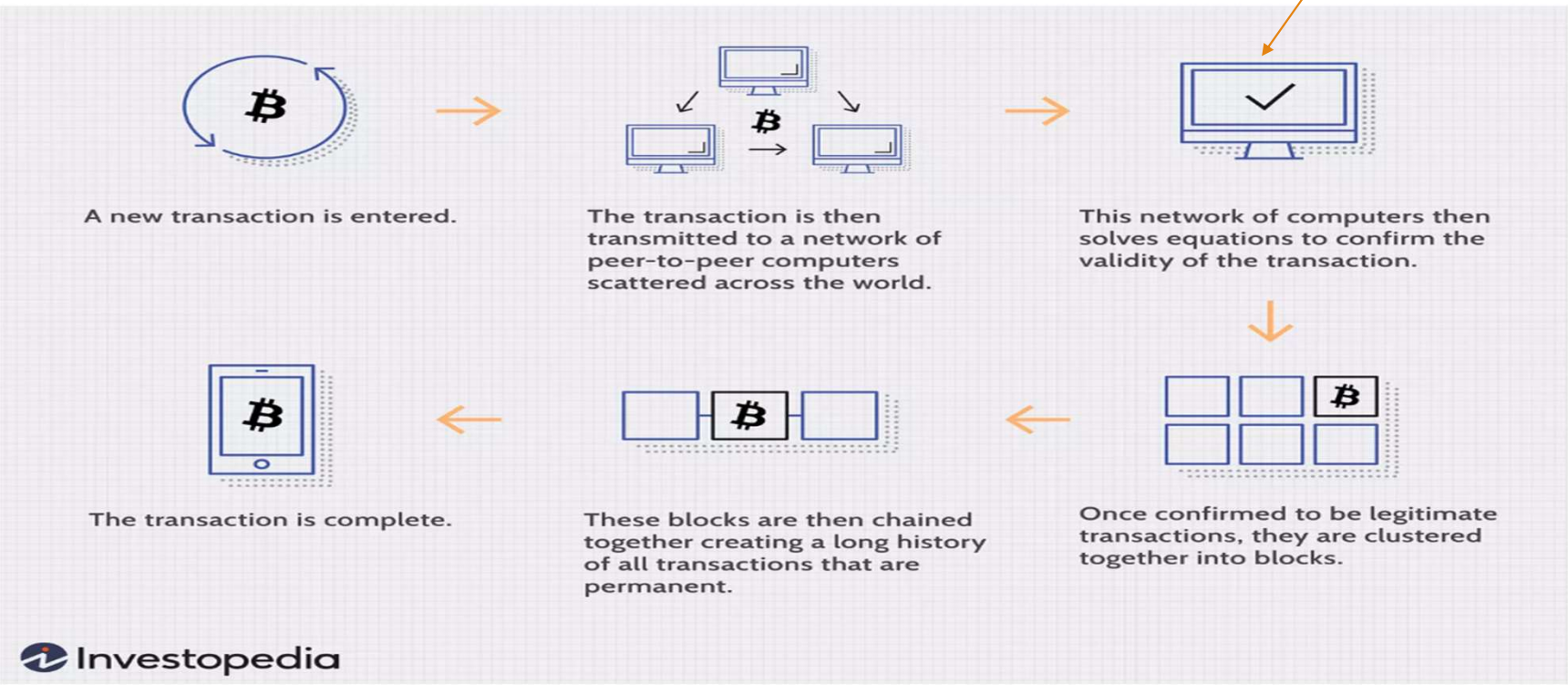


Medium.com

Blockchain

"Proof of work allows for secure peer-to-peer transaction processing without needing a trusted third party" Investopedia.com

Aka "mining" - requires a lot of computational resources



Blockchain

- A block is made up of a header with metadata, followed by 1 or more transactions
- Merkle Tree is a binary hash tree (similar to a b-tree index).
- "Merkle trees produce an overall digital fingerprint of the entire set of transactions. A Merkle tree is built by recursively hashing pairs of nodes until there is only one hash, called the root, or Merkle root"
  - Used to identify if a transaction is in a block
- The Merkle root hash is not the same as a block hash

