Data Formats

Big Data Management

Anis Ur Rahman
Faculty of Computer Science & Information Technology
University of Malaya

September 17, 2019

Borrowed from http://www.ksi.mff.cuni.cz/~svoboda/courses/171-NDBI040/

Lecture Outline

Data formats

- **XML** – Extensible Markup Language
- **JSON** – JavaScript Object Notation
- **BSON** – Binary JSON
- **RDF** – Resource Description Framework
- **CSV** – Comma-Separated Values
- **Protocol Buffers**

Outline

Introduction

XML = Extensible Markup Language

- **Representation** and **interchange** of semi-structured data
  - a family of related technologies, languages, specifications, ...
- Derived from **SGML**, developed by W3C, started in 1996

Design goals

Simplicity, generality and usability across the Internet

- **File extension.** *.xml, content type: text/xml
- **Versions.** 1.0 and 1.1
- W3C recommendation
  - `http://www.w3.org/TR/xml11/`

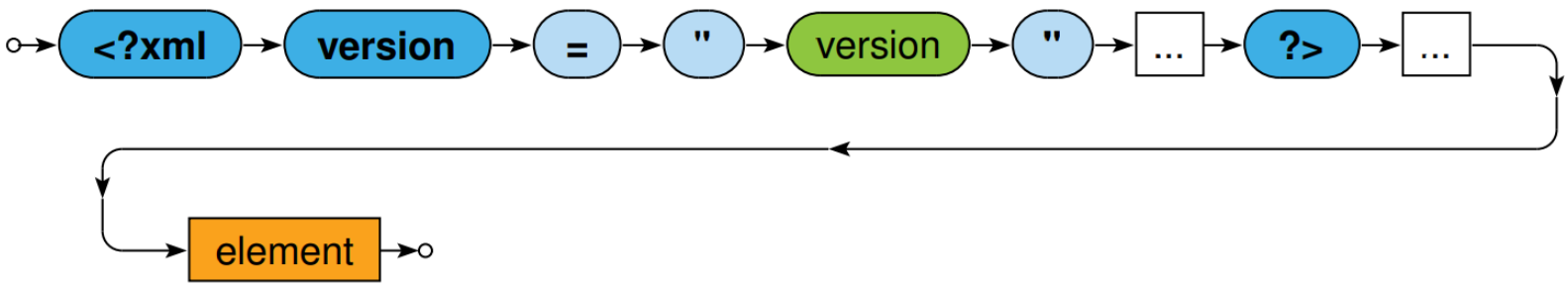
Example

```
1  <?xml version="1.1" encoding="UTF-8"?>
2  <movie year="2019">
3    <title>Murder Mystery</title>
4    <actors>
5      <actor>
6        <firstname>Adam</firstname>
7        <lastname>Sandler</lastname>
8      </actor>
9      <actor>
10       <firstname>Jennifer</firstname>
11       <lastname>Aniston</lastname>
12     </actor>
13   </actors>
14   <director>
15     <firstname>Kyle</firstname>
16     <lastname>Newacheck</lastname>
17   </director>
18 </movie>
```

Document Structure

Document

- **Prolog.** XML **version** + some other stuff
- Exactly **one root** element
  - Contains other **nested elements** and/or other content



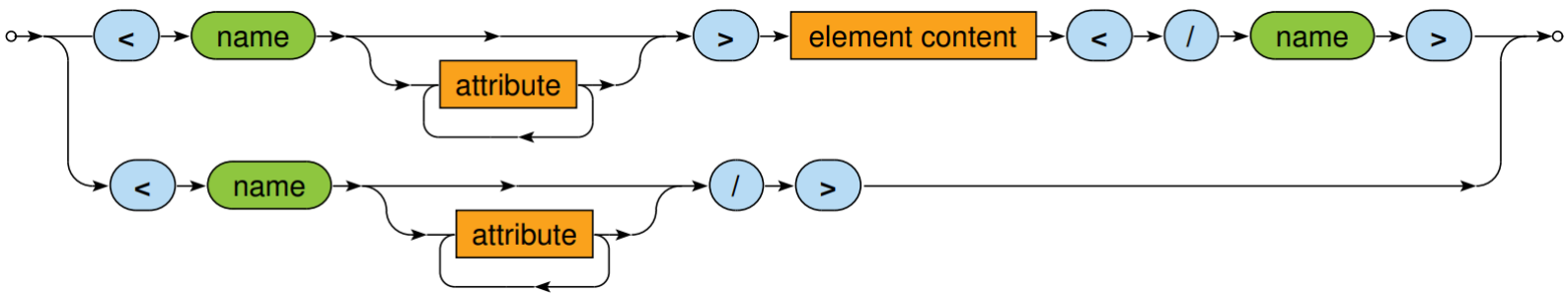
Example

```
1  <?xml version="1.1" encoding="UTF-8"?>
2  <movie>
3    ...
4  </movie>
```

Constructs

Element

- Marked using **opening** and **closing** tags
  - ... or just an abbreviated tag in case of empty elements
- Each element can be associated with a **set of attributes**



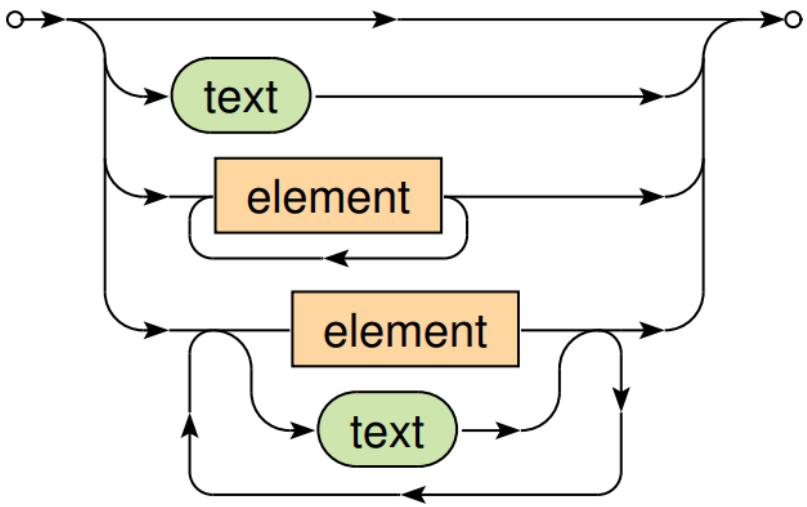
Examples

```
1 <title>...</title>
2 <actors/>
```

