



Big Data and Data Mining

***Semi-structured and
unstructured data***

Flavio Bertini

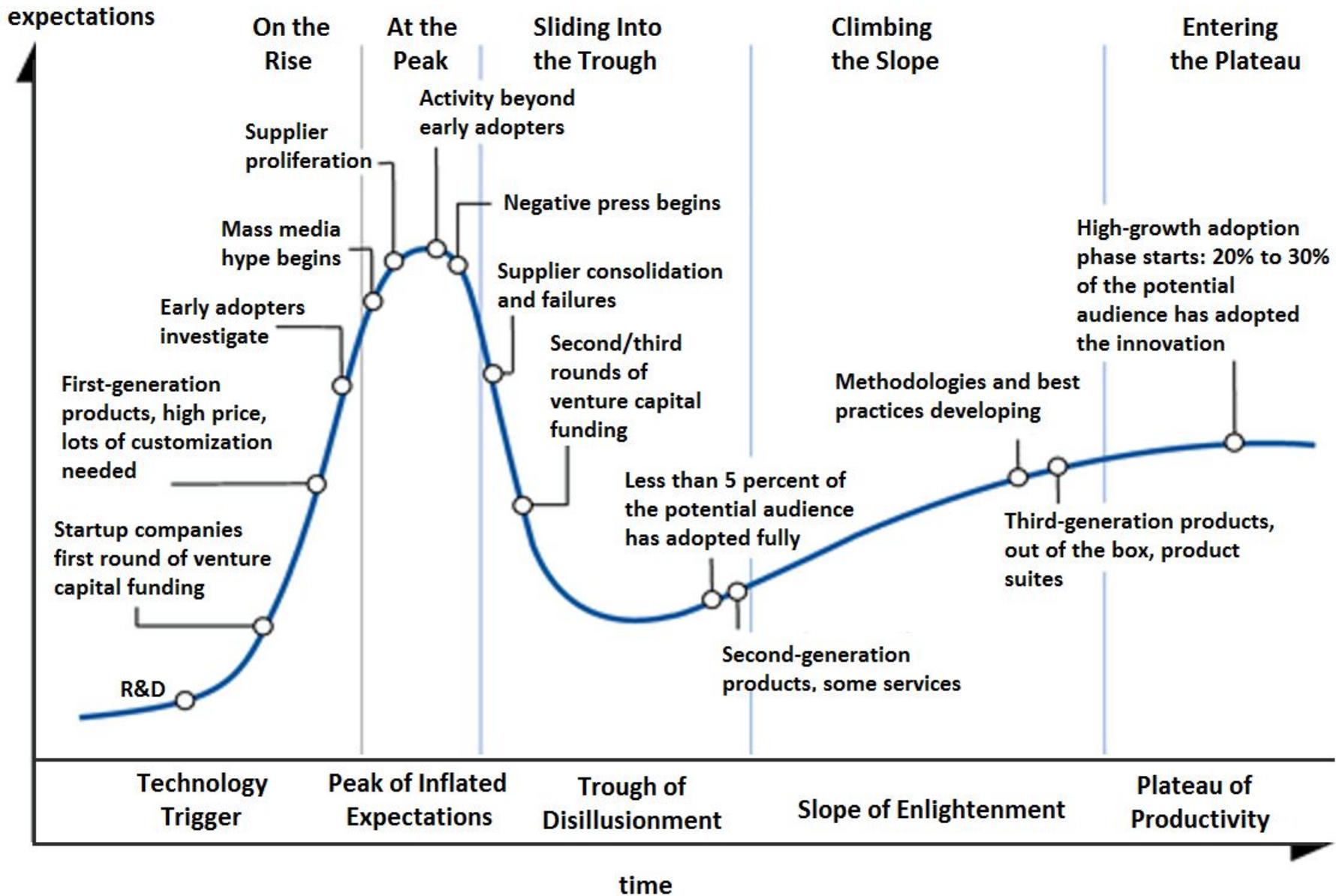
flavio.bertini@unipr.it



Motivations (1)

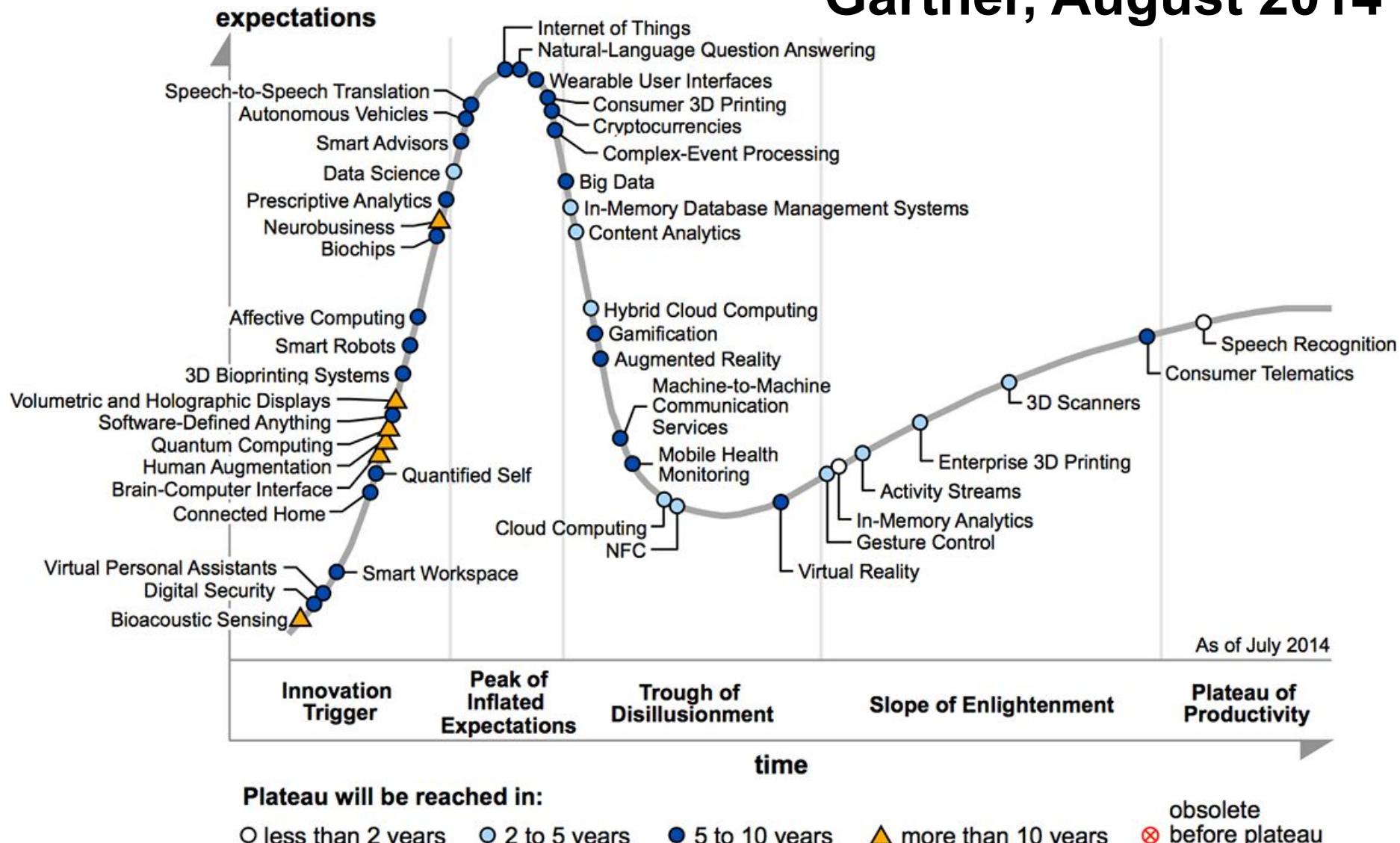


Motivations (2)



Motivations (3)

Gartner, August 2014





Introduction

- Management systems of relational data are used in many significant applications, as in information systems of banks and big companies
 - However, most digital data available today is not in relational form
- The production of massive amounts of non-relational data has intensified over time, due to the diffusion of Internet and media sharing platforms
 - This kind of data has generally different properties with respect to the data managed by relational systems

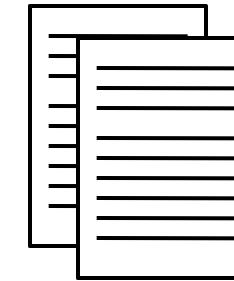
Structural classification of data

STRUCTURED

<i>id-pers</i>	<i>name</i>	<i>surname</i>
0000001	Jon	Doe
0000002	Bob	Walker

<i>id-pers</i>	<i>phone</i>
0000001	051 1234
0000001	333 3333

UNSTRUCTURED

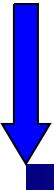




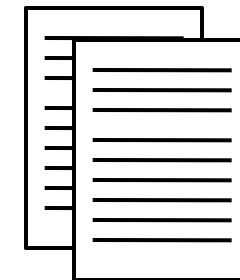
Structured data

<i>id-pers</i>	<i>name</i>	<i>surname</i>
0000001	Jon	Doe
0000002	Bob	Walker

<i>id-pers</i>	<i>phone</i>
0000001	051 1234
0000001	333 3333



STRUCTURED DATA (SCHEMA)



Unstructured data (1)

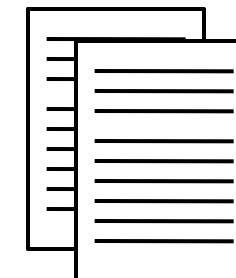


vu45s89gysJPGi
8gbyygsvs954gy
4598y9syg5vts9
4lygs98yg9s45y
g584gyt459gyg4
...



<i>id-pers</i>	<i>name</i>	<i>surname</i>
0000001	Jon	Doe
0000002	Bob	Walker

<i>id-pers</i>	<i>phone</i>
0000001	051 1234
0000001	333 3333



***RAW
DATA***

Unstructured data (2)

The old believe everything, the middle-aged suspect everything, the young know everything ...
O. Wilde

How can you prove whether at this moment we are sleeping, and all our thoughts are a dream; or whether we are awake, and talking to one another in the waking state?

Plato



<i>id-pers</i>	<i>name</i>	<i>surname</i>
0000001	Jon	Doe
0000002	Bob	Walker

<i>id-pers</i>	<i>phone</i>
0000001	051 1234
0000001	333 3333



**DATA
WITHOUT
SCHEMA**





Semi-structured data

A screenshot of the New York Times website homepage. The URL 'www.nytimes.com' is in the address bar. The page features the 'The New York Times' masthead, a navigation bar with 'SECTIONS' and 'SEARCH' buttons, and a main headline 'Obama Tells Mourning Dallas 'We Are...'' with a small image of Obama and another person. Below the headline are links for 'World', 'U.S.', 'Politics', 'N.Y.', 'Business', 'Opinion', 'Tech', and 'Sci'.

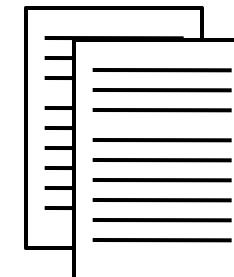
```
<html>  
<title>The New York Times</title>  
  
Tuesday, July 12, 2016 ...  
  
<img/></html>
```



<i>id-pers</i>	<i>name</i>	<i>surname</i>
0000001	Jon	Doe
0000002	Bob	Walker

<i>id-pers</i>	<i>phone</i>
0000001	051 1234
0000001	333 3333

DATA WITH
PARTIAL
STRUCTURE

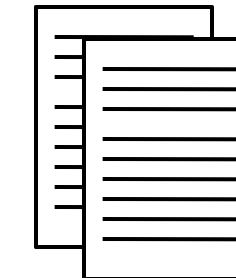


Non-relational data

- Rightmost data of this classification is called **unstructured** data, and it needs specific processing, as we will see
 - The research area which study how to manage and access these kind of data is called **Information Retrieval**
- Data in the middle is called **semi-structured** data, and it shows properties of both structured and unstructured data
 - One of the most used language for semi-structured data representation is **XML**
- We will see now some examples to show why the relational model is not suitable to manage this kind of data

<i>id-pers</i>	<i>name</i>	<i>surname</i>
0000001	Jon	Doe
0000002	Bob	Walker

<i>id-pers</i>	<i>phone</i>
0000001	051 1234
0000001	333 3333





Semi-structured data



Relational and semi-structured data: first comparison

Relational	Semi-structured
Clear distinction between schema and data	Partial schema with same properties of data
Based on the concept of set	Based on the concept of list
Unordered	Ordered
Not nested	Nested



Extensible Markup Language a.k.a. XML

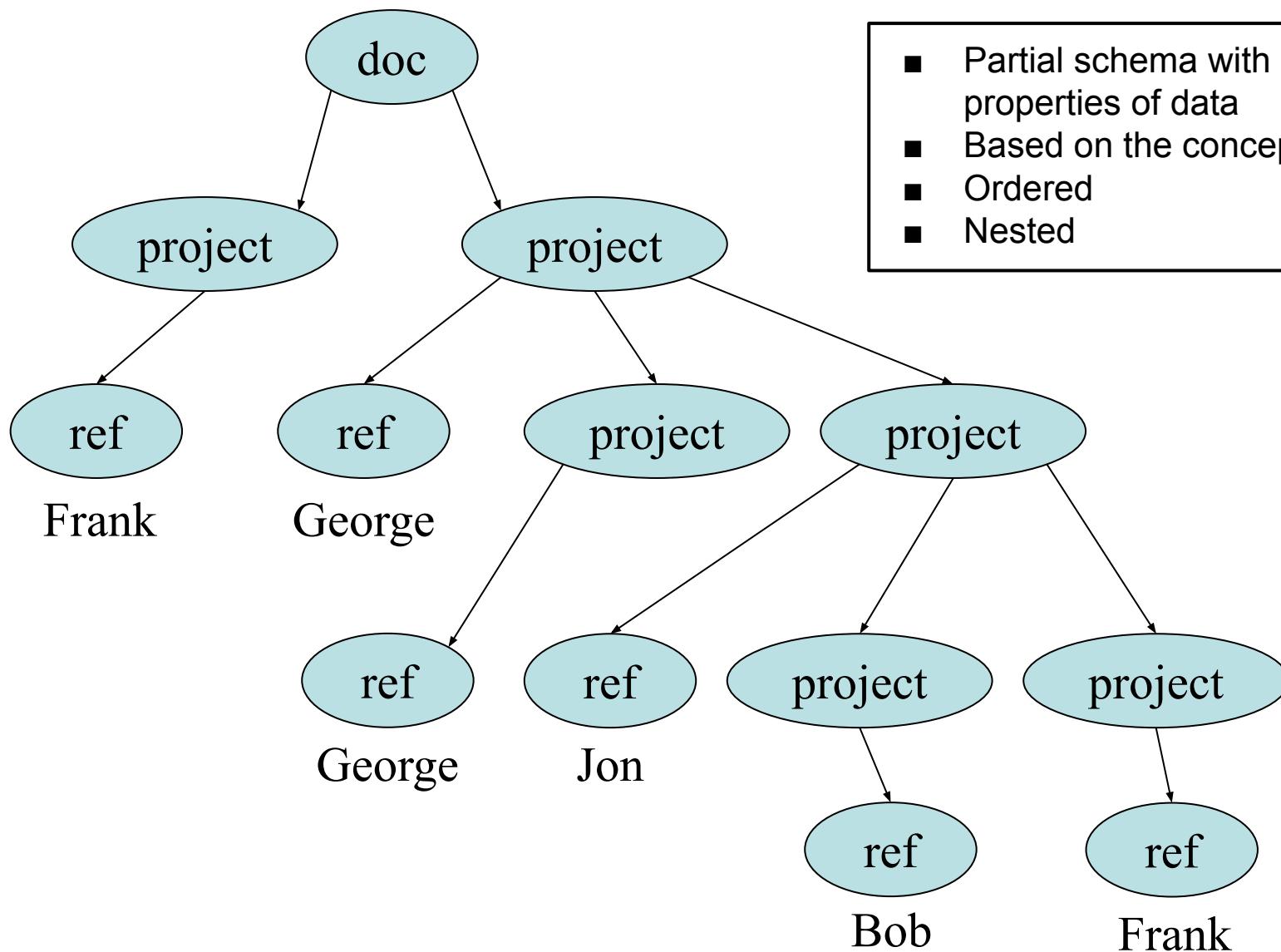
- The main format for semi-structured data representation is XML (**E**x~~t~~ensible **M**arkup **L**anguage), that is a markup language similar to HTML, but without predefined tags
- It can be used
 1. to represent structured data, for example in order to exchange them between different applications, but also
 2. to represent semi-structured data, exploiting the flexibility and the capability of indicating both the data and the schema
- In the first case, data can be first located in a relational system, and then be converted to XML
- In the second case, the relational model is not particularly suited to manage this data. Let's see an example



