Chapter X: Big Data

Database System Concepts, 6th Ed.

Chapter X: Big Data

- Map Reduce
  - The Map reduce paradigm
  - Distributed File Systems
  - Hadoop
- Big Data Storage Systems

The MapReduce Paradigm

- Platform for reliable, scalable parallel computing
- Abstracts issues of distributed and parallel environment from programmer.
- Paradigm dates back many decades
  - But very large scale implementations running on clusters with 10^3 to 10^4 machines are more recent
  - Google Map Reduce, Hadoop, ..
- Data access done using distributed file systems

Distributed File Systems

- Highly scalable distributed file system for large data-intensive applications.
  - E.g. 10K nodes, 100 million files, 10 PB
- Provides redundant storage of massive amounts of data on cheap and unreliable computers
  - Files are replicated to handle hardware failure
  - Detect failures and recovers from them
- Examples:
  - Google File System (GFS)
  - Hadoop File System (HDFS)

MapReduce: File Access Count Example

- Given log file in following format:

  ...
  2013/02/21 10:31:22.00EST /slide-dir/11.ppt
  2013/02/21 10:43:12.00EST /slide-dir/12.ppt
  2013/02/22 18:26:45.00EST /slide-dir/13.ppt
  2013/02/22 20:53:29.00EST /slide-dir/12.ppt
  ...

- Goal: find how many times each of the files in the slide-dir directory was accessed between 2013/01/01 and 2013/01/31.
- Options:
  - Sequential program too slow on massive datasets
  - Load into database expensive, direct operation on log files cheaper
  - Custom built parallel program for this task possible, but very laborious
  - Map-reduce paradigm

