










Lecture 10

Big Data Storage Technologies

Dr. Lydia Wahid

Agenda

-  Introduction
-  On-Disk Storage:
 -  Distributed File System
 -  RDBMS Databases
 -  NoSQL Databases
 -  NewSQL Databases
-  In-Memory Storage:
 -  In-Memory Data Grids
 -  In-Memory Database (Relational, NoSQL, NewSQL)



Introduction

Introduction

- Storage technology has continued to evolve over time.
- The need to **store Big Data** has radically altered the relational, database-centric view that has been embraced by Enterprise ICT.
- The bottom line is that relational technology is simply **not scalable** in a manner to support **Big Data volumes**.
- Also, businesses can find genuine value in processing **semi-structured** and **unstructured** data, which are generally **incompatible** with relational approaches.

Introduction

- Big Data has pushed the storage boundary to unified views of the available memory and **disk storage** of a cluster.
- If more storage is needed, horizontal scalability allows the expansion of the cluster through the addition of **more nodes**.
- Innovative approaches deliver **realtime** analytics via **in-memory storage**.
- Even **batch-based processing** is accelerated by the performance of Solid State Drives (SSDs), which have become less expensive.



On-Disk Storage

On-Disk Storage

- On-disk storage generally utilizes **low cost** hard-disk drives for long-term storage.
- On-disk storage can be implemented via a **distributed file system** or a **database**.



On-Disk Storage: Distributed File Systems

On-Disk Storage: Distributed File Systems

- Distributed file systems, like any file system, support **schema-less** data storage.
- In general, a distributed file system storage device provides **redundancy** and high **availability** by copying data to multiple locations via **replication**.
- A storage device that is implemented with a distributed file system provides *simple, fast access* data storage that is capable of storing large datasets that are non-relational in nature, such as **semi-structured** and **unstructured** data.

On-Disk Storage: Distributed File Systems

- Although based on straightforward file **locking** mechanisms for concurrency control, it provides **fast read/write** capability, which addresses the **velocity** characteristic of Big Data.
- Distributed file system is **not ideal** for datasets comprising a **large number of small files** as this creates excessive **disk-seek activity**, **slowing down** the overall data access.
- Due to these limitations, distributed file systems work best with **fewer but larger files** accessed in a **sequential manner**.

On-Disk Storage: Distributed File Systems

- Multiple smaller files are generally **combined** into a single file to enable optimum *storage* and *processing*.
- This allows the distributed file systems to have increased performance when data must be accessed in **streaming mode** with **no random** reads and writes.
- A distributed file system storage device is suitable when large datasets of **raw data** are to be **stored** or when **archiving** of datasets is required.
- It should be noted that distributed file systems **do not provide the ability to search** the contents of files



On-Disk Storage: RDBMS Databases

On-Disk Storage: RDBMS Databases

- Relational database management systems (RDBMSs) are good for handling transactional workloads involving **small amounts of data with random read/write properties**.
- RDBMSs are **ACID-compliant**, and they are generally restricted to a **single node**.
- To handle large volumes of data arriving at a fast pace, relational databases generally **need to scale**. RDBMSs employ **vertical scaling**, not horizontal scaling, which is a *more costly* and *disruptive* scaling strategy.