A relational model for XML data (1)

```
<doc>
  <project><ref>Frank</ref></project>
  <project>
    <ref>George</ref>
    <project><ref>George</ref></project>
    <project><ref>Jon</ref>
      <project><ref>Bob</ref></project>
      <project><ref>Frank</ref></project>
    </project>
  </project>
</doc>
```

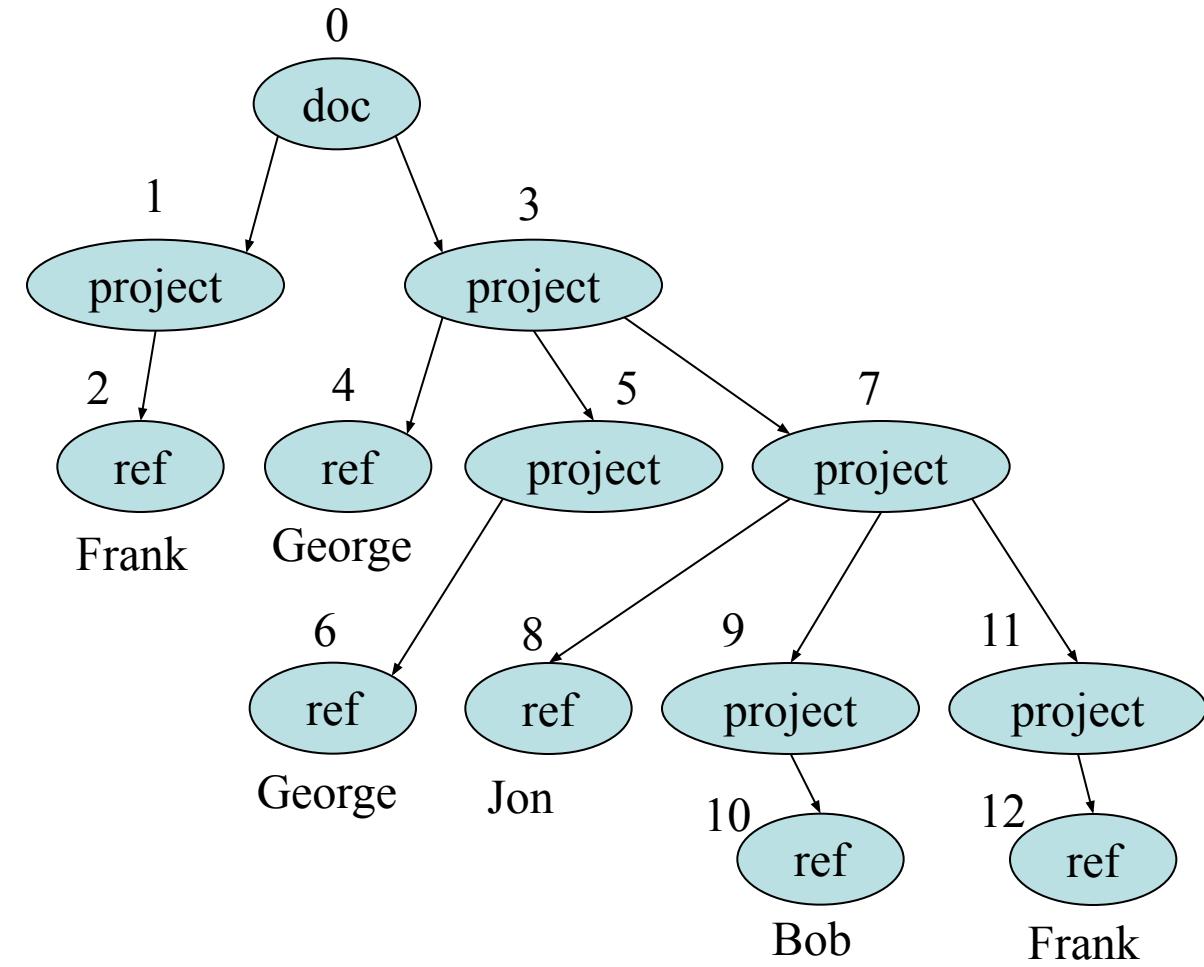
A relational model for XML data (2)



A relational model for XML data (3)

id	name
0	doc
1	project
2	ref
3	project
4	ref
5	project
6	ref
7	project
8	ref
9	project
10	ref
11	project
12	ref

id	child
0	1
0	3
1	2
3	4
3	5
3	7
5	6
7	8
7	9
7	11
9	10
11	12





Main limits of this solution

- XML was created to exchange data between applications and to represent data understandable by human beings
 - The example tables lose these properties. The data model is therefore more complex than the original format
- Some "reasonable" questions can not be written in SQL without using recursion, or they can be inefficient, requiring more time access to the same table, for example:

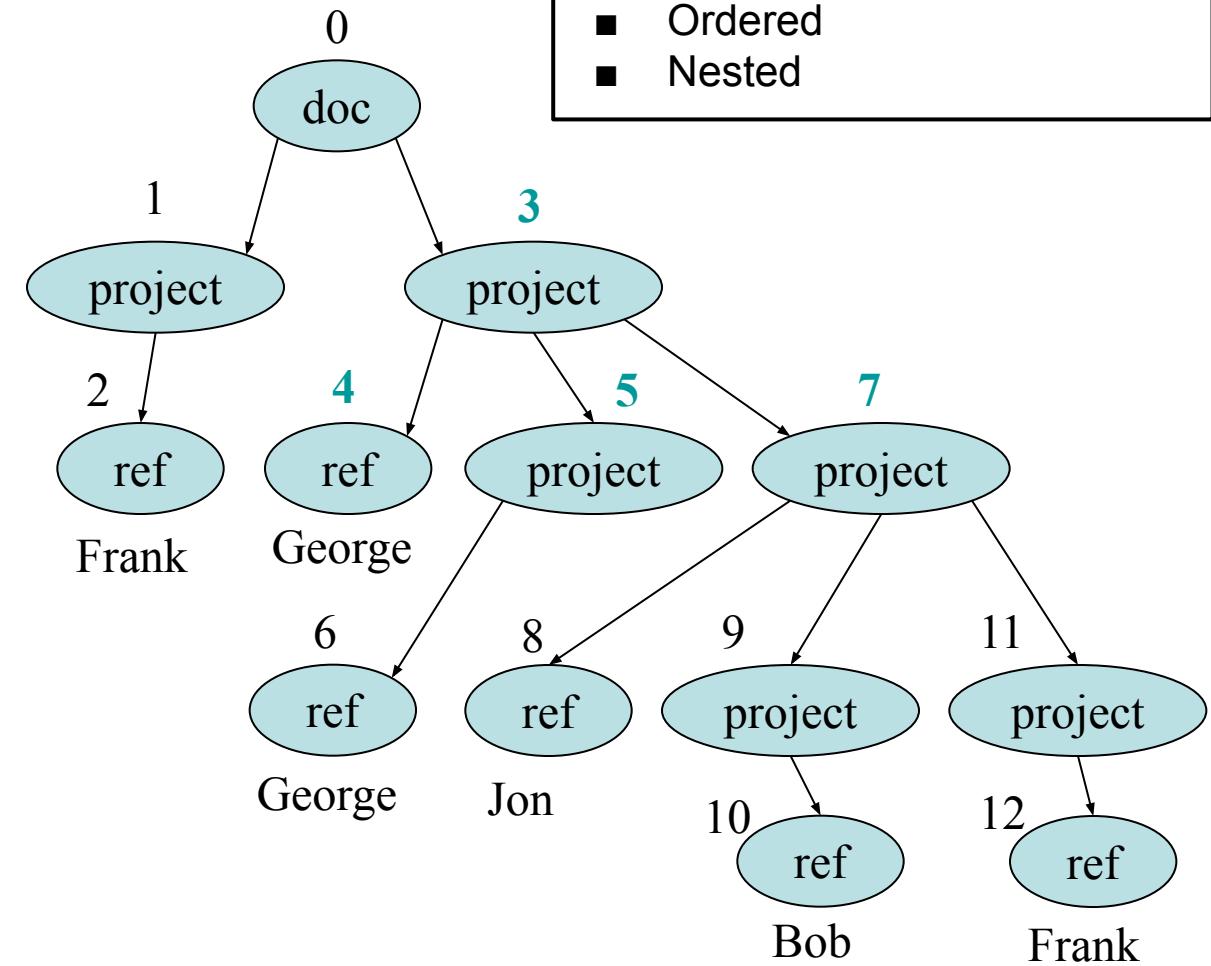
"Find all representatives participating in the second project"

 - It should be mentioned that this road, suitably improved, has been covered by the scientific community with good results
 - So this is a way forward. However, the current trend is to develop specific systems for XML

Find all representatives participating in the second project

id	name
0	doc
1	project
2	ref
3	project
4	ref
5	project
6	ref
7	project
8	ref
9	project
10	ref
11	project
12	ref

id	child
0	1
0	3
1	2
3	4
3	5
3	7
5	6
7	8
7	9
7	11
9	10
11	12

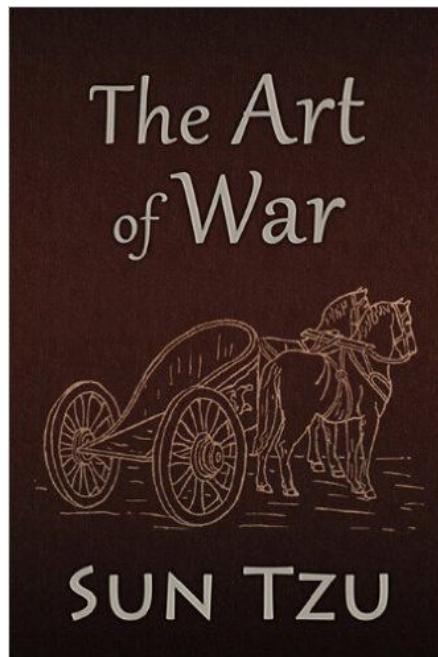




Limits of the relational model (1)

- Let's consider a book and its relational representation

Author	Title	Birth	Text
Sun Tzu	The Art of War	544 b.c.	The supreme art of war is ...

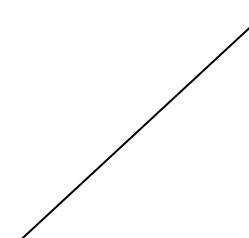
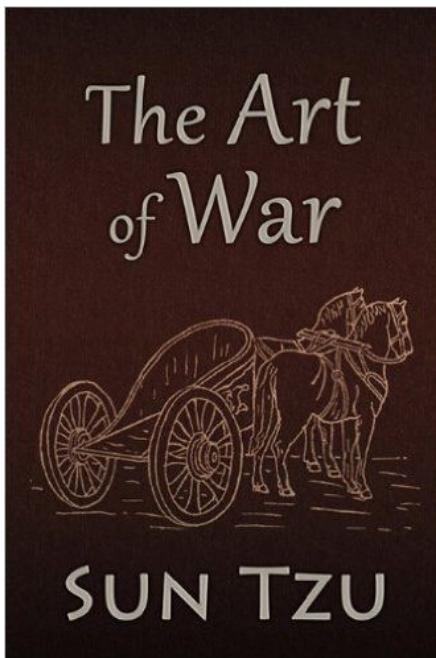




Limits of the relational model (2)

- Let's consider a book and its relational representation

Author	Title	Birth	Text
Sun Tzu	The Art of War	544 b.c.	The supreme art of war is ...



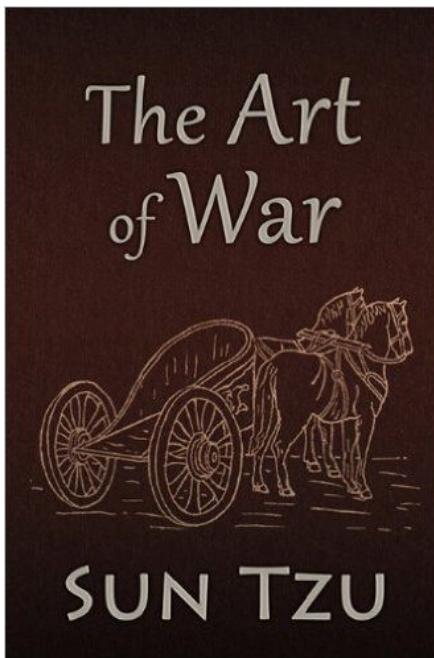
TEXT field contains thousands character without any structure (*CLOB, character large object, data type stores variable-length character data*)



Limits of the relational model (3)

- Let's consider a book and its relational representation

Author	Title	Birth	Text
Sun Tzu	The Art of War	544 b.c.	The supreme art of war is ...



Death?

Description?

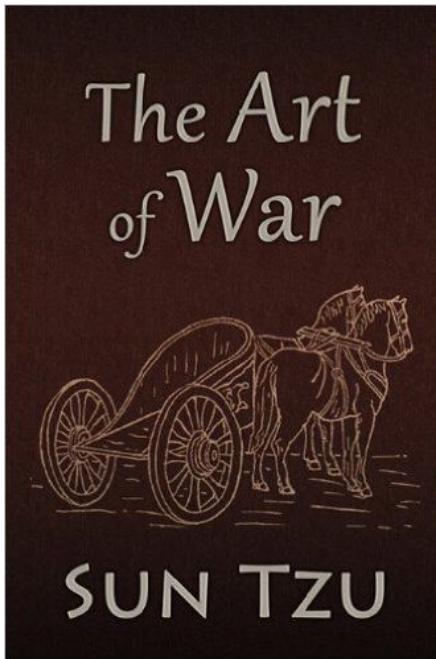
In order to add other info, we need to modify the structure of the table



Limits of the relational model (4)

- Let's consider a book and its relational representation

Sun Tzu	The Art of War	544 b.c.	The supreme art of war is ...
---------	----------------	----------	-------------------------------



Death? Birth? Publication date?

