FAKULTA ELEKTROTECHNICKÁ

České vysoké učení technické v Praze

B4M36DS2 – Database Systems 2

Lecture 5 – Types of NoSQL Databases

20. 10. 2025

Yuliia Prokop

prokoyul@fel.cvut.cz

Based on **Martin Svoboda**'s materials (https://www.ksi.mff.cuni.cz/~svoboda/courses/211-B4M36DS2/)

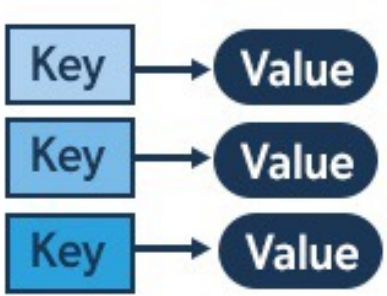
https://cw.fel.cvut.cz/b241/courses/b4m36ds2/start

Lecture Outline

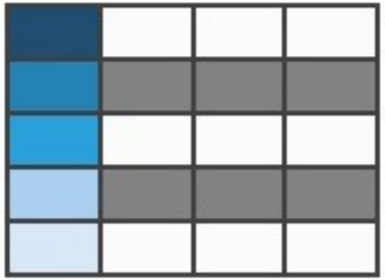
- ✓ Types of data stores
  - Key-value
  - Document
  - Wide column
  - Graph
- ✓ Non-core types
  - Native **XML** databases
  - **RDF** stores…
- ✓ Polyglot Persistence