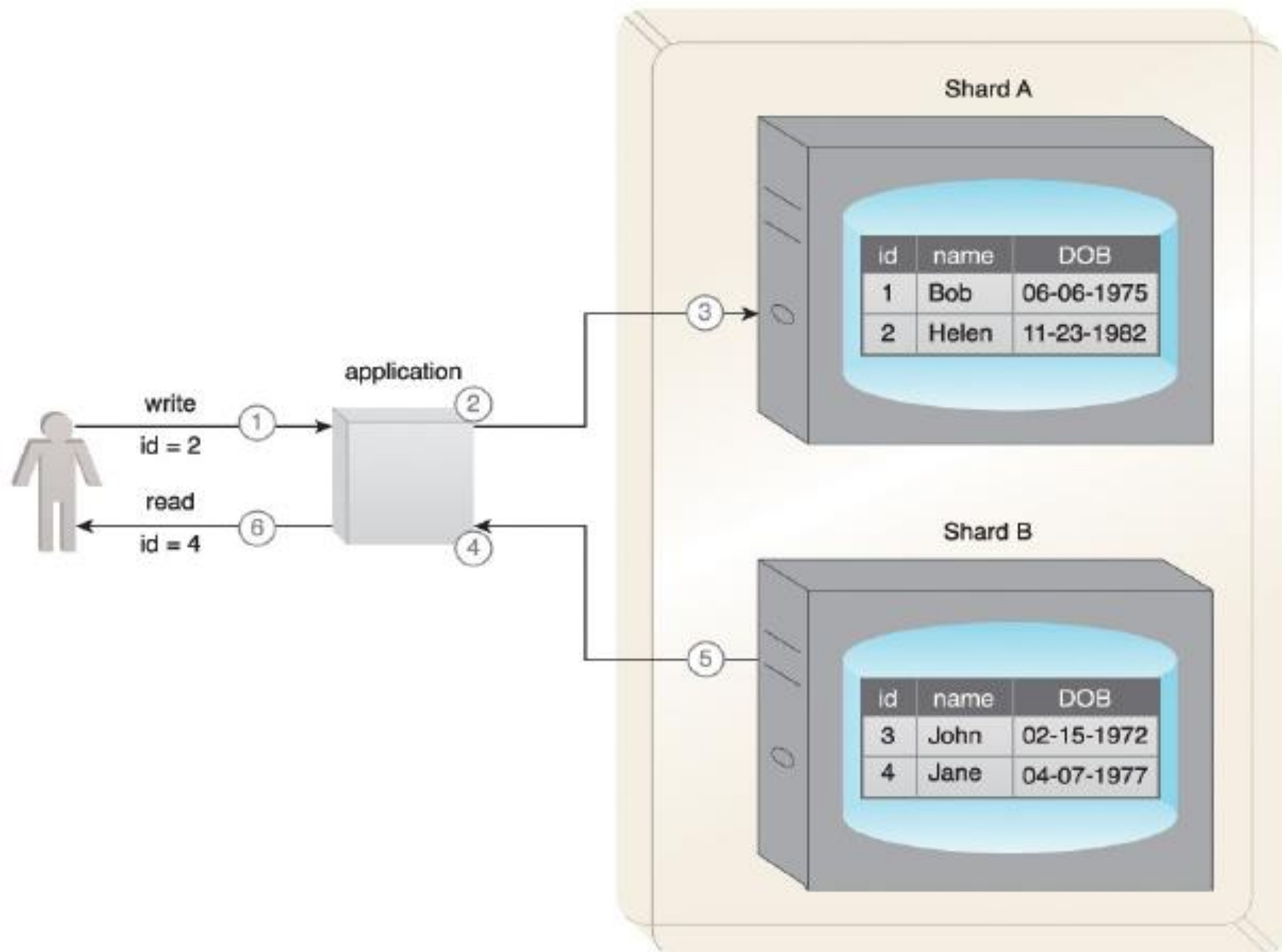
On-Disk Storage: RDBMS Databases

- Note that some relational databases, for example IBM DB2 pureScale, Sybase ASE Cluster Edition, Oracle Real Application Clusters (RAC) and Microsoft Parallel Data Warehouse (PDW), **are capable of being run on clusters**.
- However, these database clusters still **use shared storage** that can act as a single point of failure.
- Relational databases need to be **manually sharded**, mostly using application logic.
- This means that the application logic needs to know which shard to query in order to get the required data. This further **complicates** data processing when data from **multiple shards** is required.

On-Disk Storage: RDBMS Databases

➤ Consider the following scenario:

1. A user **writes** a record (id = 2).
2. The **application logic** determines **which shard** it should be written to.
3. It is **sent to the shard** determined by the application logic.
4. The user reads a record (id = 4), and the application logic determines which shard contains the data.
5. The data is read and returned to the application.
6. The application then returns the record to the user.



On-Disk Storage: RDBMS Databases

- Relational databases generally require data to **adhere to a schema**. As a result, storage of **semi-structured** and **unstructured** data whose schemas are non-relational is **not directly supported**.
- Furthermore, with a relational database schema conformance is validated **at the time of data insert or update** by checking the data against the constraints of the schema. This introduces overhead that creates **latency**.
- This latency makes relational databases a less than ideal choice for storing **high velocity** data that needs a highly available database storage device with *fast data write* capability.



On-Disk Storage: NoSQL Databases

NoSQL Databases: Characteristics

- **Not-only SQL** (NoSQL) refers to technologies used to develop next generation **non-relational** databases that are **highly scalable** and **fault-tolerant**.
- Below is a list of the principal characteristics of NoSQL storage devices that differentiate them from traditional RDBMSs:
 - **Schema-less data model** – Data can exist in its raw form.
 - **Highly available** – This is built on cluster-based technologies that provide fault.
 - **Eventual consistency** – Data reads across multiple nodes but may not be consistent immediately after a write. However, all nodes will eventually be in a consistent state.