If we want to exchange data with other applications, we need to send over also the structure, together with the data, otherwise data could be unintelligible

Properties of semi-structured data (1)

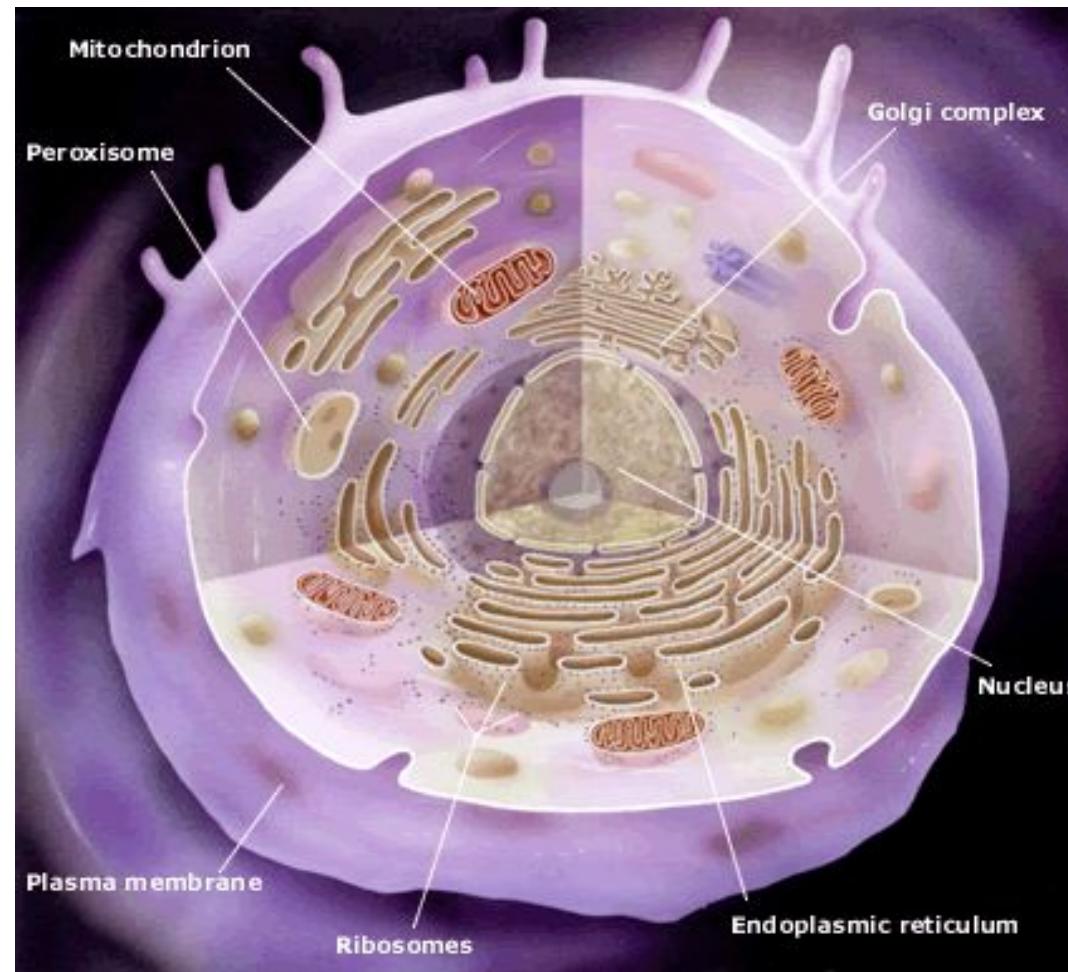
- The structure is irregular or partial

```
<html>
<head><title>The Art of War</title></head>
<body>
<h1>The Art of War</h1>
<h2>Chapter 1</h2>
The supreme art of war is ...<br/>
...

...
</body>
</html>
```

Properties of semi-structured data (2)

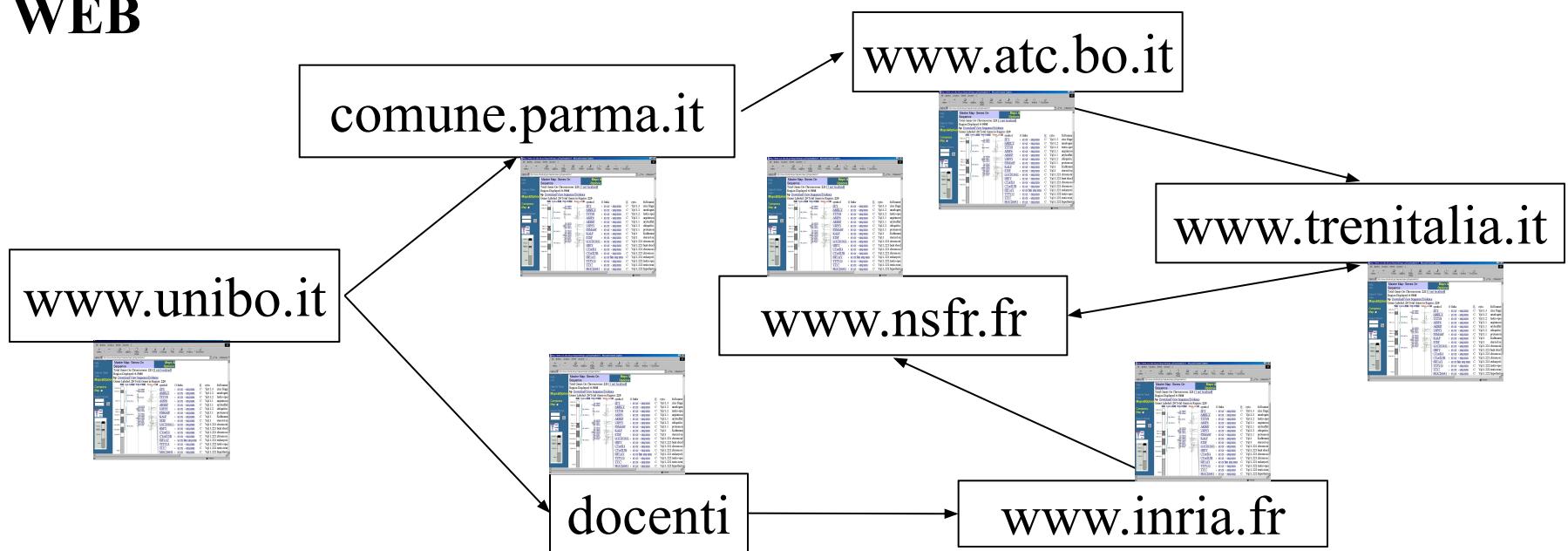
- Schema is built *a posteriori* (data guide)



Properties of semi-structured data (3)

- Schema is very broad

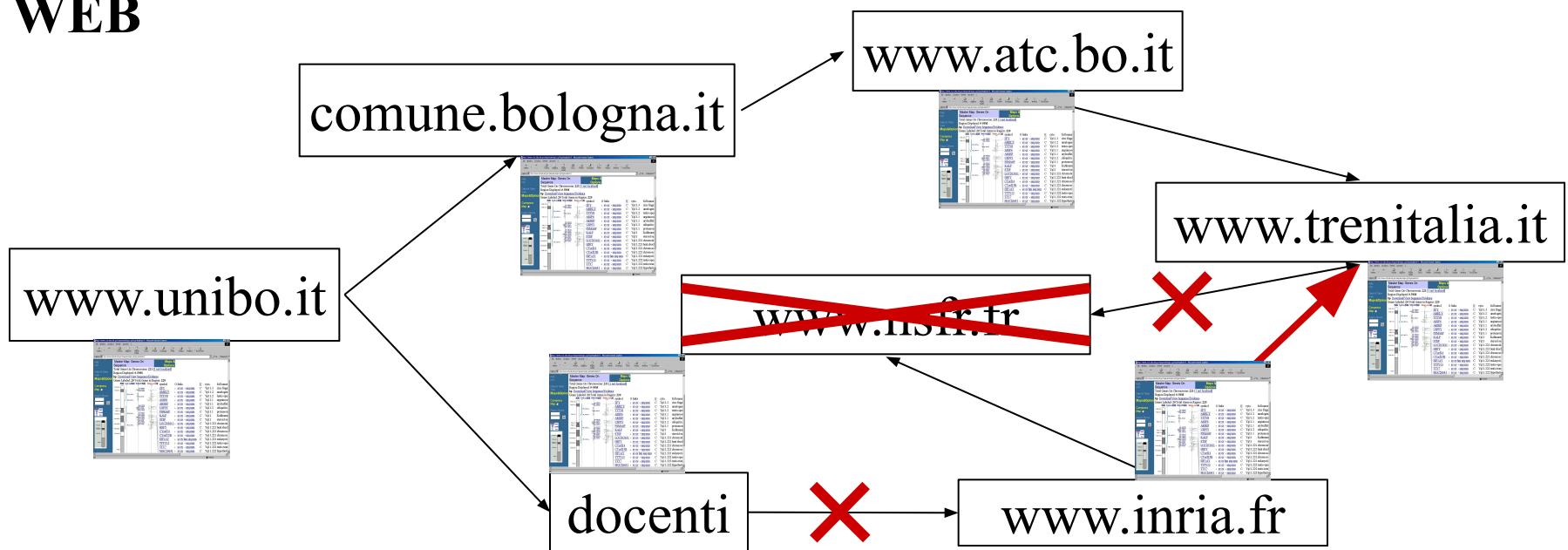
WEB



Properties of semi-structured data (4)

- The schema evolves rapidly (data as well)

WEB





In summary

- The structure is irregular or partial
- The scheme is constructed *a posteriori* (data guides)
- The scheme is very broad
- The scheme is rapidly evolving
- The differences between schema and data are not significant
 - The scheme is changed
 - The scheme is communicated with the data
 - The scheme does not impose constraints unappealable
 - The questions (query) also concern the scheme
- In the case of XML, they are relevant order and the mutual nesting of data
- Being ordered, the data are represented by lists



Unstructured data



Data without schema

- In XML and semi-structured data, the scheme has the characteristics that we saw, but it is still present
 - For example, a query in XPath (XML Path Language) on an XML document typically accesses tags of the items, that is, the schema
- If the **schema is not present**, as in the case of multimedia objects and only narrative text files, the data management change significantly
 - The main discipline that studies this data is called **Information Retrieval**
- No schema data, or which typically do not use the scheme, are of great importance: just think of the Internet and search engines, which are mostly of Web Information Retrieval systems



Simple queries

- Despite more complex query languages they have been proposed, in the majority of cases the queries on unstructured data (mostly on textual data) are very simple, usually composed of lists of keywords, like

“Return documents that contain the word «battle».”

- ... unlike to

```
SELECT name, COUNT(DISTINCT project), SUM(months)
FROM Person NATURAL JOIN Allocation
WHERE name LIKE 'M%' AND Age > 40
GROUP BY name
```



Boolean results and ranking (1)

- In a relational database, queries express precise requirements, and each tuple of the solution satisfies those requirements
- The construction of the response using a relation follows therefore a **Boolean** model: a tuple is *present* or is *not present* in the result
- Given the nature of the questions, given the large amount of possible answers, and given that several documents can respond more or less well to the requirements expressed in the question, **in Information Retrieval a Boolean model is often not usable**



Boolean results and ranking (2)

- Given the gap between data and information, due to the ambiguity of data, often you can not accurately determine whether a result is completely or not at all relevant
- The results are then ranked by degree of **relevance** (see for example Google), and the user admits possible “errors”
- Since it is usually not possible to return **correct** and **complete** results, as it happens in the case of structured data, there are metrics to describe the quality of a result

“Return the documents that have as an argument the war”

Know thy self, know thy enemy. A thousand battles, a thousand victories.

Sun Tzu

?



Relational data and unstructured data: comparison

Relational	Unstructured
Clear distinction between schema and data	No schema
Query language	Search language
Boolean model (correctness, completeness)	Ranking based model
Partial updates and queries	Total updates



In summary

- From the examples shown above emerges that the properties of data, queries and results are significantly different from those found in relational systems
- In addition to sorting by relevance, the result of a query on unstructured data typically does not provide for the manipulation of data, but only the selection of some of them
- Even in this case, as and more than for the semi-structured data, there is therefore the need to use different models and systems
- Note that well known DBMSs have already integrated some capabilities derived from Information Retrieval, such as indexing of columns that contain only text (CLOB, character large object) or columns for multimedia data (BLOB, binary large object)



Big Data and Data Mining

Introduction to XML

Flavio Bertini

flavio.bertini@unipr.it



The origins of XML

- XML (eXtensible Markup Language) derives from SGML (Standard Generalized Markup Language)
 - Both with XML and SGML it is possible to define markup languages specific to several domains, such as finance or math
 - For instance, HTML is one of the languages derived from SGML
 - With respect to SGML, XML is easier to use, and it's designed to specify markup languages to be used on the Internet
 - Maybe it is a little hard to understand, but XML does not DO anything
- Initially XML was born as a format for data exchange and XML is often used to separate data from presentation
- We will look at XML for its data storage and data lookup capabilities
- Consequently, we will not go through all the in-depth details but only the aspects relevant to data management



Example of XML document

```
<?xml version="1.0" encoding="UTF-8"?>
<?tex doctype[report] ?>
<doc isbn="2-266-04744-2">
    <!-- editor is missing! -->
    <author>T. Harris</author>
    <title xml:lang="en">The silence of the lambs</title>
    <title xml:lang="fr">Le silence des agneaux</title>
    <comment>
        A book full of <i>suspance</i>.
    </comment>
    <price currency="euro">7</price>
</doc>
```



Markup and *character data*

```
<?xml version="1.0" encoding="UTF-8"?>
<?tex doctype[report] ?>
<doc isbn="2-266-04744-2">
    <!-- editor is missing! -->
    <author>T. Harris</author>
    <title xml:lang="en">The silence of the lambs</title>
    <title xml:lang="fr">Le silence des agneaux</title>
    <comment>
        A book full of <i>suspance</i>.
    </comment>
    <price currency="euro">7</price>
</doc>
```



Prolog (1/2)

```
<?xml version="1.0" encoding="UTF-8"?>           PROLOG
<?tex doctype[report] ?>
<doc isbn="2-266-04744-2">
    <!-- editor is missing! -->
    <author>T. Harris</author>
    <title xml:lang="en">The silence of the lambs</title>
    <title xml:lang="fr">Le silence des agneaux</title>
    <comment>
        A book full of <i>suspance</i>.
    </comment>
    <price currency="euro">7</price>
</doc>
```



Prolog (2/2)

- The prolog contains information useful for the interpretation of the document
- In particular, it can contain:
 - A declaration that the document is in XML format (**optional**)
 - A *grammar* (DOCTYPE) that allows to validate the content of the document (**optional**)
 - Comments and information for software applications that will use the document (Processing Instructions, or PI) (**zero or more**)



Body of the document (1/2)

```
<?xml version="1.0" encoding="UTF-8"?>
<?tex doctype[report] ?>
<doc isbn="2-266-04744-2">
    <!-- editor is missing! -->
    DOCUMENT
    BODY
    <author>T. Harris</author>
    <title xml:lang="en">The silence of the lambs</title>
    <title xml:lang="fr">Le silence des agneaux</title>
    <comment>
        A book full of <i>suspance</i>.
    </comment>
    <price currency="euro">7</price>
</doc>
```



Body of the document (2/2)

- The body of the document is made of one **element**, which itself can contain other nested elements in its content, and also comments
- We will see, through some examples, how to write *well-formed* elements



Elements

- An element can be of two forms:

<name attributes_list> → *extended form*
content...
</name>

<name attributes_list /> → *short form or self-closed*



Well-formed elements and attributes (1/5)

- Each element must be contained between an opening **tag** and a closing **/tag** or with a *short form*

```
<?xml version="1.0"?>
<?tex doctype[report] ?>
<doc isbn="2-266-04744-2">
    <title xml:lang="en">
        The silence of the lambs
    </title>
    <price euro="7"/>
</doc>
```


ERROR: this tag
(BR) is opened
but never closed!