Types of NoSQL Databases

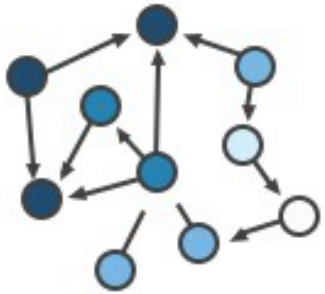
Key-Value



Column-Family



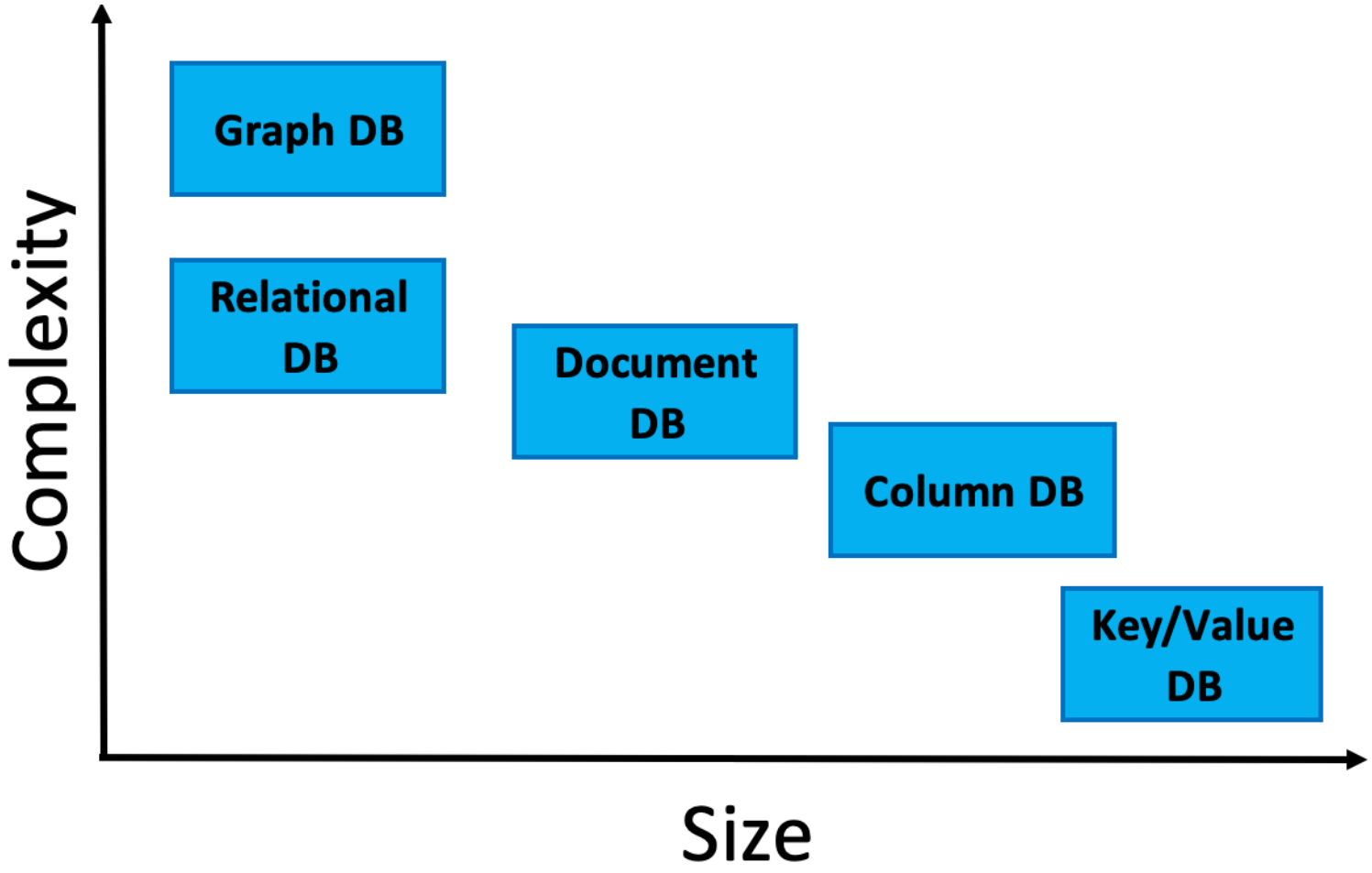
Graph



Document

Source: https://www.geeksforgeeks.org/types-of-nosql-databases/

Size / Complexity of data stores (subjective)

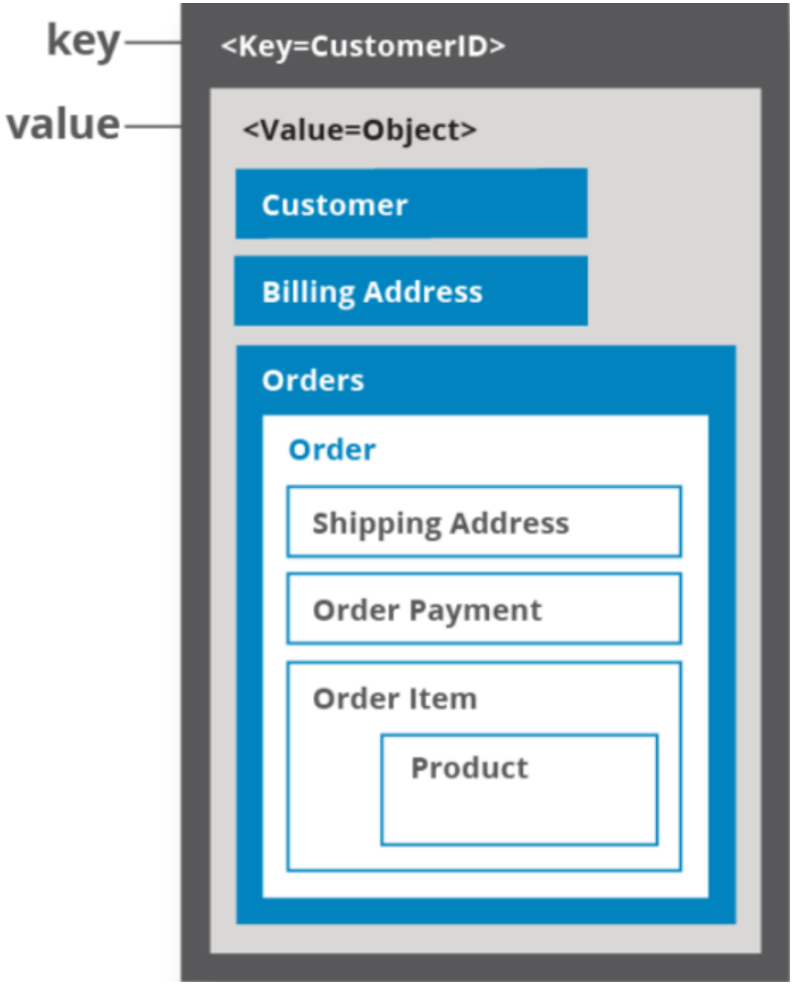


NoSQL DBS in the course

- **Document** stores (**MongoDB**)
- **Key-value** stores (**Redis**)
- **Wide column** stores (**Cassandra**)
- **Graph** DBMS (**Neo4j**)