Constructs

Types of element content

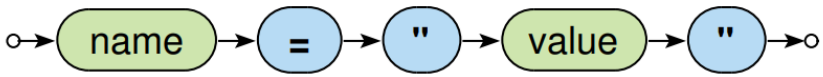
1. **Empty** content
2. **Text** content
3. **Element** content
   - **Sequence** of **nested elements**
4. **Mixed** content
   - Elements arbitrarily **interleaved** with text values text



Constructs

Attribute

- Name-value **pair**



Escape sequences (predefined entities)

- Used **within values** of attributes or text content of elements
- E.g.
  - **<** for <
  - **>** for >
  - **"** for "
  - ...

XML Conclusion

XML constructs

- **Basic.** element, attribute, text
- **Additional.** comment, processing instruction, ...

Schema languages

- DTD, XSD (XML Schema), RELAX NG, Schematron

Query languages

- XPath, XQuery, XSLT

XML formats = particular languages

- XSD, XSLT, XHTML, DocBook, ePUB, SVG, RSS, SOAP, ...

Outline

JSON

JSON = JavaScript Object Notation

- Open standard for **data interchange**

Design goals

- **Simplicity.** text-based, easy to read and write
- **Universality.** object and array data structures

- **Supported by** majority of modern programming languages
- **Based conventions** of the C-family of languages (C, C++, C#, Java, JavaScript, Perl, Python, ...)
- **Derived from** JavaScript (but language independent)
- Started in 2002
- **File extension.** *.json
- **Content type:** application/json
- `http://www.json.org/`

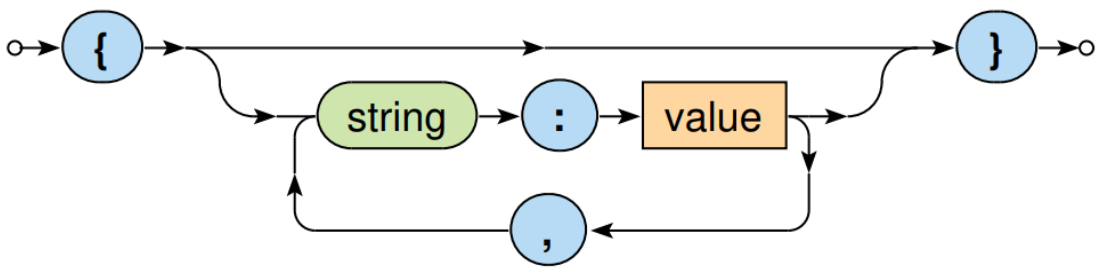
Example

```
1  {
2    "title" : "Murder Mystery",
3    "year" : 2019,
4    "actors" : [
5      {
6        "firstname" : "Adam",
7        "lastname" : "Sandler"
8      },
9      {
10       "firstname" : "Jennifer",
11       "lastname" : "Aniston"
12     }
13   ],
14   "director" : {
15     "firstname" : "Kyle",
16     "lastname" : "Newacheck"
17   }
18 }
```

Data Structure

Object

- Unordered collection of **name-value pairs** (properties)
  - Correspond to **structures** such as objects, records, structs, dictionaries, hash tables, keyed lists, associative arrays, ...
- **Values** can be of different types, names should be unique



Examples

```
1 { "name" : "Jennifer Aniston", "year" : 1964 }
```

```
1 { }
```

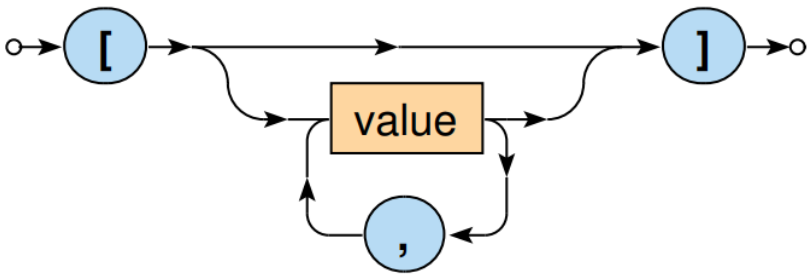
Data Structure

Array

- **Ordered collection** of values

Correspond to structures such as arrays, vectors, lists, sequences, ...

- **Values** can be of **different** types, **duplicate** values are allowed



Examples

```
1 [ 2, 7, 7, 5 ]
```

```
1 [ "Jennifer Aniston", 1964, -5.6 ]
```

```
1 [ ]
```

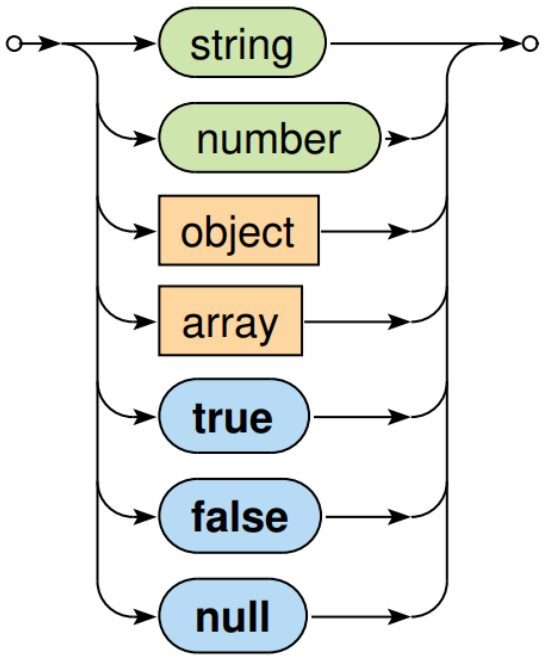
Data Structure

Value

- **Unicode** string
  - **Enclosed** with **double quotes**
  - Backslash escape sequences
  - Example:

```
1  "a \n b \" c \\ d"
```

- **Number**
  - Decimal integers or floats
  - **Examples.** 1, -0.5, 1.5e3
- Nested **object**
- Nested **array**
- **Boolean** value: true, false
- **Missing** information: **null**



JSON Conclusion

JSON constructs

- **Collections.** object, array
- **Scalar values.** string, number, boolean, null

Schema languages

- JSON Schema

Query languages

- JSONiq, JMESPath, JAQL, ...