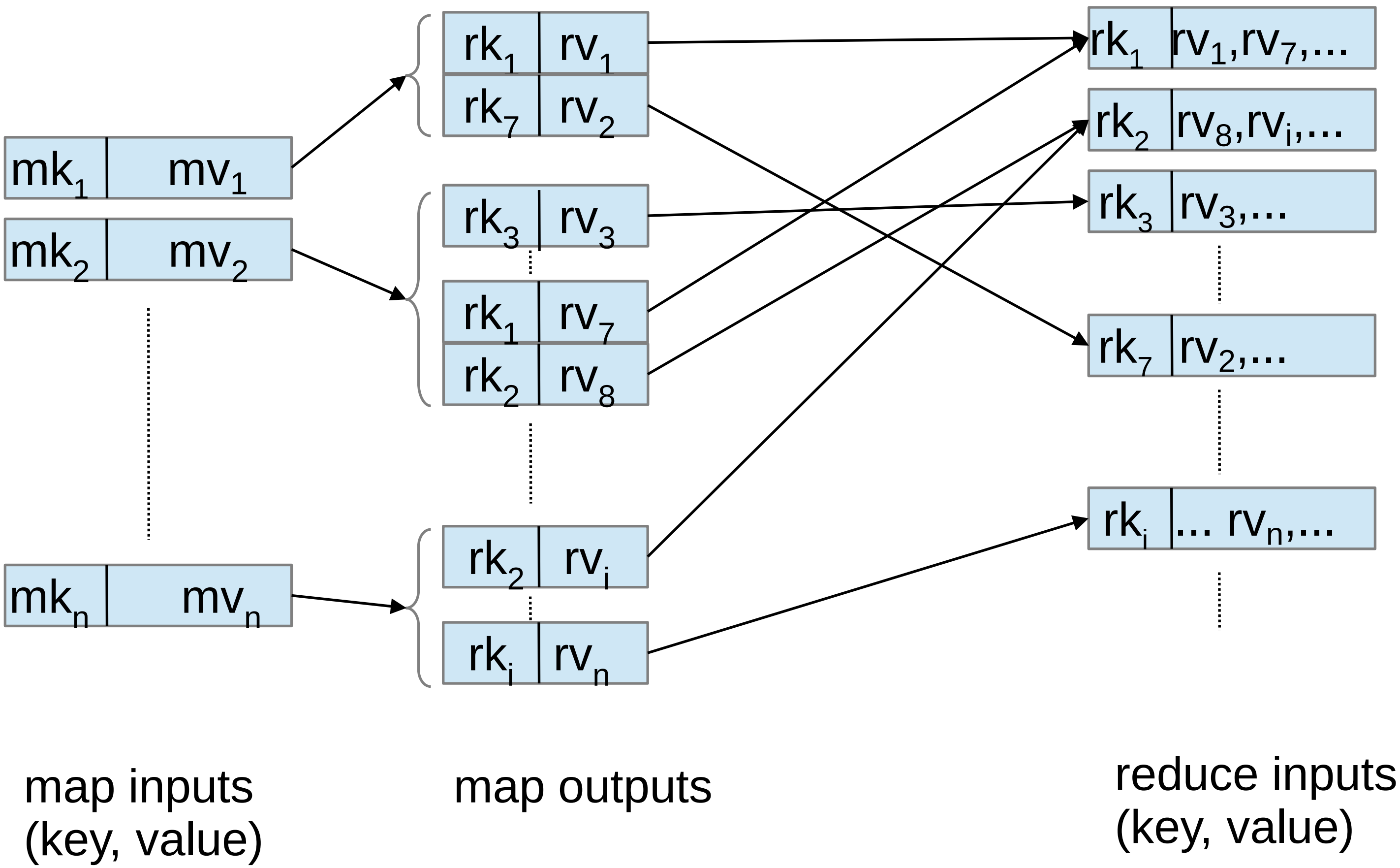
MapReduce Programming Model

- Inspired from map and reduce operations commonly used in functional programming languages like Lisp.
- Input: a set of key/value pairs
- User supplies two functions:
  - map(k,v) → list(k1,v1)
  - reduce(k1, list(v1)) → v2
- (k1,v1) is an intermediate key/value pair
- Output is the set of (k1,v2) pairs
- For our example, assume that system
  - breaks up files into lines, and
  - calls map function with value of each line
    - Key is the line number

MapReduce: File Access Count Example

```
map(String key, String record) {
    String attribute[3];
    …. break up record into tokens (based on space character), and store the
          tokens in array attributes
    String date = attribute[0];
    String time = attribute[1];
    String filename = attribute[2];
    if (date between 2013/01/01 and 2013/01/31
              and filename starts with "/slide-dir/")
        emit(filename, 1).
}
reduce(String key, List recordlist) {
    String filename = key;
    int count = 0;
    For each record in recordlist
       count = count + 1.
    output(filename, count)
}
```