Types of NoSQL Databases: Key-Value Stores

key	value
123	123 Main St.
126	(805) 477-3900



Source: https://hazelcast.com

Types of NoSQL Databases: Key-Value Stores

Data model

- The most simple NoSQL database type

  Works as a simple hash table (mapping)

- **Key-value pairs**

  Key (id, identifier, primary key)

  Value: binary object, black box for the database system

Query patterns

- Create, update or remove value for a given key
- **Get value** for a given key

Characteristics

- Simple model ⇒ **great performance, easily scaled, …**
- Simple model ⇒ **not for complex queries nor complex data**

Key-Value – Benefits & Downsides

Benefits (+):

- ✓ Can store anything
  - Value is essentially just a byte array
- ✓ Fastest possible **in-memory** read/write performance
  - No format or model associated with values
- ✓ Easy adoption of new data sources with a different format

Downsides (-):

- ✓ Lowest common denominator
  - The application needs to know the format of all the values it reads
- ✓ Complex to analyze
  - Analytics need to be written in the app tier
  - No easy means to filter or aggregate on the data tier

Key-Value – Usages

When to use:

- Fastest possible read/write performance of a value
- There are no restrictions on what you can store
- New data source formats aren't known

- **Examples**: IoT, caches, very fast lookups of individual values

When not to use:

- Sophisticated analytics
- The format is well known and/or repetitive & fastest possible performance is of no concern
- Relationships among entities
- Queries requiring access to the content of the value part
- Set operations involving multiple key-value pairs

Types of NoSQL Databases: Key-Value Stores

Suitable use cases

- Caching, session data, user profiles, user preferences, shopping carts, …

  I.e. **when values are only accessed via keys**

Representatives

- **Redis**, **Riak KV**, Berkeley DB, Oracle NoSQL, FoundationDB; Embedded: RocksDB/LevelDB, Berkeley DB;
- Managed/Cloud: Amazon DynamoDB (key-value & document – multi-model)

Types of NoSQL Databases: Document Stores


```
[
 {
  "Employee_ID": 2365,
  "Employee_Name": "Jiří Novák",
  "Department": "Finance",
  "Phone": "666555444",
  "Address": {
      "Street": "Václavské náměstí 123",
      "City": "Praha"
  },
  "Skills": [
   "Účetnictví", "Finanční analýza", "Rozpočtování"
  ]
 },
 {
  "Employee_ID": 3398,
  "Employee_Name": "Kateřina Svobodová",
  "Department": "Admin",
  "Projects": [
   {
    "Name": "Renovace kanceláří",
    "Duration": "6 měsíců"
   },
   {
    "Name": "Aktualizace HR systému",
    "Duration": "3 měsíce"
   }
  ]
 }
]
```

Types of NoSQL Databases: Document Stores

Data model

- **Documents**
  - Self-describing
  - **Hierarchical tree structures** (**JSON**, **XML**, …)
    - Scalar values, maps, lists, sets, nested documents, …
  - Identified by a **unique identifier** (key, …)
- Documents are **organized into collections.**

Query patterns

- Create, update or remove a document
- **Retrieve documents according to complex query conditions**

Observation

- Extended key-value stores where the value part is examinable!

