DISTRIBUTED AND CLOUD COMPUTING

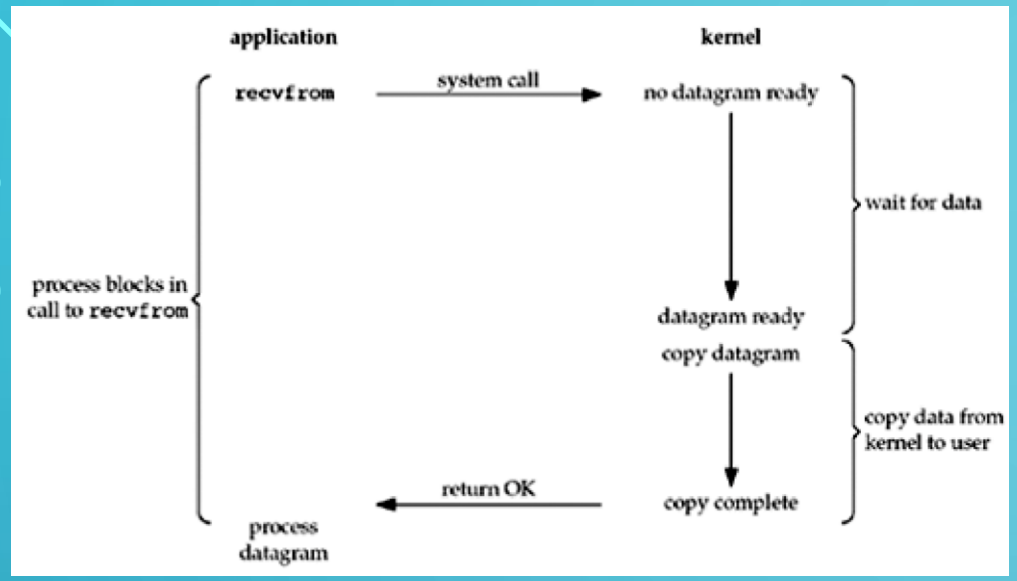
LAB5 NETWORK COMMUNICATION AND SERIALIZATION

NETWORK PROGRAMMING

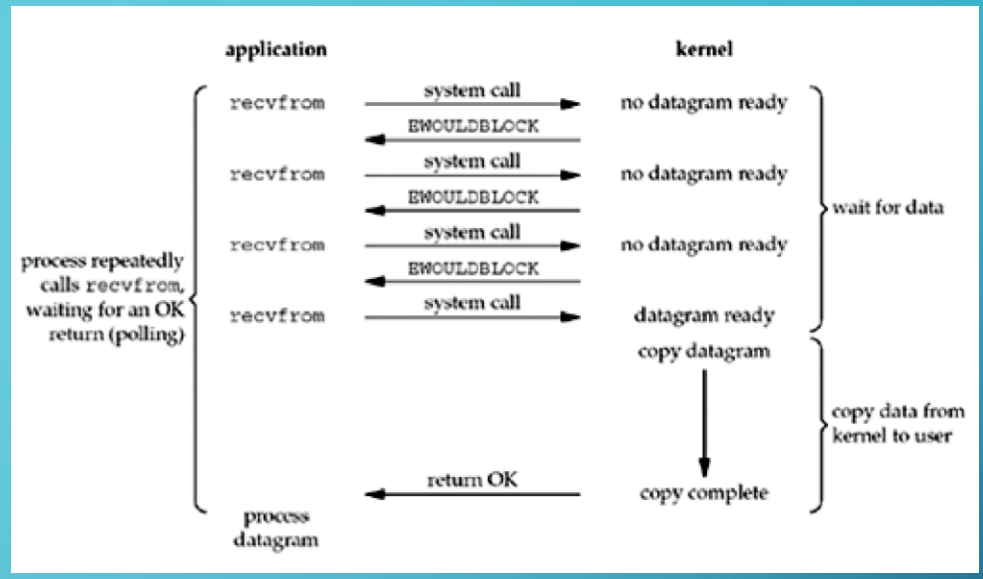
- Five I/O models under *nix:
  - Blocking I/O
  - Non-blocking I/O
  - I/O multiplexing (select/poll)
  - Signal driven I/O (SIGIO)
  - Asynchronous I/O
- They differ by behaviors in two distinct phases for input:

  1. Waiting for the data to be ready. This involves waiting for data to arrive on the network. When the packet arrives, it is copied into a buffer within the kernel

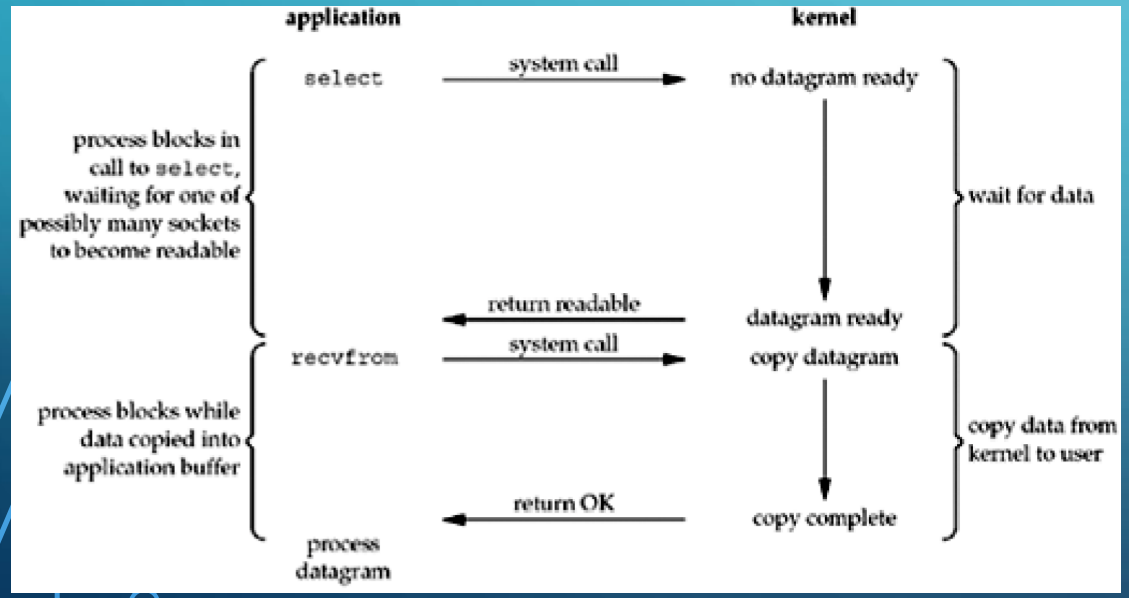
  2. Copying data from kernel to process. This means copying the (ready) data from the kernel's buffer into our application buffer