Well-formed elements and attributes (2/5)

- The names of elements and attributes are *case-sensitive*

ERROR

```
<?xml version="1.0"?>
<?tex doctype[report] ?>
<doc isbn="2-266-04744-2">
    <title xml:lang="en">
        The silence of the lambs
    </title>
    <price euro="7"/>
</DOC>
```

Well-formed elements and attributes (3/5)

- Elements must be nested correctly, and there is only one element that contains all the other elements

```
<?xml version="1.0"?>
<?tex doctype[report] ?>
<doc isbn="2-266-04744-2">
    <b><i>Missing book</b></i>
</doc>
```

ERROR



Well-formed elements and attributes (4/5)

- Values of the attributes must be contained between quotes or double quotes

```
<?xml version="1.0"?>
<?tex doctype[report] ?>
<doc isbn="2-266-04744-2">
    <title xml:lang=en>
        The silence of the lambs
    </title>
    <price euro="7"/>
</doc>
```

ERROR



Well-formed elements and attributes (5/5)

- An element cannot have more than one attribute with the same name

WRONG

```
<book author="Doe" author="Blake"/>
```

CORRECT

```
<book>
  <author>Doe</author>
  <author>Verdi</author>
</book>
```



Document Type Declaration

- In the beginning of an XML document there can be a **document type declaration**
- This declaration contains a grammar, named **Document Type Definition**, or **DTD**, with the double purpose of *constrain* and *complete* the documents
- The DTD is made of **markup declarations**, which define what can be and cannot be written in the related XML document
- The DTDs determine which elements can be included in the document, how they can be used, what are the default values of the attributes of the elements, and other constraints



DTD example (1)

XML

```
<?xml version="1.0"
encoding="UTF-8"?>
<!DOCTYPE note SYSTEM
"Note.dtd">
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this
        weekend!</body>
</note>
```

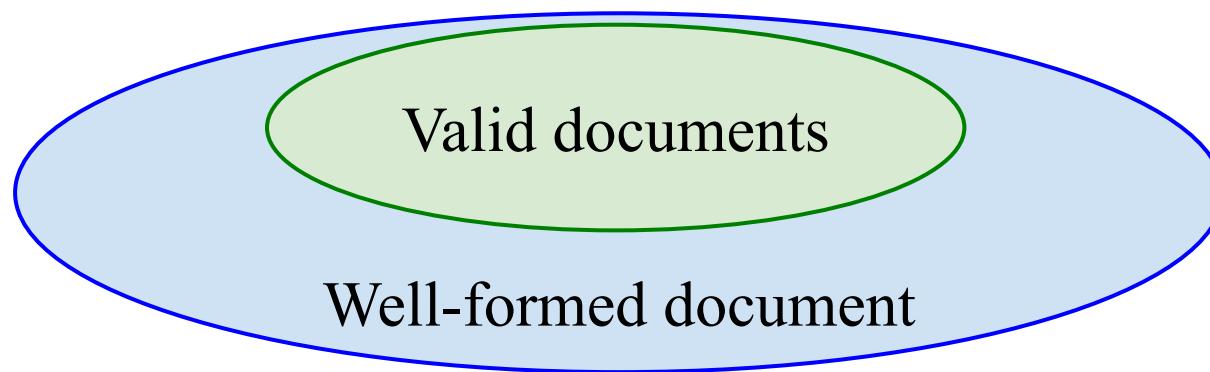
DOCTYPE

```
<!DOCTYPE note
[
  <!ELEMENT note (to,from,heading,body)>
  <!ELEMENT to (#PCDATA)>
  <!ELEMENT from (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body (#PCDATA)>
]>
```



Valid and well-formed documents

- An XML document is valid if:
 - It contains a DTD
 - It complies to it



The following is a well-formed document but it's not valid, because it doesn't have a DTD:

```
<greetings>Hello, world!</greetings>
```

Examples of Document Type Declaration 1/2

- We define a type of document named *greetings* that contains an element <greetings>, which do not contain any other element. Then, the attributes of <greetings> are declared, in this example only the *id* attribute

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE greetings [
    <!ELEMENT greetings (#PCDATA)>
    <!ATTLIST greetings
        id          ID      #REQUIRED>
]>
<greetings id="0001">Hello,
world!</greetings>
```

D
T
D



Examples of Document Type Declaration 2/2

- The same constraints can be specified inside an external file (hello.dtd) and declared in the following way

```
<?xml version="1.0"?>
<!DOCTYPE greetings SYSTEM "hello.dtd">
<greetings>Hello, world!</greetings>
```



DTD example (2)

XML

```
<?xml version="1.0"
encoding="UTF-8"?>

<!DOCTYPE note [
  !ENTITY nbsp "&#xA0;">
  !ENTITY writer "Writer: Donald Duck.">
  !ENTITY copyright "Copyright: W3Schools.">
]>

<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this
weekend!</body>
  <footer>&writer;&nbsp;&copyright;</footer>
</note>
```

OUTPUT

```
<note>

  <to>Tove</to>

  <from>Jani</from>

  <heading>Reminder</heading>

  <body>Don't forget me this
weekend!</body>

  <footer>Writer: Donald Duck.
Copyright: W3Schools.</footer>

</note>
```



DTD and XML Schema

- A DTD constrains an XML document
- It is however possible to specify constraints more complex than the one allowed by DTD
 - For instance: imported keys, uniqueness constraints, or the domains of elements and attributes (as in SQL)
- The **XML Schema** allows to specify this kind of constraints, and it is therefore an alternative to the DTD
- Moreover, the constraints of XML Schema are expressed in XML
 - XML Schemas are extensible to additions
 - XML Schemas support data types
 - XML Schemas support namespaces



XML Schema: an example

- The line `xmlns:xsi="..."` tells the parser that this document should be validated against a schema
- The line `xsi:noNamespaceSchemaLocation="shiporder.xsd"` specifies WHERE the schema resides (here it is in the same folder as "shiporder.xml")

```
<?xml version="1.0" encoding="UTF-8"?>
<shiporder orderid="889923"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="shiporder.xsd">
  <orderperson>John Smith</orderperson>
  <shipto>
    <name>Ola Nordmann</name>
    <address>Langgt 23</address>
    <city>4000 Stavanger</city>
    <country>Norway</country>
  ...

```



XML Schema

XML

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

<note
  xmlns="https://www.w3schools.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="https://www.w3schools.com/xml
  note.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

XML SCHEMA

```
<?xml version="1.0"?>
<xss:schema
  xmlns:xss="http://www.w3.org/2001/XMLSchema"
  targetNamespace="https://www.w3schools.com"
  xmlns="https://www.w3schools.com"
  elementFormDefault="qualified">

  <xss:element name="note">
    <xss:complexType>
      <xss:sequence>
        <xss:element name="to" type="xs:string"/>
        <xss:element name="from" type="xs:string"/>
        <xss:element name="heading" type="xs:string"/>
        <xss:element name="body" type="xs:string"/>
      </xss:sequence>
    </xss:complexType>
  </xss:element>

</xss:schema>
```



DTD and XML Schema - comments

- XML does not require a DTD or XML Schema. When you are experimenting with XML, or when you are working with small XML files, creating DTDs may be a waste of time
- However, with DTD and XML Schema, XML files can carry a description of its own format
 - Independent groups of people can agree on a standard for interchanging data
 - Data receive from the outside world can be verified
- One of the greatest strengths of XML Schemas is the support for data types:
 - It is easier to describe document content
 - It is easier to define restrictions on data
 - It is easier to validate the correctness of data
 - It is easier to convert data between different data types



JSON vs XML

JSON	XML
It is JavaScript Object Notation, based on JavaScript language	It is Extensible markup language, derived from SGML
It is a way of representing objects	It is a markup language and uses tag structure to represent data items
It does not provide any support for namespaces*	It supports namespaces
It supports array	It doesn't support array
Its files are very easy to read as compared to XML	Its documents are comparatively difficult to read and interpret
It doesn't use end tag	It has start and end tags
It doesn't support comments	It supports comments
It supports only UTF-8 encoding	It supports various encoding

*Valid Data Types In JSON: *string, number, (JSON) object, array, boolean, and null*



JSON vs XML: an example

JSON

```
{ "employees" : [  
    { "name" : "Shya" ,  
      "email" : "shy@mai.com" } ,  
    { "name" : "Bob" ,  
      "email" : "bob@mail.com" } ,  
    { "name" : "Jai" ,  
      "email" : "jai@gmail.com" }  
]
```

XML

```
<employees>  
  <employee>  
    <name>Shya</name>  
    <email>shy@mai.com</email>  
  </employee>  
  <employee>  
    <name>Bob</name>  
    <email>bob@mail.com</email>  
  </employee>  
  <employee>  
    <name>Jai</name>  
    <email>jai@gmail.com</email>  
  </employee>  
</employees>
```



Proper and improper usage of XML

- XML allows a great degree of freedom to the designers of XML documents
- The designer can decide the tags and attributes to use, and where to put the data
- When XML is used to store data, particular attention must be paid to the correct usage of the elements of the data model: elements, attributes, contents, hierarchies
- We will now provide some guidelines



1 – Data in elements content

- Data must be stored in the content of the elements, for instance:

```
<book>  
    <title>The Great Gatsby</title>  
</book>
```

Must be preferred to

```
<book title="The Great Gatsby" />
```

AVOID THIS



2 – Metadata in attributes or names of elements

```
<book>
  <property>
    <name>title</name>
    <value>The Great Gatsby</value>
  </property>
</book>
```

TO AVOID



- In this example, title is a metadata, therefore it must not appear as content



3 – (In)Correct usage of hierarchies

```
<db>
```

```
  <title>The Great Gatsby</title>      TO AVOID  
  <author>F Scott Fitzgerald</author>  
  <title>For Whom the Bell Tolls</title>  
  <author>Ernest Hemingway</author>
```

```
</db>
```

- The two titles and the two authors are only separated by the order of elements, while they compose well distinct objects (they refer to different books)
- **<title>** and **<author>** must be child of one element **<book>**, and not child of the element **<db>**



4 – Correct use of hierarchies

```
<db>
  <book>
    <title>The Great Gatsby</title>
    <author>F Scott Fitzgerald</author>
  </book>
  <book>
    <title>For Whom the Bell Tolls</title>
    <author>Ernest Hemingway</author>
  </book>
</db>
```

Example:

<https://www.w3schools.com/xml/Books.xml>

https://www.w3schools.com/xml/cd_catalog.xml



XML Namespaces - Name Conflicts

- In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications

```
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

XML carries HTML table information

```
<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

XML carries information about a generic table

- If these XML fragments were added together, there would be a name conflict. Both contain a `<table>` element, but the elements have different content and meaning



Solving the Name Conflict Using a Prefix

- Name conflicts in XML can easily be avoided using a name prefix

```
<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>
```

```
<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```



XML Namespaces - The xmlns Attribute

- When using prefixes in XML, a namespace for the prefix must be defined using a `xmlns` attribute in the start tag of an element

```
<root>

<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table xmlns:f="https://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>

</root>
```



