

The Semantic Web

Craig Knoblock

(based on slides by Yolanda Gil, Ian Horrocks,
Jose Luis Ambite, and Tom Russ)

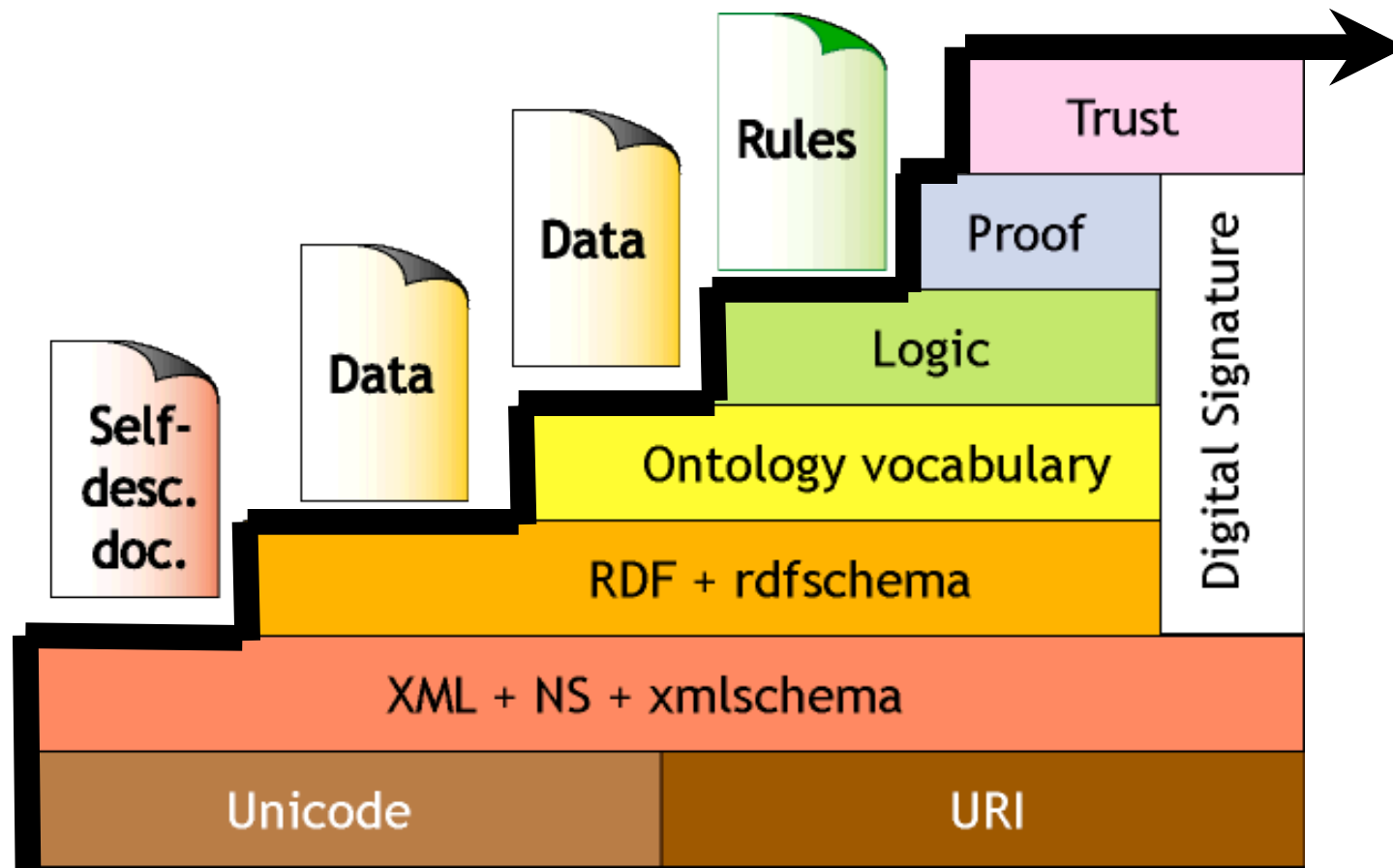
The Semantic Web

W3C's Tim Berners-Lee: "Weaving the Web":

"I have a dream for the Web... and it has two parts."

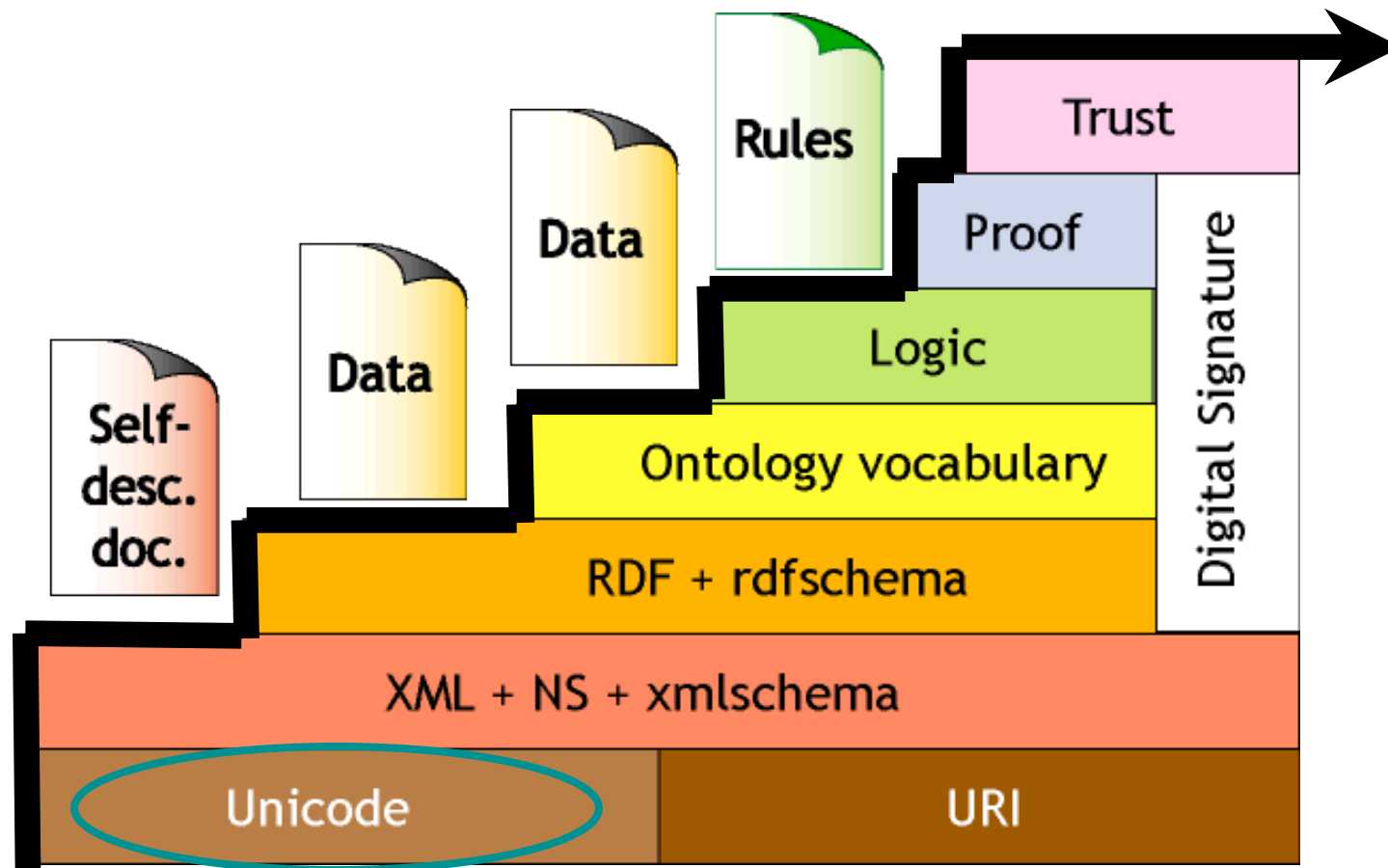
- *The first Web enables **communication between people***
 - *The Web shows how computers and networks enable the information space while getting out of the way*
- *The new Web will **bring computers into the action***
 - *Step 1 -- Describe: putting data on the Web in machine-understandable form -- a Semantic Web*
 - *RDF (based on XML)*
 - *Master list of terms used in a document (RDF schema)*
 - *Each document mixes global standards and local agreed-upon terms (namespaces)*
 - *Step 2 -- Infer and reason: apply logic inference*
 - *Operate on partial understanding*
 - *Answering why*
 - *Heuristics*

Web Semantics



Semantic Web LayerCake (Berners-Lee, 99; Swartz-Hendler, 2001)

Unicode

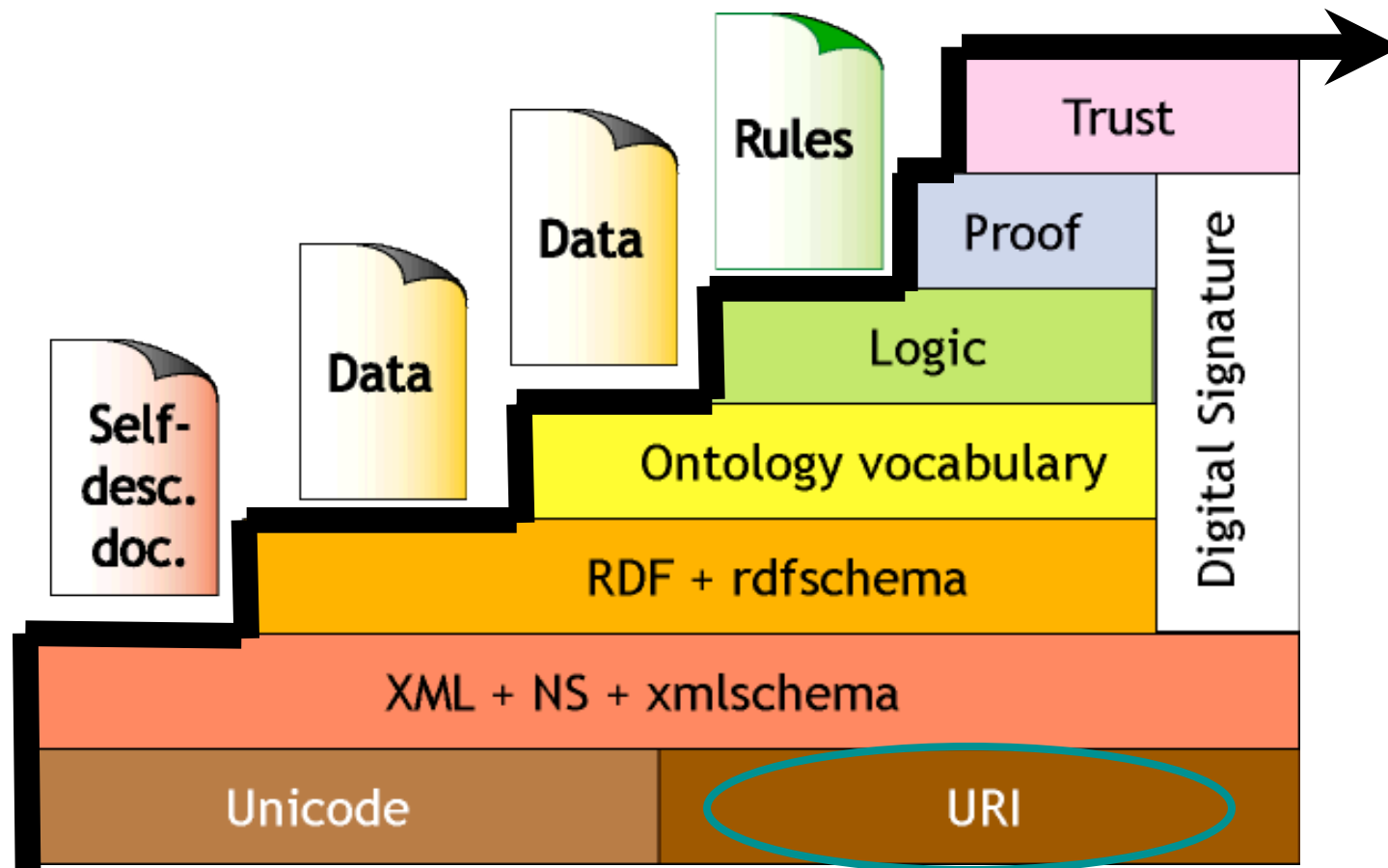


Semantic Web LayerCake (Berners-Lee, 99; Swartz-Hendler, 2001)