Schematic Flow of Keys and Values



- **Flow of keys and values in a map reduce task**

MapReduce: Word Count Example

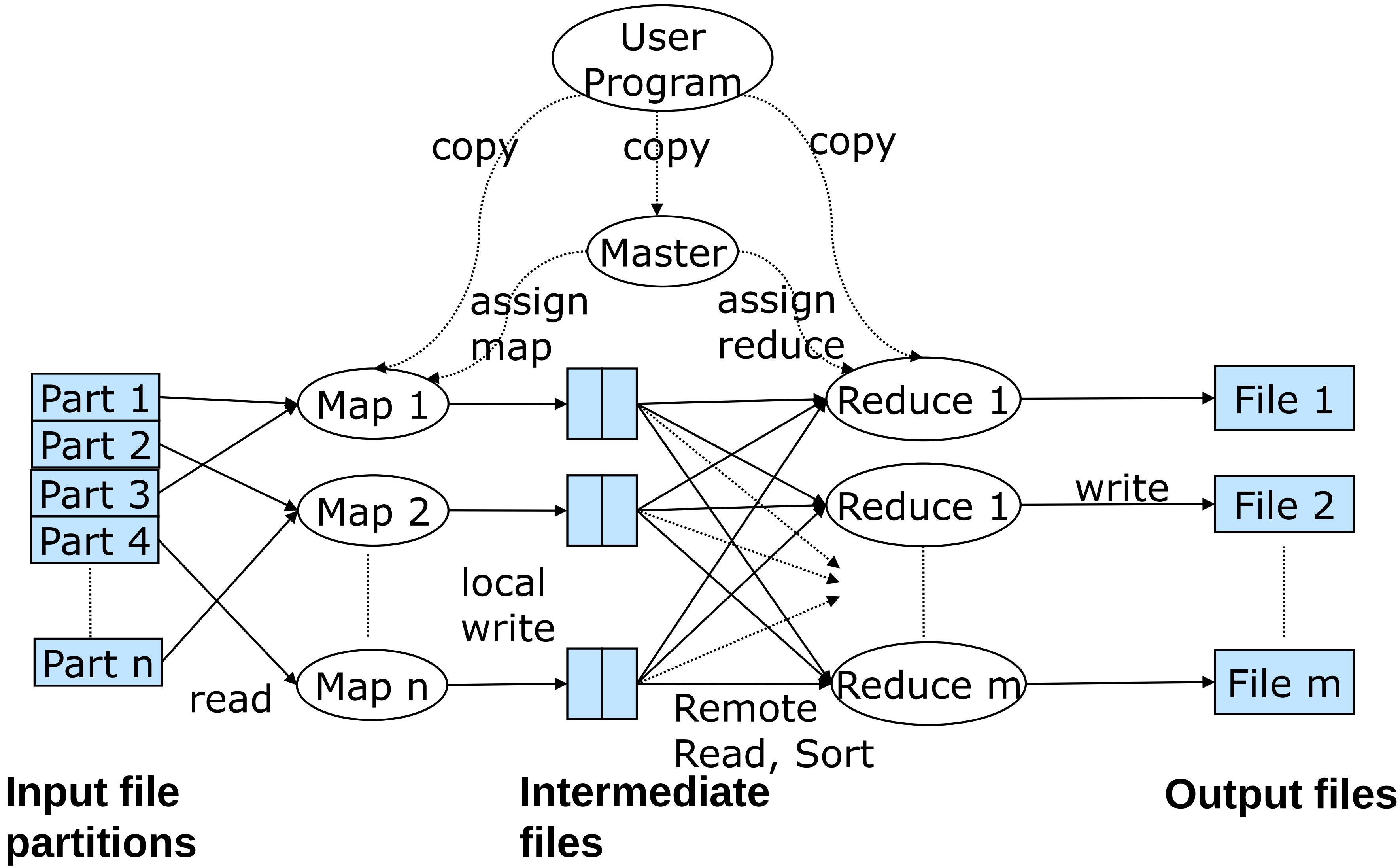
- Consider the problem of counting the number of occurrences of each word in a large collection of documents
- How would you do it in parallel ?
- Solution:
  - Divide documents among workers
  - Each worker parses document to find all words, map function outputs (word, count) pairs
  - Partition (word, count) pairs across workers based on word
  - For each word at a worker, reduce function locally add up counts
- Given input: "One a penny, two a penny, hot cross buns."
  - Records output by the map() function would be
    - ("One", 1), ("a", 1), ("penny", 1),("two", 1), ("a", 1), ("penny", 1), ("hot", 1), ("cross", 1), ("buns", 1).
  - Records output by reduce function would be
    - ("One", 1), ("a", 2), ("penny", 2), ("two", 1), ("hot", 1), ("cross", 1), ("buns", 1)

Pseudo-code

```
map(String input_key, String input_value):
// input_key: document name
// input_value: document contents
    for each word w in input_value:
        Emit(w, "1");
// Group by step done by system on key of intermediate Emit above,
// and reduce called on list of values in each group.

reduce(String output_key, Iterator intermediate_values):
// output_key: a word
// output_values: a list of counts
    int result = 0;
    for each v in intermediate_values:
        result += ParseInt(v);
    Output(result);
```

Parallel Processing of MapReduce Job



Hadoop

- Google pioneered map-reduce implementations that could run on thousands of machines (nodes), and transparently handle failures of machines
- Hadoop is a widely used open source implementation of Map Reduce written in Java
  - Map and reduce functions can be written in several different languages, we use Java.
- Input and output to map reduce systems such as Hadoop must be done in parallel
  - Google used GFS distributed file system
  - Hadoop uses Hadoop File System (HDFS)
    - File blocks partitioned across many machines
    - Blocks are replicated so data is not lost/unavailable if a machine crashes
    - Central "name node" provides metadata such as which blocks are contained in which files

Hadoop

- Types in Hadoop
  - Generic Mapper and Reducer interfaces both take four type arguments, that specify the types of the
    - input key, input value, output key and output value
  - Map class in next slide implements the Mapper interface
    - Map input key is of type LongWritable, i.e. a long integer
    - Map input value which is (all or part of) a document, is of type Text.
    - Map output key is of type Text, since the key is a word,
    - Map output value is of type IntWritable, which is an integer value.