Size	Field	Description
4 bytes	Version	A version number to track software/protocol upgrades
32 bytes	Previous Block Hash	A reference to the hash of the previous (parent) block in the chain
32 bytes	Merkle Root	A hash of the root of the Merkle-Tree of this block's transactions
4 bytes	Timestamp	The approximate creation time of this block (seconds from Unix Epoch)
4 bytes	Difficulty Target	The Proof-of-Work algorithm difficulty target for this block
4 bytes	Nonce	A counter used for the Proof-of-Work algorithm

Figure 1: Data structure of a block (Antonopoulos, 2014)

https://www.linkedin.com/pulse/blockchain-data-structure-ronald-chan/

Blockchain

- "Generating random hashes until a specific value is found is the "proof-of-work" you hear so much about – it "proves" the miner did the work. The amount of work it takes to validate the hash is why the Bitcoin network consumes so much computational power and energy"

https://www.investopedia.com/terms/b/blockchain.asp#toc-pros-and-cons-of-blockchain

- Bitcoin uses a SHA-256 cryptographic hashing algorithm as the input value can not be calculated from the hash

- "Miners employ a trial-and-error approach, in which every calculation takes a new nonce value. Blockchains employ this method as the probability of guessing a valid nonce is close to zero. Therefore, miners must possess advanced computing systems to test and discard millions of different nonce possibilities to calculate a valid block hash. The nonce that results in a valid block hash is known as a golden nonce." https://www.cnbctv18.com/cryptocurrency/blockchain-security-the-heart-of-nonce-explained-14773161.htm

Blockchain

- Use Cases – Crypto currency, banking, Non-Fungible Tokens (NFT), Supply Chain (e.g., IBM Food Trust – track the journey that food products take), healthcare (e.g., medical records - can always be available), smart contracts, long term storage of personal data (ArWeave, FileCoin).

- Why use blockchain for long term storage of personal data?
  - Will always be available due to the decentralized nature of a blockchain (no single point of failure)
  - Encrypted
  - Very difficult to tamper with

Blockchain

- For cryptocurrencies, money can be exchanged without a centralized clearing and settlement process & authority.
- Not having a central authority and faster money transfer a key driver for cryptocurrencies.

Paypal	~193/sec
ACH	3 days per transaction
Wire	A few hours - a few days per transaction
Bitcoin	~4/sec
Ethereum	~20/sec

https://medium.com/rebloc/not-all-transactions-are-equal-7a6f91286d69

Blockchain

- Is anyone here working with Blockchain? Thoughts?
- Is anyone here using Blockchain internal to their enterprise?
- What could be some possible enterprise use cases?

Graph database

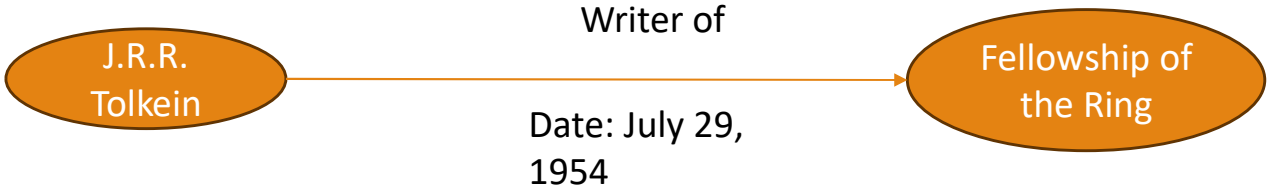
- For complex modeling and analysis of relationships where the relationship (edge) is as important as the data instance (node).
- Common use cases - social networks, network mapping, supply chain mapping, fraud detection, semantic search
- Two types of graph databases – Property Graph and Knowledge/RDF graph
- Generative AI (e.g., Chat GPT) usually just parses a bunch of text from the internet for its knowledge base (stored in a vector database) and so responses can't always be trusted. With graph databases (especially knowledge graphs) you can establish more trust in computer reasoning

