Cauvery College for Women (Autonomous)

Nationally Accredited (III Cycle) with ‘A’ Grade by NAAC

Annamalai Nagar, Tiruchirappalli-18.

Name of the Faculty	: Dr Sinthu Janita
Designation	: Professor & Head
Department	: Computer Science
Contact Number	: 9894484436
Programme	: MSc Computer Science
Batch	: 2016-2017 Onwards
Semester	: IV
Course	: Big Data Analytics
Course Code	:P16CSE5A
Unit	: V
Topics Covered	: Data Serialization And Working With Common Serialization Formats, Data Serialization

Formats

DATA SERIALIZATION AND WORKING WITH COMMON SERIALIZATION FORMATS

R.MADHUMATHI

DATA SERIALIZATION

Data serialization is the process of converting data objects present in complex data structures into a byte stream for storage,transfer and distribution purposes on physical devices.

Once the serialized data is transmitted the reverse process of creating objects from the byte sequence called *deserialization.*

Eg:

XML

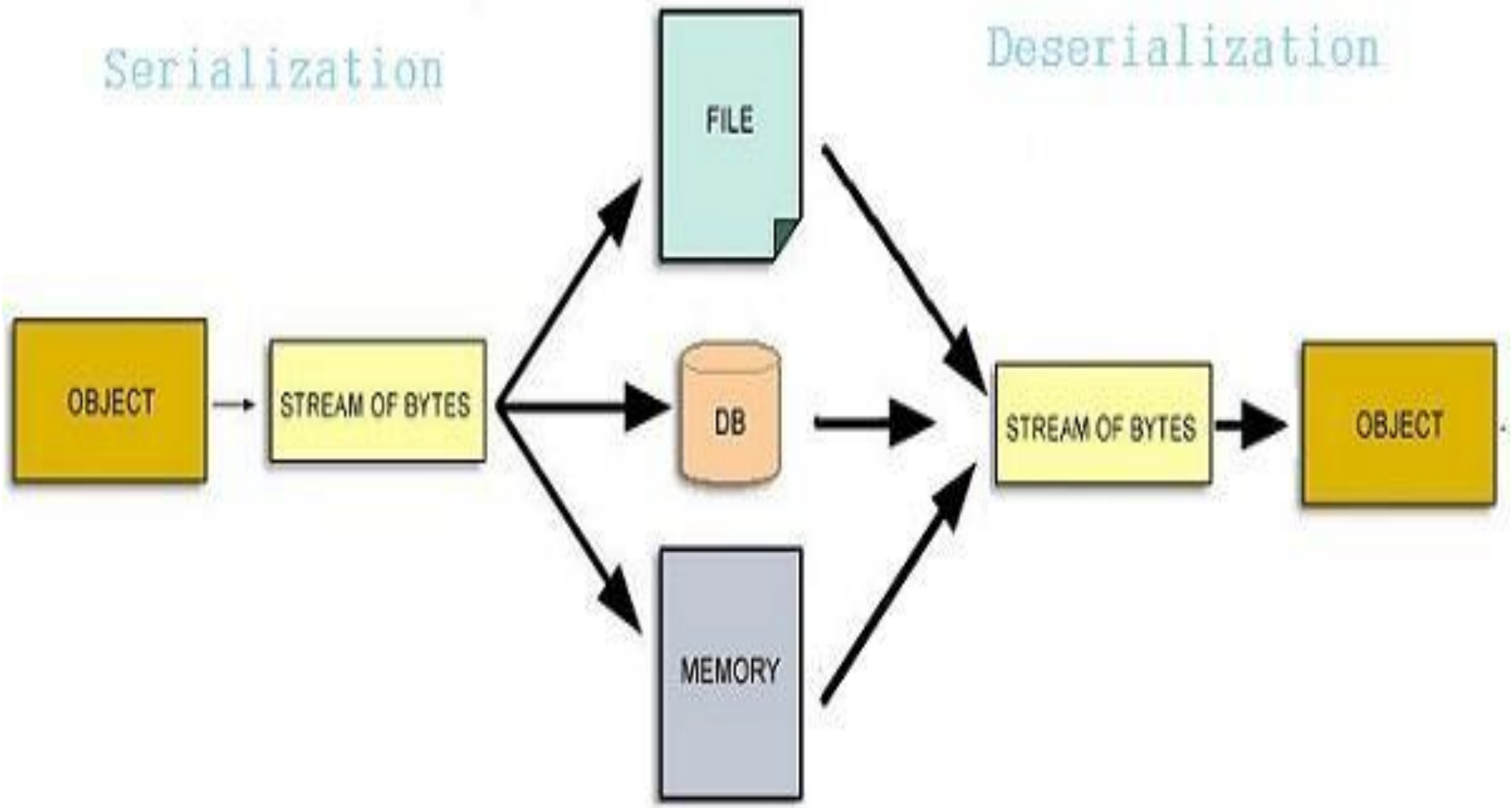
JSON

BSON

YAML

protobuf
Protocol Buffers

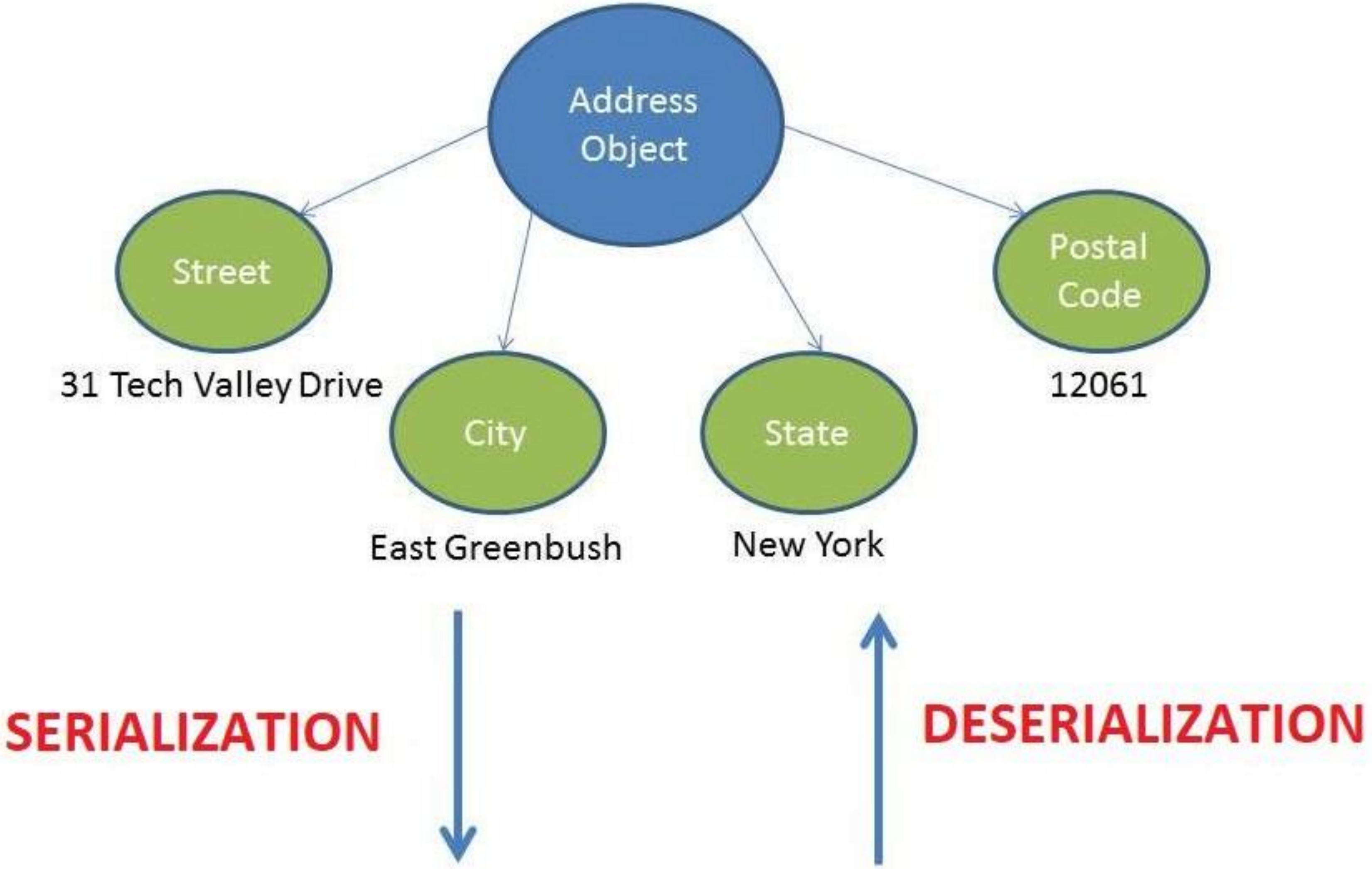
MessagePack


Serialization
Deserialization
OBJECT
STREAM OF BYTES
FILE
DB
MEMORY