Big Data and Data Mining

Relational Data and XML

Flavio Bertini

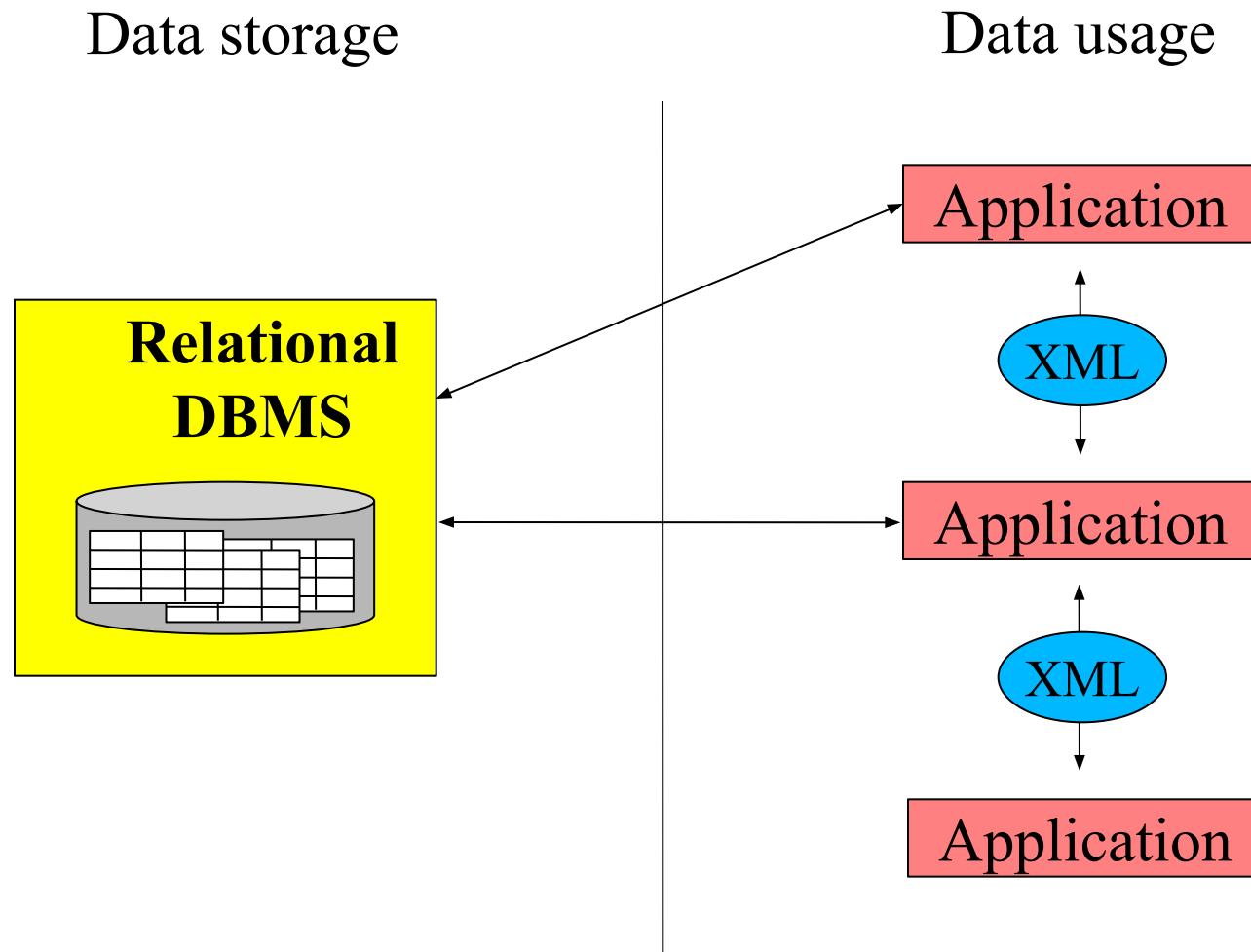
flavio.bertini@unipr.it



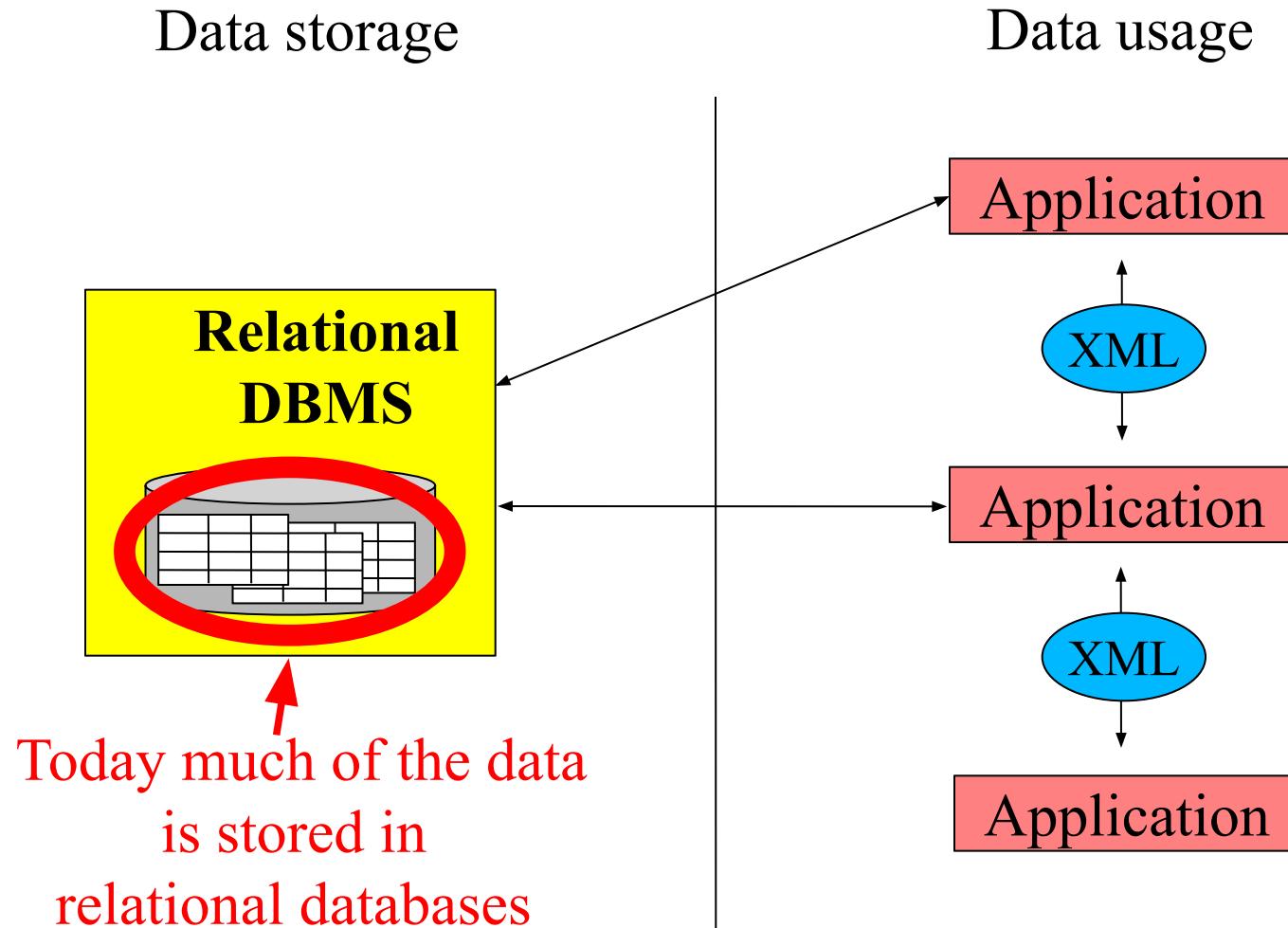
Relational Data and XML - Outline

- Combining relational data and XML
 - From SQL to XML
 - From XML to SQL
- The SQL/XML language
 - XMLELEMENT
 - XMLFOREST
 - XMLCONCAT
 - XMLAGG
 - XMLGEN

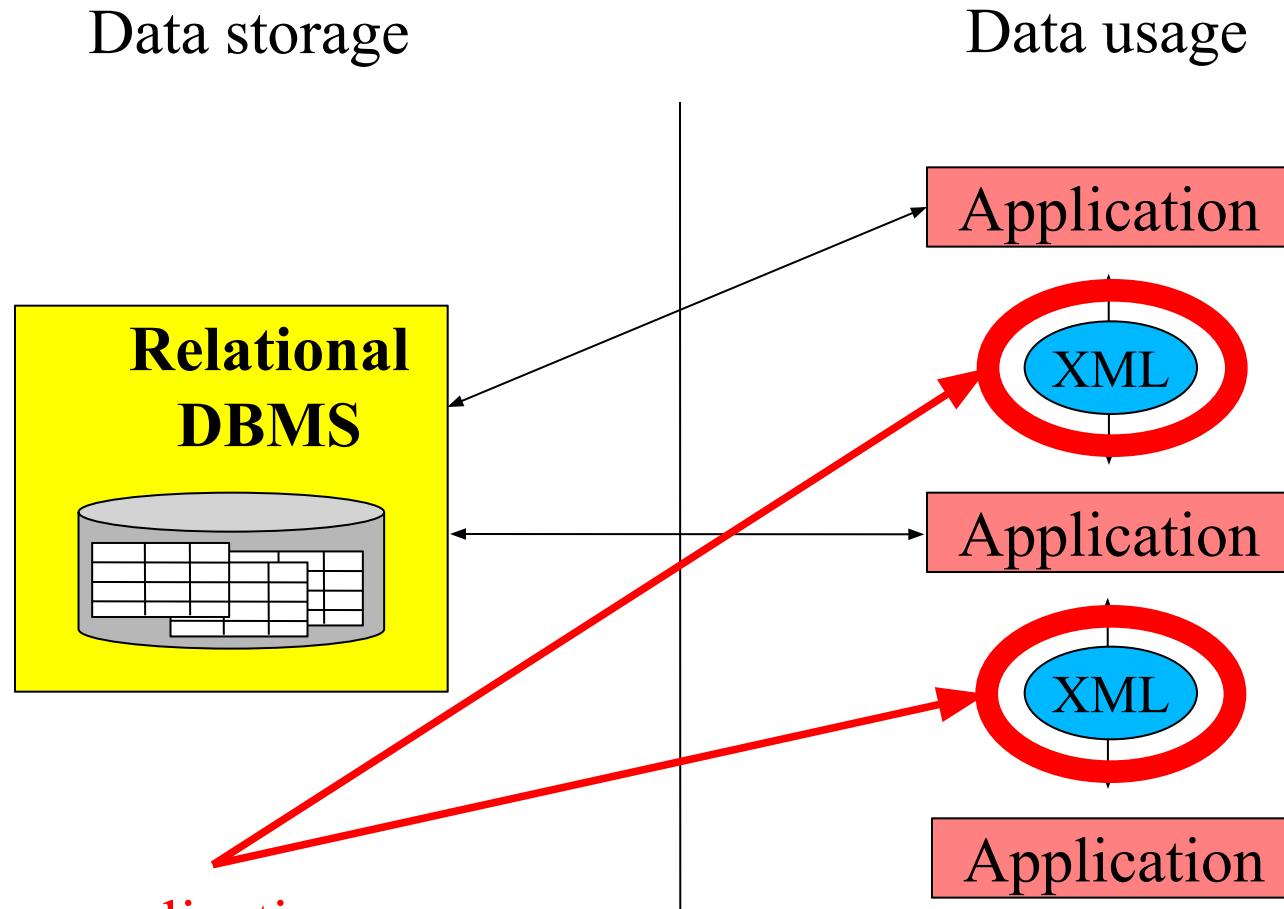
Introduction 1/4



Introduction 2/4



Introduction 3/4



Many applications use
XML as an exchange and
manipulation format

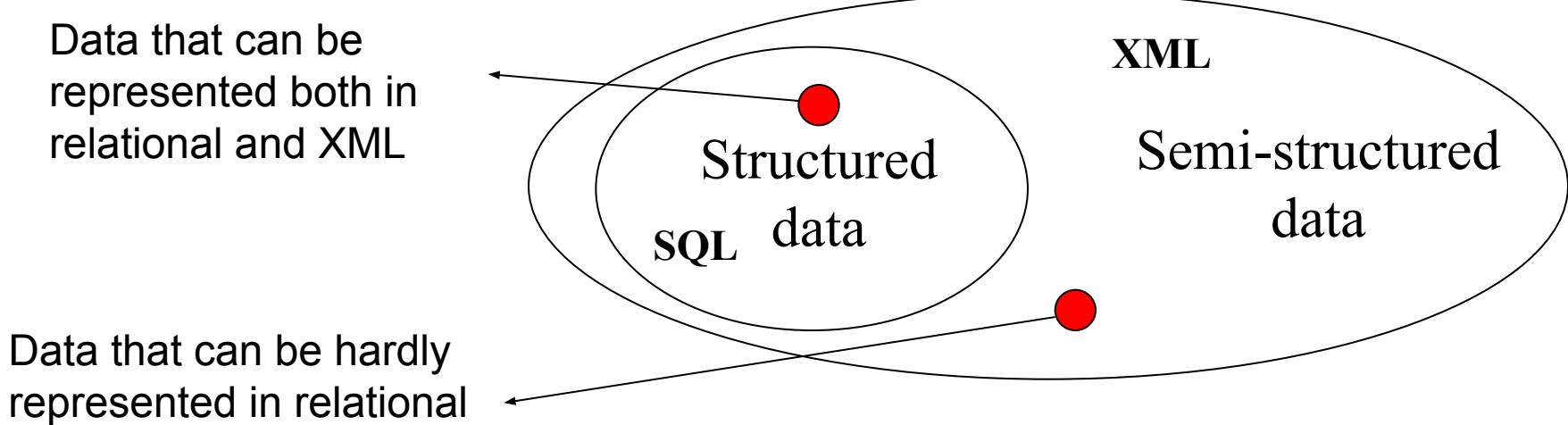


Introduction 4/4

- XML is used primarily for two types of data management activities:
 - Representation of **semi-structured data**
 - **Exchange of data** between applications
- When the data exchanged between different applications are locally stored in relational databases, a “bridge” is needed between these two formats
- SQL/XML, which is a standard extension to SQL, provides a common language to convert relational data to XML

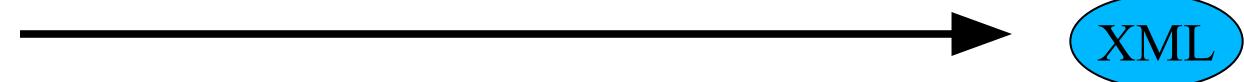
Features: SQL → XML and XML → SQL

- To use a combination of relational and XML data requires two main functions:
 - **Extracting XML** from one or more relational tables
 - **XML storage** in one (or more) relational tables
- The first feature is conceptually simpler, since XML was created in order to represent both semi-structured and structured data
- The second is more complex in general, for the same reason



Extracting XML from a table

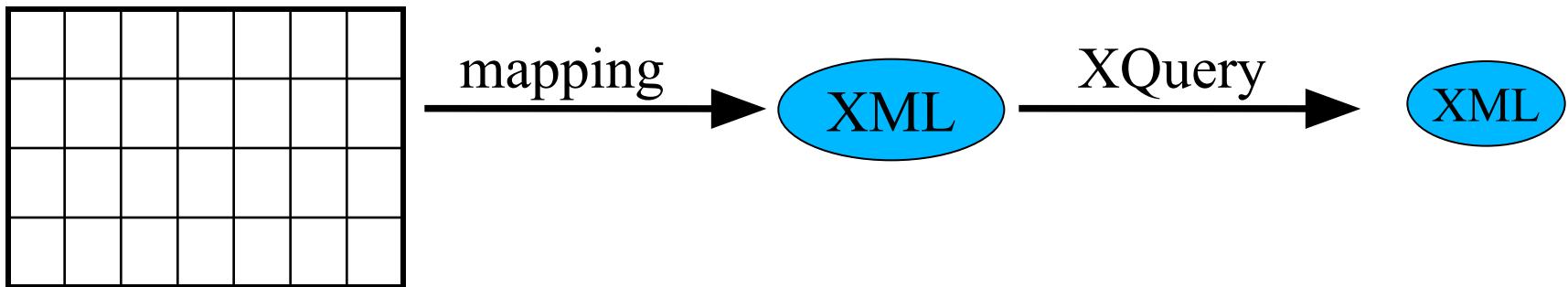
- Extracting XML data from a table is simple, because any type of data that can be represented in a table can be also represented in XML
 - Let's see two alternative ways to implement the extraction





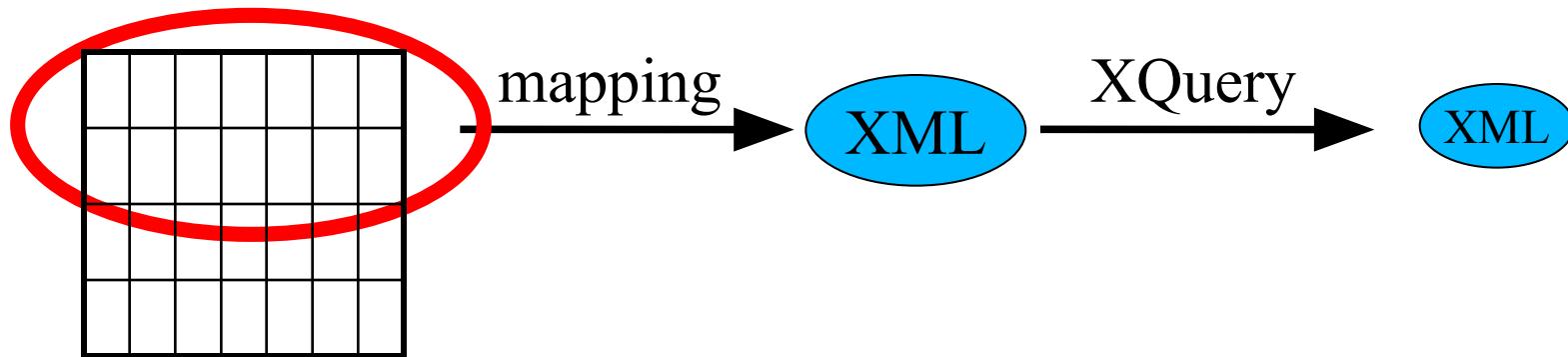
1: Extracting XML from a table (XML + XQuery)

- Represent the table in XML (mapping)
- Extract data using XML technologies (XQuery)





1: Extracting XML from a table (XML + XQuery)

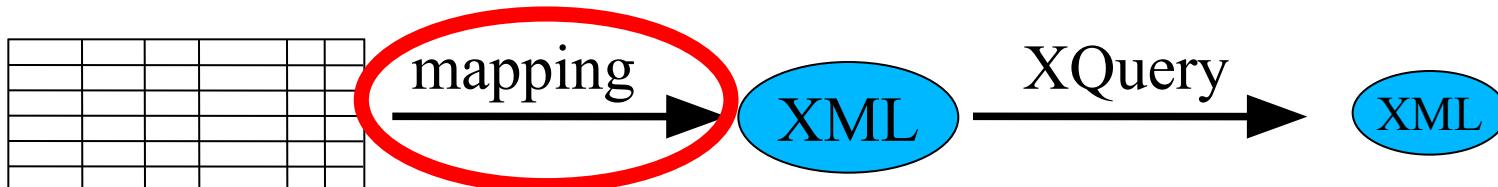


EMPLOYEES

ID	NAME	SURNAME	SALARY
emp0001	John	Doe	20000
emp0002	Jack	Black	18000

We start from a relational table

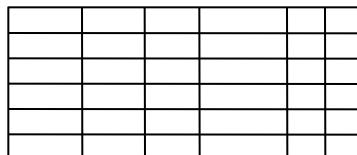
1: Extracting XML from a table (XML + XQuery)



- The mapping proceeds in the following way:
 1. The name of the table becomes the name of the document
 2. Each row is included in an element `<row>`
 3. Each value (column) is included in an element with the name of the (SQL) attribute
 4. Null values are represented using the attribute `xsi:nil="true"`