Graph database

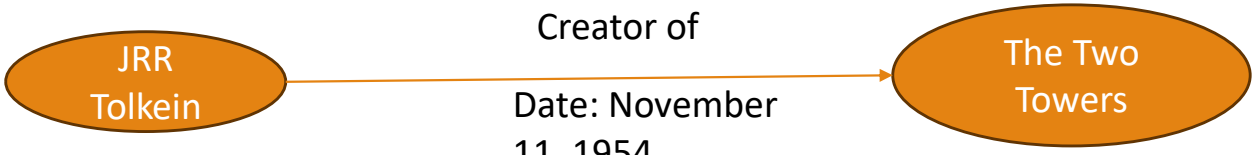
- Property graph – simpler form of a graph, easier to get started but with limitations
  - Doesn't have a schema - relationships can be named anything but with knowledge graph the relationship (edge) is richly described and can be reasoned on by a computer
  - Edges can have their own attributes
  - Good support for analytics
  - Limited inferencing – limited to what people can infer – computers can't "reason" in the way they can with a knowledge graph

Property graph database

DB1



DB2



How would the **computer** know J.R.R. Tolkein and JRR Tolkein are the same person?

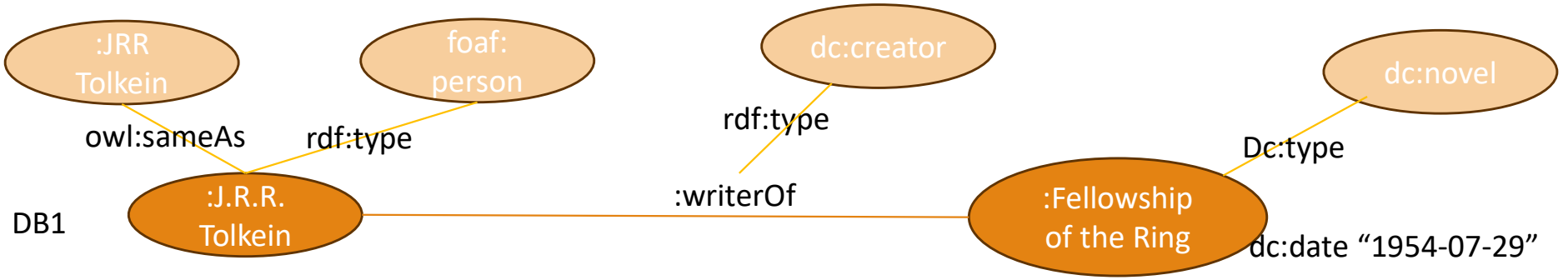
How would **computer** know that "Writer of" and "Creator of" mean the same thing?

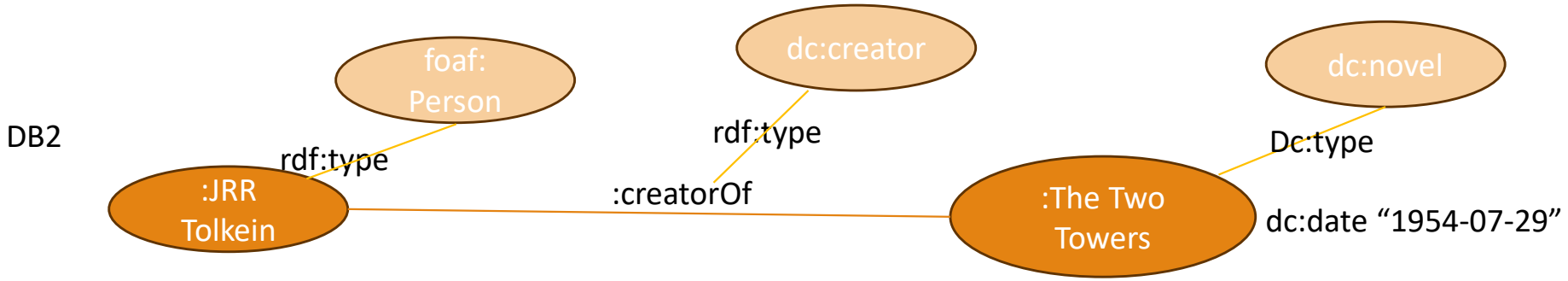
How would the **computer** know which books JRR Tolkein or J.R.R. Tolkein wrote?