Document – Benefits & Downsides

Benefits (+):

- ✓ Self-contained
  - Data & Metadata stored together
  - Self describing
- ✓ Flexible schema
  - Schema-on-read
  - Document structure can look different between documents
- ✓ Human & Machine readable

Downsides (-):

- ✓ Self-contained
  - Data and metadata is duplicated
  - Changes in data or metadata require scanning for all documents
- ✓ Flexible schema
  - Analytic workloads need to reason about the schema every time
  - Risk of becoming a "dumping ground"

Document – Usages

When to use:

- Data transfer
- Stateless communication
- Relatively static data
- Natural aggregates

- **Examples**: REST, product catalog, etc.

When not to use:

- Set operations involving multiple documents
- Frequent multi-document ACID updates
- The design of the document structure is constantly changing
- Many downstream systems consuming the data
- No natural aggregates

Types of NoSQL Databases: Document Stores

Suitable use cases

- Event logging, content management systems, blogs, web analytics, e-commerce applications, …
  - I.e. **for structured documents with similar schema**

Representatives

- **MongoDB**, **Couchbase**, Amazon **DynamoDB**, **CouchDB**, RethinkDB, RavenDB, Terrastore
- *Multi-model*: **MarkLogic**, **OrientDB**, OpenLink Virtuoso, ArangoDB

Types of NoSQL Databases: Wide Column Stores

Query patterns

- Create, update or remove a row within a given column family
- **Select rows according to a row key or simple conditions**

Warning

- Wide column stores are not just a special kind of RDBMSs with a variable set of columns!

Types of NoSQL Databases: Wide Column Stores

Data model

- **Column family** (table)
  - A **table** is a collection of **similar rows** (not necessarily identical)
- **Row**
  - A **row** is a collection of **columns**
    - Should encompass a group of data that is accessed together
  - Associated with a unique **row key**
- **Column**
  - A **column** consists of a **column name** and **a column value** (and possibly other metadata records)
  - Scalar values, but also **flat sets, lists or maps** may be allowed

Wide Column Stores – Benefits & Downsides

Benefits (+):

- Can handle massive amounts of data across distributed systems
- Optimized for fast **write** operations
- Flexible schema
- Can handle sparse data efficiently
- Fast retrieval when column keys are known

Downsides (-):

- Limited support for complex queries
- Different data model compared to traditional relational databases
- Limited ACID transactions
- Not ideal for small datasets
- Potential for data duplication
- Requires careful planning to optimize for specific query patterns

Wide-column – Usages

When to use:

- Big data applications with high write volumes
- Time-series data storage
- Content management systems with varying attributes
- Systems requiring high scalability and availability
- Applications with known query patterns

When not to use:

- Small-scale applications with simple data structures
- Systems requiring complex joins and relational data models
- Applications needing strong ACID guarantees across multiple rows or tables
- Scenarios requiring frequent, unpredictable analytical queries

Types of NoSQL Databases: Wide Column Stores

Suitable use cases

- Event logging, content management systems, blogs, …
  - I.e. **for structured flat data with similar schema**