Hadoop Code in Java: Map Function

```
public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>
{
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException
    {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                context.write(word, one);
        }
    }
}
```

Hadoop Code in Java: Reduce Function

```
public static class Reduce extends Reducer<Text, IntWritable, Text,
    IntWritable> {
   public void reduce(Text key, Iterable<IntWritable> values,
          Context context)  throws IOException, InterruptedException
   {
       int sum = 0;
       for (IntWritable val : values) {
            sum += val.get();
       }
       context.write(key, new IntWritable(sum));
   }
}
```

Hadoop Job Parameters

- The classes that contain the map and reduce functions for the job
  - set by methods setMapperClass() and setReducerClass()
- The types of the job's output key and values
  - set by methods setOutputKeyClass() and setOutputValueClass()
- The input format of the job
  - set by method job.setInputFormatClass()
    - Default input format in Hadoop is the TextInputFormat,
      - map key whose value is a byte offset into the file, and
      - map value is the contents of one line of the file
- The directories where the input files are stored, and where the output files must be created
  - set by addInputPath() and addOutputPath()
- And many more parameters

Hadoop Code in Java: Overall Program

```java
public class WordCount {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = new Job(conf, "wordcount");
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.waitForCompletion(true);
    }
}
```

Local Pre-Aggregation

- Combiners: perform partial aggregation to minimize network traffic
  - E.g. within machine
  - And/or at rack level
- In Hadoop, reduce function is used by default if combiners are enabled
  - But alternative implementation of combiner can be specified if input and output types of reducers are different

Implementations

- Google
  - Not available outside Google
- Hadoop
  - An open-source implementation in Java
  - Uses HDFS for stable storage
  - Download: http://lucene.apache.org/hadoop/
- Aster Data
  - Cluster-optimized SQL Database that also implements MapReduce
    - IITB alumnus among founders
- And several others, such as Cassandra at Facebook, etc.

Map Reduce vs. Parallel Databases

- Map Reduce widely used for parallel processing
  - Google, Yahoo, and 100's of other companies
  - Example uses: compute PageRank, build keyword indices, do data analysis of web click logs, ….
- Database people say: but parallel databases have been doing this for decades
- Map Reduce people say:
  - we operate at scales of 1000's of machines
  - We handle failures seamlessly
  - We allow procedural code in map and reduce and allow data of any type
    - many real-world uses of MapReduce that cannot be expressed in SQL.

Map Reduce vs. Parallel Databases (Cont.)

- Map Reduce is cumbersome for writing simple queries
- Current approach: declarative querying, with execution on Map Reduce infrastructure
  - Pig Latin – Declarative language supporting complex data using JSON; From Yahoo
    - Programmer has to specify parser for each input source
  - Hive – SQL based syntax; From Facebook
    - Allows specification of schema for each input source
    - Has become very popular
  - SCOPE system from Microsoft
- Many proposed extensions of Map Reduce to allow joins, pipelining of data, etc.

