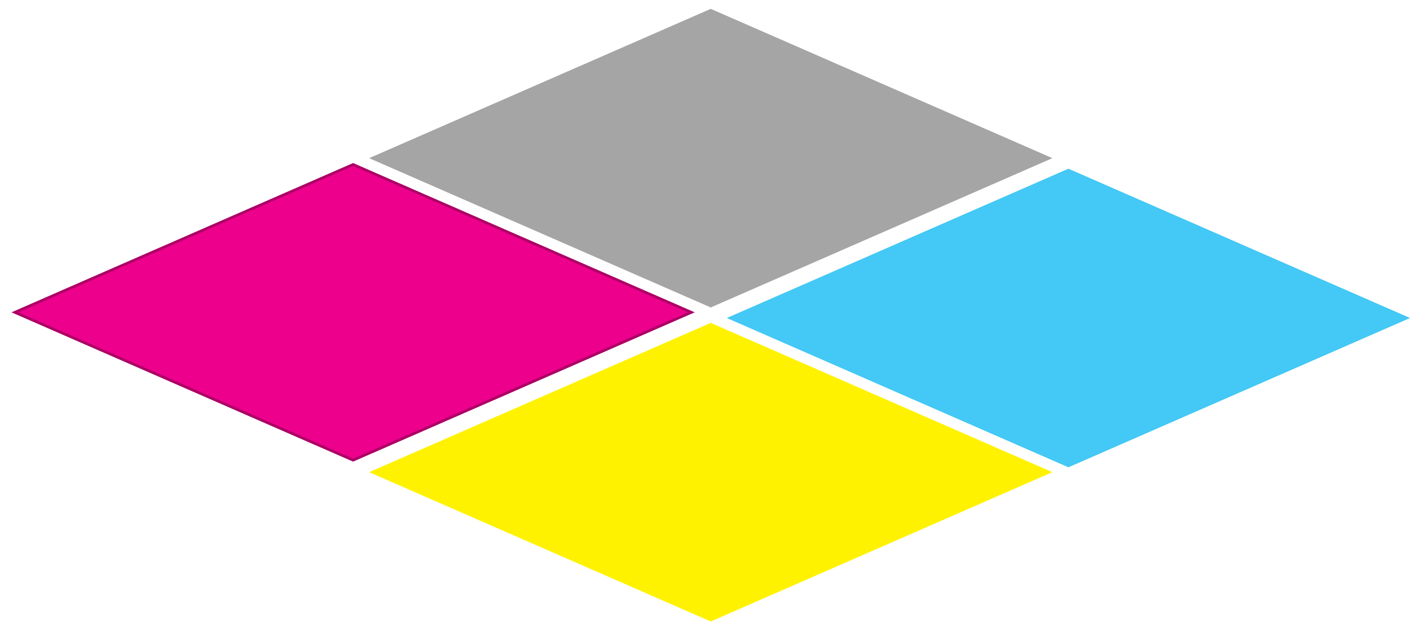
howest
hogeschool

Object Oriented Architectures and Secure Development

Client – Server

Matthias Blomme