Knowledge Graph database

- Knowledge graph (aka RDF triplestore) – provides rich semantics, modeling of ontologies i.e., "enterprise knowledge"
  - Uses globally unique URI's – everything (almost) is a resource and has a unique URI (or IRI)
  - Easier to tie in publicly available ontologies to avoid reinventing the wheel e.g., use Dublin Core to describe books, articles, etc., with industry standard elements such as creator, subject, publisher, format
  - Truly web-scale – RDF/OWL used for Semantic Web / federation
  - Can draw richer inferences in the data. Provenance of inferences (which become triples) can be provided.
  - Data expressed as triples (subject, predicate, object)
  - Often used for complex master data

Knowledge graph database





Knowledge graph database

Examples of new data (triples) that can be inferred by a reasoner (e.g., Jena, Pellet, OWLIM, BOSSAM) in RED

J.R.R. Tolkein writer of “The Fellowship of the Ring”

J.R.R. Tolkein creator of “The Fellowship of the Ring”

J.R.R Tolkein is the same person as JRR Tolkein

J.R.R. Tolkein creator of “The Two Towers”

JRR Tolkein creator of “The Two Towers”

JRR Tolkein is the same person as J.R.R. Tolkein (using inverse of owl:sameAs)

JRR Tolkein creator of “The Fellowship of the Ring”

Graph database

- Graph database capabilities are being introduced to some of the RDBMS's (e.g., Oracle, SQL Server)
- Many graph vendors (examples below) – typically divided by whether they support property graphs or knowledge/RDF graphs (some support both)

Property Graph	***BOTH***	***Knowledge Graph***
Neo4j	**Oracle**	**GraphDB**
TigerGraph	**Amazon Neptune**	**SQL Knowledge Graph**
Azure CosmosDB		**StarDog**
		AllegroGraph

- Query language(s): SparQL, Cypher, Gremlin, PGQL, GQL

Graph database

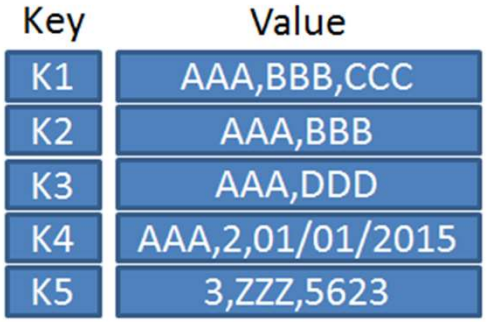
- Is anyone here working with graph databases? Property or Knowledge Graph?
- What are your use cases?
- What issues faced?

Key Value Database

- Very flexible schema-less database often used for real-time applications where the data can be easily partitioned e.g., session data
- Often used for caching (e.g., for real-time analytics), gaming, web or mobile apps
- Data is stored and queried by a unique key rather than by searching on the value which increases performance – but limits queries
- Horizontally scalable – can handle huge amount of data & numerous concurrent users

- Not as suitable for complex queries or relationships
- Limited ACID support to ensure consistency
- No built-in data validation

- Vendors/tools: Redis, Riak, Memcached, Amazon DynamoDB, Azure Cosmos DB, etc.
- Query Language(s): None, API based (could use a 3rd party ODBC/JDBC driver)



Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

Wikipedia

Key Value Database

- Is anyone here working with Key Value Databases as a data architect?
- How do you govern Key Value Databases?
- What issues faced?

Document Database

- Stores data as a document (using JSON, BSON, XML, YAML) rather than in tables
- Can include nested data structures making it ideal for messaging as an entire transaction (and perhaps its master data) can be encapsulated in a single document
- Can simplify application development as the document can become an object
- Can encode binary data as Base 64 string so images, video, etc., can be included – but for large images, etc., usually best to store separately and add a link in the doc.
- Document databases can persist the documents for long term storage and query beyond transitory messaging
  - Storing the documents long term can enable future use cases, analytics
- Some document db's e.g., MongoDB have some data quality features e.g., unique indexes, multi-document ACID transactions
- Vendors/tools: MongoDB, Amazon DynamoDB, Couchbase, MarkLogic, Azure CosmosDB, etc.
- Query Language(s): MQL, SQL, Xquery, etc.