NoSQL Databases: Characteristics

- Below is a list of the principal characteristics of NoSQL storage devices that differentiate them from traditional RDBMSs (cont.):
- **Scale out rather than scale up** – More nodes can be added to obtain additional storage with a NoSQL database.
 - **BASE, not ACID** – BASE compliance requires a database to maintain high *availability* in the event of network/node failure, while not requiring the database to be in a *consistent* state whenever an update occurs. NoSQL storage devices are generally *AP* or *CP*.
 - **API driven data access** – Data access is generally supported via API based queries, including *RESTful APIs*, whereas some implementations may also provide *SQL-like query* capability.

NoSQL Databases: Characteristics

- Below is a list of the principal characteristics of NoSQL storage devices that differentiate them from traditional RDBMSs (cont.):
- **Auto sharding and replication** – To support horizontal scaling and provide high availability, a NoSQL storage device automatically employs sharding and replication techniques.
 - **Polyglot persistence** – The use of NoSQL storage does not mandate retiring traditional RDBMSs. In fact, both can be used at the same time, thereby supporting polyglot persistence, which is an approach of *persisting data using different types of storage technologies within the same solution architecture*. This is good for developing systems requiring structured as well as semi/unstructured data.

NoSQL Databases: Characteristics

- Below is a list of the principal characteristics of NoSQL storage devices that differentiate them from traditional RDBMSs (cont.):
 - **Aggregate-focused** – Unlike relational databases that are most effective with fully normalized data, NoSQL storage devices store de-normalized aggregated data (an entity containing merged, often nested, data for an object) thereby eliminating the need for joins and extensive mapping between application objects and the data stored in the database. One exception, however, is that graph database storage devices (introduced shortly) are not aggregate-focused.


NoSQL Databases: Rational

- The emergence of NoSQL storage devices can primarily be attributed to the *volume*, *velocity* and *variety* characteristics of Big Data datasets:
 - **Volume:**
 - NoSQL storage devices provide **scale out** capability while using **inexpensive** commodity servers.
 - **Velocity:**
 - **Write latency does not occur** as **no schema check on write**.
 - It is **highly available**.
 - **Variety:**
 - NoSQL storage devices can store these different forms of **semi-structured** and **unstructured** data formats.
 - NoSQL databases support schema evolution.

NoSQL Databases: Types

- NoSQL storage devices can mainly be divided into four types based on the way they store data:
 - Key-value
 - Document
 - Column-family
 - Graph

NoSQL Databases: Types



```
{  
  invoiceId:37235,  
  date:19600801,  
  custId:29317,  
  items:[  
    {itemId:473,quantity:2},  
    {itemId:971,quantity:5}  
  ]  
}
```

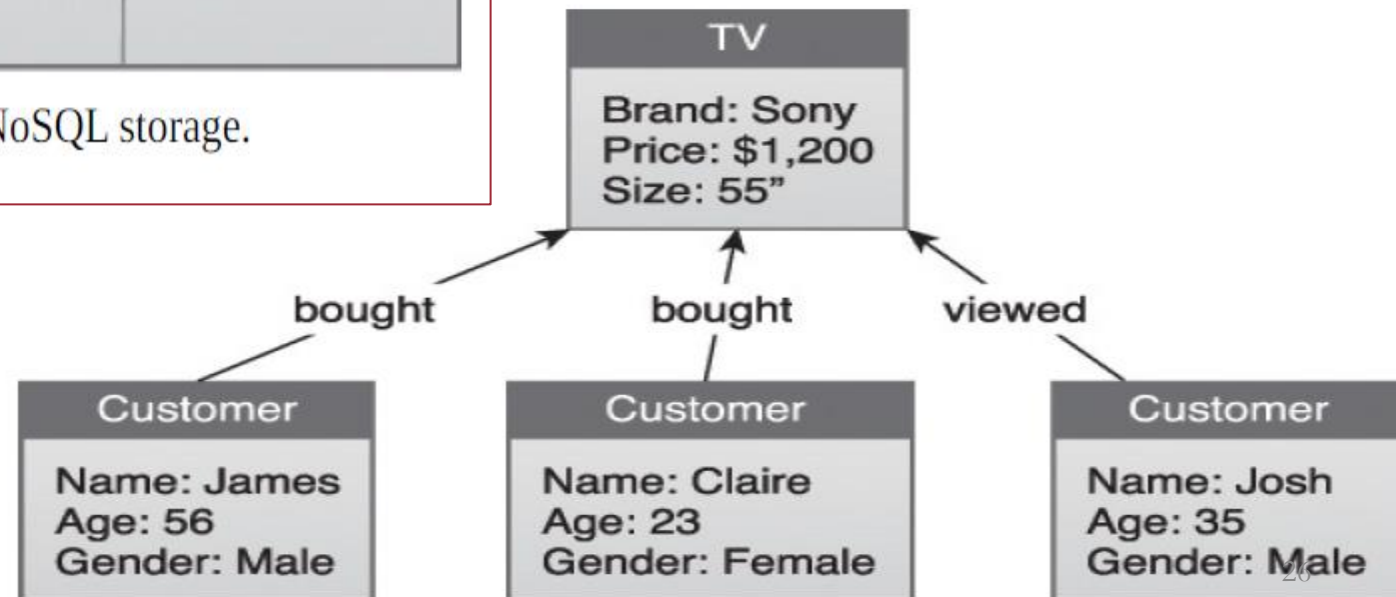
An example of document NoSQL storage.