Outline

BSON

BSON = Binary JSON

- Binary-encoded serialization of JSON documents
  - **Extends** the set of basic data types of values offered by JSON (such as a string, ...) with a few new specific ones
- **Design characteristics.** lightweight, traversable, efficient
- Used by **MongoDB**
  - Document NoSQL database for JSON documents
  - Data **storage** and network **transfer** format
- **File extension.** *.bson
- `http://bsonspec.org/`

Example

JSON

```
1 {
2   "title" : "Murder Mystery",
3   "year" : 2019
4 }
```

BSON

```
1 29 00 00 00
2 02 74 69 74 6C 65 00 0E 00 00 00 4D 75 72 64 65 72 20 4D 79 73↩
      74 65 72 79 00
3 10 79 65 61 72 00 E3 07 00 00
4 00
```

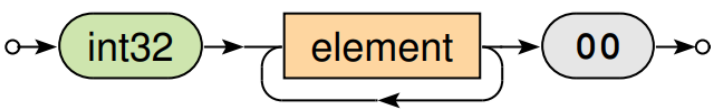
Document Structure

Document = **serialization** of one JSON object or array

- JSON **object** is serialized **directly**
- JSON **array** is first **transformed** to a JSON object
  - Property names derived from positions
  - E.g.:

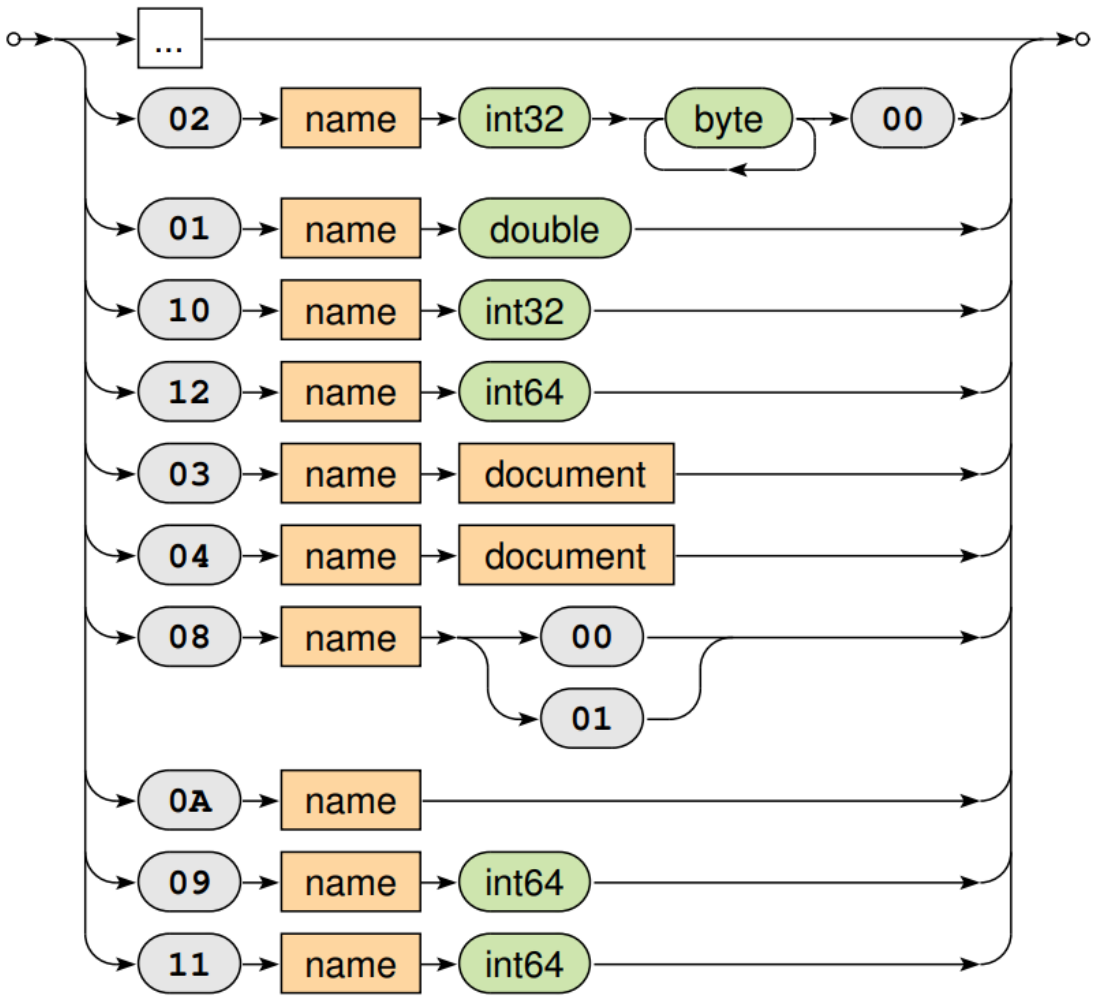
```
1 [ "Aniston", "Evans" ] ->
2 { "0" : "Aniston", "1" : "Evans" }
```

- **Structure**
  - Document size (total number of bytes)
  - **Sequence** of elements (**encoded JSON properties**)
  - **Terminating** hexadecimal **00 byte**