Unicode

- A character encoding system, like ASCII, designed to help developers who want to create software applications that work in any language in the world
- Unicode provides a unique number for every character, no matter what the platform, no matter what the program, no matter what the language

URI

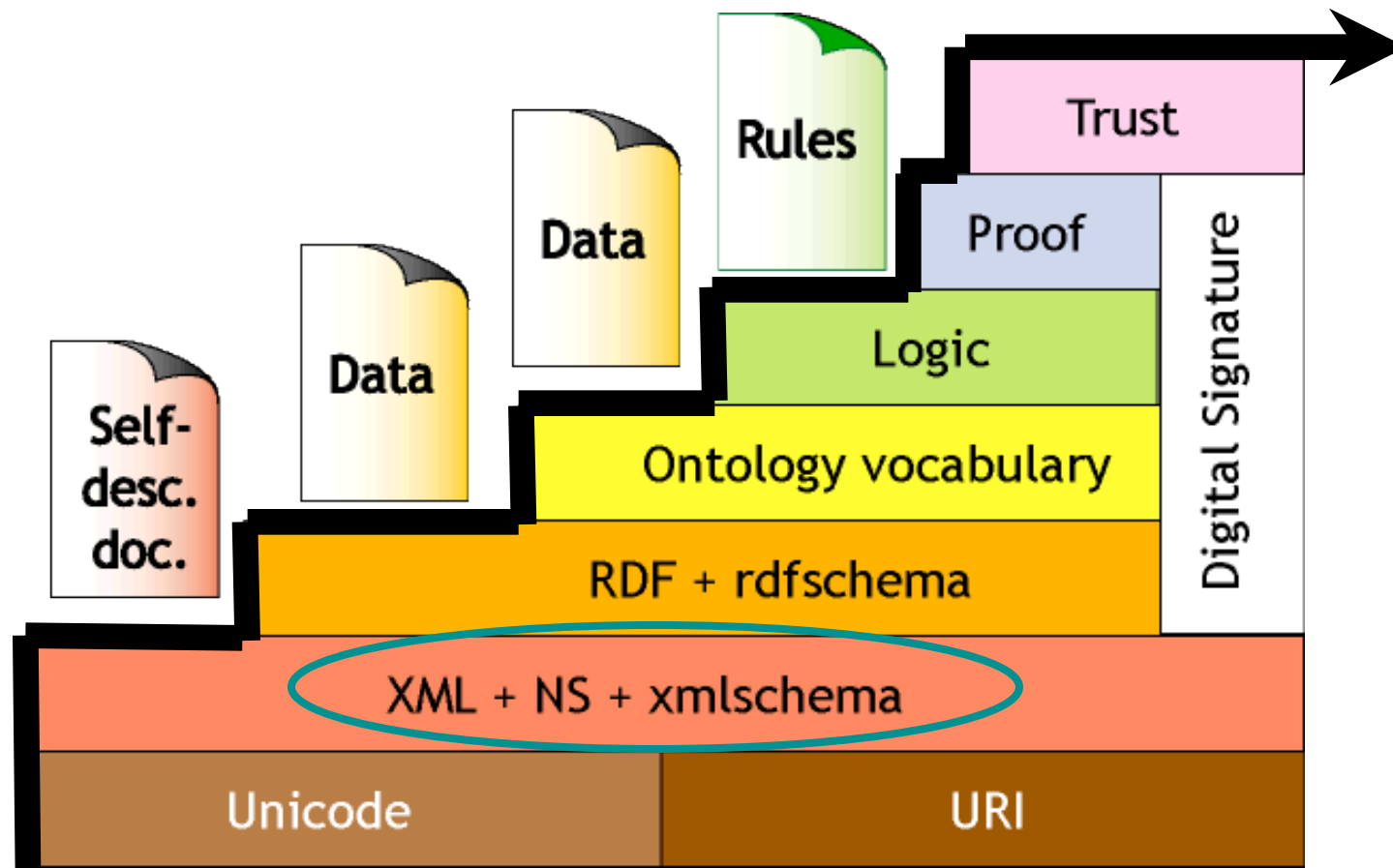


Semantic Web LayerCake (Berners-Lee, 99; Swartz-Hendler, 2001)

URIs: Uniform Resource Identifiers (aka URLs)

- The Web is an information space. URIs are the points in that space.
- Short strings that identify **resources** in the web: documents, images, downloadable files, services, electronic mailboxes, and other resources.
- They make resources addressable in the same simple way. They reduce the tedium of "log in to this server, then issue this magic command ..." down to a single click.

XML and Namespaces



Semantic Web LayerCake (Berners-Lee, 99; Swartz-Hendler, 2001)

Why XML (eXtensible Markup Language)

Problems with HTML

HTML design

- HTML is **intended for presentation** of information as Web pages.
- HTML contains a **fixed set of markup tags**.

This design is not appropriate for data:

- Tags don't convey meaning of the data inside the tags.
- Tags are not extensible.

The Design of XML

- Tags can be used to represent the meaning of data/information
 - separates syntax (structural representation) from semantics => **only syntax is considered in XML**
- There is no fixed set of markup tags - **new tags can be defined**
- Underlying data model is a **tree structure**
- “XML is the new ASCII” -- Tim Bray

<http://www.w3.org/TR/2000/REC-xml-20001006>

Simple XML Example

```
<Bookstore>
  <Book ID="101">
    <Author>John Doe</Author>
    <Title>Introduction to XML</Title>
    <Date>12 June 2001</Date>
    <ISBN>121232323</ISBN>
    <Publisher>XYZ</Publisher>
  </Book>
  <Book ID="102">
    <Author>Foo Bar</Author>
    <Title>Introduction to XSL</Title>
    <Date>12 June 2001</Date>
    <ISBN>12323573</ISBN>
    <Publisher>ABC</Publisher>
  </Book>
</Bookstore>
```

Make up your own tags

Sub-elements

XML by itself is just hierarchically structured text

An important diversion: Namespaces

- What is a Namespace ?

The Namespace of an element, is the scope within which, it (and thus it's name) is valid

- Why do we need Namespaces ?

- If elements were defined within a global scope, it becomes a problem when combining elements from multiple documents
- Modularity: If a markup vocabulary exists which is well understood and for which there is useful software available, it is better to reuse it

- Namespaces in XML:

An XML namespace is a collection of names, identified by a URI reference.

Names from XML namespaces may appear as qualified names, which contain a single colon, separating the name into a **prefix** and a **local** part. The prefix, which is mapped to a URI reference, selects a namespace

XSD: XML Schema Definition

- Written in the same syntax as XML documents (unlike XML DTDs!)
- Elements and attributes
- Enhanced set of primitive datatypes.
 - Wide range of primitive data types, supporting those found in databases (string, boolean, decimal, integer, date, etc.)
 - Can create your own datatypes (complexType)
- Can derive new type definitions on the basis of old ones (refinement)
- Can have constraints on attributes
 - Examples: maxLength, precision, enumeration, maxInclusive (upper bound), minInclusive (lower bound), etc.

XSD (XML Schema) Example

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.books.org"
  xmlns="http://www.books.org">
  <xsd:element name="Bookstore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Book" minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Book">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Title" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="Author" minOccurs="1" maxOccurs="unbounded"/>
        <xsd:element ref="Date" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="ISBN" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="Publisher" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Title" type="xsd:string"/>
  <xsd:element name="Author" type="xsd:string"/>
  <xsd:element name="Date" type="xsd:date"/>
  <xsd:element name="ISBN" type="xsd:integer"/>
  <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>
```

Prefix “**xsd**” refers to the XMLSchema namespace

“**xmlns**” refers to the **default** namespace

Defining the element “Bookstore” as a complex Type

Containing a sequence of 1 or more “Book” elements

When referring to another Element, use “ref”

The Author can be 1 or more

Element definitions

Notice the use of more meaningful data types

Summary of the XML+ NS +XSD Layer

The Power of Simplicity

- “When I designed HTML, I chose to avoid giving it more power than it absolutely needed – a “**principle of least power**”, which I have stuck to ever since. I could have used a language like Knuth’s Tex but...” -- TBL
- Keeps the principles of SGML in place but its spec is thin enough to wave 😊
- To say you are “Using XML” is sort of like saying you are using ASCII
- Using XSD (XML Schema) makes a lot more sense

Where XML & XML Schemas Fail

- No semantics!