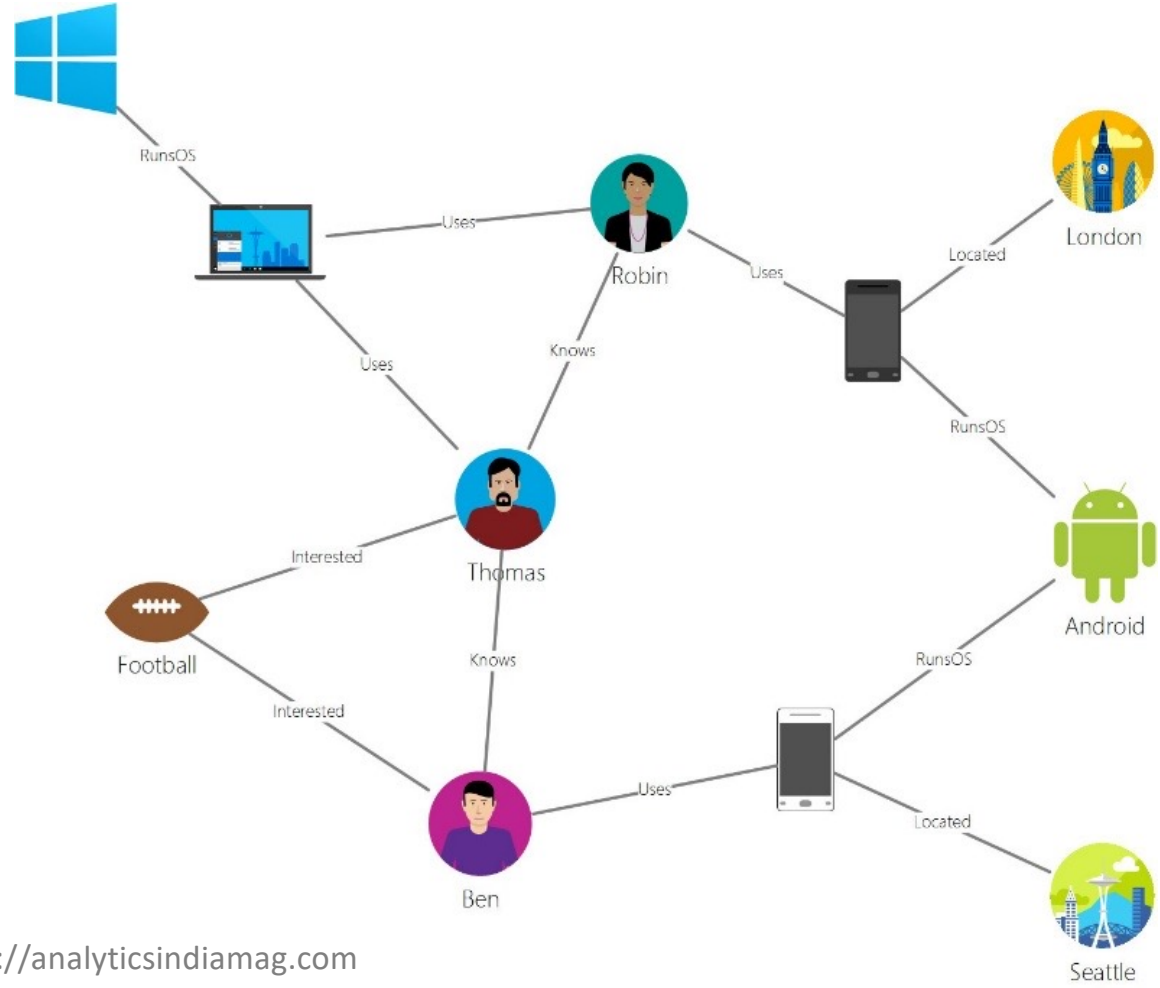
When not to use

- **ACID transactions** are required
- **Complex queries**: aggregation (SUM, AVG, …), joining, …
- Early prototypes: i.e. when **database design may change**

Representatives

- Apache **Cassandra**, Apache **HBase**, Apache Accumulo, Hypertable, **Google Bigtable**

Types of NoSQL Databases: Graph Databases



Source: https://analyticsindiamag.com

Types of NoSQL Databases: Graph Databases

Data model

- Property graphs
  - **Directed / undirected graphs**, i.e. collections of …
    - nodes (vertices) for real-world entities, and
    - relationships (edges) between these nodes
  - Both the nodes and relationships can be associated with additional properties

Types of databases

- Few very large connected graphs
- A large number of small graphs

Types of NoSQL Databases: Graph Databases

Query patterns

- Create, update or remove a node / relationship in a graph
- **Graph algorithms** (shortest paths, spanning trees, …)
- General **graph traversals**
- **Sub-graph** queries or **super-graph** queries
- Similarity based queries (approximate matching)

Representatives

- **Neo4j**, JanusGraph (Titan successor), Amazon Neptune
- *Multi-model*: **OrientDB**, OpenLink **Virtuoso**, **ArangoDB**

Graph Databases – Benefits & Downsides

Benefits (+):

- ✓ Easy analytics for finding relationships between entities
  - Walking the graph
- ✓ Great for finding "hidden" relationships between entities
  - Visualizing the graph

Downsides (-):

- ✓ Anything not to do with finding relationships between entities
  - It is much more cumbersome and/or impractical to model anything that isn't to do with relationships between nodes

Graph Databases – Usages

When to use:

- LinkedIn – who knows who
- Facebook – friends of friends
- Recommendation engines – people who bought, etc.
- Fraud detection
- etc.