Mattias De Wael
Frédéric Vlummens

Single layered application



The n-tier application

UI

FXML

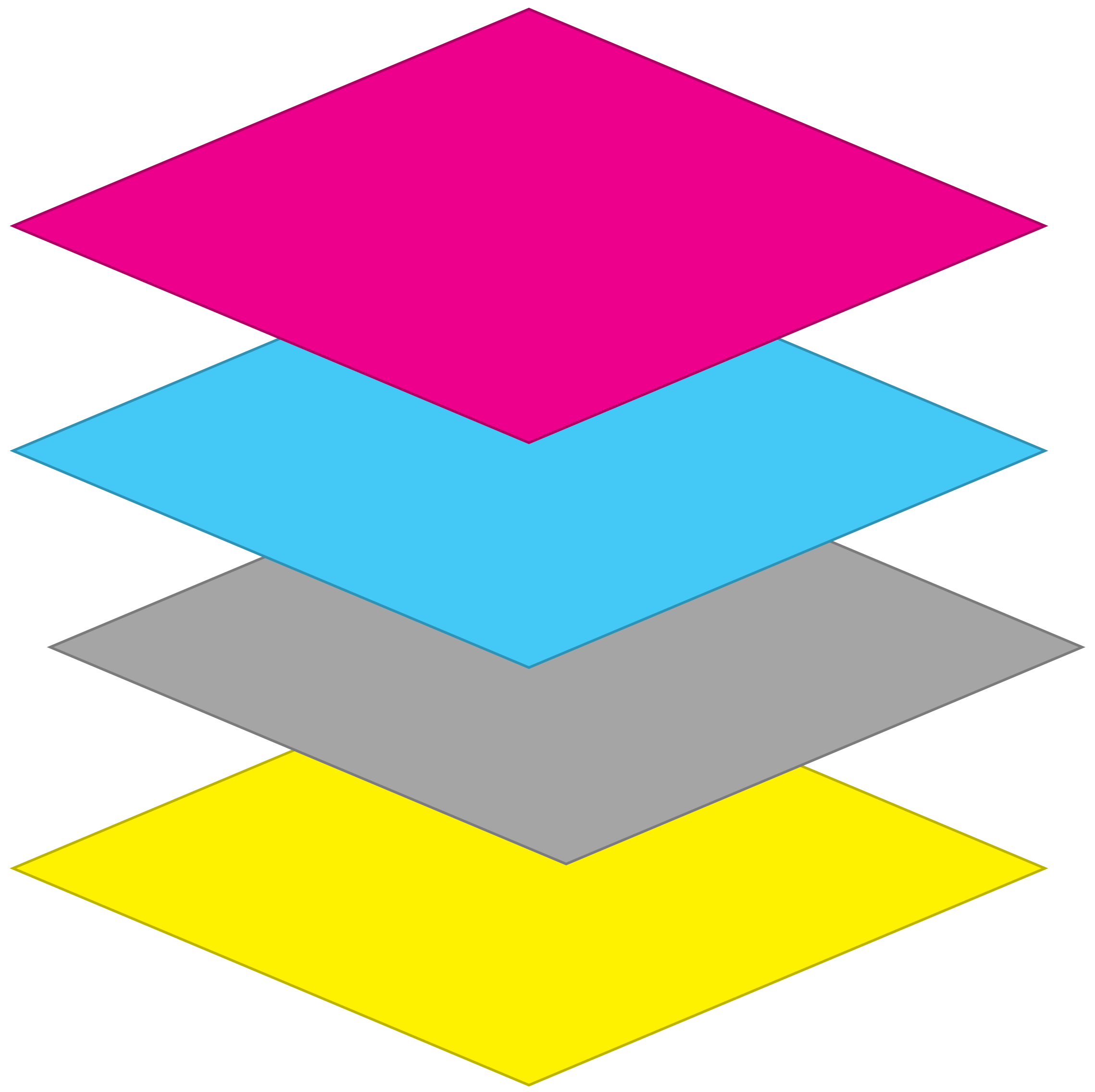