```
<book>
  <title> ... </title>
  <author> ... </author>
  <isbn> ... </isbn>
</book>
```

```
<bookstore>
  <book> ... </book>
  <mgzine> ... </mgzine>
</bookstore>
```

- Will XML scale in the metadata world?

1. The order in which elements appear in an XML document is often meaningful. This seems highly unnatural in the metadata world.

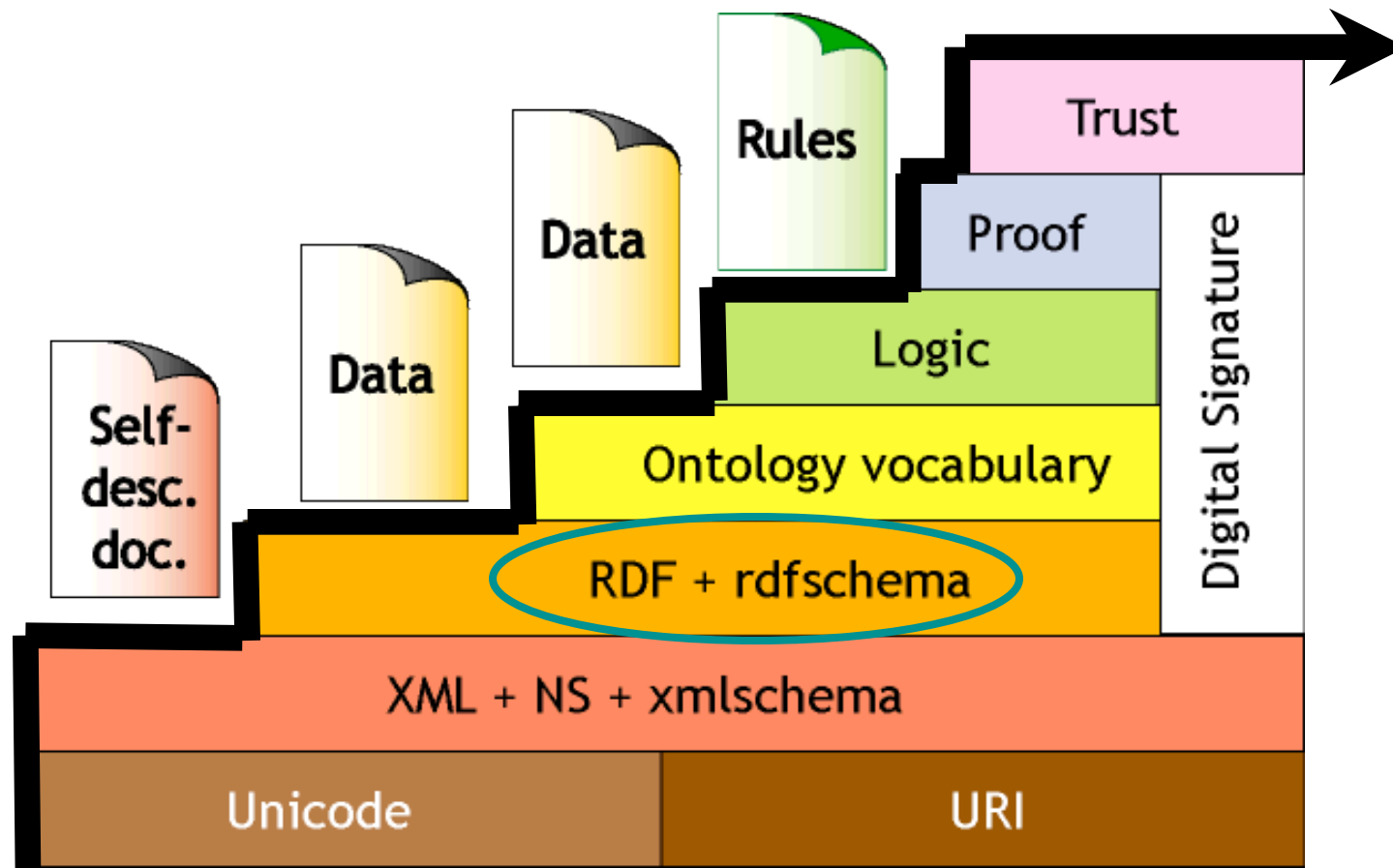
Furthermore, maintaining the correct order of millions of data items is impractical.

2. XML allows constructions that mix up some text along with child elements, which are hard to handle.

Ex.

```
<topelem>This is some character string data
  <elem>
    this is a child
    <subelem>this is another child</subelem>
  </elem>
</topelem>
```


Resource Description Framework



Semantic Web LayerCake (Berners-Lee, 99; Swartz-Hendler, 2001)

RDF (Resource Description Framework)

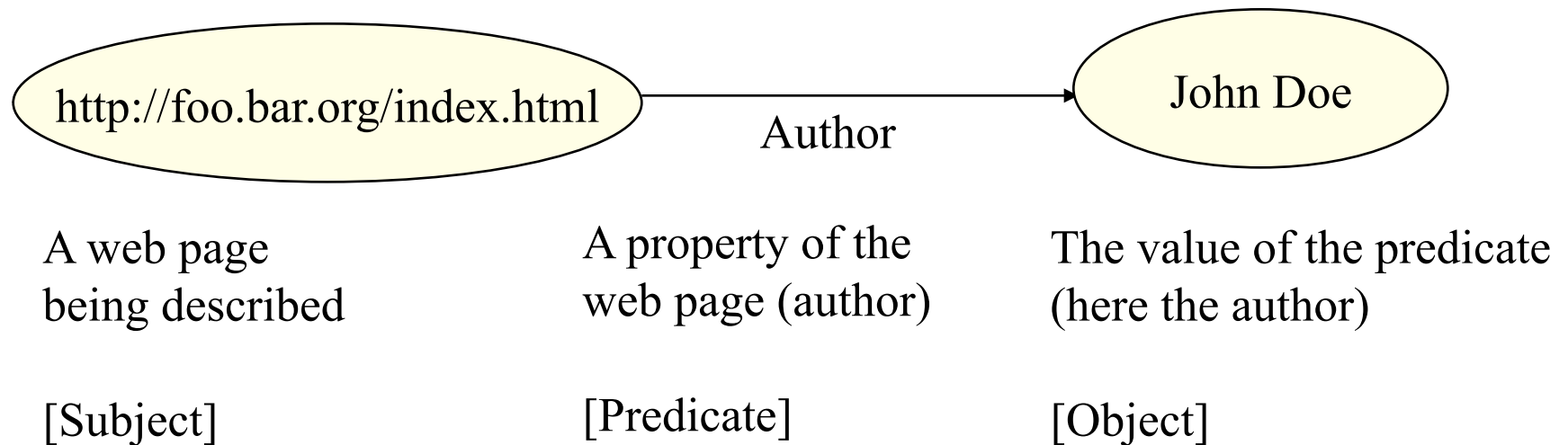
- RDF provides a way of describing resources via metadata (data about data)
It restricts the description of resources to **triples (subject,predicate,object)**
- It provides interoperability between applications that exchange machine understandable information on the Web.
- The original broad goal of RDF was to define a mechanism for describing resources that makes no assumptions about a particular application domain, nor defines (a priori) the semantics of any application domain.
- Provides a **lightweight ontology system**
 - Labeled graph model
 - Subclass of, instance of
 - Property domain and range
- Uses XML as the interchange syntax.
- The formal specification of RDF is available at: <http://www.w3.org/RDF/>

RDF Syntax

Subject, Predicate and Object Triples (Triples)

- Subject: The resource being described.
- Predicate: A property of the resource
- Object: The value of the property

A combination of them is said to be a Statement



RDF Example

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/TR/WD-rdf-syntax#"
  xmlns:s="http://description.org/schema/">

  <rdf:Description about="http://foo.org/index.html">
    <s:Author>John Doe</s:Author>
  </rdf:Description>
</rdf:RDF>
```

Annotations for the RDF XML snippet:

- Namespace for the RDF spec (points to `xmlns:rdf="http://www.w3.org/TR/WD-rdf-syntax#"`)
- Custom namespace 's' (points to `xmlns:s="http://description.org/schema/"`)
- Subject: a resource (points to `about="http://foo.org/index.html"`)
- Property: a resource (points to `<s:Author>`)
- Object: a resource or a literal (points to `John Doe`)

In Triples notation: `<http://foo.org/index.html> <s:Author> "John Doe"` .

Both statements say: The Author of <http://foo.org/index.html> is "John Doe"

In this way, we can have different objects (resources) pointing to other objects (resources) , thus forming a Directed Labeled Graph

You can also make statements about statements – reification

Ex: 'xyz' says that ' The Author of <http://foo.org/index.html> is John Doe'

RDF Schema (Triples Notation)

- A schema defines the terms that will be used in the RDF statements and gives specific meanings to them.

<http://www.w3.org/TR/rdf-schema/>

Example:

ex:MotorVehicle rdf:type rdfs:Class . —→ MotorVehicle is an **instance of** rdfs:Class
ex:PassengerVehicle rdf:type rdfs:Class .
ex:Van rdf:type rdfs:Class .
ex:Truck rdf:type rdfs:Class .
ex:MiniVan rdf:type rdfs:Class .
ex:PassengerVehicle rdfs:subClassOf ex:MotorVehicle .
ex:Van rdfs:subClassOf ex:MotorVehicle .
ex:Truck rdfs:subClassOf ex:MotorVehicle .
ex:MiniVan rdfs:subClassOf ex:Van .
ex:MiniVan rdfs:subClassOf ex:PassengerVehicle .

PassengerVehicle is a **subclass of** MotorVehicle

} → **Multiple Inheritance**

RDF Schema (RDF/XML notation)

```
<rdf:RDF xml:lang="en"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
```

→ RDF Schema Namespace

```
  <rdf:Description rdf:ID="MotorVehicle">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
  </rdf:Description>
```

→ An rdf:ID attribute names a new resource

→ (“Resource” is the top level class)

```
  <rdf:Description rdf:ID="PassengerVehicle">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
  </rdf:Description>
```

→ PassengerVehicle is a **subclass** of MotorVehicle

```
  <rdf:Description rdf:ID="Truck">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
  </rdf:Description>
```

RDF Schema Example (cont..)

```
<rdf:Description rdf:ID="Van">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>
```

```
<rdf:Description rdf:ID="MiniVan">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#Van"/>
  <rdfs:subClassOf rdf:resource="#PassengerVehicle"/>
</rdf:Description>
```

} → **Multiple Inheritance**

```
<rdf:Description rdf:ID="registeredTo">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdfs:domain rdf:resource="#MotorVehicle"/>
  <rdfs:range rdf:resource="#Person"/>
</rdf:Description>
```

→ **Domain** of a property

→ **Range** of a property

```
<rdf:Description rdf:ID="weight">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdfs:domain rdf:resource="#MotorVehicle"/>
  <rdfs:domain rdf:resource="#Book"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/03/example/classes#Number"/>
</rdf:Description>
```

} → **Multiple Domains**

```
</rdf:RDF>
```

“Typed Node” Abbreviation

```
<rdf:RDF
```

```
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://example.org/schemas/vehicles">
```

```
  <rdfs:Class rdf:ID="MotorVehicle"/>
```

```
  <rdfs:Class rdf:ID="PassengerVehicle">
```

```
    <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
```

```
  </rdfs:Class>
```

```
  <rdfs:Class rdf:ID="Van">
```

```
    <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
```

```
  </rdfs:Class>
```

```
  <rdfs:Class rdf:ID="MiniVan">
```

```
    <rdfs:subClassOf rdf:resource="#Van"/>
```

```
    <rdfs:subClassOf rdf:resource="#PassengerVehicle"/>
```

```
  </rdfs:Class>
```

```
</rdf:RDF>
```

—————→ the rdf:type of MotorVehicle is rdfs:Class
(i.e., MotorVehicle is a Class)

```
<rdf:RDF
```

```
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:ex="http://example.org/schemas/vehicles#"
  xml:base="http://example.org/things">
```

```
  <ex:MiniVan rdf:ID="minivan123"/>
```

```
</rdf:RDF>
```

—————→ the rdf:type of minivan123 is ex:MiniVan
(i.e., minivan123 is a MiniVan)

N3 (Notation 3)

- This is a language which is a compact and readable alternative to RDF's XML syntax

```
@prefix rdf <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
@prefix cd <http://www.recshop.fake/cd>
```

```
<http://www.recshop.fake/cd/Empire Burlesque>
```

```
  cd:artist  Bob Dylan;
```

```
  cd:country USA;
```

```
  cd:company Columbia;
```

```
  cd:price  10.90;
```

```
  cd:year  1985.
```

```
<http://www.recshop.fake/cd/Hide your heart>
```

```
  cd:artist  Bonnie Tyler;
```

```
  cd:country  UK;
```

```
  cd:company  CBS Records;
```

```
  ....
```

<http://www.w3.org/DesignIssues/Notation3.html> (Berners-Lee)

SPARQL Query Language for RDF

Data:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:name "Johnny Lee Outlaw" .
_:a foaf:mbox <mailto:jlow@example.com> .
_:b foaf:name "Peter Goodguy" .
_:b foaf:mbox <mailto:peter@example.org> .
_:c foaf:mbox <mailto:carol@example.org> .
```

Query:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE
{ ?x foaf:name ?name .
  ?x foaf:mbox ?mbox }
```

Query Result:

| name | mbox |
|---------------------|----------------------------|
| "Johnny Lee Outlaw" | <mailto:jlow@example.com> |
| "Peter Goodguy" | <mailto:peter@example.org> |

Data:

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix : <http://example.org/book/> .
@prefix ns: <http://example.org/ns#> .