key	value
631	John Smith, 10.0.30.25, Good customer service
365	10010101110110111101110101010101001110011010
198	<CustomerId>32195</CustomerId><Total>43.25</Total>

An example of key-value NoSQL storage.

studentId	personal details	address	modules history
821	FirstName: Cristie LastName: Augustin DoB: 03-15-1992 Gender: Female Ethnicity: French	Street: 123 New Ave City: Portland State: Oregon ZipCode: 12345 Country: USA	Taken: 5 Passed: 4 Failed: 1
742	FirstName: Carlos LastName: Rodriguez MiddleName: Jose Gender: Male	Street: 456 Old Ave City: Los Angeles Country: USA	Taken: 7 Passed: 5 Failed: 2

An example of column-family NoSQL storage.



An example of graph NoSQL storage.

NoSQL Databases: Key-Value

- A key-value storage device is **appropriate** when:
- unstructured data storage is required
 - high performance read/writes are required
 - the value is fully identifiable via the key alone
 - value is a standalone entity that is not dependent on other values
 - values have a comparatively simple structure or are binary
 - query patterns are simple, involving insert, select and delete operations only

NoSQL Databases: Key-Value

- A key-value storage device is **inappropriate** when:
- applications require searching or filtering data using attributes of the stored value
 - relationships exist between different key-value entries
 - a group of keys' values need to be updated in a single transaction
 - multiple keys require manipulation in a single operation
 - schema consistency across different values is required
 - update to individual attributes of the value is required

NoSQL Databases: Document

- The main differences between document storage devices and key-value storage devices are as follows:
- document storage devices are value-aware
 - the stored value is self-describing; the schema can be inferred from the structure of the value or a reference to the schema for the document is included in the value
 - a select operation can reference a field inside the aggregate value
 - a select operation can retrieve a part of the aggregate value
 - partial updates are supported; therefore a subset of the aggregate can be updated

NoSQL Databases: Document

- A document storage device is **appropriate** when:
 - storing semi-structured document-oriented data comprising
 - schema evolution is a requirement as the structure of the document is either unknown or is likely to change
 - applications require a partial update of the aggregate stored as a document
 - searches need to be performed on different fields of the documents
 - query patterns involve insert, select, update and delete operations
- A document storage device is **inappropriate** when:
 - multiple documents need to be updated as part of a single transaction
 - performing operations that need joins between multiple documents
 - schema enforcement for achieving consistent query design is required

NoSQL Databases: Column-Family

- A column-family storage device is **appropriate** when:
 - data represents a tabular structure
 - support for schema evolution is required
 - certain fields are mostly accessed together, and searches need to be performed using field values
 - query patterns involve insert, select, update and delete operations
- A column-family storage device is **inappropriate** when:
 - relational data access is required; for example, joins
 - ACID transactional support is required
 - SQL-compliant queries need to be executed
 - query patterns are likely to change frequently because that could initiate a corresponding restructuring of how column-families are arranged

NoSQL Databases: Graph

- A graph storage device is **appropriate** when:
 - Consistency is required (generally, graph storage devices provide consistency via ACID compliance)
 - interconnected entities need to be stored
 - querying entities based on the type of relationship with each other rather than the attributes of the entities
 - finding groups of interconnected entities
 - finding distances between entities in terms of the node traversal distance

NoSQL Databases: Graph

- A graph storage device is **inappropriate** when:
 - updates are required to a large number of node attributes or edge attributes, as this involves searching for nodes or edges, which is a costly operation
 - entities have a large number of attributes or nested data—it is best to store lightweight entities in a graph storage device while storing the rest of the attribute data in a separate non-graph NoSQL storage device
 - queries based on the selection of node/edge attributes dominate node traversal queries



On-Disk Storage: NewSQL Databases

NewSQL Databases

- NoSQL storage devices are **highly scalable, available, fault-tolerant** and **fast** for read/write operations.
- However, they do not provide the same transaction and consistency support as exhibited by **ACID** compliant RDBMSs. Following the BASE model, NoSQL storage devices provide **eventual consistency** rather than **immediate consistency**.
- They therefore will be in a soft state while reaching the state of eventual consistency.
- As a result, they are **not appropriate** for use when **implementing large scale transactional systems**.

