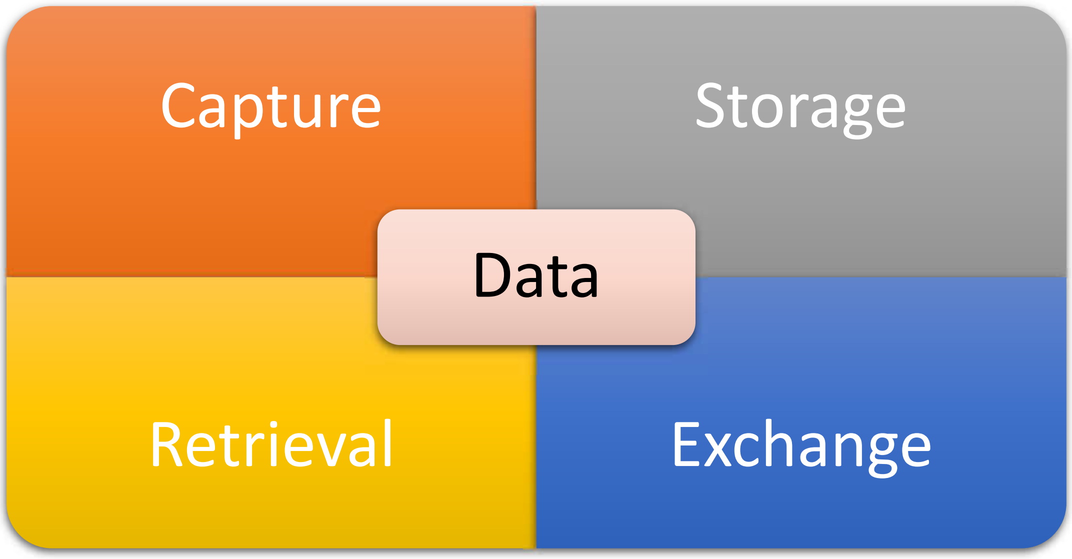
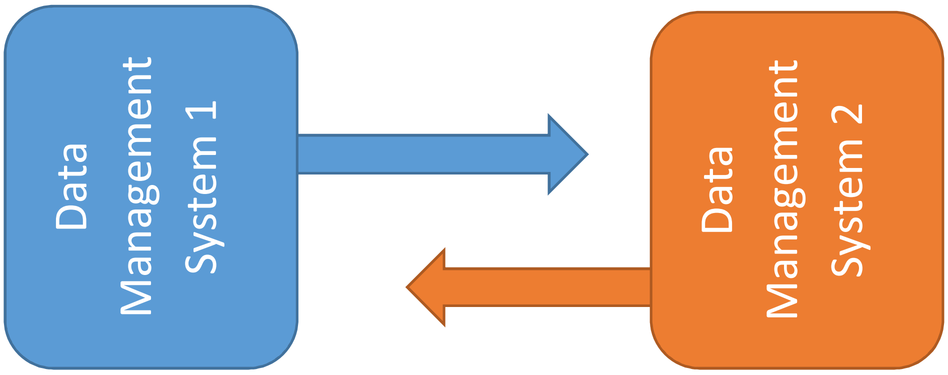
Data Exchange

Data management domains:



- **Data capture**
  - how to get data
  - various types of digitization (cameras, various scanners, microphones, etc.), calculations of statistics and metadata, data found on the Internet, filled forms from users, data mining …
- **Storage**
  - how to store data to make them available as soon as possible while keeping them safe
  - storage devices, tiered storage (from Tier 0 – critical, most wanted data – towards lower levels)
    - example: Tier 0 - SSD drives, Tier 1 – HDDs in RAIDs, Tier 2 – NAS, SAN, Tier 3 – CDs, DVDs, tapes
- **Retrieval**
  - How to find and represent the appropriate data as quickly and simply as possibly – from various types of data (documents, images, databases …)
- **Exchange**

Data Exchange



Goal:

- Transfer of data and their relationships between various systems

Challenges:

- Various structures and types of data (relational databases, spreadsheets, data warehouses, PDFs, HTML documents, presentations etc.),
- Different encodings (UTF-8, UTF-16, ASCII, ANSI, Base64)
- Incompatibility of file formats
- Security and privacy
- … and there are some more ☺

Single-purpose Exchange Systems

- Dedicated to one domain
- Examples:
  - GPX – GPS data
  - CSV, DIF – spreadsheets

Universal Exchange Systems

- They can be used for various types of data and systems
- Examples:
  - XML
  - YAML
  - JSON
  - Atom