Document Structure

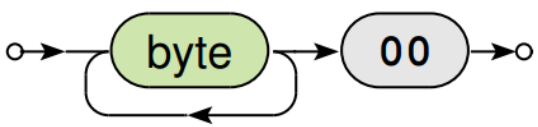
Element = serialization of one JSON property



Document Structure

Element = serialization of one JSON property

- **Structure**
  - Type selector
    - 02 (string)
    - 01 (double), 10 (32-bit integer), 12 (64-bit integer)
    - 03 (object), 04 (array)
    - 08 (boolean)
    - 0A (null)
    - 09 (datetime), 11 (timestamp)
    - ...
  - Property name
    - **Unicode string** terminated by 00



  - Property value

Outline

RDF: Resource Description Framework

RDF = Resource Description Framework

- Language for representing **information about resources** in the World Wide Web
  - a **family** of related technologies, languages, specifications, ...
  - Used in the **context** of the **Semantic Web**, **Linked Data**, ...
- Developed by W3C
- Started in 1997
- Versions: 1.0 and 1.1
- W3C recommendations
  - `https://www.w3.org/TR/rdf11-concepts/`
    - Concepts and Abstract Syntax
  - `https://www.w3.org/TR/rdf11-mt/`
    - Semantics

Statements

Resource

- Any **real-world** entity
  - Referents = resources **identified by IRI**
    - E.g. physical things, documents, abstract concepts, ...
  - Values = resources for **literals**
    - E.g. numbers, strings, ...

Statement about resources = one RDF triple

- **Three components.** subject, predicate, and object

Examples

```
1 <http://db.my/movies/Murder+Mystery>
2 <http://db.my/terms#actor>
3 <http://db.my/actors/Aniston> .
```

```
1 <http://db.my/movies/Murder+Mystery>
2 <http://db.my/terms#year>
3 "2019" .
```

Statements

Triple components

- **Subject**
  - Describes a resource the given statement is **about**
  - IRI or blank node identifier
- **Predicate**
  - Describes the **property** or **characteristic** of the **subject**
  - **IRI**
- **Object**
  - Describes the **value** of that property
  - **IRI** or **blank** node identifier or literal

Although triples are **inspired by** natural languages, they have nothing to do with processing of natural languages.

Example

```
<http://db.my/movies/Murder+Mystery>
<http://db.my/terms#actor> <http://db.my/actors/Sandler> .

<http://db.my/movies/Murder+Mystery>
<http://db.my/terms#actor> <http://db.my/actors/Aniston> .

<http://db.my/movies/Murder+Mystery>
<http://db.my/terms#year> "2019" .

<http://db.my/movies/Murder+Mystery>
<http://db.my/terms#director> _:n18 .

_:n18
<http://db.my/terms#firstname> "Kyle" .

_:n18
<http://db.my/terms#lastname> "Newacheck" .
```

Identifiers and Literals

IRI = Internationalized Resource Identifier

- **Absolute** (not relative) IRIs with optional fragment identifiers
- RFC 3987
- Unicode characters
- Examples

```
1 http://db.my/movies/Murder+Mystery
2 http://db.my/terms#actor
3 mailto:anis.rahman@um.edu.my
4 urn:issn:0167-6423
```

URLs are often used in practice → information about given resources are then intended to be **published/retrieved** via standard **HTTP**

Identifiers and Literals

Literals

1. **Plain** values

```
"Murder Mystery", "2019"
```

2. **Typed** values
   - XML Schema simple data types are adopted and used

```
"Murder Mystery"^^xs:string, "2019"^^xs:integer
```

3. **Strings** with **language tags**

```
"Murder Mystery"@cs
```

Types and **language tags** **cannot** be mutually **combined**

Identifiers and Literals

Blank node identifiers

- Blank nodes (**anonymous** resources)
  - Allow to express statements about resources **without explicitly naming** (identifying) them
- Blank node identifiers only have **local scope** of validity
  - E.g. within a given file, query expression, ...
- Particular syntax depends on a serialization format

```
1 _:node18
```