Document Database

- Structure of the document type can vary from one document to another
  - Best to use JSON Schema or XSD's to define a model for documents to help ensure data quality
    - Can enforce datatypes, formats, data lengths, mandatory fields, etc.
  - Messages / docs should be validated against the schema (a separate step but a simple programming call) before being transmitted
  - Be sure to use schema version numbers in the schema and docs to know which schema a doc is aligned with
  - Use a tool such as Hackolade or XMLSpy to model JSON Schemas or XSD's

Document Database

- Who here is using a document database?
- How often do you query document databases? What do you use to query? Are you querying document databases for analytics?
- Are you leveraging JSON Schema or XSD's to model documents? If so, do your developers validate documents against the model?
- What issues faced?

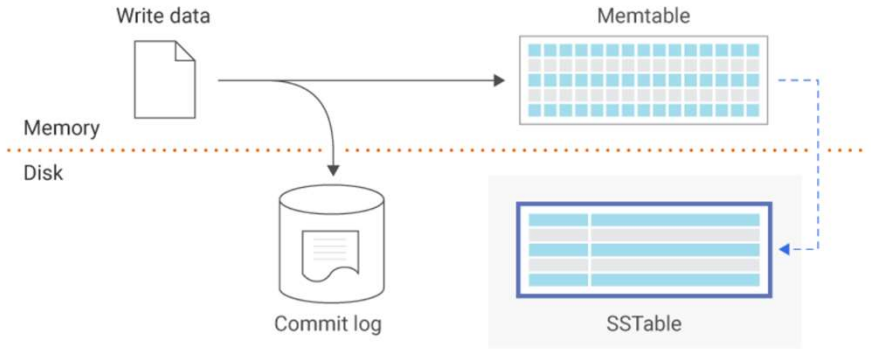
Wide Column (or Column Family) DB

- Used for large volume and high velocity use cases, designed for horizontal scalability. May have many nodes, multiple masters (e.g., Cassandra) across many data centers
- Use cases: Data warehousing, real-time analytics, IOT, highly variable data
- Data organized using column families (similar to tables). Allows for a variable number of columns which don't have to be pre-defined.
  - E.g., 1 row can have 10 columns the next could have 1,000.
  - Usually want to have all the data needed for a single query stored in a single column family for read performance. Joins not supported (e.g., Hbase, Cassandra).
  - A column cell contains the column name, the value, and a timestamp (when inserted – updates actually result in inserting a new column cell for versioning or "tombstoning")
- Can support random-access as opposed to Hadoop / batch / sequential reads (e.g., for Hbase which uses Hadoop HDFS)

Wide Column (or Column Family) DB

- The data is actually stored in persistent, ordered, and immutable SSTable (Sorted String Table) files. For the same record, with multiple column cell versions – old versions can be cleaned up by compaction which writes a new SSTable
- Updates to data creates a new SSTable file
- Each SSTable has an index file and a data file

- Vendors / Tools: Apache Hbase, Apache/DataStax/S CosmosDB, etc.
- Query Language(s): CQL, SQL, API



https://www.scylladb.com/glossary/sstable/

Vector Database

- **Database that stores unstructured data in vector embeddings in order to measure distance between things (e.g., words)**
- **"Optimized for efficient storage, indexing, and querying of high-dimensional vector data" db-engines.com**
- **Use cases: AI (e.g., ChatGPT), NLP, recommendation engines, question answering applications**
- **Vendors/tools: Kdb, Chroma, Pinecone, etc.**
- **Query tool(s): API**

Source text

```
(1) John likes to watch movies. Mary likes movies too.
(2) Mary also likes to watch football games.
```

This would be our vocabulary (removing stopwords like "to"):

```
V = {John, likes, watch, movies, Mary, too, also, football, games}
```

And the original sentences could be converted to these vectors by counting the number of words each sentence has in the vocabulary:

Source text as vectors

```
(1) [1, 2, 1, 2, 1, 1, 0, 0, 0]
(2) [0, 1, 1, 0, 1, 0, 1, 1, 1]
```

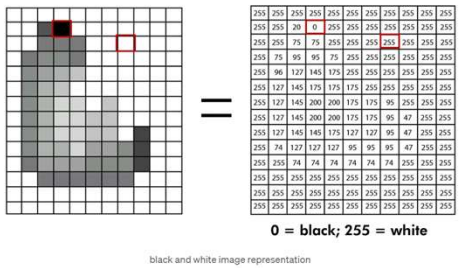
The idea behind BOW models is that the weight of words in the text is valuable to represent the text meaning, even though order and syntax are lost.

https://www.baeldung.com/cs/text-sequence-to-vector

Many different ways to create vectors for text e.g., bag of words (BOW), bag of n-grams, term frequency

Vector Database

- **Is anyone here using Vector databases?**
- **What are your use cases?**
- **What are some possible use cases in your enterprise?**



black and white image representation

https://rom1504.medium.com/image-embeddings-ed1b194d113e

https://www.imdb.com/title/tt0080339/characters/nm0039149

Impacts of new database types

Data Governance

- Some of the new database types can present a challenge for data governance, and integration with data catalog tools (e.g., graph databases)
- Ideal is to be able to know where all your data is, what it means, how it's being used, etc., but some of the databases (e.g., key value pairs) are more geared towards developers (i.e., intermediary in nature)
- Might focus instead on key data sources and your data warehouse / data lake for governance efforts i.e., business user facing
- Of course, Data Governance should assess the security, reliability, backup capabilities of the database technology and understand what data is being stored there and why (without necessarily having detailed metadata)

Data Architecture

- **Data architects are still needed in order to:**
  - Translate complex (and often vague/contradictory) business requirements into a data architecture (regardless of how data will be stored)
  - Need to take existing technical environment/infrastructure into account, consider potential data and technical integration problems…
  - Data architects need to thoroughly understand the implications of using any specific database technology
  - Data architects should not be bullied by developers *(who might be looking to pad their resume's)* into using a certain technology, but of course should be open minded.
  - Data architects need to align the data architecture with data governance