1: Extracting XML from a table (XML + XQuery)



mapping



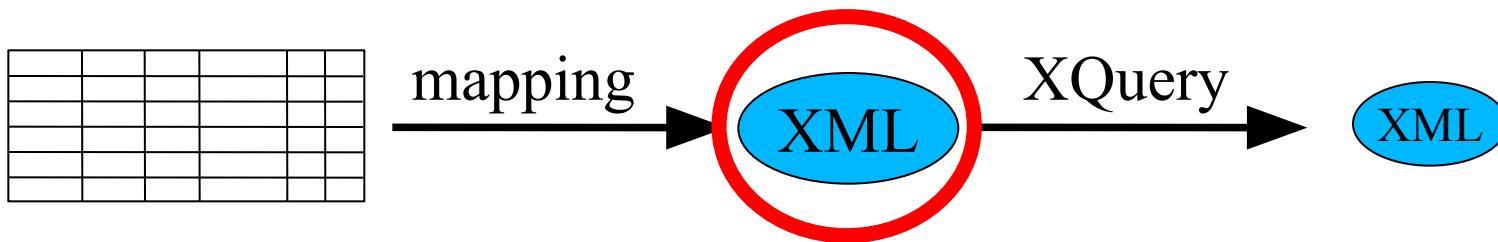
XQuery



This mapping is
defined by SQL/XML

```
<EMPLOYEES>
  <row>
    <ID>emp0001</ID>
    <NAME>John</NAME>
    <SURNAME>Doe</SURNAME>
    <SALARY>20000</SALARY>
  </row>
  <row>
    <ID>emp0002</ID>
    <NAME>Jack</NAME>
    <SURNAME>Black</SURNAME>
    <SALARY>18000</SALARY>
  </row>
</EMPLOYEES>
```

1: Extracting XML from a table (XML + XQuery)



- The standard defines also an XML schema with the definitions of each type of data in SQL and each XML element
 - For instance, CHARACTER(6) produces:

```
<xsd:sempleType name="CHAR_6">
  <xsd:restriction base="xsd:string">
    <xsd:length value="6" />
  </xsd:restriction>
</xsd:sempleType>
```

Fragment of XML Schema,
provided as an example (it
is not necessary to
remember its details)



1: Extracting XML from a table (XML + XQuery)

XML schema

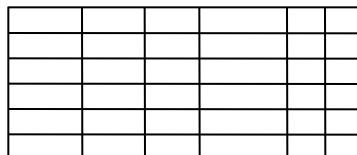
```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Address">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Recipient" type="xs:string" />
        <xs:element name="House" type="xs:string" />
        <xs:element name="Street" type="xs:string" />
        <xs:element name="Town" type="xs:string" />
        <xs:element name="County" type="xs:string"
                     minOccurs="0" />
        <xs:element name="PostCode" type="xs:string" />
        <xs:element name="Country">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="FR" />
              <xs:enumeration value="DE" />
              <xs:enumeration value="ES" />
              <xs:enumeration value="UK" />
              <xs:enumeration value="US" />
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML file

```
<?xml version="1.0" encoding="utf-8"?>
<Address
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="SimpleAddress.xsd">
  <Recipient>Mr. Walter C. Brown</Recipient>
  <House>49</House>
  <Street>Featherstone Street</Street>
  <Town>LONDON</Town>
  <PostCode>EC1Y 8SY</PostCode>
  <Country>UK</Country>
</Address>
```



1: Extracting XML from a table (XML + XQuery)



mapping

XML

XQuery

XML

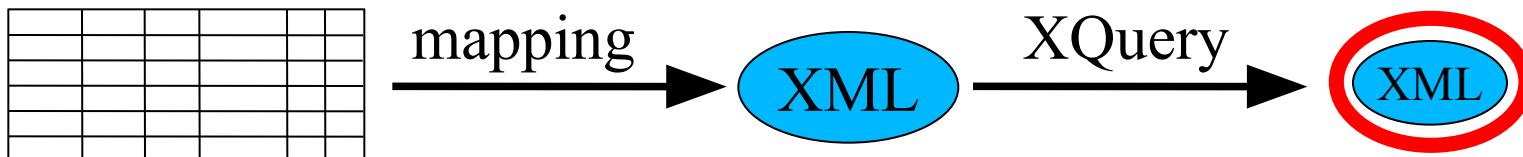
Non-standard XQuery function

```
<highSalaries>
{ for $e in table("EMPLOYEES")/EMPLOYEES/row
  where $e/salary > 18000
  return
    <employee>
      {$e/surname, $e/name}
    </employee>
}
</highSalaries>
```

An XQuery query, provided as
an example (we will not see
XQuery in details in these
slides)



1: Extracting XML from a table (XML + XQuery)



EMPLOYEES

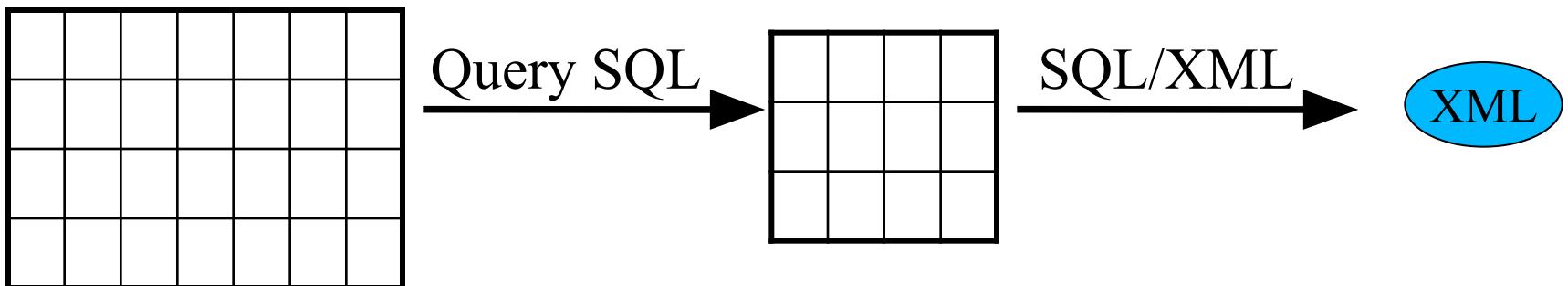
ID	NAME	SURNAME	SALARY
emp0001	John	Doe	20000
emp0002	Jack	Black	18000

- The result is the following:

```
<highSalaries>
  <employee>
    <name>John</name>
    <surname>Doe</surname>
  </employee>
</highSalaries>
```

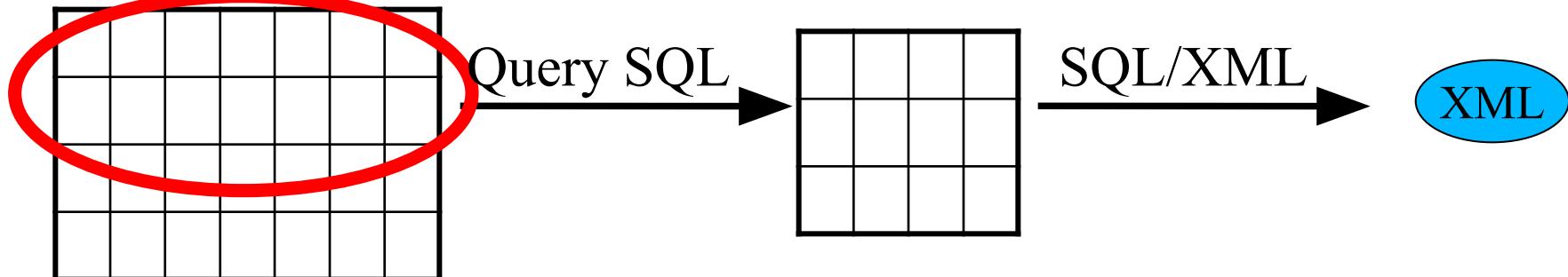
2: Extracting XML from a table (SQL/XML)

- Extract data from the table (SQL)
- Transform this data in XML (SQL/XML)





2: Extracting XML from a table (SQL/XML)



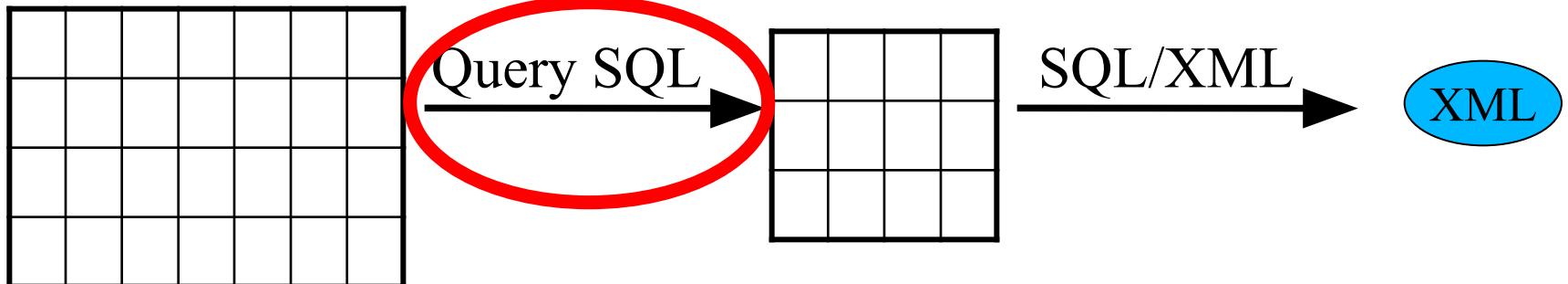
EMPLOYEES

ID	NAME	SURNAME	SALARY
emp0001	John	Doe	20000
emp0002	Jack	Black	18000

We start from the same table as in the previous example



2: Extracting XML from a table (SQL/XML)



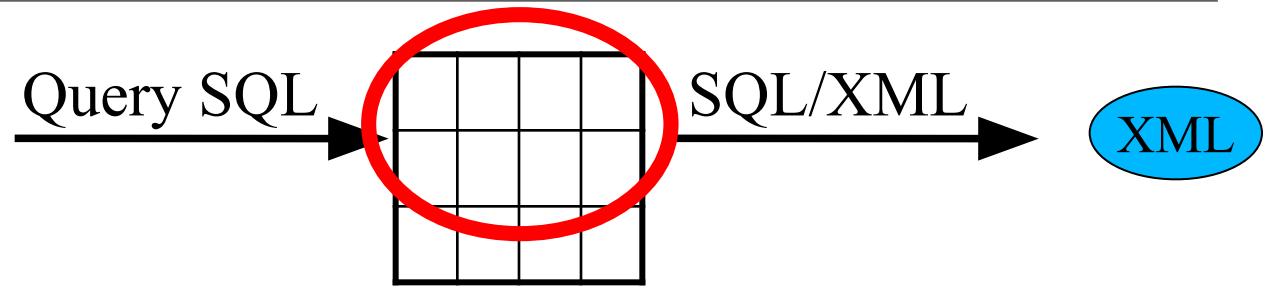
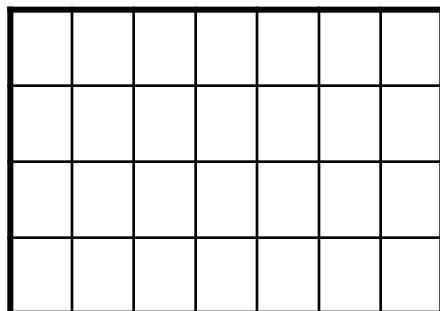
```
SELECT i.name AS employeeName
```

```
FROM EMPLOYEES i WHERE salary > 18000
```

We write an SQL query to get the data of interest



2: Extracting XML from a table (SQL/XML)



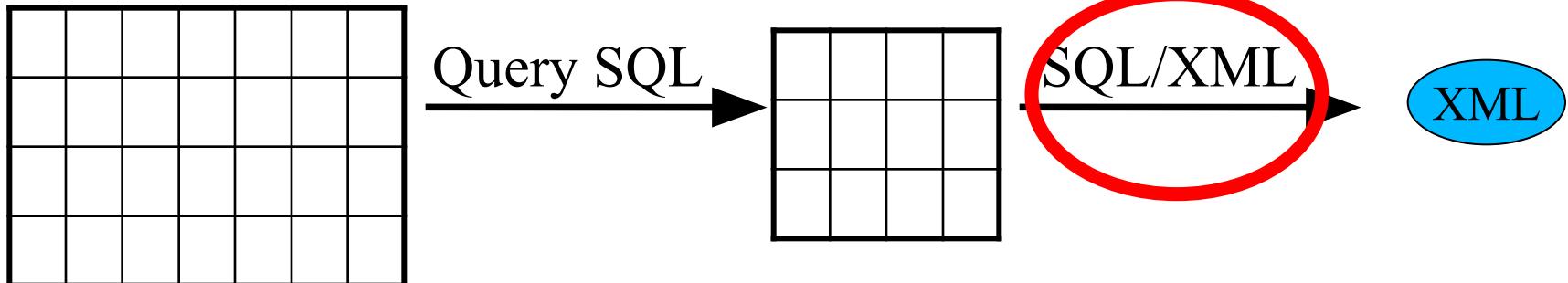
```
SELECT i.name AS employeeName
```

```
FROM EMPLOYEES i WHERE salary > 18000
```

<i>employeeName</i>
John



2: Extracting XML from a table (SQL/XML)

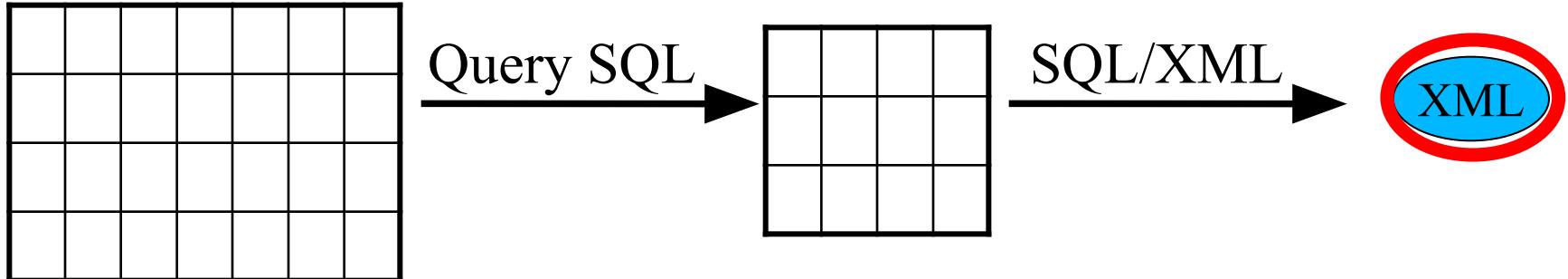


```
SELECT XMLEMENT(  
    NAME "emp",  
    i.name ) AS employeeName  
FROM EMPLOYEES i WHERE salary > 18000
```

We add the XML constructors



2: Extracting XML from a table (SQL/XML)



```
SELECT XMLEMENT(  
    NAME "emp",  
    i.name ) AS employeeName  
FROM EMPLOYEES i WHERE salary > 18000
```

employeeName
<emp>John</emp>

More properly, in this way we are not extracting XML code, but tables containing XML code, which can be then restored and used as XML



Storing XML in Relational DBs

- To provide this functionality, different systems use *ad hoc* techniques
- These can be traced to two main modes:
 - Using Object-Relational columns to store entire XML fragments in a single field
 - *Shredding* ("chopping") of the documents, for which different items are stored in different fields

Inserting XML 1/2

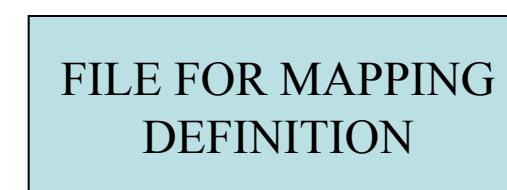
- An XML document can be viewed as a type of SQL data
 - In this case, an entire document is stored in an attribute (column) of a table

Inserting XML 2/2 (Shredding)

- With some limitations, an XML document can be splitted in fragments and stored in pieces