STREAM OF BYTES
OBJECT

HOW IT WORKS?

- Computer data is generally organized in data structures such as arrays, tables, trees, classes. When data structures need to be stored or transmitted to another location, such as across a network, they are serialized.

- Serialization becomes complex for nested data structures and object references.

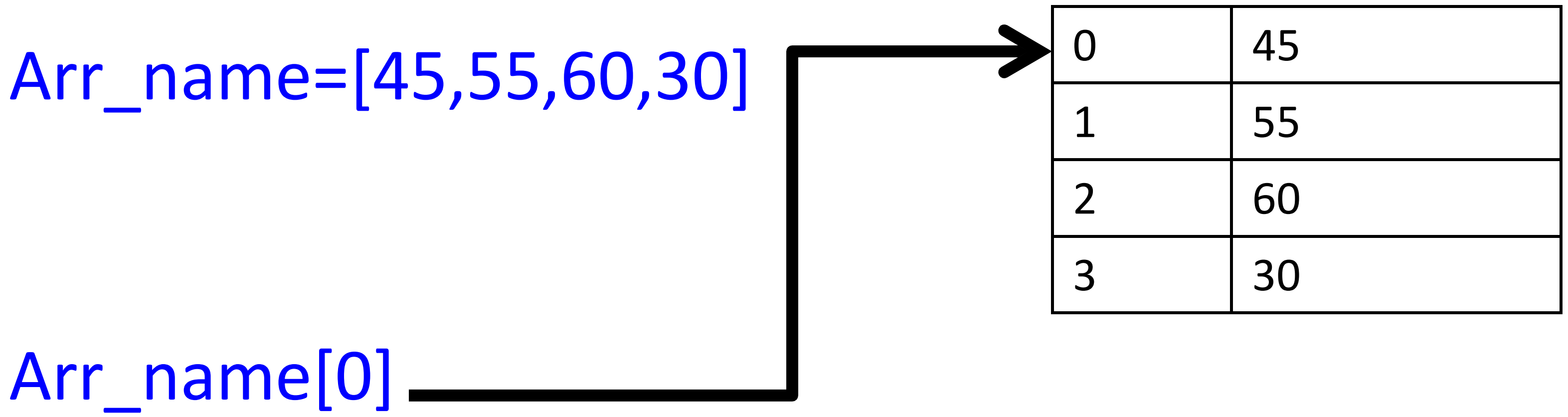


<Address><Street>31 Tech Valley Drive</Street><City>East Greenbush</City><State>New York</State><Postal Code>12061</Postal Code></Address>

Simple Array

Memory(RAM)

Arr_name=[45,55,60,30]