Data Model

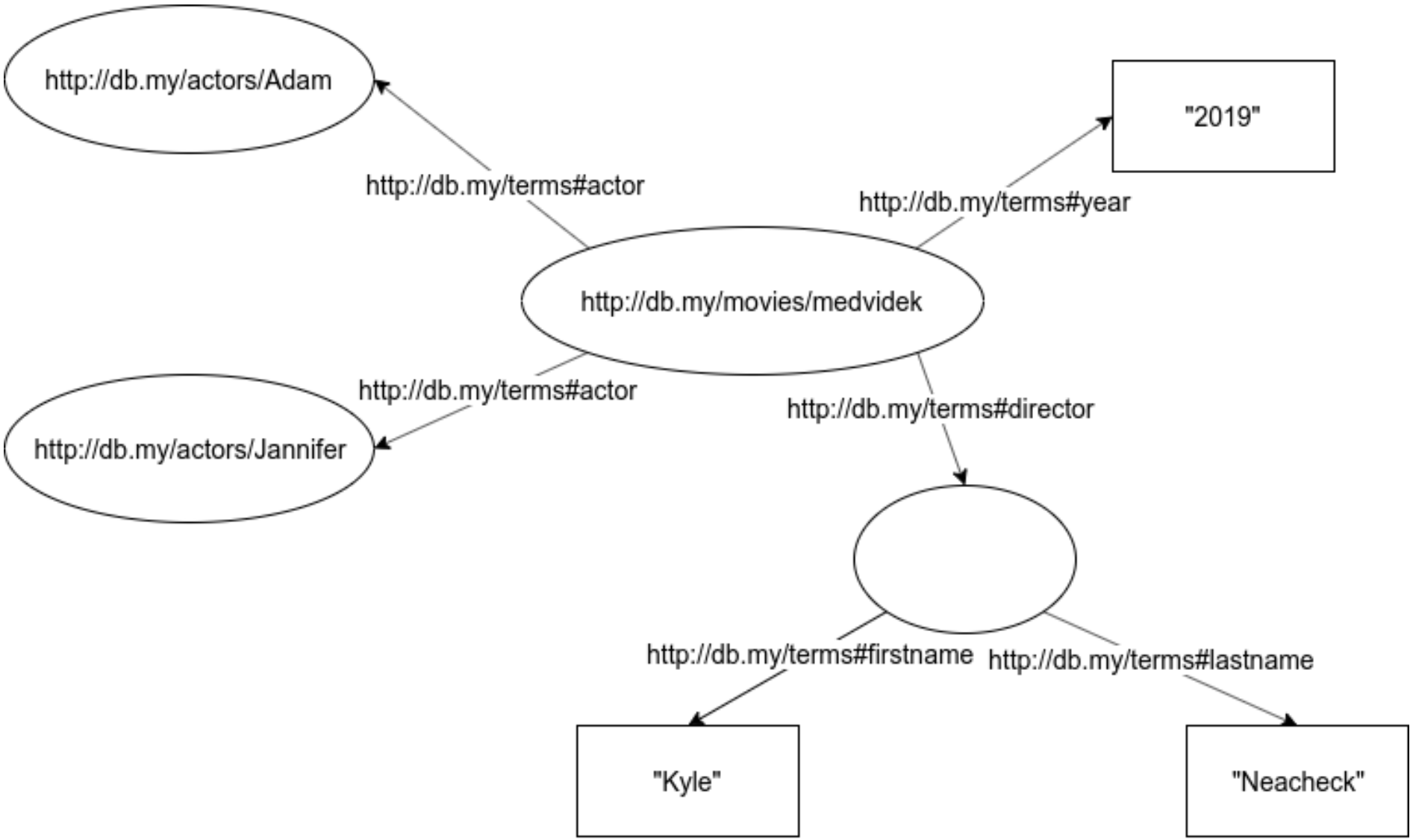
Directed labeled multigraph

1. **Vertices**
   - **One vertex** for **each** **IRI** or **literal** value
2. **Edges**
   - **One edge** for each individual **triple**
   - Edges are **directed**

```
          predicate
subject ----------> object
```

   - **Property names** (predicate IRIs) are used as **edge labels**

Example


http://db.my/actors/Adam
http://db.my/terms#actor
"2019"
http://db.my/terms#year
http://db.my/movies/medvidek
http://db.my/terms#actor
http://db.my/actors/Jannifer
http://db.my/terms#director
http://db.my/terms#firstname
http://db.my/terms#lastname
"Kyle"
"Neacheck"

Serialization

Available approaches

- **N-Triples** notation
  - `https://www.w3.org/TR/n-triples/`
- **Turtle** notation (Terse RDF Triple Language)
  - `https://www.w3.org/TR/turtle/`
- **RDF/XML** notation
  - XML syntax for RDF
  - `https://www.w3.org/TR/rdf-syntax-grammar/`
- **JSON-LD** notation
  - JSON-based serialization for Linked Data
  - `https://www.w3.org/TR/json-ld/`
- ...

N-Triples Notation

RDF N-Triples notation = A line-based syntax for an RDF graph

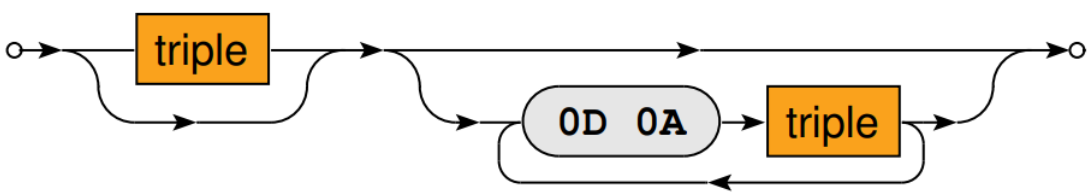
- Simple, line-based, plain text format
- File extension: *.rdf
- `https://www.w3.org/TR/n-triples/`

Example

- Already presented...

Document

- Statements are **terminated** by dots, **delimited** by EOL triple



N-Triples Notation

Statement

- Individual triple **components** are **delimited** by spaces

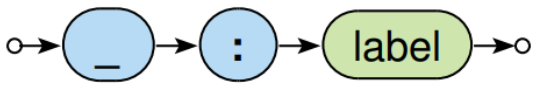
Triple components. subject, predicate, object

N-Triples Notation

IRI reference

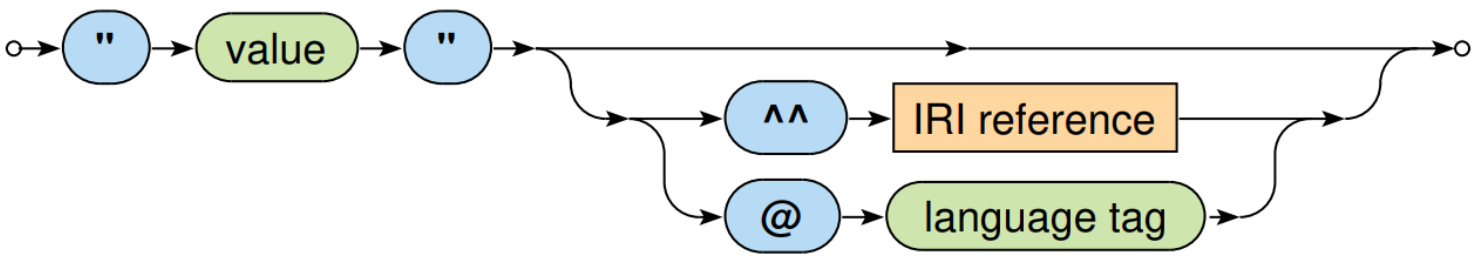
- IRIs are **enclosed** in **angle brackets**

Blank node identifier



Literal

- Literals are **enclosed** in **double quotes**



Turtle Notation

Turtle = Terse RDF Triple Language

- Compact text format, various abbreviations for common usage patterns
- File extension: *.ttl
- Content type: text/turtle
- https://www.w3.org/TR/turtle/