NewSQL Databases

- NewSQL is a class of relational databases that combines the **ACID properties** of **RDBMS** with the **scalability and fault tolerance** offered by **NoSQL** storage devices.
- NewSQL databases generally **support SQL** compliant syntax for data definition and data manipulation operations, and they often use a logical **relational data model** for data storage.

NewSQL Databases

- NewSQL databases can be used for developing **OLTP** (online transaction processing) systems with very high volumes of transactions, for example a banking system.
- They can also be used for **real-time analytics** as some implementations leverage **in-memory storage**.
- Examples of NewSQL databases include Apache Trafodion, Google spanner, VoltDB and NuoDB.



In-Memory Storage

In-Memory Storage

- This section presents in-memory storage as a means of providing options for **highly performant**, and **advanced data storage**.
- An in-memory storage device generally utilizes **RAM**, the main memory of a computer, as its storage medium to provide **fast data access**.
- The growing capacity and decreasing cost of RAM, coupled with the increasing read/write speed of solid state hard drives, has made it possible to develop in-memory data storage solutions.

In-Memory Storage

- Storage of data in memory **eliminates the latency of disk I/O** and the data transfer time between the main memory and the hard drive.
- This **overall reduction** in data read/write **latency** makes data processing **much faster**.
- In-memory storage device capacity can be increased massively by **horizontally scaling the cluster**.
- In-memory storage significantly **reduces the overall execution time** of Big Data analytics, thus **enabling real-time Big Data analytics**.

In-Memory Storage

- When compared with an on-disk storage device, an in-memory storage device is **expensive** because of the higher cost of memory as compared to a disk-based storage device.
- Apart from being expensive, in-memory storage devices do not provide the same level of support for **durable data storage**. The **price** factor further affects the achievable **capacity** of an in-memory device when compared with an on-disk storage device.

In-Memory Storage

- An in-memory storage device is **appropriate** when:
 - data arrives at a **fast pace** and **requires real-time analytics** or **event stream processing**
 - **continuous** or **always-on analytics** is required, such as operational BI and operational analytics
 - **interactive query processing** and real-time data visualization needs to be performed
 - developing **low latency** Big Data solutions

In-Memory Storage

- An in-memory storage device is **inappropriate** when::
 - data processing consists of **batch processing**
 - **very large amounts of data** need to be persisted in-memory for a long time in order to perform in-depth data analysis
 - datasets are **extremely large and do not fit** into the available memory
 - an enterprise has a **limited budget**, as setting up an in-memory storage device may require upgrading nodes, which could either be done by node replacement or by adding more RAM