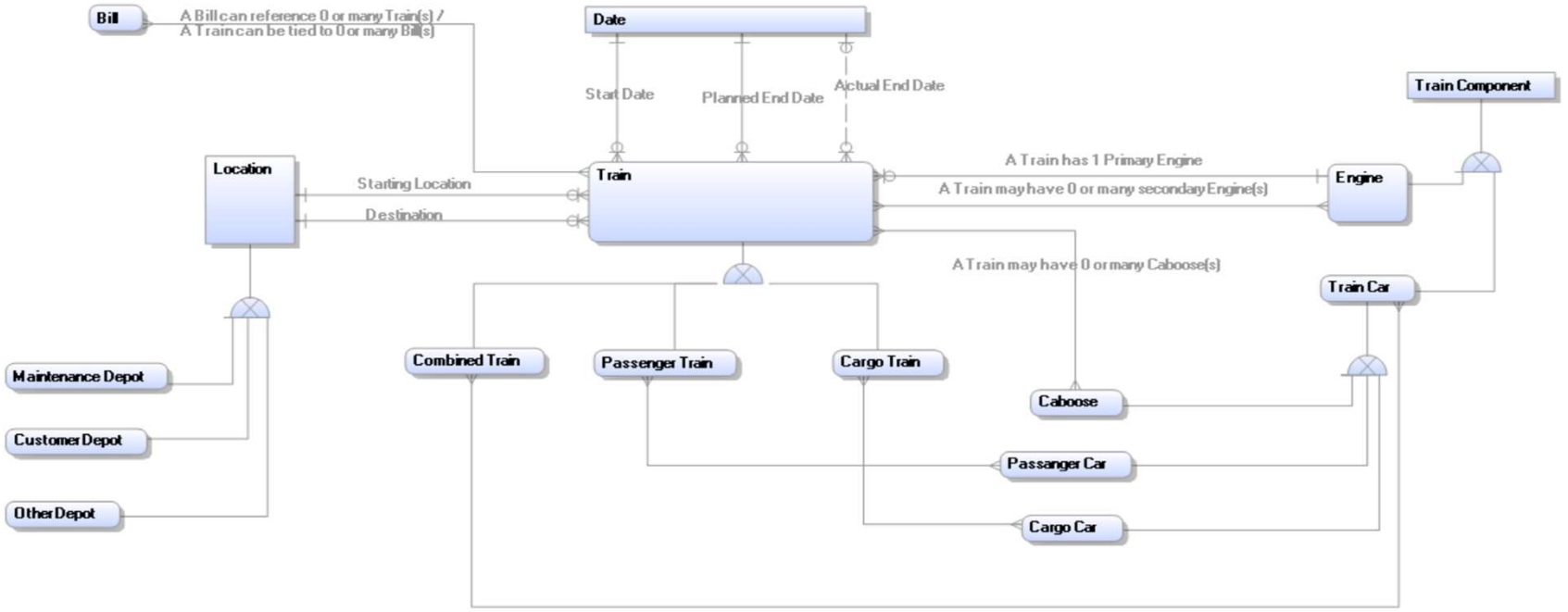
Data Modeling

- The Physical Data Model for many of these databases is radically different from relational models
- Some might think data architects and/or data models no longer needed
- Still need data architects to translate business requirements into a data design, still need models to facilitate communications with the business, between IT teams
- Conceptual Data Modeling still as important as ever. Logical Data Modeling may still be useful if relational constructs aren't enforced.
  - E.g., may just attributize the conceptual model rather than resolving many to many relationships, resolving subtyping

Conceptual Data Model

- AKA ERD (Entity Relationship Diagram)
- Identifies the key business entities/objects (customer, product, sale, store…) and their relationships (and relationship cardinality!)
- Mis-understanding a key relationship can have dramatic impacts – on data quality, application design, analytics, and data science!!!
- Is a key tool for defining scope and ensuring alignment on terminology and business rules
- Is completely non-technical – a businessperson should recognize their business in the model
- Downstream models / databases based on the CDM will be much more aligned with business and will result in better run projects e.g., less re-work, less quality issues, greater integration…