:book1 dc:title "SPARQL Tutorial" .
:book1 ns:price 42 .
:book2 dc:title "The Semantic Web" .
:book2 ns:price 23 .
```

Query:

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>
SELECT ?title ?price
WHERE { ?x ns:price ?price .
        FILTER (?price < 30.5)
        ?x dc:title ?title . }
```

Query Result:

| title | price |
|--------------------|-------|
| "The Semantic Web" | 23 |

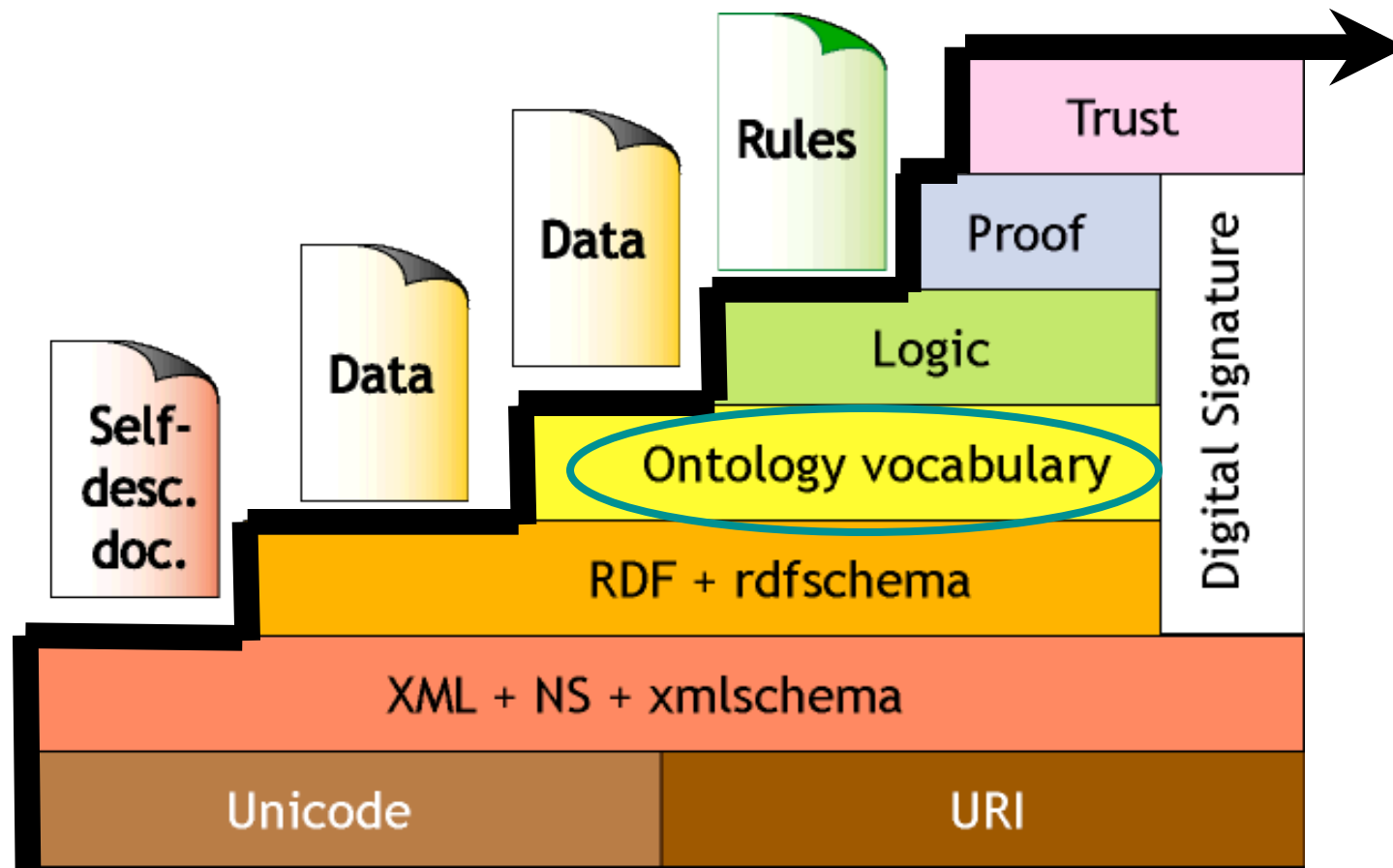
Summary: RDF & RDF Schema layer

- Minimalist model:
 - Class, Property
 - Subclass, Subproperty
 - Domain & Range
- RDF Schema: W3C recommendation, Feb 2004
 - <http://www.w3.org/RDF/>
- SPARQL: W3C recommendation, Jan 2008
 - <http://www.w3.org/TR/rdf-sparql-query/>
- Efficient storage and retrieval
 - “Triple store” using database backends

Limitations of RDF

- Cannot define properties of properties (unique, transitive)
- No equivalence, disjointness, etc.
- No mechanism of specifying necessary and sufficient conditions for class membership.
- Example: If it is given that 'XYZ' has a 'car' which is '7ft high', has 'wide wheels' and 'loading space is 4 cub.m', then we should be able to reason that 'XYZ' has an 'SUV', as given by the necessary and sufficient conditions for being an 'SUV' :
height > 4ft & wide wheels & loading space > 2 m³

Ontology Vocabulary



Semantic Web LayerCake (Berners-Lee, 99; Swartz-Hendler, 2001)

OWL: Web Ontology Language

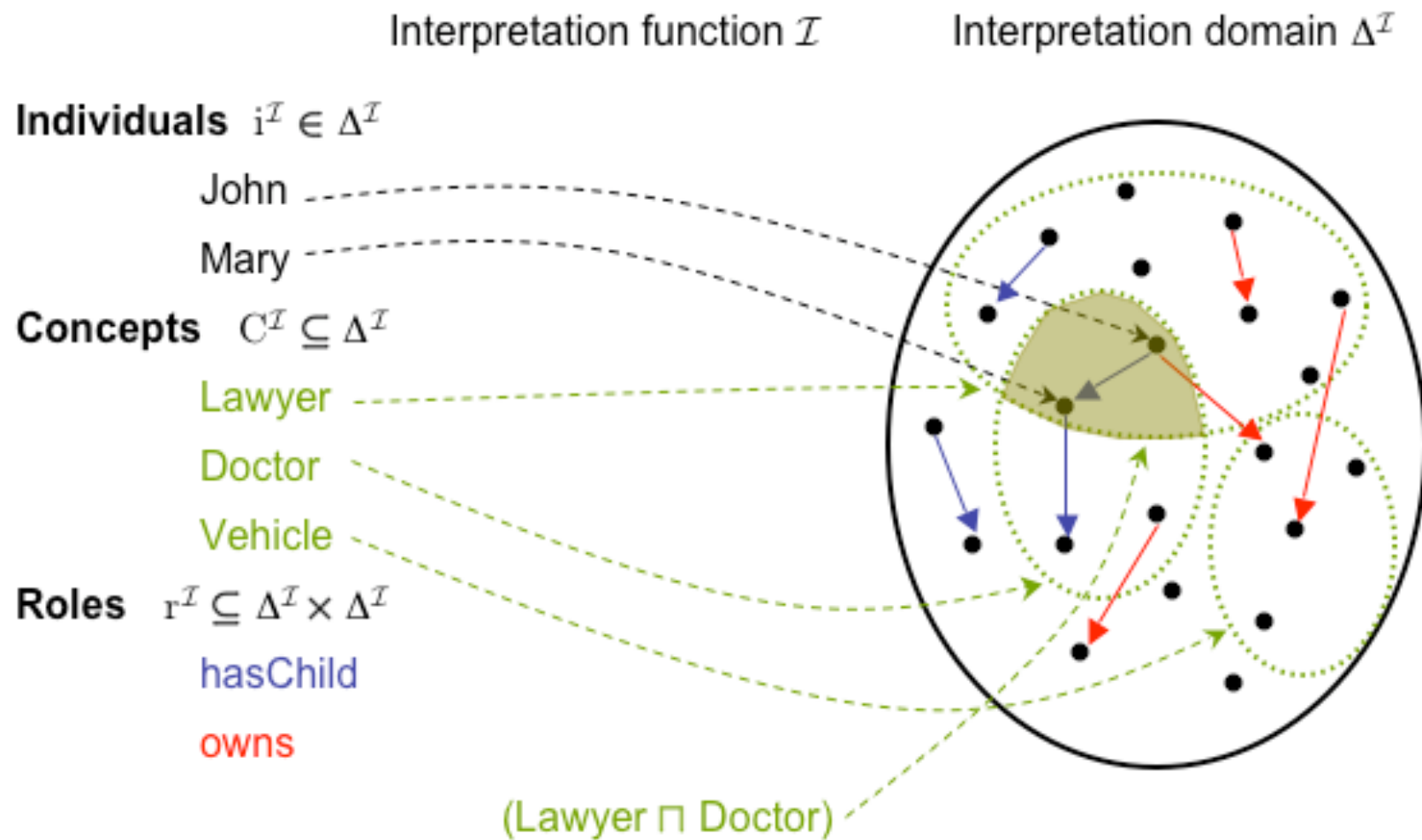
- W3C Recommendation (Feb 10, 2004)
- Description logic substrate
 - Extension of RDF schema
 - Concepts, roles, instances
 - Concept constructors
- OWL comes in three flavors
 - OWL Lite (*SHIF* description logic)
 - OWL DL (*SHOIN* description logic)
 - OWL Full
- OWL Web Ontology Language Overview
 - <http://www.w3.org/TR/owl-features/>
- Full details at:
 - <http://www.w3.org/2004/OWL/#specs>

Description Logic Basics

- **Concepts:** unary predicates/formulae with one free variable $p(x)$
 - E.g., Person, Doctor, HappyParent, (Doctor \wedge Lawyer)
- **Roles:** binary predicates/formulae with two free variables $r(x,y)$
 - E.g., hasChild, loves
- **Individuals:** constants
 - E.g., John, Mary, Italy
- **Concept/Role constructors** restricted so that:
 - Satisfiability/subsumption is decidable and, *if possible*, of low complexity
 - No need for explicit use of variables
 - Restricted form of \forall and \exists
 - Features such as counting can be succinctly expressed