0	45
1	55
2	60
3	30

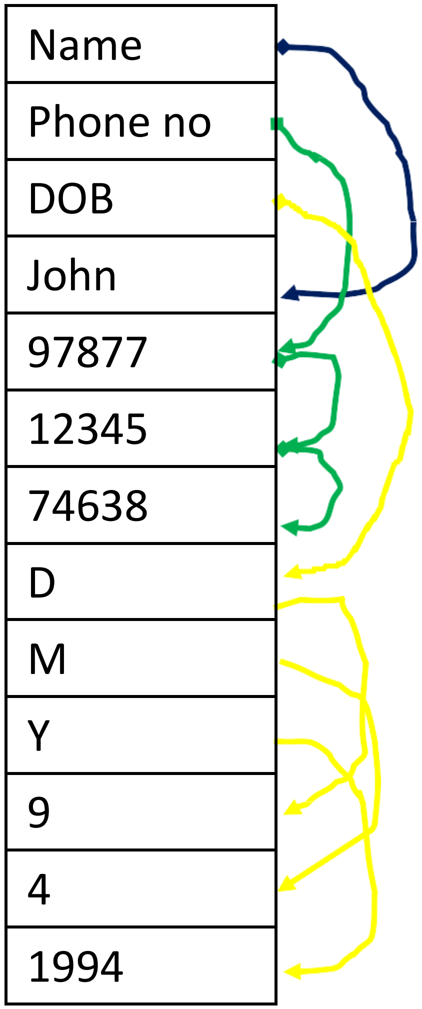
Arr_name[0]

SERIALIZED FORMAT

{45,55,60,30}

COMPLICATED STRUCTURE

Name:	John	
Phone no:	97877	
	12345	
	74638	
DOB:	D:	9
	M:	4
	Y:	1994



SERIALIZED FORMAT

{“Name”: “ John”,
“Phone no”: [97877,
12345,
74638]
“DOB”: {“ D”: 9,
“ M”: 4,
“ Y” : 1994}}

JSON

IN ONE LINE

{“Name”: “John”,“Phone no”: [97877,12345,74638”],“DOB”: {“ D”: 9,“ M”: 4,“ Y” : 1994}}

JSON

JAVA SCRIPT OBJECTNOTATION

```
{"Name": " John",
"Phone no": [ 97877,
              12345,
              74638]
"DOB":   {" D": 9,
          " M": 4,
          " Y" : 1994}}
```

, SEPARATOR
[] LIST OF ELEMENTS
{} KEY-VALUE PAIR

YAML

YAML Ain't Markup Language

```
Name: John
Phone no:
          97877
          12345
          74638
DOB:
          D: 9,
          M: 4,
          Y : 1994
```

SPACING DENOTES THE LEVEL OF NESTING

XML

EXTENSIBLE MARKUP LANGUAGE

```
<Name> John</Name>
<Phone no>97877</Phone no>
<Phone no>12345</Phone no>
<Phone no>74638</Phone no>
<DOB>
          <D> 9</D>
          <M>4</M>
          <Y>1994</Y>
</DOB>
```

BOTH ARE MARKUP LANGUAGES

APPLICATIONS OF DATA SERIALIZATION

- Serialization allows a program to save the state of an object and recreate it when needed.
- **Persisting data onto files** – happens mostly in language-neutral formats such as CSV or XML. However, most languages allow objects to be serialized directly into binary using APIs
- **Storing data into Databases** – when program objects are converted into byte streams and then stored into DBs, such as in Java JDBC.
- **Transferring data through the network** – such as web applications and mobile apps passing on objects from client to server and vice versa.
- **Sharing data in a Distributed Object Model** – When programs written in different languages need to share object data over a distributed network .However, SOAP, REST and other web services have replaced these applications now.

POTENTIAL RISK DUE TO SERIALIZATION

- It may allow a malicious party with access to the serialization byte stream to read private data, create objects with illegal or dangerous state, or obtain references to the private fields of deserialized objects. Workarounds are tedious, not guaranteed.
- Open formats too have their **security issues**.
- XML might be tampered using external entities like macros or unverified schema files.
- JSON data is vulnerable to attack when directly passed to a JavaScript engine due to features like JSONP requests.

protobuf
Protocol Buffers

AVRO™

YAML

MessagePack

CSV

BSON

XML

DATA SERIALIZATION FORMATS

R.MADHUMATHI

Serialization Formats in Hadoop

- XML
- CSV
- YAML
- JSON
- BSON
- MessagePack
- Thrift
- Protocol buffers
- Avro

TEXT-BASED DATA SERIALIZATION FORMATS

XML (Extensible Markup Language) :

- Nested textual format. Human-readable and editable.
- Schema based validation.
- Used in metadata applications, web services data transfer, web publishing.