Conceptual Data Model example



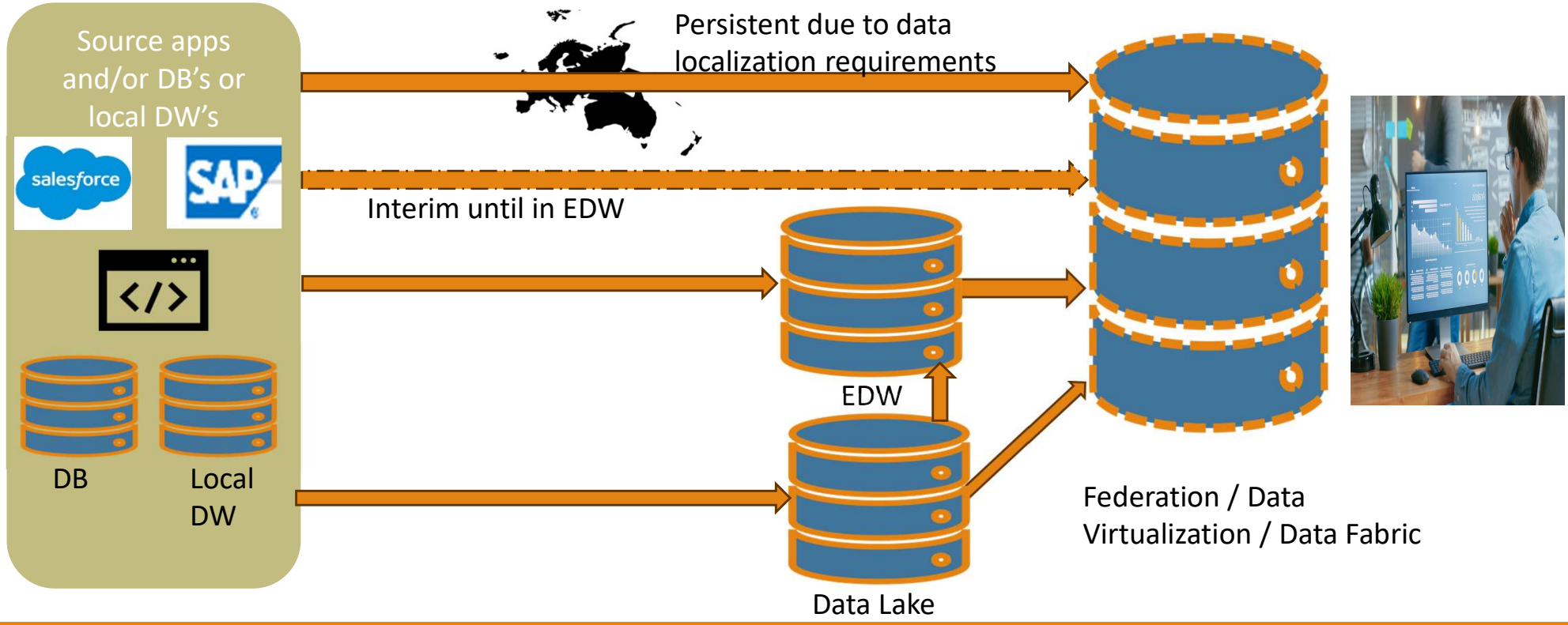
Data Quality

- Nice to have a “separation of concern” where data architects identify/implement data quality constraints (e.g., foreign keys, unique indexes, referential integrity, value constraints) separate from the app code.
- With many of the new database types – data quality is mostly left up to the programmers as the constraints that can be enforced by newer database technologies are often limited / non-existent.
- Data architects should still be involved with capturing detailed data requirements including data quality rules
  - Time should be added to the project plan so developers can code the data quality checks

Data Federation

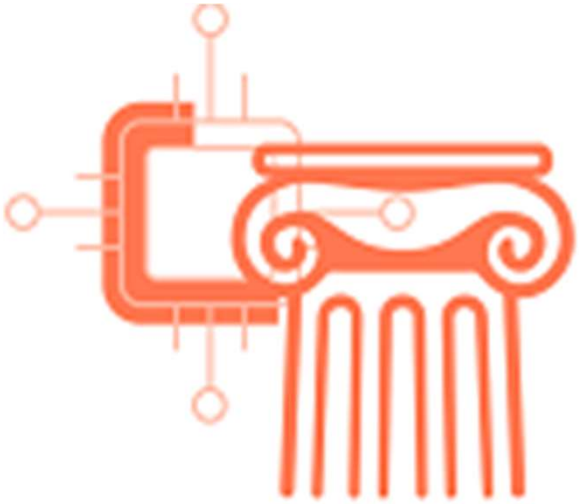
- With data spread out across multiple clouds, multiple databases, multiple database technologies with varying data localization requirements – using data federation technologies can make sense
- Leaving the data in place vs centralizing – i.e., virtualizing the data and bringing it together at query time (possibly leveraging some caching)
- However, an EDW still makes sense as need a place to store historical data - operational systems often only store limited history
- For difficult integration, standardization, cleansing, and enrichment processes can make sense to do that in an EDW – but takes time to integrate data into an EDW
- Federation can provide more immediate access – while longer term integration into an EDW takes place

Data Federation



Specialized database types - conclusion

- These are exciting times… many interesting types of DB technologies to choose from
- Many different paradigms for how data can be stored and managed – with different types of models (graph, key value pair, document, etc.)
- Data architects need to really understand the paradigms, benefits/drawbacks of using a specific DBMS
- Must have solid use cases identified for leveraging a new DBMS platform
- Must understand impacts on functional and non-functional requirements, especially for data governance and data quality



Contact us to continue the conversation!!

Pstiglich@data-principles.com
https://www.data-principles.com/
https://www.linkedin.com/in/petestiglich/

Data Principles, LLC

Realize analytics value with data principles