When not to use:

- Anything not to do with finding relationships between entities

Types of NoSQL Databases: Graph Databases

Suitable use cases

- Social networks, routing, dispatch, and location-based services, recommendation engines, chemical compounds, biological pathways, linguistic trees, …
  - I.e. simply **for graph structures**

When not to use

- **Extensive batch operations** are required
  - Multiple nodes/relationships are to be affected
- **Only too large graphs** to be stored
  - Graph distribution is difficult or impossible at all

Types of NoSQL Databases: Native XML Databases

Data model

- **XML documents**
  - Tree structure with nested elements, attributes, and text values (besides other less important constructs)
  - Documents are organized into collections

Query languages

- **XPath**: *XML Path Language* (navigation)
- **XQuery**: *XML Query Language* (querying)
- **XSLT**: *XSL Transformations* (transformation) Representatives
- **Sedna**, **Tamino**, BaseX, eXist-db
- *Multi-model*: **MarkLogic**, **Virtuoso**

Types of NoSQL Databases: Native XML Databases

```xml
<?xml version = "1.0"?>
<contact-info>
   <contact1>
      <name>Tanmay Patil</name>
      <company>TutorialsPoint</company>
      <phone>(011) 123-4567</phone>
   </contact1>

   <contact2>
      <name>Manisha Patil</name>
      <company>TutorialsPoint</company>
      <phone>(011) 789-4567</phone>
   </contact2>
</contact-info>
```

https://www.tutorialspoint.com

MarkLogic™

OPENLINK® VIRTUOSO UNIVERSAL SERVER

sedna

Native XML Database System

Tamino XML Server

existdb

Types of NoSQL Databases: RDF Stores

Data model

- **RDF triples**
  - Components: **subject**, **predicate**, and **object**
  - Each triple represents a statement about a real-world entity
- Triples can be viewed as **graphs**
  - **Vertices** for subjects and objects
  - **Edges** directly correspond to individual statements

Query language

- **SPARQL**: *SPARQL Protocol and RDF Query Language*

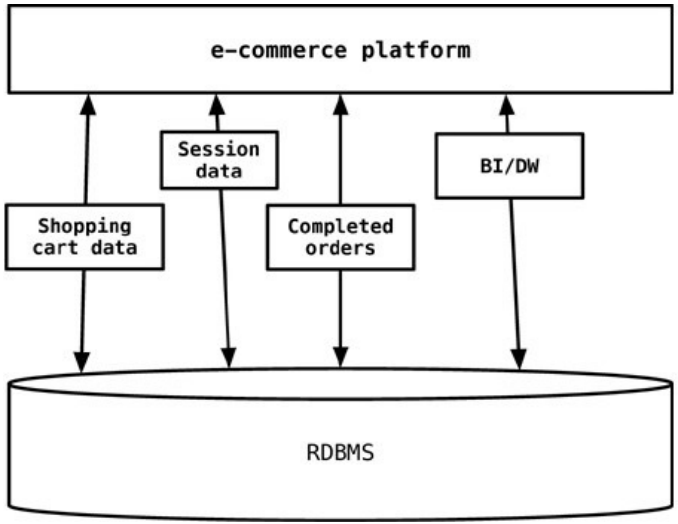
Representatives

- Apache Jena TDB/Fuseki, RDF4J, Blazegraph, Stardog
- *Multi-model*: MarkLogic, Virtuoso