Services

Domain

POJO

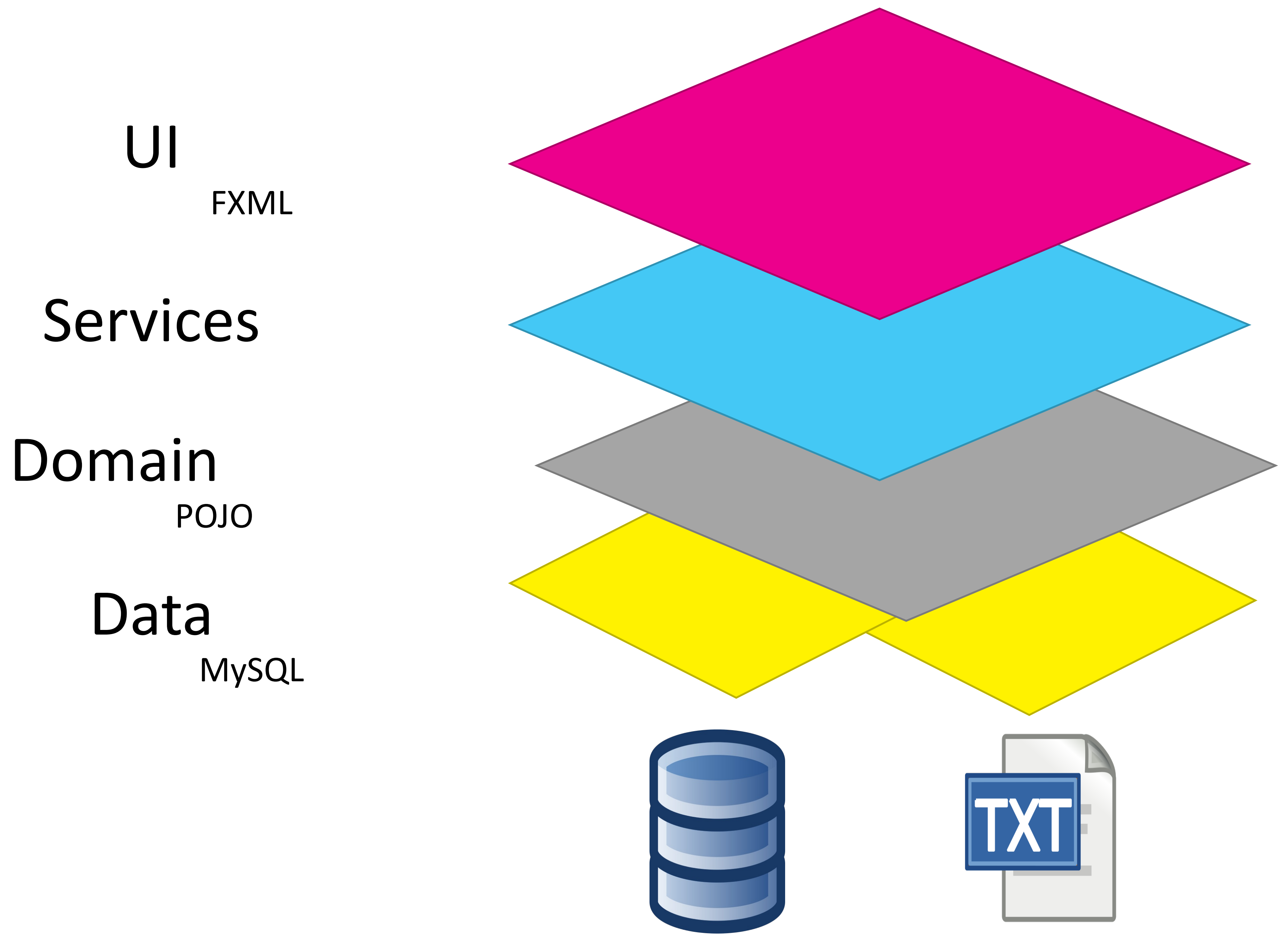
Data

MySQL

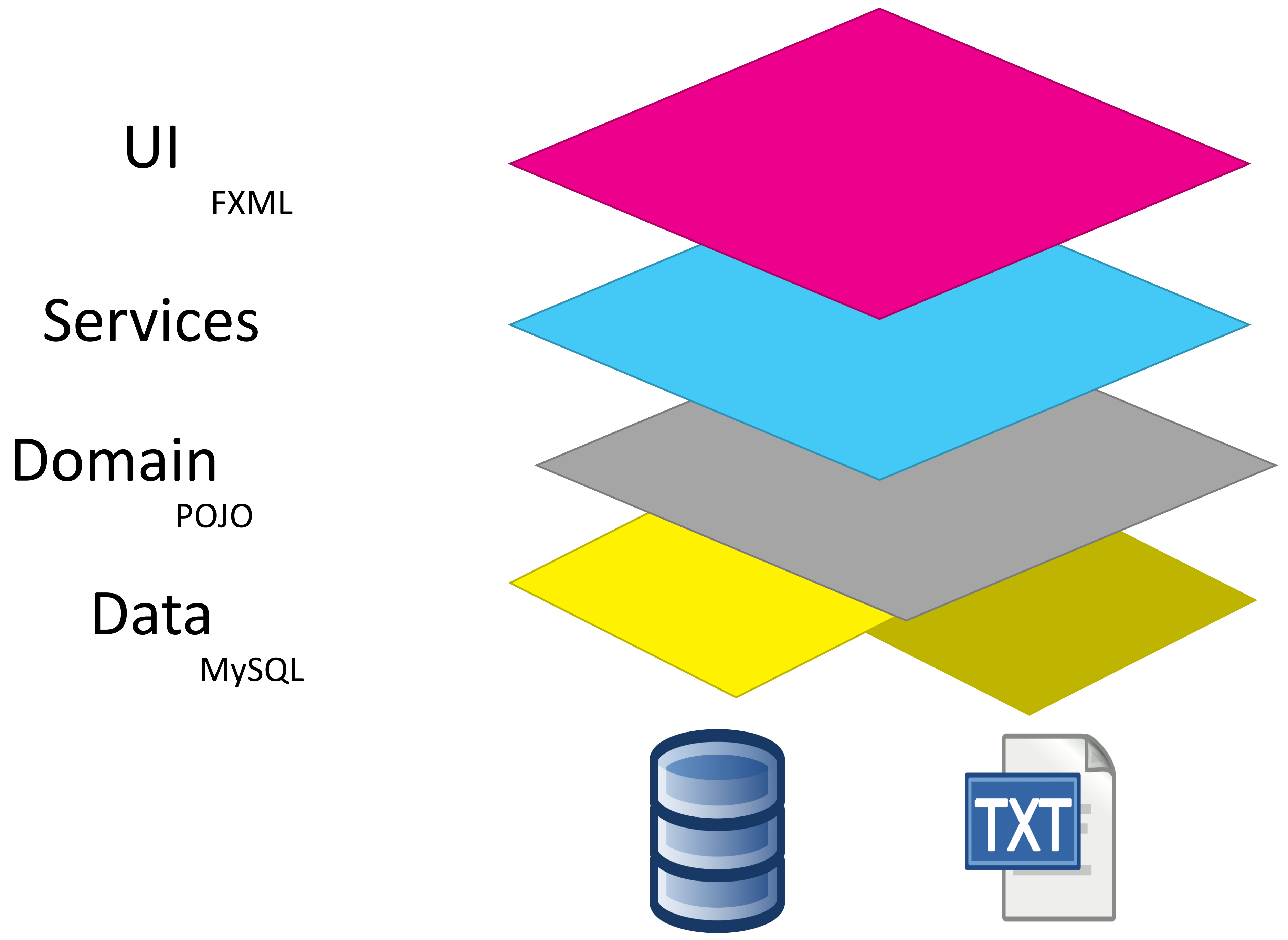


POJO = Plain Old Java Object