Description Logic Semantics

Semantics given by standard first-order model:



The Description Logic Family

- Many description logics: depending on choice of concept/role constructors
- Smallest propositionally closed DL is **ALC**
 - Concepts constructed using boolean operators:
 \wedge (and), \vee (or), \neg (complement)
 - plus restricted quantifiers
 \exists (some), \forall (all)
 - Only atomic roles
- Example: Person all of whose children are either Doctors or have a child who is a Doctor:

$\text{Person} \wedge \forall \text{hasChild} . (\text{Doctor} \wedge \exists \text{hasChild} . \text{Doctor})$

OWL RDF/XML Exchange Syntax

E.g., $\text{Person} \wedge \forall \text{hasChild} . (\text{Doctor} \vee \exists \text{hasChild} . \text{Doctor})$:

```
<owl:Class>
  <owl:intersectionOf rdf:parseType=" collection">
    <owl:Class rdf:about="#Person"/>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasChild"/>
      <owl:allValuesFrom>
        <owl:unionOf rdf:parseType=" collection">
          <owl:Class rdf:about="#Doctor"/>
          <owl:Restriction>
            <owl:onProperty rdf:resource="#hasChild"/>
            <owl:someValuesFrom rdf:resource="#Doctor"/>
          </owl:Restriction>
        </owl:unionOf>
      </owl:allValuesFrom>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

Class/Concept Constructors

| Constructor | DL Syntax | Example | FOL Syntax |
|----------------|---------------------------------------|---------------------------|--------------------------------------|
| intersectionOf | $C_1 \sqcap \dots \sqcap C_n$ | Human \sqcap Male | $C_1(x) \wedge \dots \wedge C_n(x)$ |
| unionOf | $C_1 \sqcup \dots \sqcup C_n$ | Doctor \sqcup Lawyer | $C_1(x) \vee \dots \vee C_n(x)$ |
| complementOf | $\neg C$ | \neg Male | $\neg C(x)$ |
| oneOf | $\{x_1\} \sqcup \dots \sqcup \{x_n\}$ | {john} \sqcup {mary} | $x = x_1 \vee \dots \vee x = x_n$ |
| allValuesFrom | $\forall P.C$ | \forall hasChild.Doctor | $\forall y.P(x, y) \rightarrow C(y)$ |
| someValuesFrom | $\exists P.C$ | \exists hasChild.Lawyer | $\exists y.P(x, y) \wedge C(y)$ |
| maxCardinality | $\leqslant_n P$ | $\leqslant 1$ hasChild | $\exists \leqslant_n y.P(x, y)$ |
| minCardinality | $\geqslant_n P$ | $\geqslant 2$ hasChild | $\exists \geqslant_n y.P(x, y)$ |

C is a concept (class); P is a role (property); x is an individual name

Ontology Axioms

| OWL Syntax | DL Syntax | Example |
|--------------------|-----------------------|--|
| subClassOf | $C_1 \sqsubseteq C_2$ | Human \sqsubseteq Animal \sqcap Biped |
| equivalentClass | $C_1 \equiv C_2$ | Man \equiv Human \sqcap Male |
| subPropertyOf | $P_1 \sqsubseteq P_2$ | hasDaughter \sqsubseteq hasChild |
| equivalentProperty | $P_1 \equiv P_2$ | cost \equiv price |
| transitiveProperty | $P^+ \sqsubseteq P$ | ancestor ⁺ \sqsubseteq ancestor |

| OWL Syntax | DL Syntax | Example |
|------------|----------------------------|---|
| type | $a : C$ | John : Happy-Father |
| property | $\langle a, b \rangle : R$ | $\langle \text{John}, \text{Mary} \rangle : \text{has-child}$ |

The Description Logic Family (2)

- S often used for ALC extended with transitive roles
- Additional letters indicate other extensions, e.g.:
 - H for role hierarchy (e.g., $\text{hasDaughter} \sqsubseteq \text{hasChild}$)
 - O for nominals/singleton classes (e.g., $\{\text{Italy}\}$)
 - I for inverse roles (e.g., isChildOf **inverse of** hasChild)
 - N for number restrictions (e.g., $\geq 2 \text{ hasChild}$, $\leq 3 \text{ hasChild}$)
 - Q for qualified number restrictions (e.g., $> 2 \text{ hasChild.Doctor}$)
 - F for functional number restrictions ($\text{Functional}(\text{hasMother})$)
- S + role hierarchy (H) + inverse (I) + QNR (Q) = SHIQ
- SHIQ is the basis for OWL
 - OWL Lite SHIQ with functional restrictions (i.e., SHIF)
 - OWL DL SHIQ extended with nominals (i.e., SHOIQ)

OWL-Lite

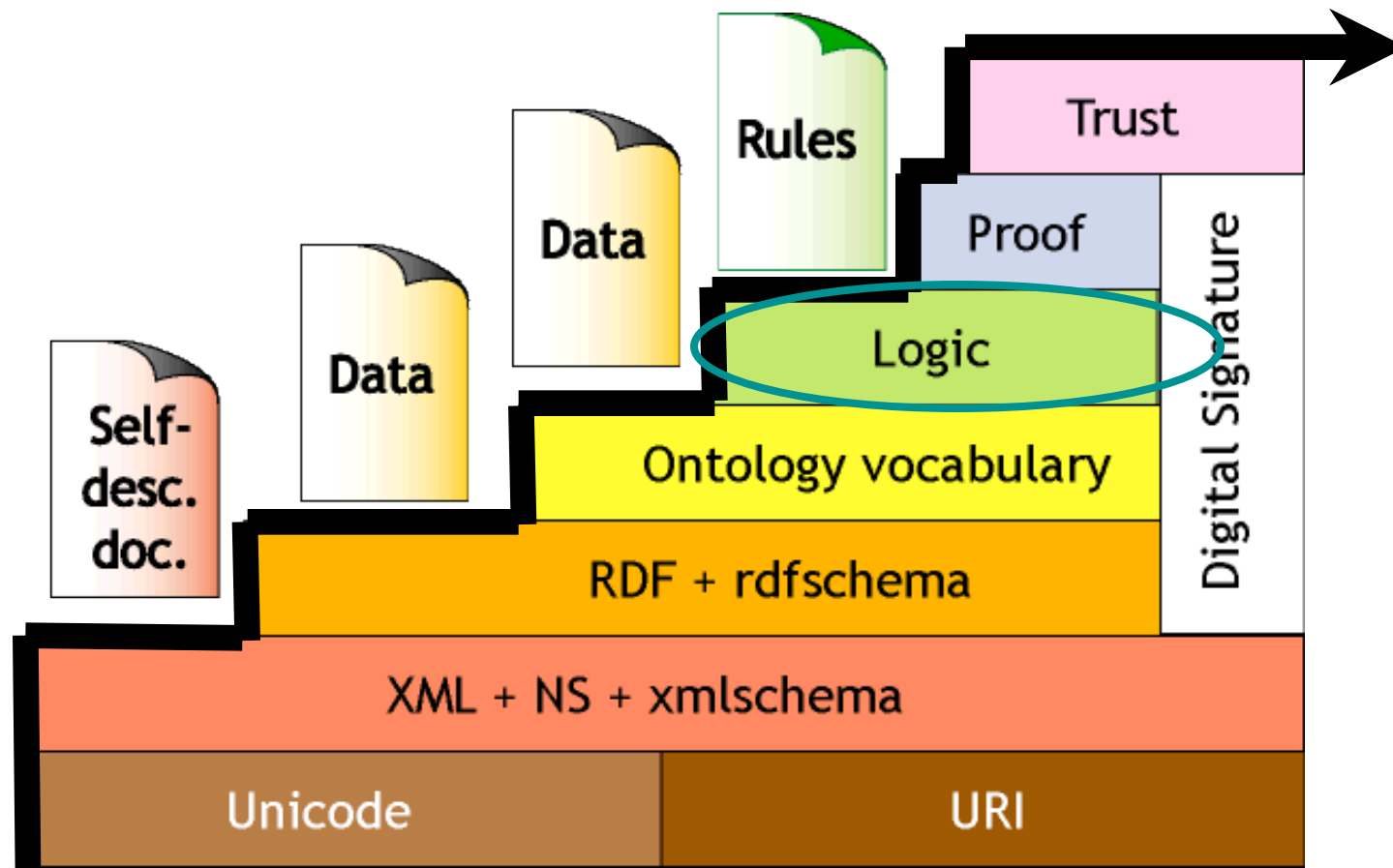
- **RDF Schema Features:**
 - [Class \(+ Thing, Nothing\)](#)
 - [Individual](#)
 - [rdfs:subClassOf](#)