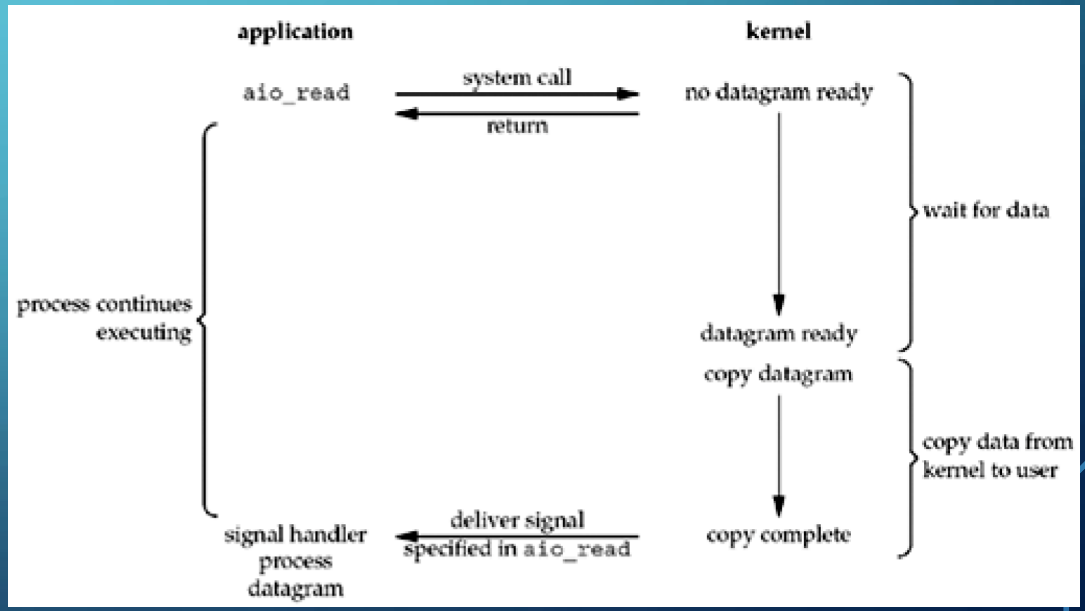
Blocking



Non-blocking

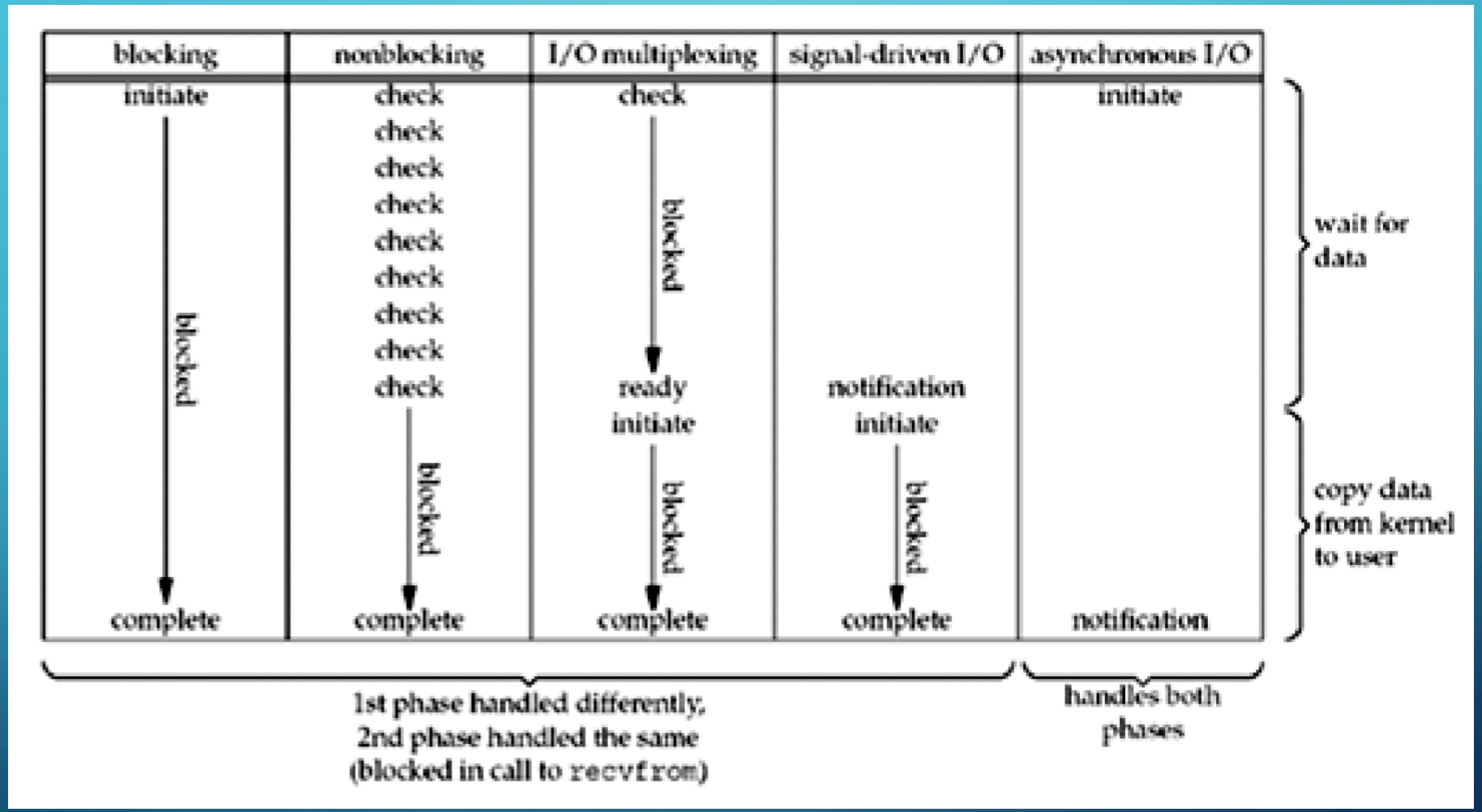


IO multiplexing



Asynchronous IO

Ref: UNIX® Network Programming Volume 1, Third Edition: The Sockets Networking Chapter 6



Ref: UNIX® Network Programming Volume 1, Third Edition: The Sockets Networking Chapter 6
https://notes.shichao.io/unp/ch6/

SERIALIZATION

- The process of converting an object or data structure into a format that can be easily transmitted or stored, typically as a sequence of bytes. The resulting serialized data can then be transmitted across a network, written to disk, or stored in a database. The reverse process of converting the serialized data back into an object or data structure is called deserialization.
- Commonly used protocols:
  - XML (Text, human-readable)
  - JSON (Text, human-readable)
  - Protobuf (Binary)
  - Avro (JSON/Binary)
  - Thrift (Binary, actually an RPC framework)
  - Java Object Serialization Stream Protocol (Binary)

https://www.w3schools.com/js/js_json_intro.asp

JSON VS. XML

```
{"employees":[
  { "firstName":"John", "lastName":"Doe" },
  { "firstName":"Anna", "lastName":"Smith" },
  { "firstName":"Peter", "lastName":"Jones" }
]}
```

JSON

```
<employees>
  <employee>
    <firstName>John</firstName> <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName> <lastName>Smith</lastName>
  </employee>
  <employee>
    <firstName>Peter</firstName> <lastName>Jones</lastName>
  </employee>
</employees>
```

XML

JAVA SERIALIZATION

- Java's Serializable interface
- Object is serialized using Java's built-in object binary stream
- Relatively large in size, slow in speed and not cross-language

JAVA SERIALIZATION

```
public class Employee implements java.io.Serializable {
   public String name;
   public String address;
   public transient int SSN;
   public int number;

   private static final long serialVersionUID = 1234567L;

   public void mailCheck() {
      System.out.println("Mailing a check to " + name + " " + address);
   }
}
```