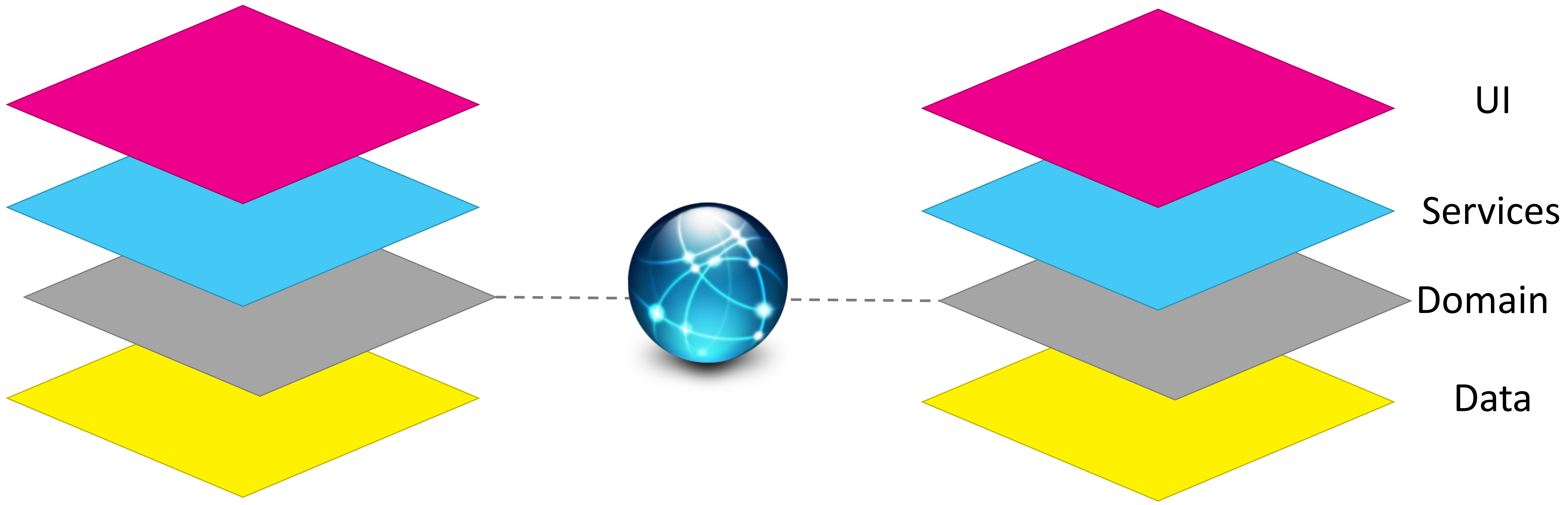
The n-tier application: multiple data access strategies



What if some data is not available locally?