 - [rdf:Property](#)
 - [rdfs:subPropertyOf](#)
 - [rdfs:domain](#)
 - [rdfs:range](#)
- **Class Intersection:**
 - [intersectionOf](#)
- **(In)Equality:**
 - [equivalentClass](#)
 - [equivalentProperty](#)
 - [sameAs](#)
 - [differentFrom](#)
 - [AllDifferent](#)
 - [distinctMembers](#)
- **Property Characteristics:**
 - [ObjectProperty](#)
 - [DatatypeProperty](#)
 - [inverseOf](#)
 - [TransitiveProperty](#)
 - [SymmetricProperty](#)
 - [FunctionalProperty](#)
 - [InverseFunctionalProperty](#)
- **Property Restrictions:**
 - [allValuesFrom](#)
 - [someValuesFrom](#)
- **Restricted Cardinality:**
 - [minCardinality](#) (only 0 or 1)
 - [maxCardinality](#) (only 0 or 1)
 - [cardinality](#) (only 0 or 1)

OWL-DL

OWL-Lite +

- **Class Axioms:**
 - [oneOf, dataRange](#)
 - [disjointWith](#)
 - [equivalentClass](#) (applied to class expressions)
 - [rdfs:subClassOf](#) (applied to class expressions)
- **Boolean Combinations of Class Expressions:**
 - [unionOf](#)
 - [complementOf](#)
 - [intersectionOf](#)
- **Arbitrary Cardinality:**
 - [minCardinality](#)
 - [maxCardinality](#)
 - [cardinality](#)
- **Filler Information:**
 - [hasValue](#)

Logic

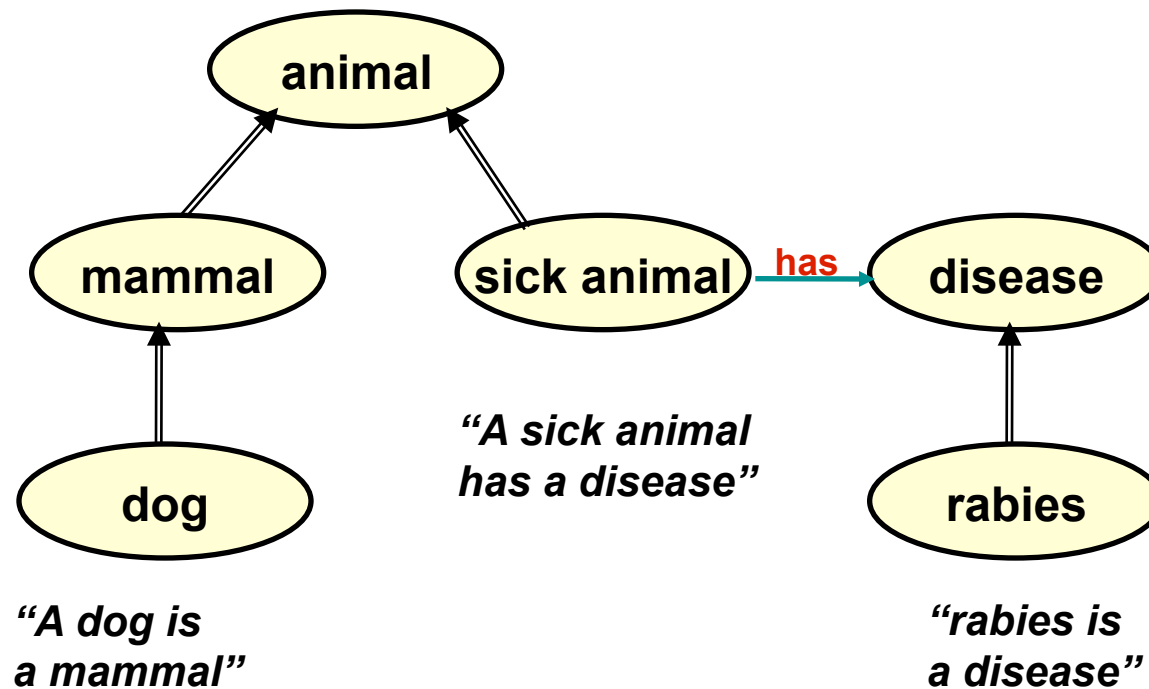


Semantic Web LayerCake (Berners-Lee, 99; Swartz-Hendler, 2001)

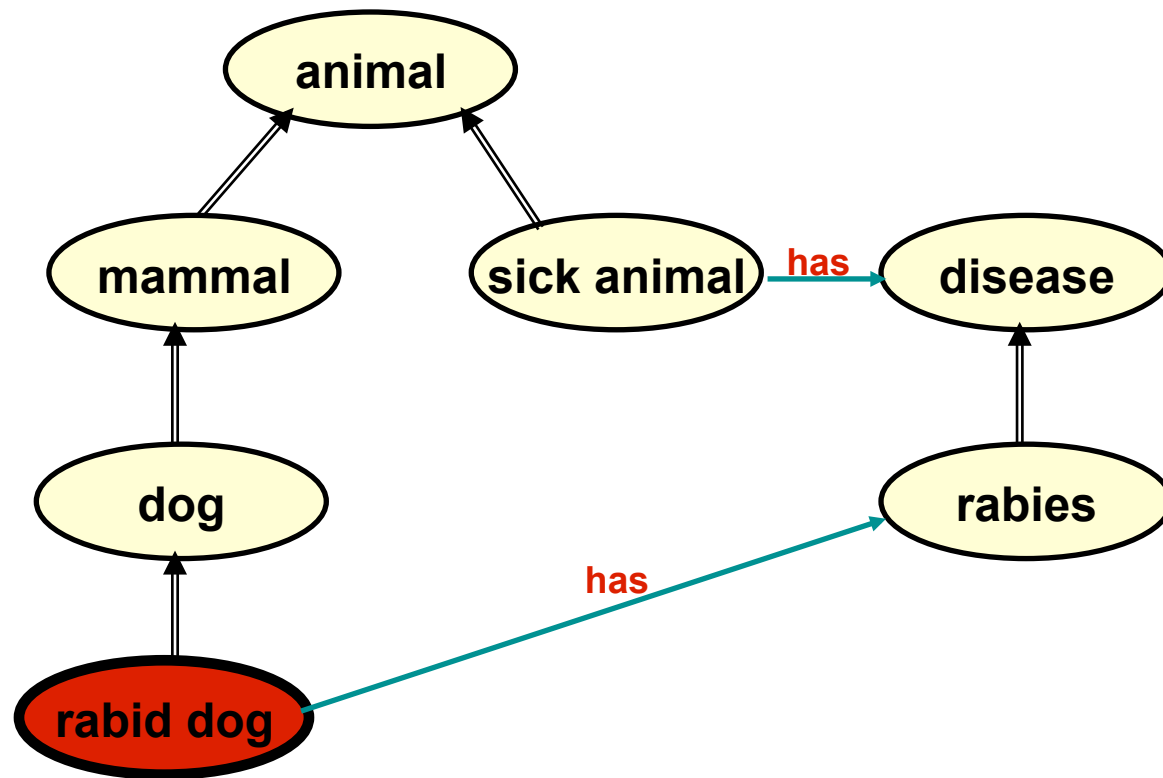
Description Logics

- Classes are defined in terms of other classes/relations
- Powerful inference algorithms:
 - **Subsumption**: is classA a subclass of classB given their definitions?
 - **Recognition**: is instanceA of classA?
 - **Classification**: automatic reorganization of class hierarchy based on definitions of classes
- Logical proofs

Classification: Defining an Ontology

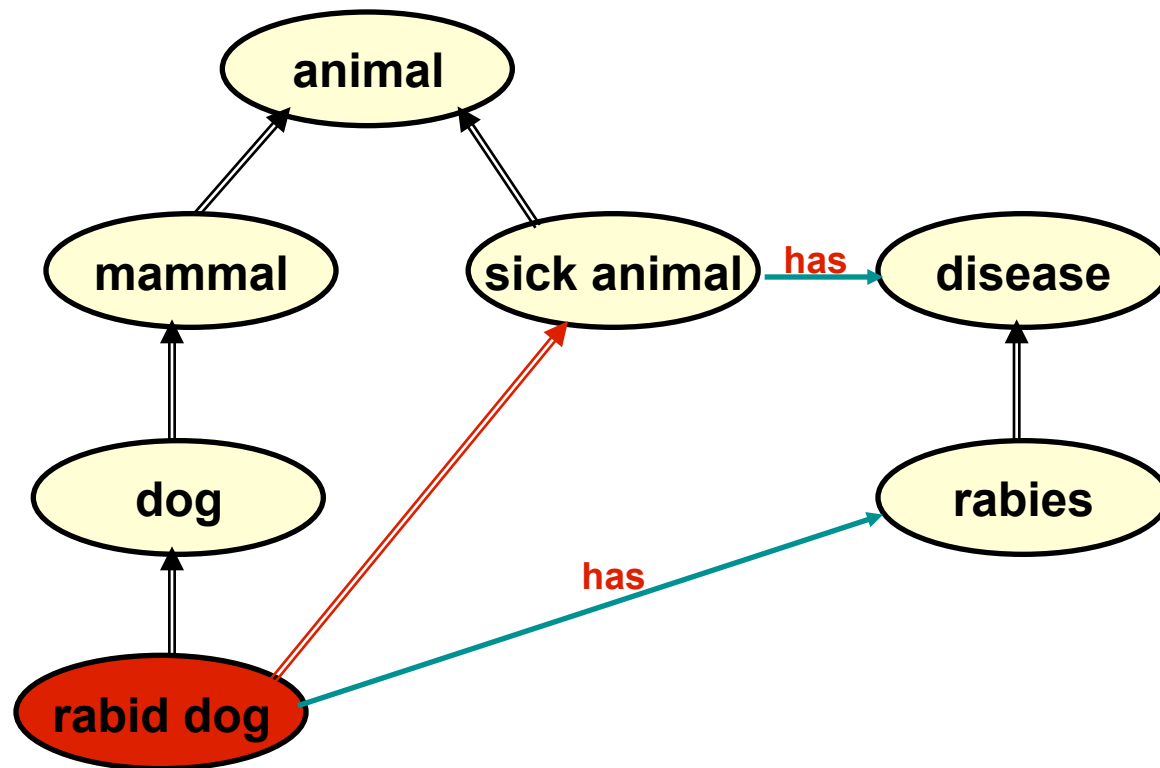


Classification: Defining a “rabid dog”

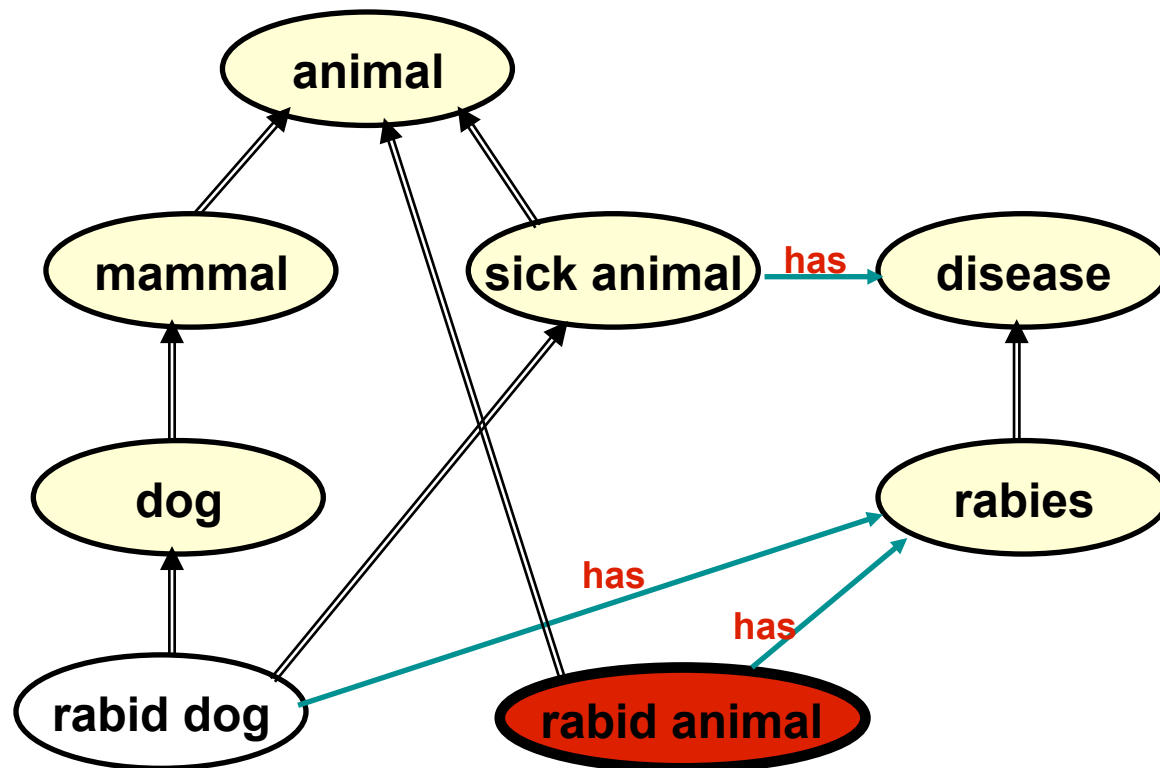


Classification:

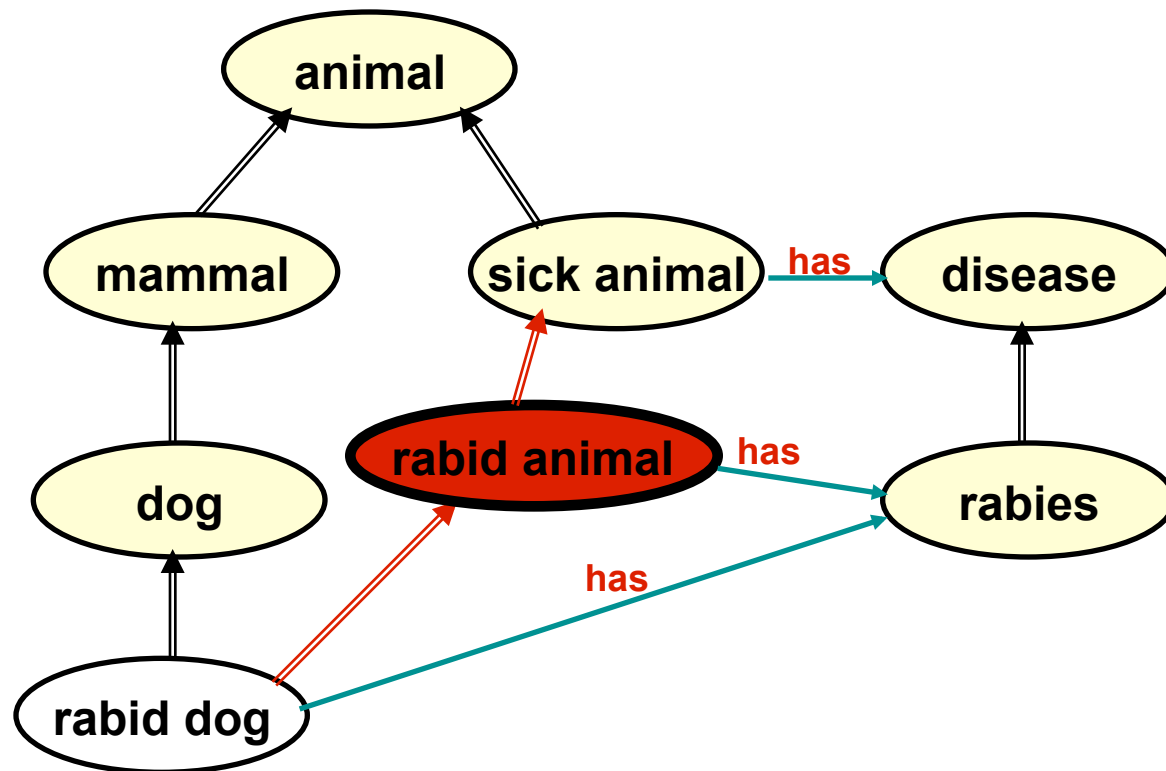
Classifier Infers “sick animal”



Classification: Defining “rabid animal”



Classification: Concept Placed in Hierarchy





Complexity of reasoning in Description Logics

Note: the information here is (always) incomplete and updated often

Base description logic: *Attributive Language with Complements*

$\mathcal{ALC} ::= \perp \mid A \mid \neg C \mid C \wedge D \mid C \vee D \mid \exists R.C \mid \forall R.C$



Concept constructors:

- ☐ \mathcal{F} – functionality²: $(\leq 1 R)$
- ☐ \mathcal{N} – (unqualified) number restrictions: $(\geq n R), (\leq n R)$
- ☐ \mathcal{Q} – qualified number restrictions: $(\geq n R.C), (\leq n R.C)$
- ☐ \mathcal{O} – nominals: $\{a\}$ or $\{a_1, \dots, a_n\}$ ("one-of" constructor)
- ☐ μ – least fixpoint operator: $\mu X.C$
- ☐ $R \subseteq S$ – role-value-maps
- ☐ $f = g$ – agreement of functional role chains ("same-as")

TBox options:

- ☒ Empty TBox
- ☐ Acyclic TBox ($A \sqsubseteq C$, A is a concept name; no cycles)
- ☐ General TBox ($C \sqsubseteq D$ for arbitrary concepts C and D)

Role constructors:

- ☐ \mathcal{I} – role inverses: R^-
- ☐ \cap – role intersection³: $R \cap S$
- ☐ \cup – role union: $R \cup S$
- ☐ \neg – role complement: full
- ☐ \circ – role chain (composition): $R \circ S$
- ☐ $*$ – reflexive-transitive closure⁴: R^*
- ☐ id – concept identity: $id(C)$
- complex roles⁵ in number restrictions⁶

ALC

Role axioms (RBox):

- ☐ \mathcal{S} – Role transitivity: $\text{Trans}(R)$
- ☐ \mathcal{H} – Role hierarchy: $R \sqsubseteq S$
- ☐ \mathcal{R} – Complex role inclusions: $R \circ S \sqsubseteq R, R \circ S \sqsubseteq S$
- ☐ s – some additional features

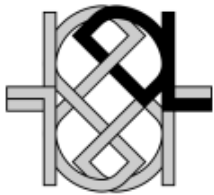
You have selected the Description Logic: \mathcal{ALC}

Complexity of reasoning problems⁷

| Reasoning problem | Complexity ⁸ | Comments and references |
|------------------------|-------------------------|--|
| Concept satisfiability | PSpace-complete | <ul style="list-style-type: none"> • <u>Hardness</u> for \mathcal{ALC}: see [73]. • <u>Upper bound</u> for \mathcal{ALCQ}: see [77, Theorem 4.6]. |