In-Memory Storage

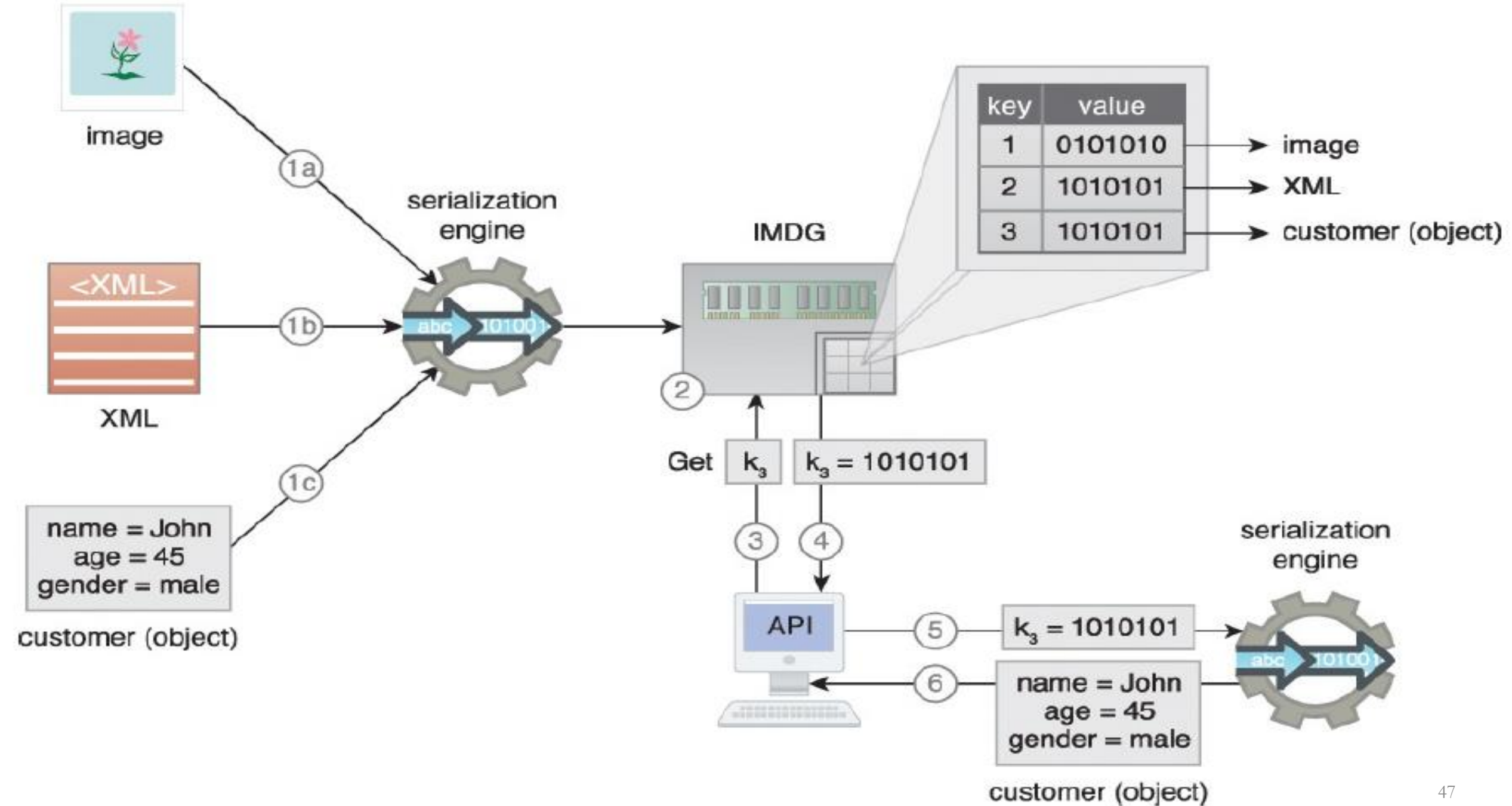
- In-memory storage devices can be implemented as:
 - In-Memory Data Grid (IMDG)
 - In-Memory Database (IMDB)
- Although both of these technologies use memory as their underlying data storage medium, what makes them distinct is **the way data is stored in the memory.**



In-Memory Storage: In-Memory Data Grids

In-Memory Storage: In-Memory Data Grid (IMDG)

- IMDGs store data in memory as **key-value pairs** across multiple nodes where the keys and values can be any business object or application data in serialized form.
- This supports schema-less data storage through storage of **semi/unstructured** data. Data access is typically provided via APIs.



An IMDG storage device.

In-Memory Storage: In-Memory Data Grid (IMDG)

- The previous figure describes the following scenario:
1. An image (a), XML data (b) and a customer object (c) are first serialized using a serialization engine.
 2. They are then stored as key-value pairs in an IMDG.
 3. A client requests the customer object via its key.
 4. The value is then returned by the IMDG in serialized form.
 5. The client then utilizes a serialization engine to deserialize the value.
 6. The original object is returned for manipulation.

In-Memory Storage: In-Memory Data Grid (IMDG)