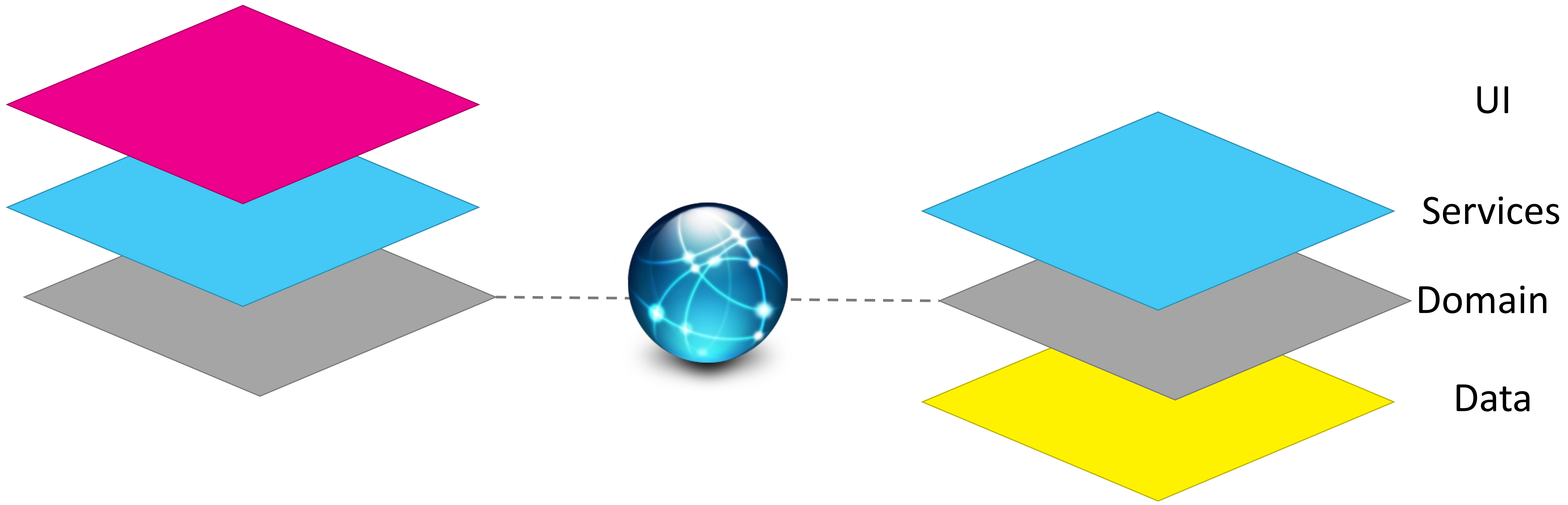


Client-server architecture



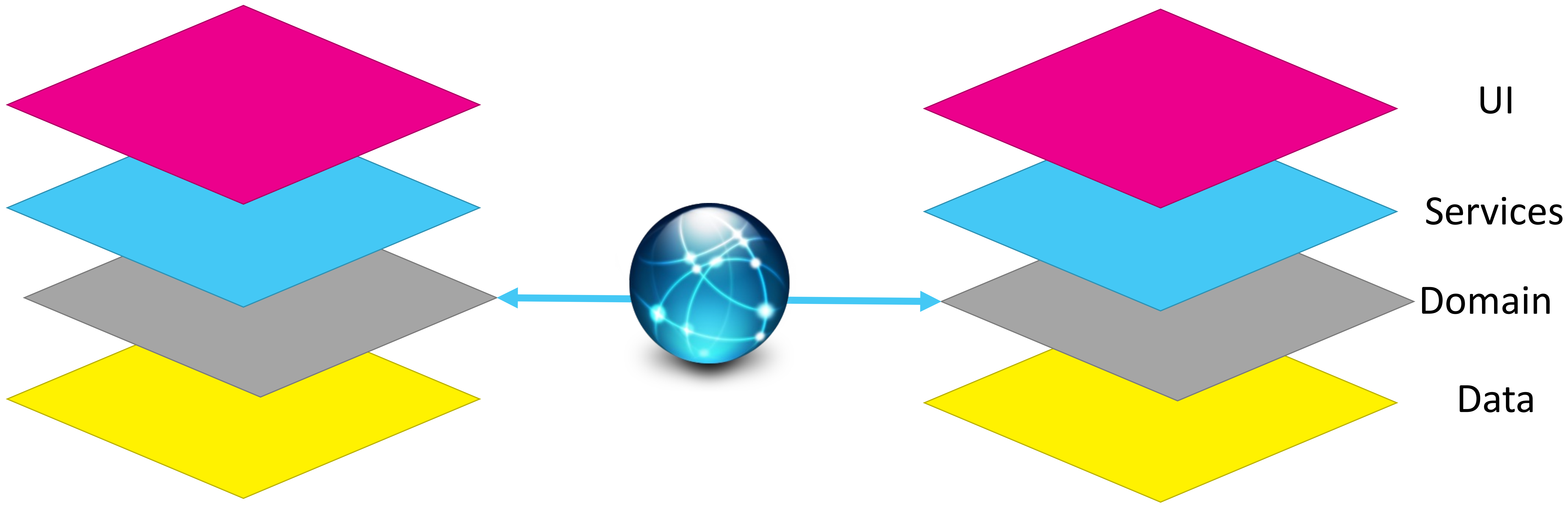
Usually, (a large part of) the model/domain is shared

Client-server architecture



Not all layers need to be present in both client and server

Client-server architecture



Communication is done through sockets

Sockets

- Server

```
ServerSocket serverSock = new ServerSocket(1234);
Socket sock = serverSock.accept();
```

Pick a port

Wait for a connection

- Client

```
Socket sock = new Socket("localhost", 1234);
```

The server's address and port

```
sock.getInputStream();
sock.getOutputStream();
```

We know its family (file streams, System.in, System.out, …)