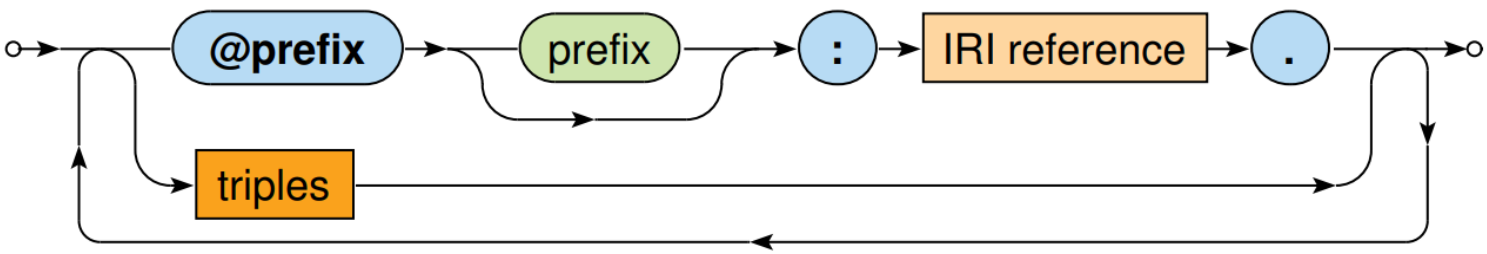
Example

```
1 @prefix i: <http://db.my/terms#> .
2 @prefix m: <http://db.my/movies/> .
3 @prefix a: <http://db.my/actors/> .
4 m:Murder Mystery
5 i:actor a:Sandler , a:Aniston ;
6 i:year "2019" ;
7 i:director [ i:firstname "Kyle" ; i:lastname "Newacheck" ] .
```

Turtle Notation

Document

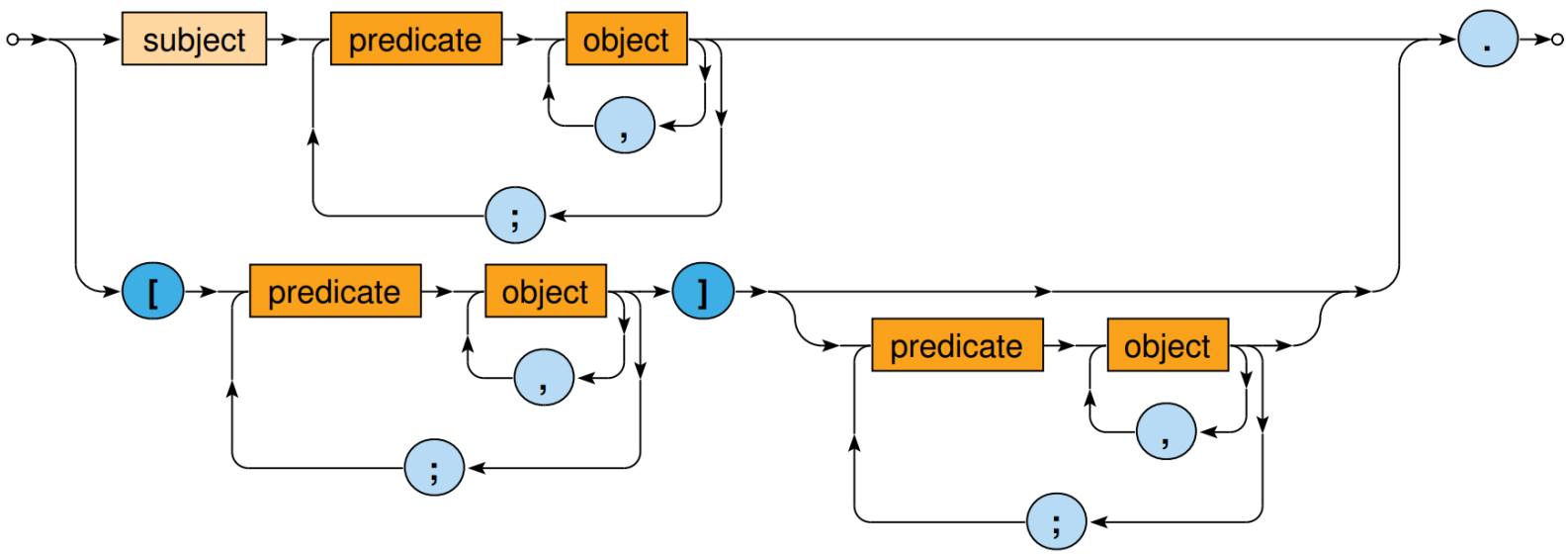
- Contains a sequence of triples and/or declarations
- **Prefix declarations**
  - Prefixed names can then be used **instead of** full IRI references
- **Groups of triples**
  - Individual groups are **terminated** by dots



Turtle Notation

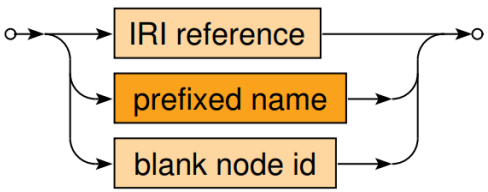
Triples

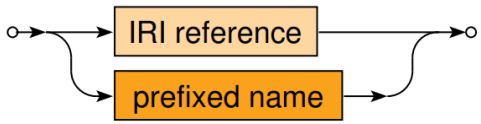
- Triples sharing the **same subject** and **object** OR at least the **same subject** can be **grouped** together
  - **object list** for a **shared subject** and **predicate**
  - **predicate-object list** for a **shared subject**
- **Brackets** can be used to define **blank nodes**