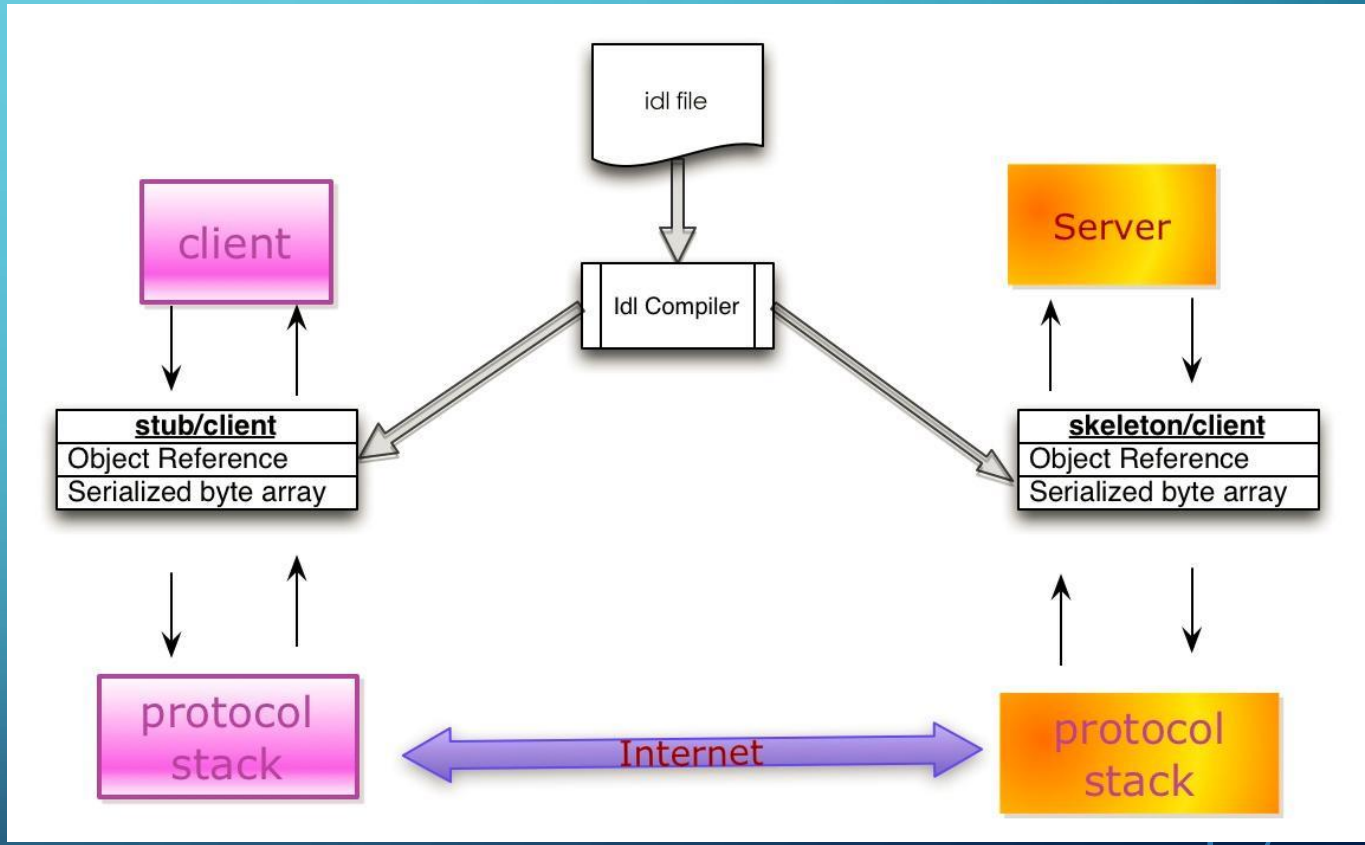
https://www.tutorialspoint.com/java/java_serialization.htm

```
import java.io.*;
public class SerializeDemo {

   public static void main(String [] args) {
      Employee e = new Employee();
      e.name = “Reyan Ali”;
      e.address = “Phokka Kuan, Ambehta Peer”;
      e.SSN = 11122333;
      e.number = 101;

      try {
         FileOutputStream fileOut = new FileOutputStream("/tmp/employee.ser");
         ObjectOutputStream out = new ObjectOutputStream(fileOut);
         out.writeObject(e);
         out.close();
         fileOut.close();
         System.out.printf("Serialized data is saved in /tmp/employee.ser");
      } catch (IOException i) {
         i.printStackTrace();
      }
   }
}
```

```
import java.io.*;
public class DeserializeDemo {
   public static void main(String [] args) {
      Employee e = null;
      try {
         FileInputStream fileIn = new FileInputStream("/tmp/employee.ser");
         ObjectInputStream in = new ObjectInputStream(fileIn);
         e = (Employee) in.readObject();
         in.close();
         fileIn.close();
      } catch (IOException i) {
         i.printStackTrace();
         return;
      } catch (ClassNotFoundException c) {
         System.out.println("Employee class not found");
         c.printStackTrace();
         return;
      }
      System.out.println("Name: " + e.name);
      System.out.println("Address: " + e.address);
      System.out.println("SSN: " + e.SSN);
      System.out.println("Number: " + e.number);
   }
}
```