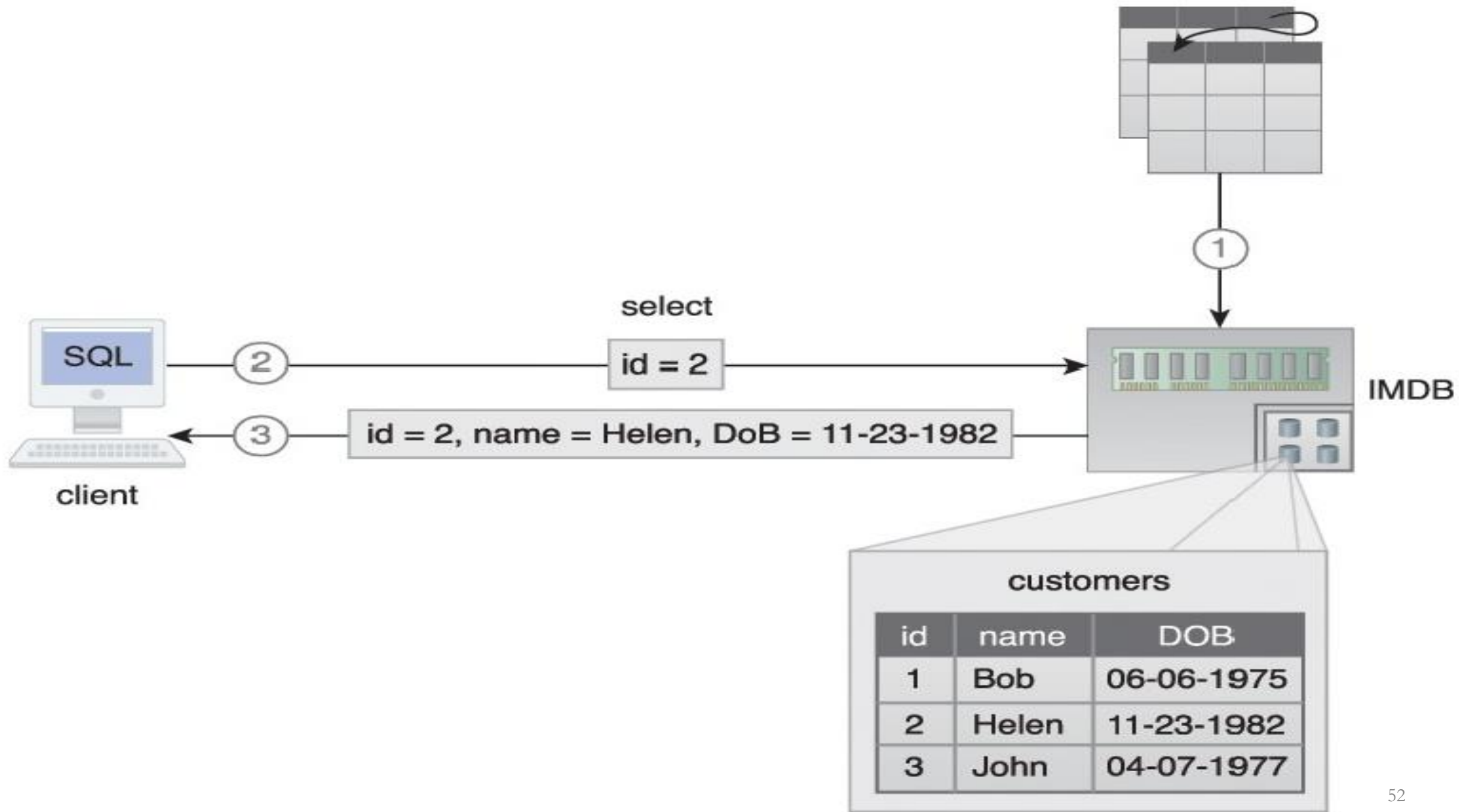
- Nodes in IMDGs keep themselves synchronized and collectively provide **high availability**, **fault tolerance** and **consistency**.
- In comparison to **NoSQL**'s eventual consistency approach, IMDGs support **immediate consistency**.
- As compared to relational **IMDBs** (discussed next), IMDGs provide **faster** data access as IMDGs store non-relational data as objects.
- IMDGs may also support **in-memory MapReduce** that helps to reduce the latency of disk based MapReduce processing.
- Examples include In-Memory Data Fabric, Hazelcast and Oracle Coherence.



In-Memory Storage: In-Memory Database

In-Memory Storage: In-Memory Database (IMDB)