| ABox consistency | PSpace-complete | <ul style="list-style-type: none"> • <u>Hardness</u> follows from that for concept satisfiability. • <u>Upper bound</u> for \mathcal{ALCQO}: see [7, Appendix A]. |

Important properties of the description logic

| | | |
|-----------------------|------------|---|
| Finite model property | Yes | \mathcal{ALC} is a notational variant of the multi-modal logic \mathbf{K}_m (cf. [70]), for which the finite model property can be found in [2, Sect. 2.3]. |
|-----------------------|------------|---|



Complexity of reasoning in Description Logics

Note: the information here is (always) incomplete and [updated](#) often

Base description logic: *Attributive Language with Complements*

$\mathcal{ALC} ::= \perp \mid A \mid \neg C \mid C \wedge D \mid C \vee D \mid \exists R.C \mid \forall R.C$



Concept constructors:

- ☒ \mathcal{F} – functionality²: $(\leq 1 R)$
- ☐ \mathcal{N} – (unqualified) number restrictions: $(\geq n R), (\leq n R)$
- ☐ \mathcal{Q} – qualified number restrictions: $(\geq n R.C), (\leq n R.C)$
- ☐ \mathcal{O} – nominals: $\{a\}$ or $\{a_1, \dots, a_n\}$ ("one-of" constructor)
- ☐ μ – least fixpoint operator: $\mu X.C$
- ☐ $R \subseteq S$ – role-value-maps
- ☐ $f = g$ – agreement of functional role chains ("same-as")

TBox is *internalized* in extensions of \mathcal{SH} , see [40, 46].

- ☒ Empty TBox
- ☐ Acyclic TBox ($A \sqsubseteq C$, A is a concept name; no cycles)
- ☐ General TBox ($C \sqsubseteq D$ for arbitrary concepts C and D)

Role constructors:

- ☒ \mathcal{I} – role inverses: R^-
- ☐ \cap – role intersection³: $R \cap S$
- ☐ \cup – role union: $R \cup S$
- ☐ \neg – role complement:
- ☐ \circ – role chain (composition): RoS
- ☐ $*$ – reflexive-transitive closure⁴: R^*
- ☐ id – concept identity: $id(C)$
- ⁵ in number restrictions⁶

Role axioms (RBox):

- ☒ \mathcal{S} – Role transitivity: $\text{Trans}(R)$
- ☒ \mathcal{H} – Role hierarchy: $R \subseteq S$
- ☐ \mathcal{R} – Complex role inclusions: $RoS \subseteq R, RoS \subseteq S$
- ☐ s – some additional features

You have selected the Description Logic: *SHIF*

Complexity of reasoning problems⁷

| Reasoning problem | Complexity ⁸ | Comments and references |
|------------------------|-------------------------|--|
| Concept satisfiability | ExpTime-complete | <ul style="list-style-type: none">• <u>Hardness</u> follows from internalization of TBoxes in any extension of \mathcal{SH} and ExpTime-hardness of \mathcal{ALC}-concept satisfiability w.r.t. general TBoxes.• <u>Upper bound</u> for <i>SHIQ</i> see [77, Corollary 6.29, 6.30]; for <i>SHIO</i> see [33, Th.3]; for <i>SHOQ</i> see [?].• Important: in number restrictions, only <i>simple</i> roles (i.e. which are neither transitive nor have a transitive subroles) are allowed; however, according to [46], it is an open problem whether <i>SHF</i> or <i>SHIF</i> becomes undecidable without this restriction. |

OWL-Lite



Complexity of reasoning in Description Logics

Note: the information here is (always) incomplete and [updated](#) often

Base description logic: *Attributive Language with Complements*

$\mathcal{ALC} ::= \perp \mid A \mid \neg C \mid C \wedge D \mid C \vee D \mid \exists R.C \mid \forall R.C$



Concept constructors:

- ☐ \mathcal{F} – functionality²: $(\leq 1 R)$
 - ☒ \mathcal{N} – (unqualified) number restrictions: $(\geq n R)$, $(\leq n R)$
 - ☐ \mathcal{Q} – qualified number restrictions: $(\geq n R.C)$, $(\leq n R.C)$
 - ☒ \mathcal{O} – nominals: $\{a\}$ or $\{a_1, \dots, a_n\}$ ("one-of" constructor)
-
- ☐ μ – least fixpoint operator: $\mu X.C$
 - ☐ $R \subseteq S$ – role-value-maps
 - ☐ $f = g$ – agreement of functional role chains ("same-as")

TBox is *internalized* in extensions of \mathcal{ALCIO} , see

[76, Lemma 4.12], [54, p.3]

- ☒ Empty TBox
- ☐ Acyclic TBox ($A \sqsubseteq C$, A is a concept name; no cycles)
- ☐ General TBox ($C \sqsubseteq D$ for arbitrary concepts C and D)

Role constructors:

- ☒ \mathcal{I} – role inverses: R^-
- ☐ \cap – role intersection³: $R \cap S$