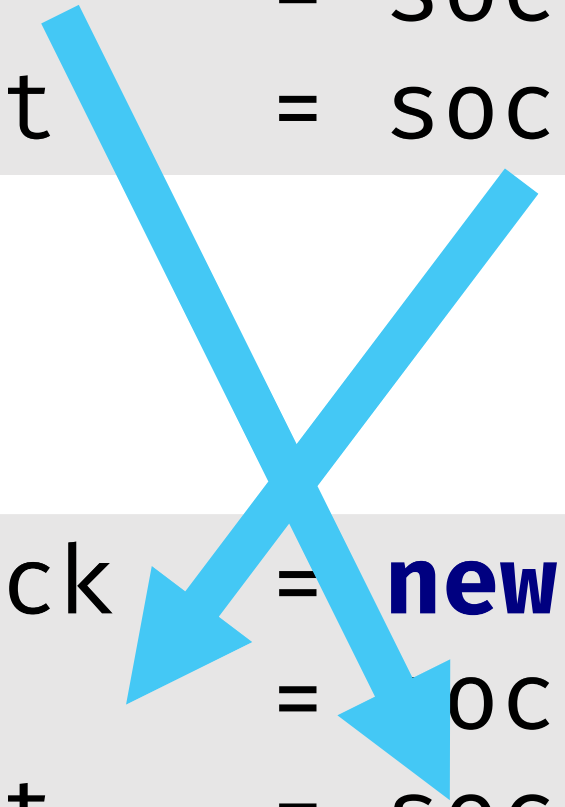
Sockets

- Server

```
Socket       sock   = serverSock.accept();
InputStream  in     = sock.getInputStream();
OutputStream out    = sock.getOutputStream();
```

- Client

```
Socket       sock   = new Socket("localhost", 1234);
InputStream  in     = sock.getInputStream();
OutputStream out    = sock.getOutputStream();
```



Sending and receiving messages over the sockets

```java
Socket sock = new Socket("localhost", 1234);

Scanner in = new Scanner(sock.getInputStream());
PrintStream out = new PrintStream(sock.getOutputStream(), true);
```

Simple example: echo server

```
try (ServerSocket serverSocket = new ServerSocket(1234)) {
    while (true) {
        try (Socket socket = serverSocket.accept()) {
            Scanner in = new Scanner(socket.getInputStream());
            PrintStream out = new PrintStream(socket.getOutputStream());

            while (in.hasNextLine()) {
                String line = in.nextLine();
                out.println(line.toUpperCase());
            }
        }
    }
} catch (IOException ex) {
     …
}
```

SERVER

Simple example: echo server

```
try (Socket socket = new Socket("localhost", 1234)) {
    Scanner in = new Scanner(socket.getInputStream());
    PrintStream out = new PrintStream(socket.getOutputStream());

    Scanner kbd = new Scanner(System.in);       ← keyboard input

    String line = kbd.nextLine();

    while (!line.equals("STOP")) {
        out.println(line);
        String response = in.nextLine();
        System.out.println(response);
        line = kbd.nextLine();
    }
} catch (IOException ex) {
      …
}
```

CLIENT

Sending and receiving custom objects as messages

- We need to tell Java objects can be converted to byte streams for transmission over the wire…
- Solution: serialization!

```
public class Product implements Serializable {
        …
}
```

Serialization: Let Java do the conversion for us

- We need to tell Java that an object can be (de)serialized
- This is done by implementing the **Serializable** interface on our POJOs
- That is it: no need to do anything else (the **Serializable interface** does not require any method to be implemented.

```java
public class Product implements
Serializable {

    private int id;
    private String name;
    private double price;

// ...
```

Everything you want to persist/transmit must be serializable!

```
public class Product implements Serializable {

    private int id; OK
    private String name; OK
    private double price; OK
    private List<Review> reviews = new
ArrayList<>();    NOK
      OK


// ...

    public class Review { }
                NOK
```

Everything you want to persist/transmit must be serializable!

```
public class Product implements Serializable {

    private int id; OK
    private String name; OK
    private double price; OK
    private List<Review> reviews = new
ArrayList<>();     OK
      OK


// ...

    public class Review implements Serializable { }
                    OK
```

Everything you want to persist/transmit must be serializable!

```
public class Product implements Serializable {

    private int id;
    private String name;
    private double price;
    private List<Review> reviews = new ArrayList<>();
    // small detail: List is not necessarily Serializable, but we know it is an ArrayList and
Arraylist (and LinkedList) is Serializeable.
```

OK

OK

OK

OK

```
//  public class Review implements Serializable {
```

Pay attention to security!

Note: Deserialization of untrusted data is inherently dangerous and should be avoided.