```
<employee>
  <ID>emp0001</ID>
  <SURNAME>Doe</SURNAME>
  <SALARY>20000</SALARY>
</employee>
<employee>
  <ID>emp0002</ID>
  <SURNAME>Black</SURNAME>
  <SALARY>18000</SALARY>
</employee>
```



ID	SURNAME	SALARY
emp0001	Doe	20000
emp0002	Black	18000



The SQL/XML language

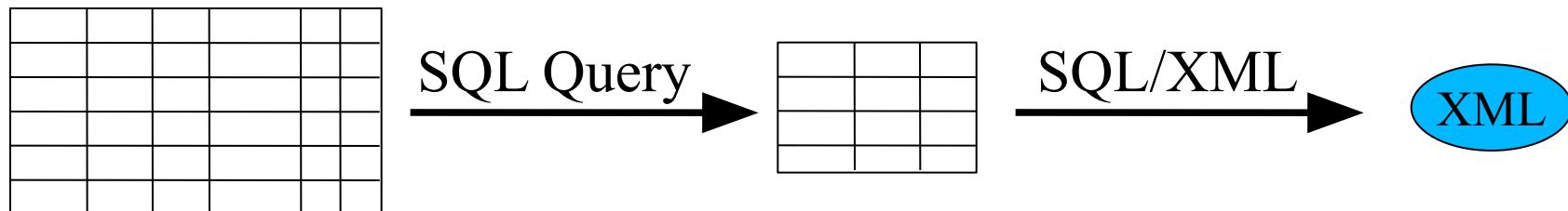


SQL/XML: a SQL extension for XML

- SQL/XML is an extension of SQL
- It is a part of the Structured Query Language (SQL) specification
- It comprises of constructors, routines, functions to support manipulation and storage of XML in a SQL Database

Extracting XML with SQL/XML 1/2

- SQL/XML defines the following operators
 - XMLELEMENT
 - XMLFOREST
 - XMLCONCAT
 - XMLAGG
 - XMLGEN



<https://oracle-base.com/articles/misc/sqlxml-sqlx-generating-xml-content-using-sql>



Extracting XML with SQL/XML 2/2

- An SQL/XML query has the following structure:

SELECT Attr1, Attr2, ..., *Constructor XML*

FROM...

WHERE...

- To compute its result

- FIRST: only **SQL** is considered, and a table is computed as it was a **SELECT ***
- THEN: the result is built, selecting the requested attributes (**Attr1, Attr2...**) and building the **XML code FOR EACH TUPLE**



Table used in the examples

EMPLOYEE

<i>id</i>	<i>name</i>	<i>surname</i>	<i>department</i>	<i>dismissed</i>	<i>salary</i>
emp0001	John	Doe	Sales	null	20000
emp0002	Jack	Black	Sales	null	18000
emp0003	Donald	Mason	Accounting	null	15000
emp0004	Samuel	Wood	Accounting	01/01/03	15000



XMLEMENT

- XMLEMENT allows to create an XML element
- It takes the following arguments:
 - The name of the element
 - An optional list of attributes
 - The content of the element



XMLELEMENT – example

```
SELECT i.id, XMLELEMENT(  
    NAME "emp",  
    i.name ) AS result  
FROM EMPLOYEES i
```



XMLELEMENT – example

```
SELECT i.id, XMLELEMENT(  
    NAME "emp",  
    i.name) AS result  
FROM EMPLOYEES i
```

The code snippet illustrates the use of the XMLELEMENT function in a SQL query. The XMLELEMENT function creates an XML element named 'emp' with the content 'i.name'. The 'NAME' parameter is highlighted with a red box and an arrow pointing to the text 'name of the element'. The 'content' parameter is also highlighted with a red box and an arrow pointing to the text 'content'.



XMLELEMENT – result

```
SELECT i.id, XMLELEMENT(  
    NAME "emp",  
    i.name) AS result  
FROM EMPLOYEES i
```

The code uses the XMLELEMENT function to generate XML elements. The first argument is the element name, and the second argument is the content. Red boxes highlight these two parts, with arrows pointing to their descriptions: 'name of the element' and 'content'.

id	result
emp0001	<emp>John</emp>
emp0002	<emp>Jack</emp>
emp0003	<emp>Donald</emp>
emp0003	<emp>Samuel</emp>



XMLELEMENT

- The content of an element can be built by concatenating more values of SQL
- The concatenation operator is ||



XMLEMENT

```
SELECT i.id, XMLEMENT(  
    NAME "emp",  
    i.name || ' ' || i.surname) AS result  
FROM EMPLOYEES i
```

id	result
emp0001	<emp>John Doe</emp>
emp0002	<emp>Jack Black</emp>
emp0003	<emp>Donald Mason</emp>
emp0003	<emp>Samuel Wood</emp>



XMLATTRIBUTES

- In order to declare a list of attributes, we use the operator **XMLATTRIBUTES**
- Each parameter of **XMLATTRIBUTES** is inserted in an attribute which, if not explicitly declared, takes the name of the relational column from which it has been selected



XMLELEMENT and XMLATTRIBUTES

```
SELECT i.id, XMLELEMENT(  
    NAME "emp",  
    XMLATTRIBUTES(i.salary as "sal"),  
    i.name || ' ' || i.surname ) AS result  
FROM EMPLOYEES i
```

id	result
emp0001	<emp sal="20000">John Doe</emp>
emp0002	<emp sal="18000">Jack Black</emp>
emp0003	<emp sal="15000">Donald Mason</emp>
emp0003	<emp sal="15000">Samuel Wood</emp>



XMLELEMENT

- In the content of an element it is possible to specify several objects, both elements and strings of characters
- Let's try to create an element that contains other two elements and some text



XMLELEMENT – example

As we said before,
the first parameter
specifies the name of
the resulting element

```
SELECT i.id,  
       XMLELEMENT(  
           → NAME "emp",  
           XMLELEMENT(  
               NAME "co",  
               i.surname),  
               ' of department ',  
               XMLELEMENT(  
                   NAME "dep",  
                   i.department)  
           ) AS result  
      FROM EMPLOYEES AS i
```



XMLELEMENT – example

Here we list the contents. We specify an element <co>, then a text content “of department”, then a <dep> element

```
SELECT i.id,  
      XMLELEMENT(  
        NAME "emp",  
        XMLELEMENT(  
          NAME "co",  
          i.surname),  
        ' of department ').  
      XMLELEMENT(  
        NAME "dep",  
        i.department)  
    ) AS result  
  FROM EMPLOYEES AS i
```



XMLELEMENT – result

id	result
emp0001	<emp><co>Doe</co> of department <dep>Sales</dep></emp>
emp0002	<emp><co>Black</co> of department <dep>Sales</dep></emp>
emp0003	<emp><co>Mason</co> of department <dep>Accounting</dep></emp>
emp0003	<emp><co>Wood</co> of department <dep>Accounting</dep></emp>



XMLFOREST

- XMLFOREST is a quick way to produce a list of simple elements
- The behaviour of its parameters is the same of XMLATTRIBUTES



XMLFOREST – example

```
SELECT i.id,  
       XMLEMENT(  
           NAME "emp",  
           XMLFOREST(  
               i.name, ←  
               i.surname AS "co",  
               i.department AS "dip")  
           ) AS result  
      FROM EMPLOYEES i
```

If this explicit name is missing, the name of the corresponding column is used instead



XMLFOREST – result

id	result
emp0001	<emp><name>John</name> <co>Doe</co><dip>Sales</dip></emp>
emp0002	<emp><name>Jack</name> <co>Black</co><dip>Sales</dip></emp>
emp0003	<emp><name>Donald</name><co>Mason </co><dip>Accounting</dip></emp>
emp0003	<emp><name>Samuel</name><co>Wood </co><dip>Accounting</dip></emp>



XMLCONCAT

- XMLCONCAT concatenates its arguments, producing a forest of elements
- XMLCONCAT can concatenate also elements built using XMLEMENT



XMLCONCAT – example

```
SELECT i.id,  
       XMLCONCAT(  
           XMLELEMENT(  
               NAME "co",  
               i.surname),  
           XMLELEMENT(  
               NAME "dep",  
               i.department)  
       ) AS result  
FROM EMPLOYEES i
```

XMLCONCAT – result

id	result
emp0001	<co>Doe</co> <dep>Sales</dep>
emp0002	<co>Black</co> <dep>Sales</dep>
emp0003	<co>Mason</co> <dep>Accounting</dep>
emp0003	<co>Wood</co> <dep>Accounting</dep>



XMLAGG

- If we need to group more tuples depending on one or more attributes, SQL use the GROUP BY construct
- XMLAGG allows to restore the tuples aggregated using GROUP BY



XMLAGG

<i>id</i>	<i>nome</i>	<i>surname</i>	<i>department</i>	<i>dismissed</i>	<i>salary</i>
emp0001	John	Doe	Sales	null	20000
emp0002	Jack	Black	Sales	null	18000
emp0003	Donald	Mason	Accounting	null	15000
emp0004	Samuel	Wood	Accounting	01/01/03	15000

```
<dip nome="Sales">
    <emp></emp>
    <emp></emp>
</dip>
```

```
<dip nome="Accounting">
    <emp></emp>
    <emp></emp>
</dip>
```



XMLAGG

```
SELECT XMLELEMENT(
    NAME "department",
    XMLATTRIBUTES(i.department AS "name"),
    XMLAGG(
        XMLELEMENT(
            NAME "emp",
            i.surname) )
    ) AS result
FROM EMPLOYEES i
GROUP BY department
```



XMLAGG

result

```
<department name="Sales">
    <emp>Doe</emp>
    <emp>Black</emp>
</department>
```

```
<department name="Accounting">
    <emp>Mason</emp>
    <emp>Wood</emp>
</department>
```



XMLGEN

- Another way to create XML is XMLGEN
- It is possible to directly specify XML code, inserting data through variables, expressed using “{” and “}”
- Variables can be also used for the names of the elements, which was not possible using XMLELEMENT, which instead requires that the names of the elements must be provided explicitly using a constant



XMLGEN

```
SELECT XMLGEN(  
    '<employee>  
        <name>{$name}</name>  
        <salary>{$sal}</salary>  
    </employee>',  
    i.name,  
    i.salary AS "sal"  
) AS result  
FROM EMPLOYEES i  
WHERE salary > 15000
```

Reference by name



XMLGEN

result

```
<employee>
    <name>Doe</name>
    <salary>20000</salary>
</employee>
```

```
<employee>
    <name>Black</name>
    <salary>18000</salary>
</employee>
```



Specifications of SQL/XML

- ISO/IEC 9074-14:2003
 - Corresponding to the things presented in these slides
- ISO/IEC 9075-14:2006
 - It adds the possibility to execute queries in XQueries, whose data model is adopted, and to validate XML documents
- ISO/IEC 9075-14:2008
 - It adds some features like the TRUNCATE TABLE statement
- ISO/IEC 9075-14:2011
 - Most recent version with feature for new data types (e.g., temporal)



References

- Some articles on SQL/XML:
 - A description of the standard:
<https://dl.acm.org/doi/10.1145/565117.565141>
 - An in-depth view of its new features:
<https://dl.acm.org/doi/10.1145/1031570.1031588>
- Some of the systems supporting SQL/XML:
 - PostgreSQL
 - Oracle database server
 - IBM DB2
 - MS SQL Server



Big Data and Data Mining

XQuery Language

Flavio Bertini

flavio.bertini@unipr.it



XQuery in Brief

- XQuery is a query language for data expressed in XML, and can thus be used to access structured and semi-structured documents
 - It became a W3C Recommendation in January 2007
- XQuery is a case sensitive language (such as XML), it covers the latest version of XPath and has similar functionality to XSLT (eXtensible Stylesheet Language Transformations)
- The XQuery language is typed expression (functional) languages
- We will see the essential aspects (i.e., widely implemented and used) of the current version 3.1. So we do not treat:
 - The definition of functions
 - Extensions for information retrieval (XQuery-FullText)
 - The documents for updating extensions (XQuery-Update)



Subjects outline

- XQuery data models
- Navigation expressions (Path Expressions)
- FLWOR expressions
- Other commonly used expressions:
 - conditional expressions (if-then-else)
 - comparison, arithmetic and logical operators
 - quantifiers (exists / each)
- Some key standard features



XQuery Data Model



Sequences 1/2

- Unlike SQL, which operates on relationships, XQuery operates on sequences, which may contain:
 - **Atomic values**, as the string “hello” or the integer 3
 - **Nodes**
- An XQuery expression receives zero (in the case of constructors) or more sequences and produces a sequence
- The main features of the sequences are the following:
 - The sequences are **ordered**, therefore:
 $(1, 2)$ is different from $(2, 1)$
 - The sequences are **not nested**, therefore:
 $(((), 1, (2, 3)))$ is equal to $(1, 2, 3)$
 - There is no difference between an *item* and a sequence with the same *item*:
 (1) is equal to 1



Sequences 2/2

- To manipulate sequences XQuery provides the following operators:
 - , (*comma*) and **to**
 - examples of alternative syntaxes to define the sequence of integers 1, 2 and 3:
 - (1, 2, 3)
 - (1, (), (2, 3))
 - (1 to 3)
 - (1, 2 to 3)
 - **union** (also in its equivalent form **|**), **intersect**, **except**
 - (A) union (A, B) → (A, B)
 - (A, B) intersect (B, C) → (B)
 - (A, B) except (B) → (A)

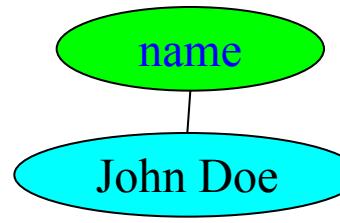


Nodes 1/3

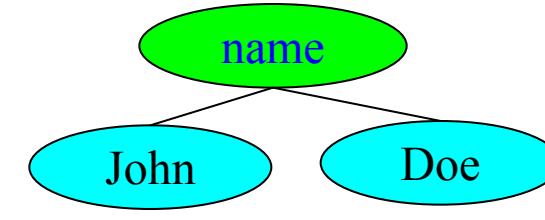
- In addition to atomic values, a sequence can contain nodes
- XML documents and fragments are represented as trees, whose nodes can be of type: **document**, **element**, **attribute**, **namespace**, **text**, **comment**, and **processing instruction**
 - The **namespace** nodes such as <xmlNs:xsi>, are not represented in the XQuery data model, so we do not consider them
 - In a similar way, we will not treat **comments** and **processing instructions**, which are not (or should not) be used to store data
- The string value of a document node is equal to the concatenated contents of all its descendant text nodes, in the order in which they are in the document

Nodes 2/3

- The nodes of type attribute are not sorted - if extracted using a query in XQuery, the order does not necessarily correspond to the one in which they are encountered in the source XML document
- Once extracted, however, they are inserted in a sequence, of which they keep the order
- Two **text nodes** can not be adjacent:
 - an element `<name>John Doe</name>` is therefore represented as follows



CORRECT



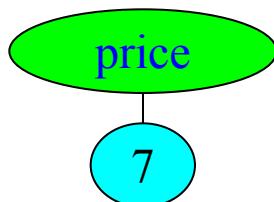
WRONG!

Nodes (3/3)

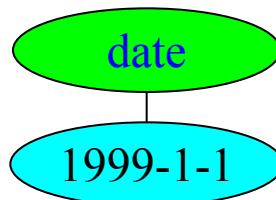
- If a schema (XML Schema) is associated with the document, the nodes can have a type, as in the following examples:



It's of type *string* with the *text value* “T.Harris”



It's of type *integer* with *integer value* 7



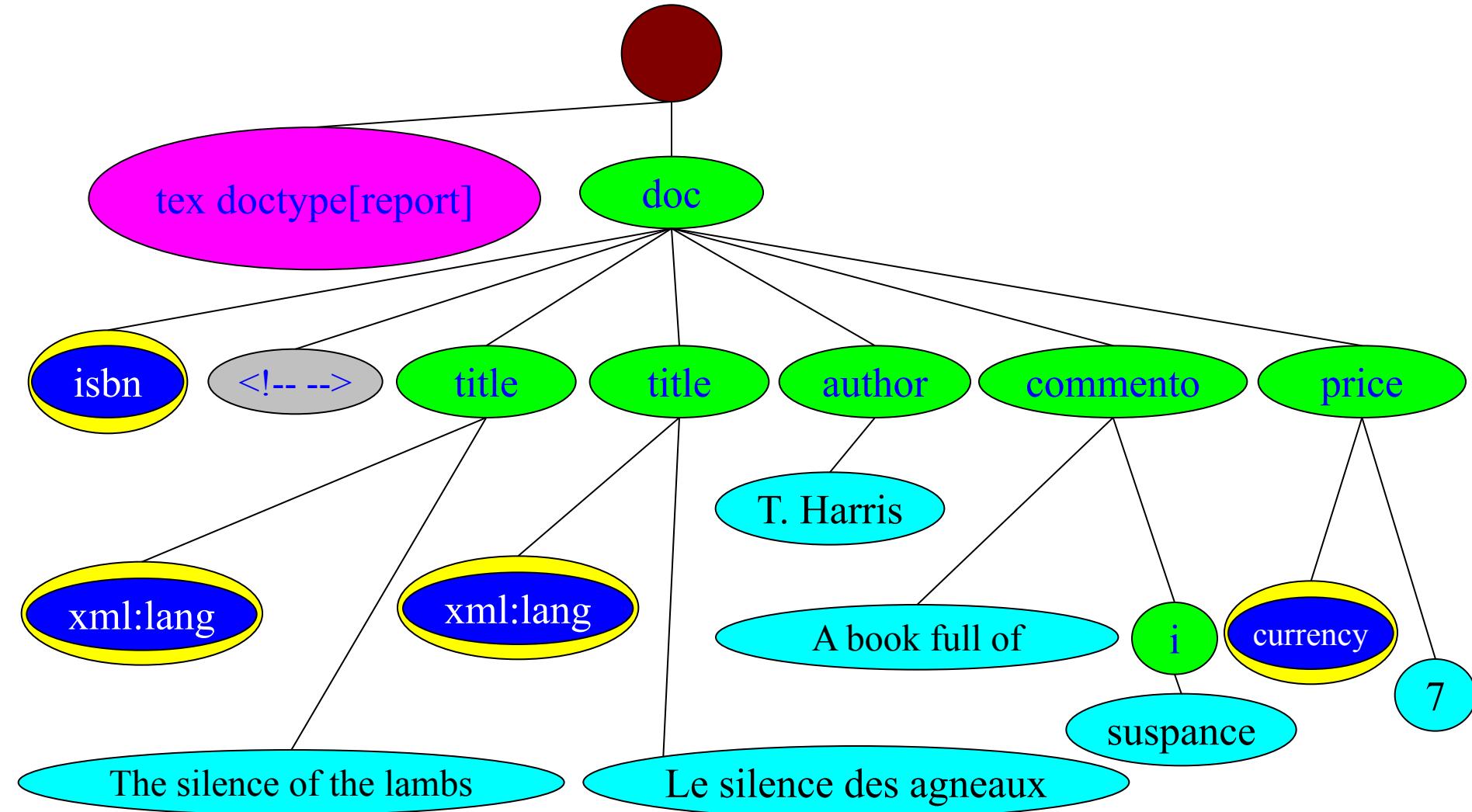
It's of type *date* with value “1st of January 1999”



Examples of nodes 1/2