CSV (Comma-Separated Values) :

- Table structure with delimiters.
- Human-readable textual data.
- Opens as spreadsheet or plaintext.
- CSV file is the most commonly used data file format.
- Easy to read, Easy to parse, Easy to export data from an RDBMS table.

It has three major drawbacks when used for HDFS.

1. All lines in a CSV file is a record, therefore, we should not include any headers or footers. In other word, CSV file cannot be stored in HDFS with any meta data.
2. CSV file has very limited support for schema evolution. Because the fields for each record are ordered, we are not able to change the orders. We can only append new fields to the end of each line.
3. It does not support block compression which many other file formats support. The whole file has to be compressed and decompressed for reading, adding a significant read performance cost to the files.

JSON (JavaScript Object Notation) :

- Short syntax textual format with limited data types.
- Human-readable. Derived from JavaScript data formats.
- No need of a separate parser (like XML) since they map to JavaScript objects. No direct support for DATE data type. All data is dynamically processed
- It is in text format that stores meta data with the data, so it fully supports schema evolution and also spiltable.
- We can easily add or remove attributes for each datum. However, because it’s text file, it doesn’t support block compression.

YAML Ain't Markup Language :

- It is a data serialization language which is designed to be human -friendly and works well with other programming languages for everyday tasks.
- Superset of JSON
- Supports complex data types. Maps easily to native data structures.

Binary JSON:

- It is a binary-encoded serialization of JSON-like documents.
- MongoDB uses BSON ,when storing documents in collections
- It deals with attribute-value pairs like JSON. Includes datetime, bytearray and other data types not present in JSON.

- It is Created by Google
- It is Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data
- Protocol buffers currently support generated code in Java, Python, Objective-C, and C++.

- It is designed for data to be transparently converted from/to JSON.
- Support rich set of data structures
- It create schema based annotation
- Primary use is network communication

AVRO

- Apache **Avro** is a language-neutral data
- serialization system, developed by **Doug Cutting.**

  the father of Hadoop.
- It also called a schema-based serialization technique.

FEATURES

- Avro uses JSON format to declare the data structures. Presently, it supports languages such as Java, C, C++, C#, Python, and Ruby.
- Avro creates binary structured format that is both **compressible** and **splitable**. Hence it can be efficiently used as the input to Hadoop MapReduce jobs.
- Avro provides **rich data structures**.

- Avro **schemas** defined in **JSON**, facilitate implementation in the languages that already have JSON libraries.
- Avro creates a self-describing file named *Avro Data File,* in which it stores data along with its schema in the metadata section.
- Avro is also used in Remote Procedure Calls (RPCs).

Thrift and **Protocol Buffers** are the most competent libraries with Avro. Avro differs from these frameworks in the following ways

- Avro supports both dynamic and static types as per the requirement. Protocol Buffers and Thrift use Interface Definition Languages (IDLs) to specify schemas and their types. These IDLs are used to generate code for serialization and deserialization.
- Avro is built in the Hadoop ecosystem. Thrift and Protocol Buffers are not built in Hadoop ecosystem.

PERFORMANCE CHARACTERISTICS

- Speed – Binary formats are faster than textual formats. A late entrant, **protobuf reports the best times**. JSON is preferable due to readability and being schema-less.
- Data size – This refers to the physical space in bytes post serialization. For small data, compressed JSON data occupies more space compared to binary formats like protobuf. Generally, **binary formats always occupy less space**.
- Usability – Human readable formats like JSON are naturally preferred over binary formats. For editing data, YAML is good. Schema definition is easy in protobuf, with in-built tools.
- Compatibility-Extensibility – JSON is a closed format. XML is average with schema versioning. Backward compatibility (extending schemas) is best handled by protobuf.