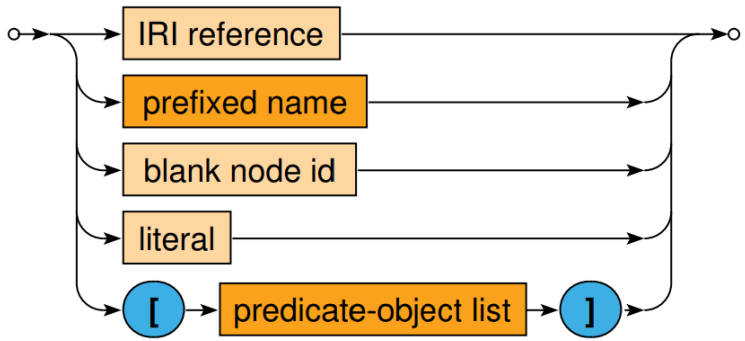


Turtle Notation

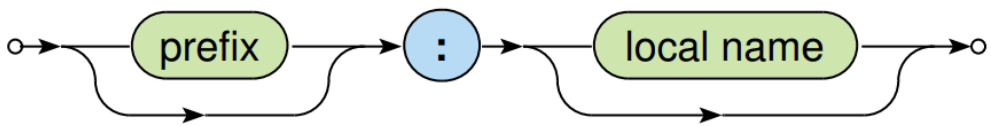
Triple components. subject, predicate, object







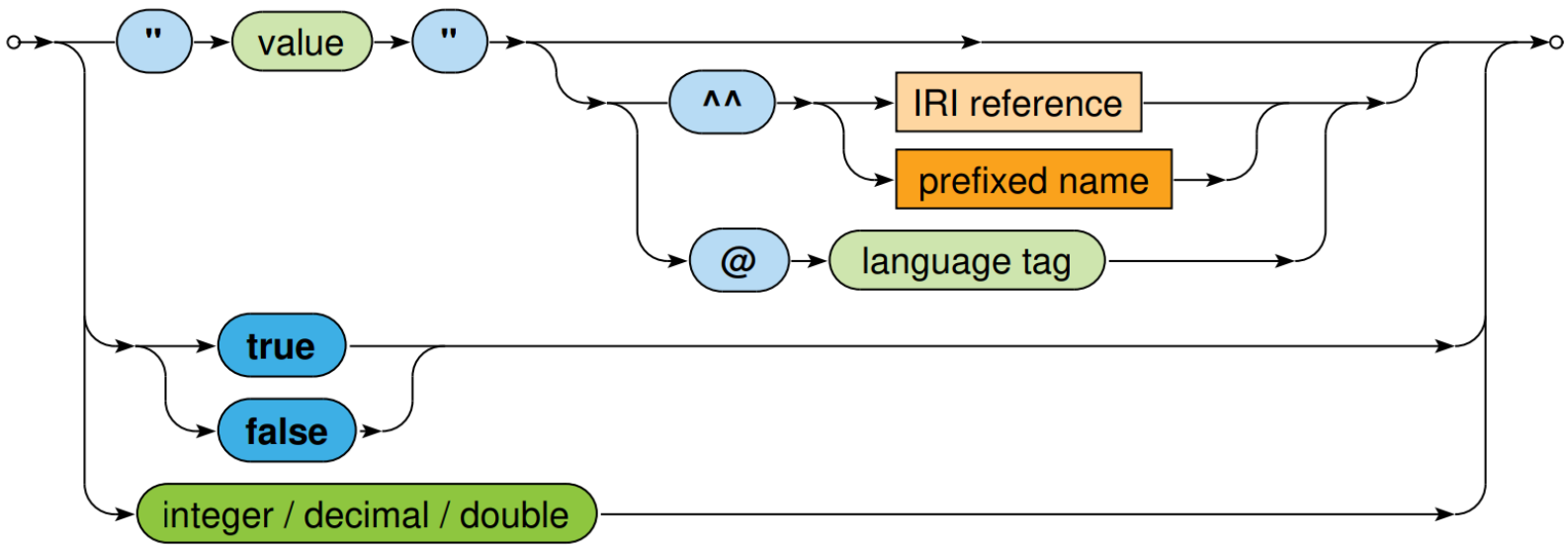
IRI reference/prefixed name



Turtle Notation

Literal

- **Traditional** literals
  - **new abbreviated forms** of numeric and boolean literals



Example

Example revisited

```
1 @prefix i: <http://db.my/terms#> .
2 @prefix m: <http://db.my/movies/> .
3 @prefix a: <http://db.my/actors/> .
4 m:Murder Mystery
5 i:actor a:Sandler , a:Aniston ;
6 i:year "2019" ;
7 i:director [ i:firstname "Kyle" ; i:lastname "Newacheck" ] .
```

RDF Conclusion

RDF statements

- Subject, predicate, and object components

Schema languages

- **RDFS** (RDF Schema)
- **OWL** (Web Ontology Language)

Query languages

- SPARQL (SPARQL Protocol and RDF Query Language)

CSV

CSV = Comma-Separated Values

- Unfortunately **not fully standardized**
  - **Different** field **separators** (commas, semicolons)
  - **Different escape sequences**
  - **No encoding information**
- RFC 4180, RFC 7111
- File extension: ***.csv**
- Content type: text/csv

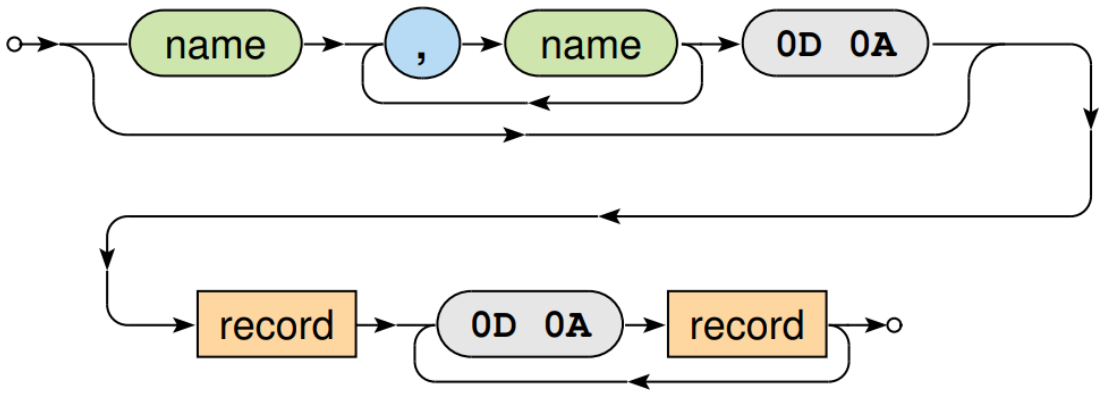
Example

```
1 firstname,lastname,year
2 Jennifer,Aniston,1964
3 Adam,Sandler,1966
4 Luke,Evans,1973
5 Terence,Stamp,1936
6 Gemma,Arterton,1976
```