Polyglot Persistence

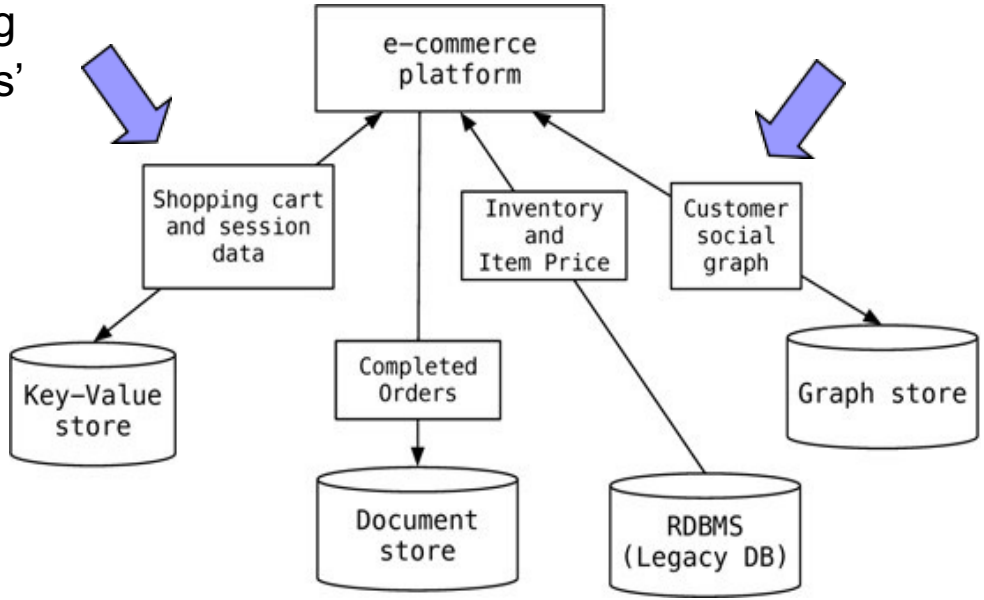
Polyglot Persistence

- Different databases are designed to solve different kinds of problems
- Using a single database engine for all of the requirements usually leads to partially non-performant solutions
- Example: e-commerce
  - Many types of data
    - Business transactions, session management data, reporting, data warehousing, logging information, …
  - Do not need the same properties of availability, consistency, or backup requirements



Polyglot Persistence

- Polyglot programming (2006)
  - Applications should be written in a mix of languages
  - Different languages are suitable for tackling different problems
- Polyglot persistence
  - Hybrid approach to persistence
  - e.g., a data store for the shopping cart, which is highly available vs. finding products bought by the customers' friends



Polyglot Persistence

Functionality	Considerations	Database Type
User Sessions	Rapid Access for reads and writes. Short-lived; occasional loss acceptable	Key-value
Financial Data	Needs transactional updates. Tabular structure fits data	RDBMS
POS Data	Depending on size and rate of ingest. Lots of writes, infrequent reads, mostly for analytics	RDBMS (if modest), Key-value or Document (if ingest very high), or Wide-column if analytics is key.
Shopping Cart	High availability across multiple locations. Can merge inconsistent writes	Key-value or Document
Recommendations	Rapidly traverse links between friends, product purchases, and ratings	Graph
Product Catalog	Lots of reads, infrequent writes. Products make natural aggregates	Document or RDBMS
Reporting	SQL interfaces well with reporting tools	RDBMS, Wide-column
Analytics	Large scale analytics on large cluster	Wide-column or Hadoop
User activity logs, CSR logs, Social Media analysis	High volume of writes on multiple nodes	Key-value or Document