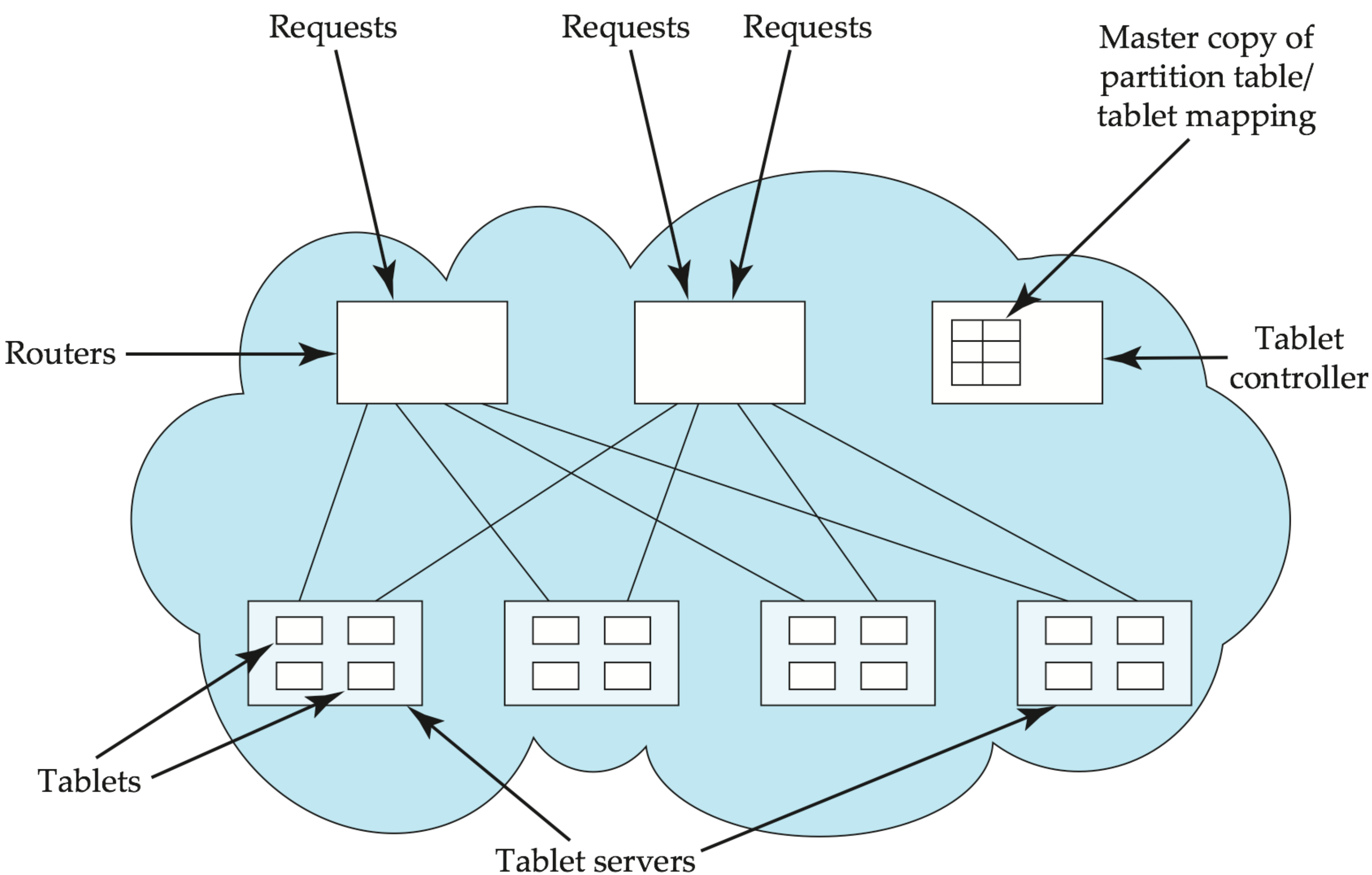
Big Data Storage Systems

- Need to store massive amounts of data
- Scalability: ability to grow the system by adding more machines
  - Scalability of storage volume, and access rate
- Distributed file systems good for scalable storage of unstructured data
  - E.g. log files
- Massively parallel database ideal for storing records
  - But hard to support all database features across thousands of machines
- Massively parallel key-value stores built to support scalable storage and access of records
  - E.g. Big Table from Google, PNUTS/Sherpa from Yahoo, HBase from Apache Hadoop project
- Applications using key-value stores manage query processing on their own

Key Value Stores

- Key-value stores support
  - put(key, value): used to store values with an associated key,
  - get(key): which retrieves the stored value associated with the specified key.
- Some systems such as Bigtable additionally provide range queries on key values
- Multiple versions of data may be stored, by adding a timestamp to the key

PNUTS Parallel Storage System Architecture



Data Representation

- Records in many big data applications need to have a flexible schema
  - Not all records have same structure
  - Some attributes may have complex substructure
- XML and JSON data representation formats widely used
- An example of a JSON object is:

```
{
    "ID": "22222",
    "name": {
            "firstname: "Albert",
            "lastname: "Einstein"
    },
    "deptname": "Physics",
    "children": [
            { "firstname": "Hans", "lastname": "Einstein" },
            { "firstname": "Eduard", "lastname": "Einstein" }
    ]
}
```