Document Structure

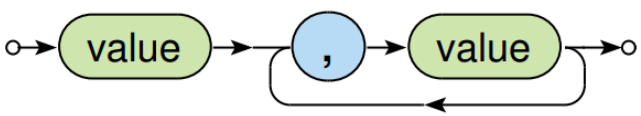
Document

- Optional **header** + **list** of records



Record

- Comma separated list of fields



Outline

Protocol Buffers

Protocol Buffers

- **Extensible mechanism** for serializing structured data
  - Used in communication protocols, data storage, ...

Design goals

- Language-neutral, platform-neutral
- Small, fast, simple

- Developed (and widely used) by Google
- Started in 2008 internally and 2011 publicly
- Versions: proto2, proto3
- File extension: ***.proto**
- `https://developers.google.com/protocol-buffers/`
- Real-world usage: RiakKV, HBase

Introduction

Intended usage

- Schema creation → automatic source code generation → sending messages between applications

Components

- Interface description language
- Source code generator (protoc compiler)
- Supported languages
  - Official: C++, C#, Java, Python, Ruby ...
  - 3rd party: Perl, PHP, Scala, ...
- **Binary serialization format**
  - Compact, not self-describing

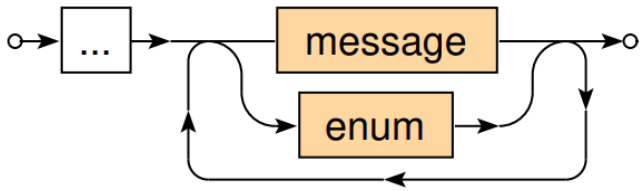
Example

```
1  syntax = "proto3";
2  message Actor {
3    string firstname = 1;
4    string lastname = 2;
5  }
6  message Movie {
7    string title = 1;
8    int32 year = 16;
9    repeated Actor actors = 17;
10   enum Genre {
11     UNKNOWN = 0;
12     COMEDY = 1;
13     FAMILY = 2;
14   }
15   repeated Genre genres = 2048;
16 }
```

Schema Structure

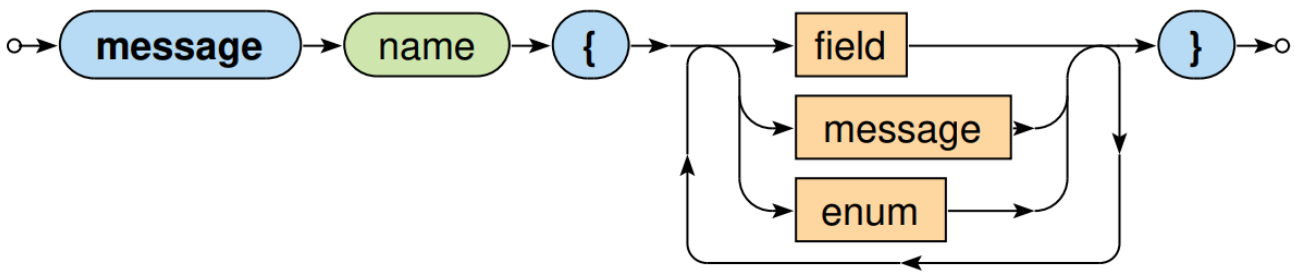
Schema

- One schema may contain multiple **message descriptions**
  - Other constructs are allowed as well, e.g. enumerations



Message

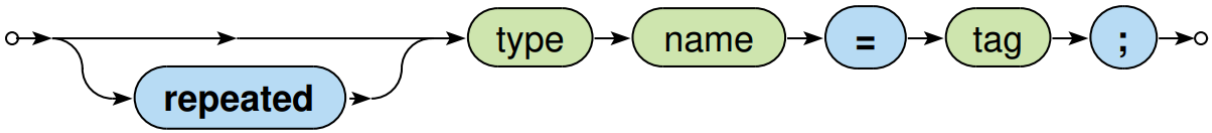
- Represents a **small logical record** of information
  - Defines a set of uniquely **numbered fields**
  - **Nested messages** or **enumerations** are allowed too



Schema Structure

Field

- Describes **one data value** repeated



- **Rule** – allowed number of **value occurrences**
  - **Default** = 0 or 1 value
  - **repeated** = 0 or more values (i.e. an arbitrary number)
    - The **order** of individual values is **preserved**
- ...

Schema Structure

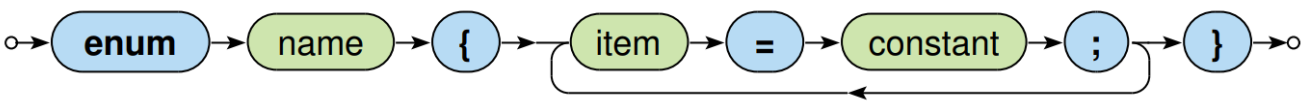
Field

- Type
  1. **Atomic.** int32, int64, double, string, bool, bytes, ...
     - Mappings to data types of particular programming languages as well as default values are introduced
  2. **Composed.** messages, enumerations, ...
- Name – name of a given field
- Tag – internal integer identifier
  - Used to **identify individual fields** of a message in a binary format
  - **Frequently used** fields should be assigned **lower tags**
    - Since **lower** **number of bytes** will then be needed

Schema Structure

Enumeration

- Description of a **predefined list of values**
- The **first item** is considered to be the **default value** and its value **must** be **equal to 0**



A few other constructs are available too (e.g. maps)

Lecture Conclusion

Data formats

- Tree: XML, JSON
- Graph: RDF
- Relational: CSV

Binary serializations

- BSON, Protocol Buffers