https://www.jamesserra.com/archive/2015/07/what-is-polyglot-persistence/

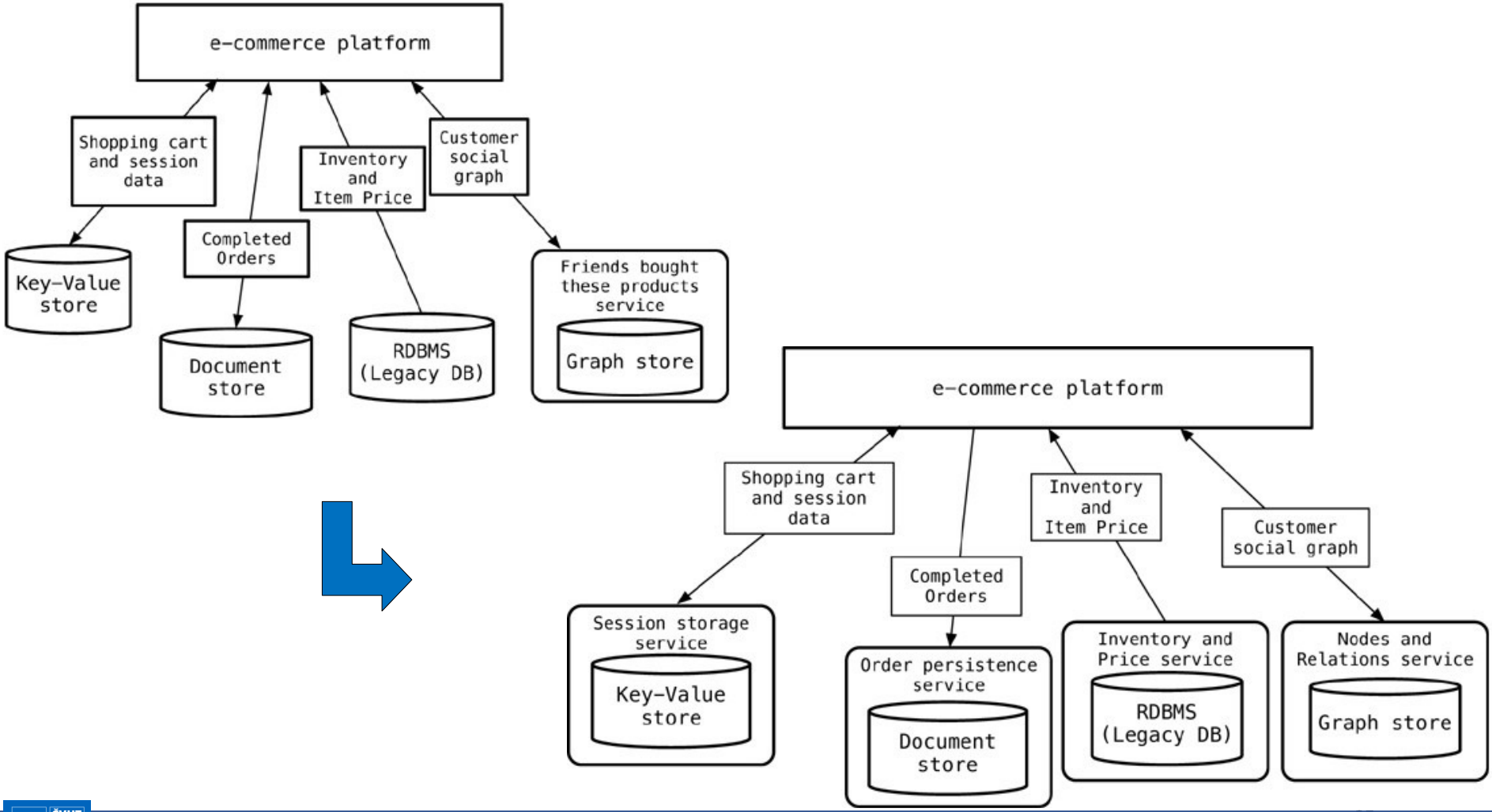
Polyglot Persistence

- There may be other applications in the enterprise
  - e.g., the graph data store can serve data to applications that need to understand which products are being bought by a particular segment of the customer base.

⇒ Instead of each application talking independently to the graph database, we can wrap the graph database into a service

- Assumption:
  - Nodes can be saved in one place
  - Queried by all the applications
- Allows for the databases inside the services to evolve without having to change the dependent applications

Polyglot Persistence



Lecture Conclusion

- ✓ **Key-Value**
  - Caching, sessions, high-speed lookups
  - Simplest model, fastest performance
- ✓ **Document**
  - Content management, flexible schemas
  - Self-contained aggregates (JSON/XML)
- ✓ **Wide Column**
  - Time-series, massive scale writes
  - Distributed, tunable consistency
- ✓ **Graph**
  - Social networks, relationships
  - Connections are first-class citizens

Choose based on:

- Access patterns (how you query data)
- Scale requirements (volume and velocity)
- Consistency vs. availability needs
- Data structure (relationships, hierarchy, flat)

Polyglot Persistence

- Different problems need different solutions
- Single application can use multiple databases
- Wrap databases in services for flexibility