CROSS-LANGUAGE SERIALIZATION

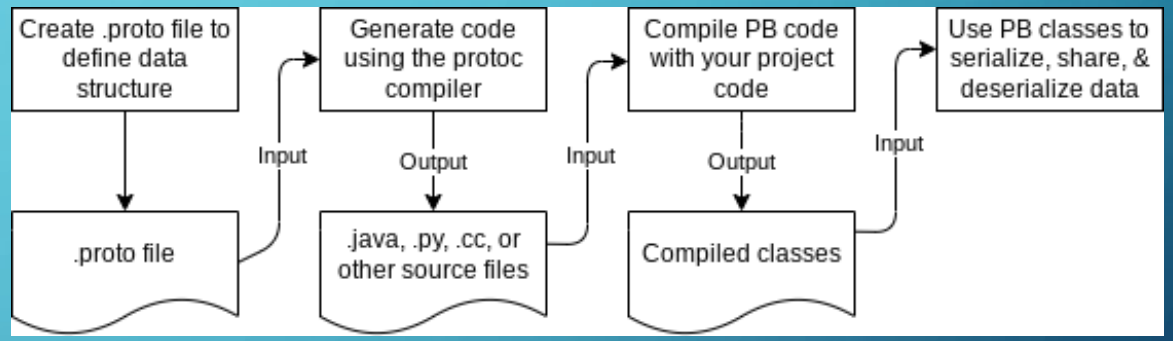
- IDL: **I**nterface **D**escription **L**anguage
- IDL compiler: Convert IDL to dynamic libraries for different languages



https://www.baeldung.com/google-protocol-buffer
https://protobuf.dev/overview/

EXAMPLE OF IDL: PROTOBUF

```
syntax = "proto2";
package protobuf;

option java_package = "com.baeldung.protobuf";
option java_outer_classname = "AddressBookProtos";

message Person {
   required string name = 1;
   required int32 id = 2;
   optional string email = 3;
   repeated string numbers = 4;
}

message AddressBook {
   repeated Person people = 1;
}
```




```
$ protoc -I=. --java_out=. addressbook.proto
```