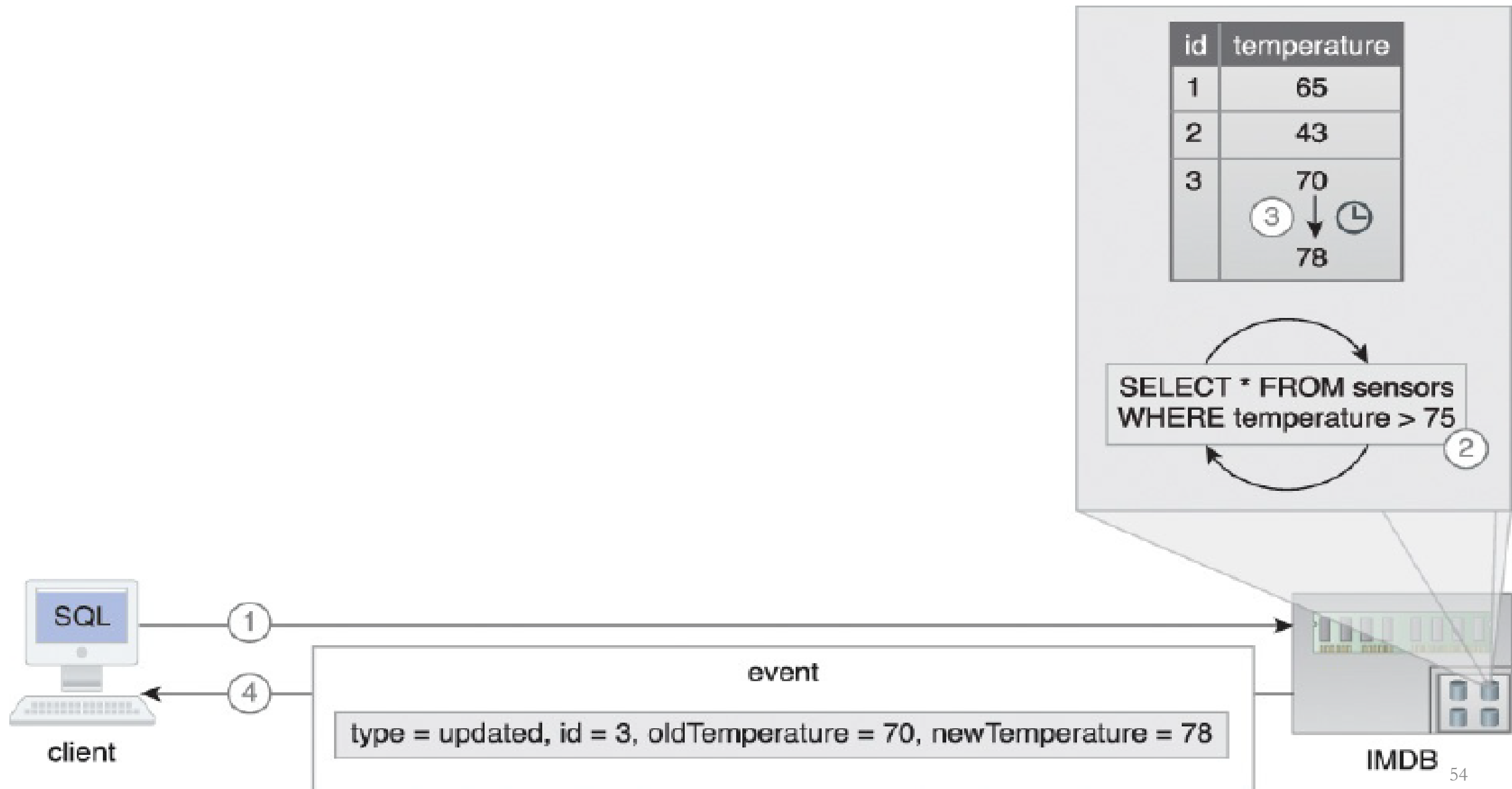
- IMDBs are in-memory storage devices that **employ database technology** and leverage the performance of **RAM** to **overcome runtime latency** issues of the on-disk storage devices.
- An IMDB can be **relational** in nature (**relational IMDB**) for the storage of **structured** data, or may leverage **NoSQL** technology (**non-relational IMDB**) for the storage of **semistructured** and **unstructured** data.
- IMDBs are heavily used in **real-time analytics** and can further be used for developing **low latency applications** requiring full ACID transaction support (relational IMDB).
- Examples include Aerospike, MemSQL, Altibase HDB, eXtreme DB and Pivotal GemFire XD



An example depicting the retrieval of data from an IMDB.

In-Memory Storage: In-Memory Database (IMDB)

- The previous example describes the following scenario:
1. A relational dataset is **stored into an IMDB**.
 2. A **client requests** a customer record (id = 2) via SQL.
 3. The relevant customer **record is then returned** by the **IMDB**, which is directly manipulated by the client without the need for any deserialization.



An example of IMDB storage configured with a continuous query.

In-Memory Storage: In-Memory Database (IMDB)

- The previous example describes the following scenario:
1. A client issues a **continuous query**
(select * from sensors where temperature > 75).
 2. It is registered in the **IMDB**.
 3. When the temperature for any sensor exceeds 75F ...
 4. ... an **updated event** is sent to the subscribing **client** that contains various details about the event.

In-Memory Storage: In-Memory Database (IMDB)

- In the case of replacing an RDBMS with a relational IMDB, **little or no application code** change is required due to SQL support provided by the relational IMDB.
- However, when replacing an RDBMS with a NoSQL IMDB, **code change may be required** due to the need to implement the IMDB's NoSQL APIs.
- In the case of replacing an on-disk NoSQL database with a relational IMDB, **code change** will often be required to establish SQL-based access.
- However, when replacing an on-disk NoSQL database with a NoSQL IMDB, **code change** may still be required due to the implementation of new APIs.



Thank You