- ☐ \cup – role union: $R \cup S$
- ☐ \neg – role complement:
- ☐ \circ – role chain (composition): $R \circ S$
- ☐ $*$ – reflexive-transitive closure⁴: R^*
- ☐ id – concept identity: $id(C)$
- complex roles⁵ in number restrictions⁶

Role axioms (RBox):

- ☒ \mathcal{S} – Role transitivity: $\text{Trans}(R)$
- ☒ \mathcal{H} – Role hierarchy: $R \sqsubseteq S$
- ☐ \mathcal{R} – Complex role inclusions: $R \circ S \sqsubseteq R$, $R \circ S \sqsubseteq S$
- ☐ s – some additional features

You have selected the Description Logic: *SHOIN*

Complexity of reasoning problems⁷

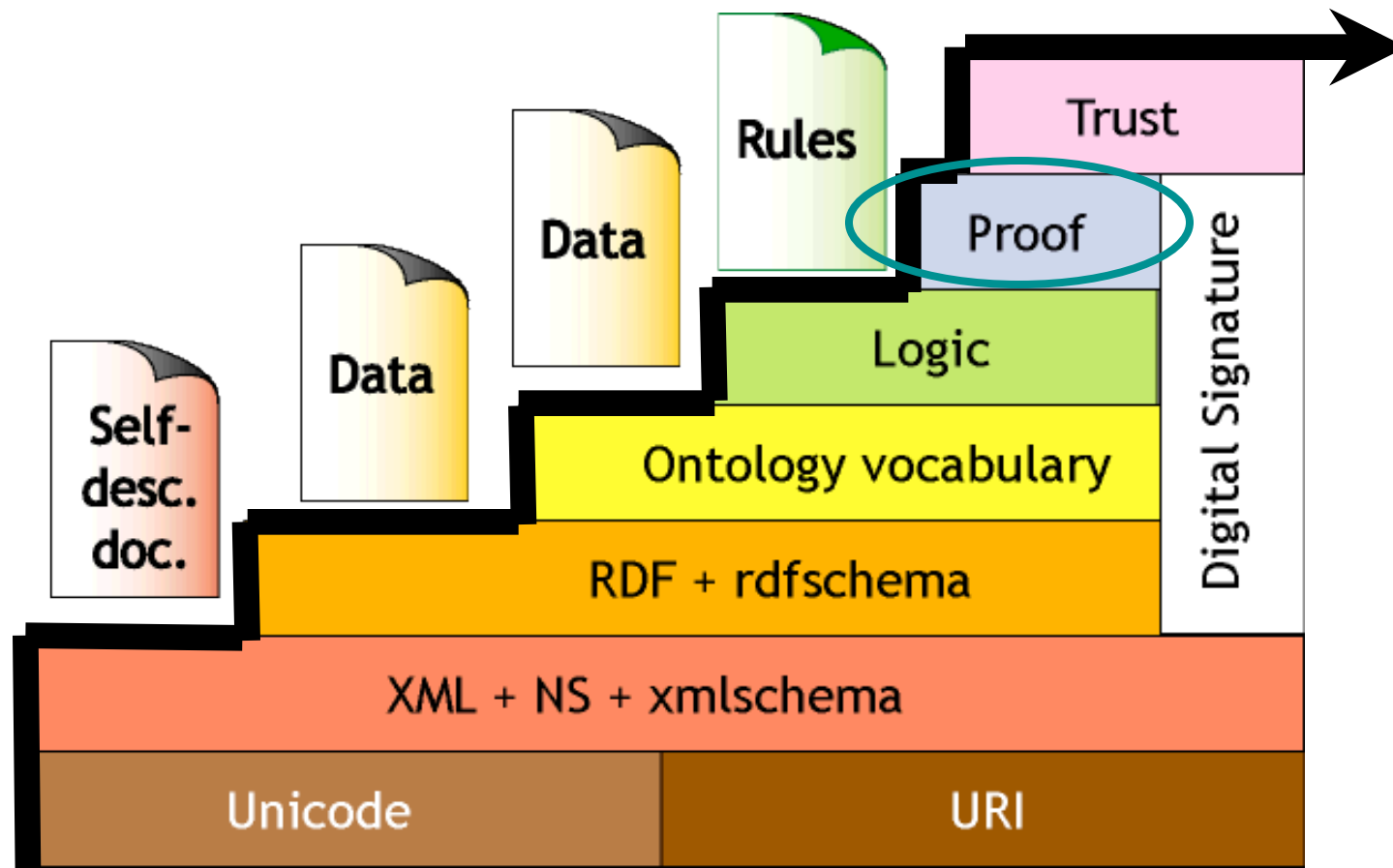
| Reasoning problem | Complexity ⁸ | Comments and references |
|------------------------|--------------------------|---|
| Concept satisfiability | NExpTime-complete | <ul style="list-style-type: none">• <u>Hardness</u> of even \mathcal{ALCFIO} is proved in [76, Corollary 4.13]. In that paper, the result is formulated for \mathcal{ALCQIO}, but only number restrictions of the form $(\leq 1R)$ are used in the proof.• A different proof of the NExpTime-hardness for \mathcal{ALCFIO} is given in [54] (even with 1 nominal, and role inverses not used in number restrictions).• <u>Upper bound</u> for \mathcal{SHOIQ} is proved in [77, Corollary 6.31] with numbers coded in unary (for binary coding, the upper bound remains an open problem for all logics in between \mathcal{ALCNIO} and \mathcal{SHOIQ}).• Important: in number restrictions, only <i>simple</i> roles (i.e. which are neither transitive nor have a transitive subrole) are allowed; otherwise |

OWL-DL

Resources for OWL and DL

- Description Logic Handbook, Cambridge University Press
 - <http://books.cambridge.org/0521781760.htm>
- Description Logic: <http://dl.kr.org/>
 - complexity: <http://www.cs.man.ac.uk/~ezolin/dl>
- Web Ontology Language (OWL): <http://www.w3.org/2004/OWL/>
- Reasoners:
 - Pellet (open source): <http://pellet.owldl.com/>
 - FaCT++ (open source): <http://owl.man.ac.uk/factplusplus/>
 - Racer (comercial): <http://www.racer-systems.com/>
 - (Loom and Powerloom: <http://www.isi.edu/isd/LOOM/>)
- Ontology Editors:
 - Protégé: <http://protege.stanford.edu/>
- Ian Horrocks has great slides on description logics and OWL:
 - <http://web.comlab.ox.ac.uk/oucl/work/ian.horrocks/>

Proof

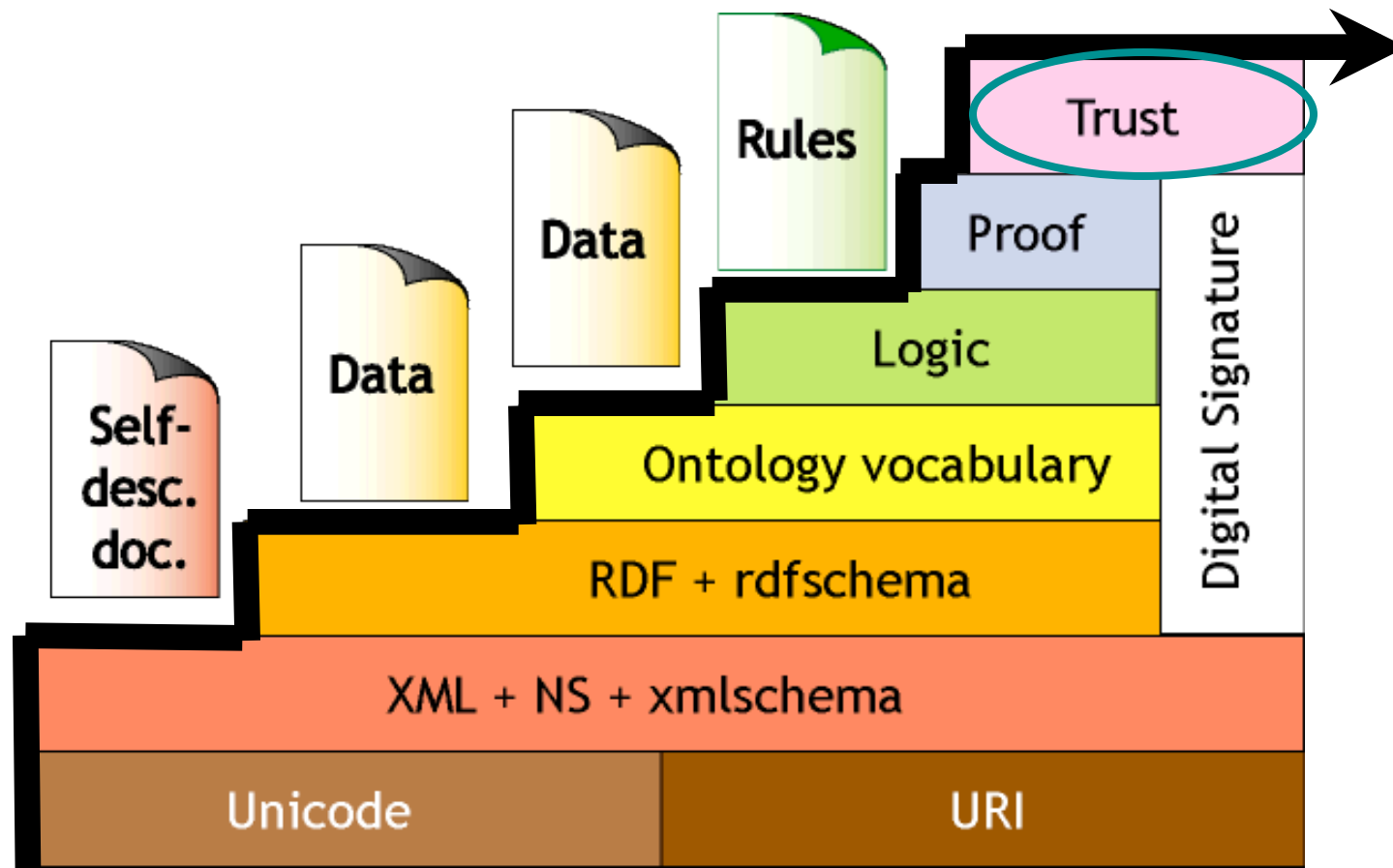


Semantic Web LayerCake (Berners-Lee, 99; Swartz-Hendler, 2001)

Proofs: Logical Derivations

- Use the logic to prove things given the set of facts provided
- The derivation of the proof provides the support for the derived facts
- Easier to verify a proof than it is to find one

Trust



Semantic Web LayerCake (Berners-Lee, 99; Swartz-Hendler, 2001)

Can We Trust the Result

- Need a mechanism to determine who to trust
- Exploit digital signatures to verify that information comes from a trusted source
- Define a “Web of Trust”
 - You tell the system who you want to trust

W3C's Semantic Web Principles

- Everything identifiable is in the Semantic Web (URIs!)
- Partial information
 - Anyone can say anything about anything
- Web of trust
 - All statements on the Web occur in some context
- Evolution
 - Allow combining independent work done by different communities
- Minimalist design
 - Make the simple things simple, and the complex things possible
 - Standardize no more than is necessary

Hypertext: Then and Now

- SOTA circa 1990: Dynatext's electronic book
 - A book had to be compiled (like a program) in order to be displayed efficiently
 - A central link database, to make sure there were no broken links
 - Text that was fixed and consistent (a whole book)
- WWW:
 - Links can be added and used at any time
 - Distributed (must live with broken links!)
 - Decentralized

Knowledge Representation: Now and Tomorrow

“To webize KR in general is, in many ways, the same as to webize hypertext. Replace identifiers with URIs. Remove any requirement for global consistency. Put any significant effort into getting critical mass. Sit back.”

-- TBL