XML – eXtensible Markup Language

- Markup language designed to describe data and their relations
- **Markup language**
    - based on **tags** – the tags can be defined by the author of the XML document (unlike e.g. HTML tags, which are predefined by W3C organisation)
    - XML tags define **XML elements –** entities of data
    - XML elements can contain
        - attributes
        - text content
        - element content (other XML elements)
- Example of the XML file[1]

```xml
<?xml version="1.0"?>
<catalog>

   <book id="bk101">
      <author>Gambardella, Matthew</author>
      <title>XML Developer's Guide</title>
      <genre>Computer</genre>
      <price currency="euro">44.95</price>
      <publish_date>2000-10-01</publish_date>
      <description>An in-depth look at creating applications
      with XML.</description>
   </book>

   <book id="bk102">
      <author>Ralls, Kim</author>
      <title>Midnight Rain</title>
      <genre>Fantasy</genre>
      <price currency="euro">5.95</price>
      <publish_date>2000-12-16</publish_date>
      <description>A former architect battles corporate zombies,
      an evil sorceress, and her own childhood to become queen
      of the world.</description>
   </book>

   <book id="bk103">
      <author>Corets, Eva</author>
      <title>Maeve Ascendant</title>
      <genre>Fantasy</genre>
      <price currency="euro">5.95</price>
      <publish_date>2000-11-17</publish_date>
      <description>After the collapse of a nanotechnology
      society in England, the young survivors lay the
      foundation for a new society.</description>
   </book>

</catalog>
```

[1]Modified version of the source: http://msdn.microsoft.com/en-us/library/ms762271%28v=vs.85%29.aspx

`<?xml version="1.0"?>`	**Version of the XML**
`<catalog>`	Beginning of the *catalog* element – its start tag
`  <book id="bk101">`	Beginning of the *book* element –its start tag; it also contains the attribute *id* of the value *bk101*
`    <author>Gambardella, Matthew</author>`	Complete element *author* with both tags – start and end ones
`    <title>XML Developer's Guide</title>`	
`    <genre>Computer</genre>`	
`    <price currency="euro">44.95</price>`	This element contains an attribute, which specifies the value of the element – the number 44.95 expresses the price in euro

XML:

- Strict syntax rules – each element must be properly closed; case-sensitive; proper nesting of elements …
- Liberal semantic rules – both examples below are correct – which is better? It is difficult to say:

```
<name first="John"
last="Doe"></name>
```

```
<name>
  <first>John</first>
  <last>Doe</last>
</name>
```

- Human-readable
- Relatively slow processing, esp. reading
- It is the basis for various data formats with limited sets of tags, e.g.
  - XHTML
  - SVG
  - Office Open XML (e.g. in DOCX, XLSX, PPTX files of Microsoft Office)
  - OpenDocument
  - GPX
  - BeerXML ☺ (exchange of brewing data)
  - MathML (notation of mathematical expressions)
  - SBML (biological processes)

JSON

JavaScript Object Notation

An alternative to XML – more limited; however, much simpler

It uses JavaScript syntax, but it is language-independent – it can be used in C/C++, Python, Perl …

Example[2]

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

Equivalent in XML:

```
<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>
```

Essential JSON syntax rules:

- Data is in name/value pairs (e.g. `"firstName":"John"`)
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

[2] http://www.w3schools.com/json/

YAML

YAML = YAML Ain't Markup Language

Essential rules:

- The hierarchy of objects is defined by their indentation (made exclusively by spaces).
- Each file starts with --- (3 dashes).
- Items are represented in pairs **key : value**
- List of values are lines at the same indentation level starting with –

Example of the YAML:

```
---
# An employee record
name: Example Developer
job: Developer
skill: Elite
employed: True
foods:
    - Apple
    - Orange
    - Strawberry
    - Mango
languages:
    ruby: Elite
    python: Elite
    dotnet: Lame
```

CSV

Comma-Separated Values

- Pseudo-standard – supported by many platforms and programs

Essential rules

- **Each record is one line** (ended by CR LF symbols in Windows, LF in POSIX systems like Linux, UNIX, MacOS – however; the end of line is ignored, if it is in a zone surrounded by quotation marks)
- **Fields are separated with commas** – the comma "," can be replaced by semicolon ";", colon ":" or Tab
- **The first record in a CSV file *may* be a header record containing column (field) names** – this is not mandatory, just recommended item in CSV files

There are problems in cases, when values contain symbols, which have special function in the CSV – the most common case is the presence of commas in addresses.

Example:

```
Transaction_date,Product,Price,Payment_Type,Name,City
1/2/09 6:17,Product1,1200,Mastercard,carolina,Basildon
1/2/09 4:53,Product1,1200,Visa,Betina,Parkville
1/2/09 13:08,Product1,1200,Mastercard,Federica e Andrea,Astoria
1/3/09 14:44,Product1,1200,Visa,Gouya,Echuca
```

Exercise

1. Create an XML file, which will represent teachers of the school (name, surname, age) incl. the subject they can teach. Then validate the created file in an online validator.
2. Create a JSON file, which would be ready to keep data about all countries of the world. The file should express the hierarchy **world → continent → country.** The data about countries should be as follows:
   name, capital city, TLD domain (e.g. Slovakia has **.sk**, Germany .**de**), calling prefix (e.g. Slovakia +421).
   Then check the file in this online validator: **http://jsonviewer.stack.hu/** .
3. Create a CSV file, which would keep data about the sights of Slovakia (name, kind – castle, church, cave … - its longitude and latitude, and admission fees for adults and children). Then try to load the file in the MS Excel/LibreOffice Calc.