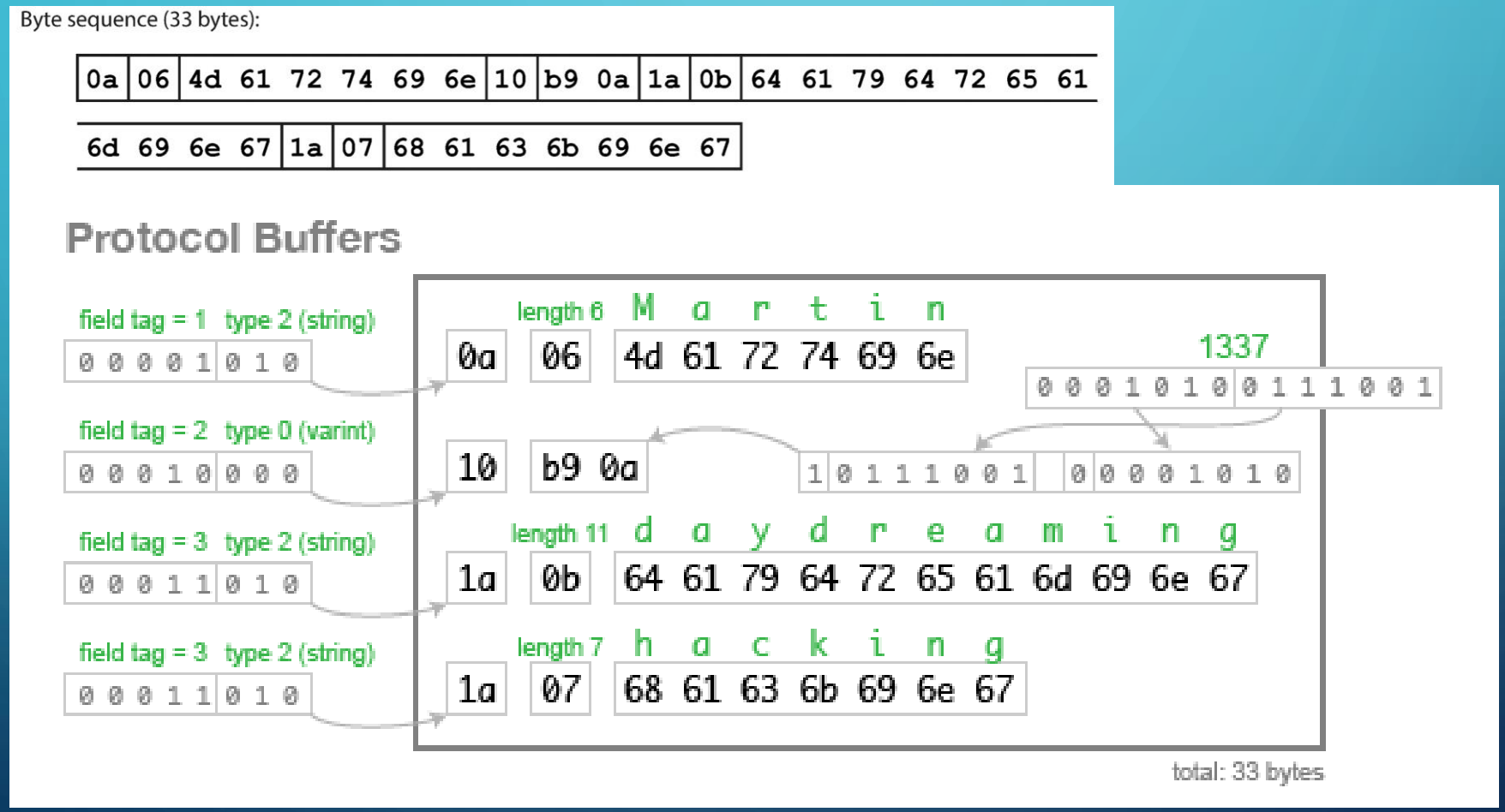
EXAMPLE OF IDL: PROTOBUF(2)

```
String email = "j@baeldung.com";
int id = new Random().nextInt();
String name = "Michael Program";
String number = "01234567890";

AddressBookProtos.Person person =
  AddressBookProtos.Person.newBuilder()
    .setId(id)
    .setName(name)
    .setEmail(email)
    .addNumbers(number)
    .build();
```

```
AddressBookProtos.AddressBook addressBook =
        AddressBookProtos.AddressBook.newBuilder()
        .addPeople(person).build();
FileOutputStream fos = new FileOutputStream(filePath);
addressBook.writeTo(fos);


AddressBookProtos.AddressBook deserialized =
      AddressBookProtos.AddressBook.newBuilder()
      .mergeFrom(new FileInputStream(filePath)).build();
```

SCHEMA AND DATA

```
{
        "userName": "Martin",
        "favouriteNumber": 1337,
        "interests": ["daydreaming", "hacking"]
}
```



https://www.oreilly.com/library/view/designing-data-intensive-applications/9781491903063/ch04.html

TUTORIALS

- https://protobuf.dev/overview/
- https://protobuf.dev/getting-started/javatutorial/