Java Serialization provides an interface to classes that sidesteps the field access control mechanisms of the Java language. As a result, care must be taken when performing serialization and deserialization. Furthermore, deserialization of untrusted data should be avoided whenever possible, and should be performed carefully when it cannot be avoided.

https://www.oracle.com/technetwork/java/seccodeguide-139067.html#8

- Guideline 8-1 / SERIAL-1: Avoid serialization for security-sensitive classes
- Guideline 8-2 / SERIAL-2: Guard sensitive data during serialization
- Guideline 8-3 / SERIAL-3: View deserialization the same as object construction
- Guideline 8-4 / SERIAL-4: Duplicate the SecurityManager checks enforced in a class during serialization and deserialization
- Guideline 8-5 / SERIAL-5: Understand the security permissions given to serialization and deserialization
- **Guideline 8-6 / SERIAL-6: Filter untrusted serial data**

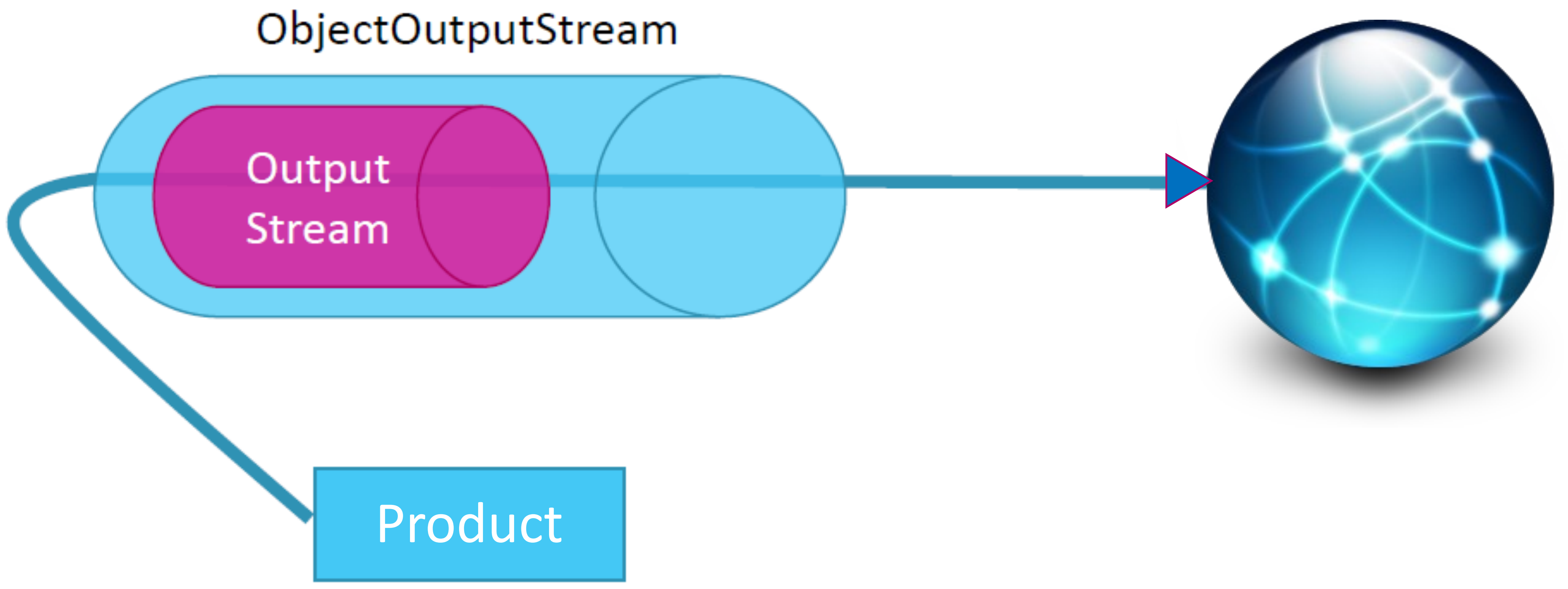
Attention points

- Not limited to network, you can also store your objects in the database for example or persist them to a file. The other side just needs to know Java as well.
- Both techniques are examples of serialization.
- Manual (previous class) ⇔ using built-in Java mechanism (this class).

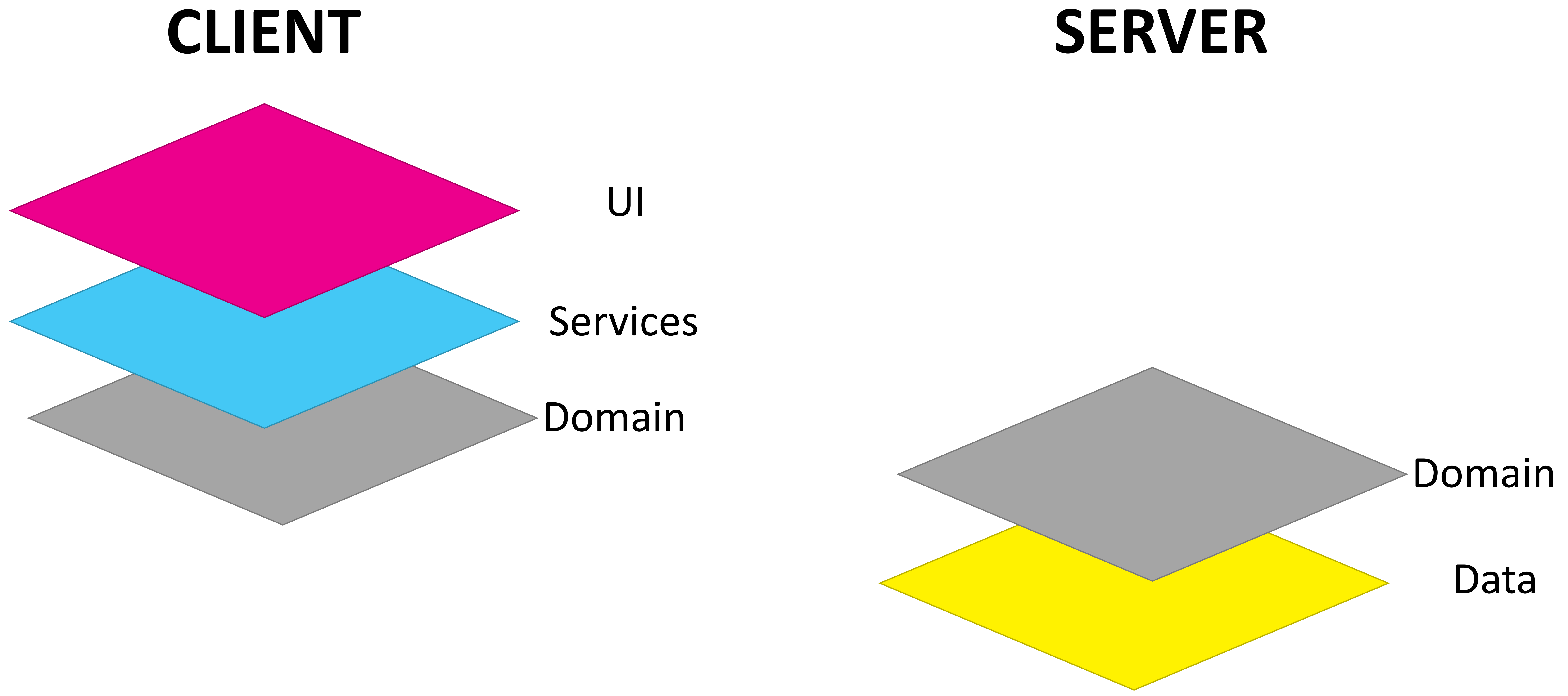
- Manual serialization results in human-readable files, which can in certain situations be considered an advantage over built-in serialization.
- For other user cases, built-in serialization may be more interesting…
- However, appending is not possible. You need to (load and) **re-**store a whole object, or in the case of appending, a list !

Using ObjectOutputStream and ObjectInputStream

- Java will do the conversion for us if we **decorate** our OutputStreams and InputStreams with ObjectOutputStream and ObjectInputStream, respectively.
- *Decorating is design pattern which "adds new functionality" to an object by wrapping it in another (more powerful) object.*



Writing our client-server product application



Server

Product.java

```
public class Product implements Serializable { ... }
```

Server.java

```
serverSocket = new ServerSocket(1234);

while (true) {
    try (Socket socket = serverSocket.accept();
         ObjectOutputStream oos = new ObjectOutputStream(socket.getOutputStream());
    ) {
        oos.writeObject(Repositories.getProductRepository().getProducts());
    } catch (IOException ex) {
        LOGGER.log(Level.WARNING, "Exception during communication with client.", ex);
    }
}
```

Client: just plug in a new implementation of ProductRepository

```
public class NetworkProductRepository implements ProductRepository {
    private static final Logger LOGGER = Logger.getLogger(NetworkProductRepository.class.getName());

    @Override
    public List<Product> getProducts() {
        try (Socket socket = new Socket("localhost", 1234);
              ObjectInputStream ois = new ObjectInputStream(socket.getInputStream())) {
            List<Product> products = (List<Product>) ois.readObject();
            return products;
        } catch (IOException | ClassNotFoundException ex) {
            LOGGER.log(Level.SEVERE, "Unable to retrieve products from network.", ex);
            throw new ShopException("Unable to retrieve products.");
        }
    }
}
```

We know it is a list of products.