```
<?xml version="1.0"?>
<?tex doctype[report] ?>
<doc isbn="2-266-04744-2">
    <!-- editor is missing ! -->
    <author>T. Harris</author>
    <title xml:lang="en">The silence of the lambs</title>
    <title xml:lang="fr">Le silence des agneaux</title>
    <commento>
        A book with a lot of <i>suspance</i>
    </commento>
    <price currency="euro">7</price>
</doc>
```

Examples of nodes 2/2





Path Expressions



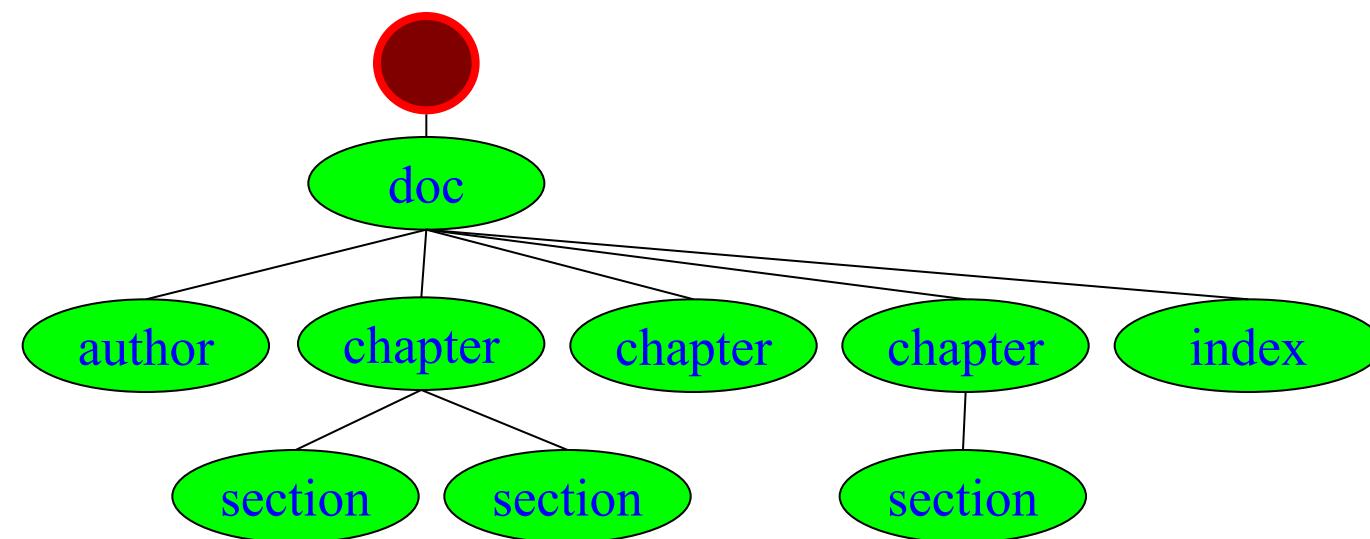
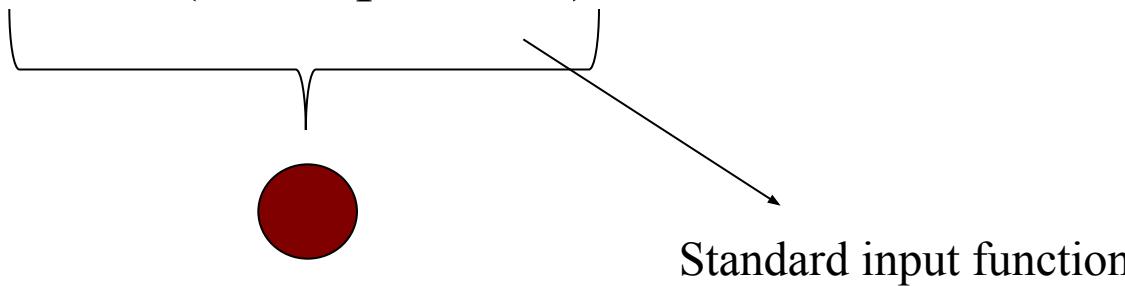
Observations

- The Path Expressions can be used to extract values from XML nodes and trees, and to check their properties
 - Recall that, as with any other expression in XQuery, the Path Expressions elaborate **sequences**
- A Path Expression consists of a series of steps, separated by the character /
 - Each step is evaluated in a **context**, i.e., a sequence of nodes (with additional information, for example the position of the node), and produces a sequence
 - The next step is evaluated using as context the sequence of nodes produced by the previous step



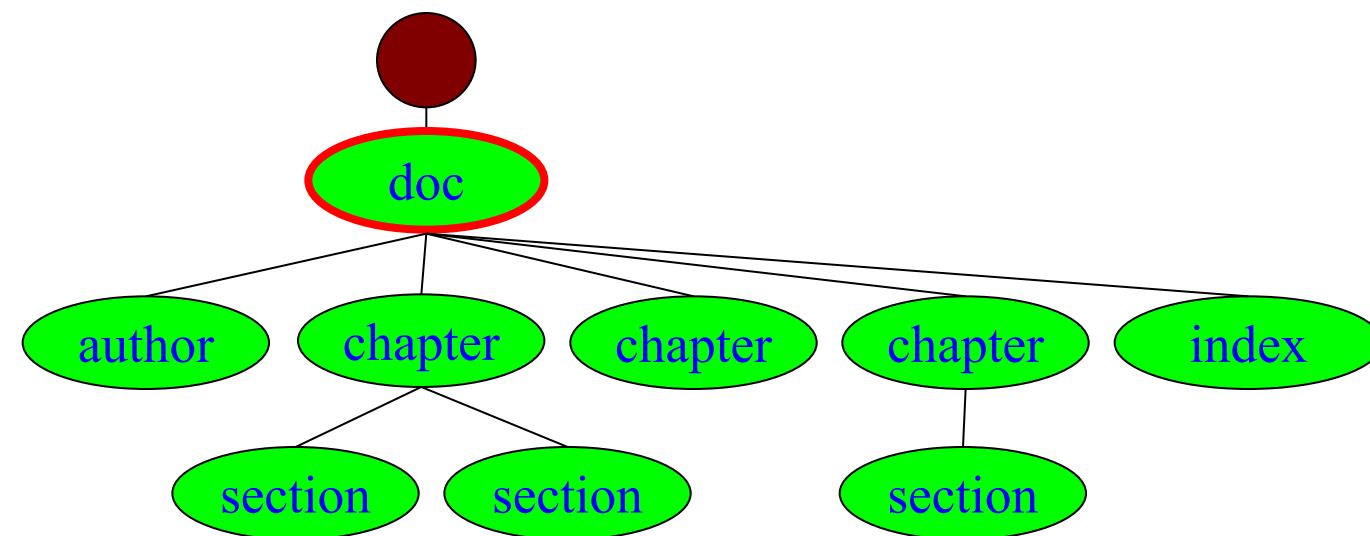
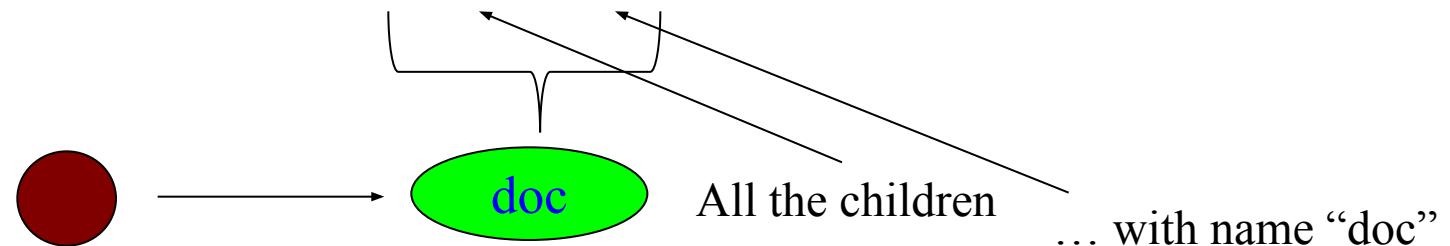
Example of evaluation (step 1)

`fn:doc('example.xml') / child::doc / child::chapter / child::section`



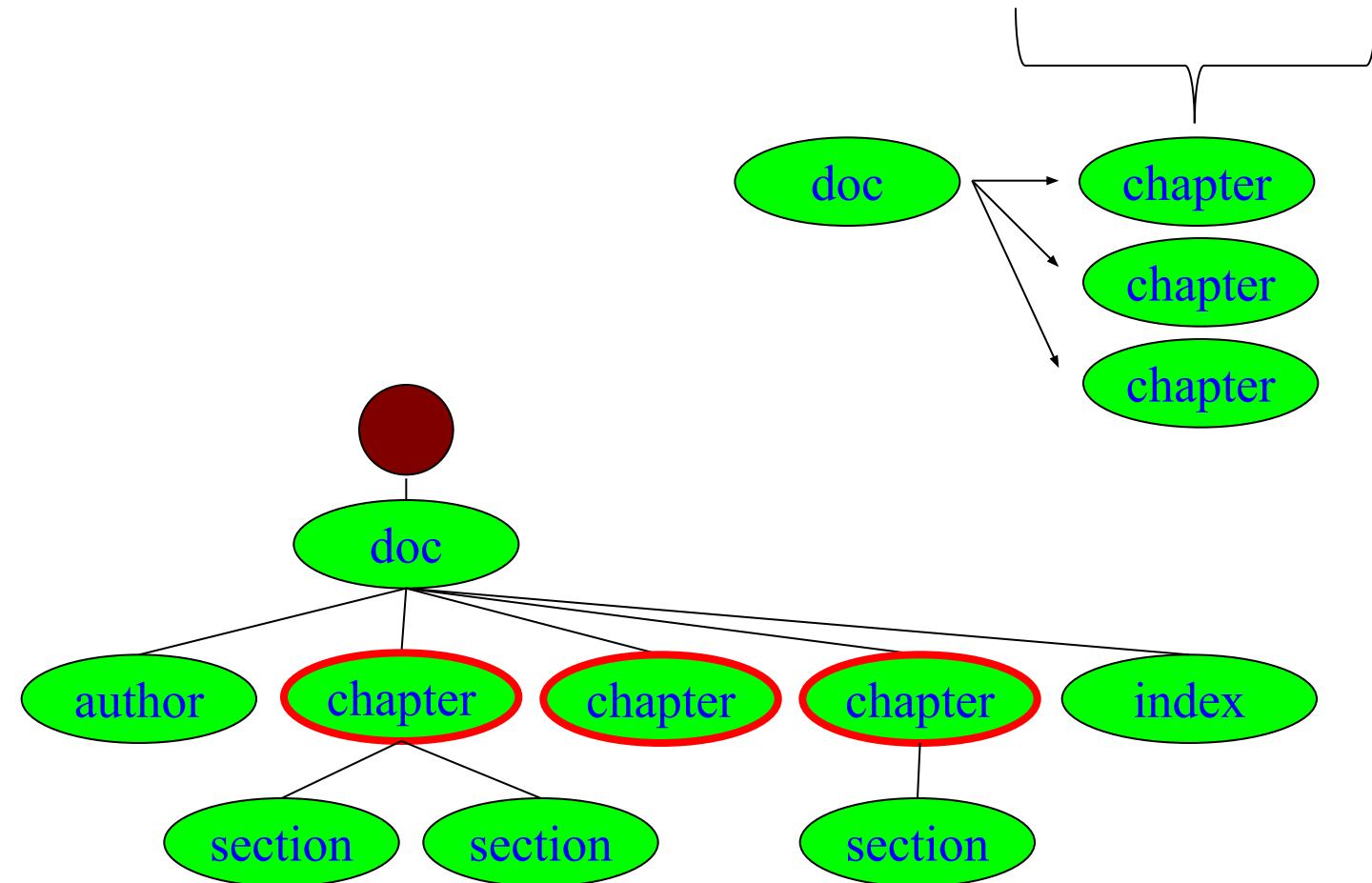
Example of evaluation (step 2)

`fn:doc('example.xml') / child::doc / child::chapter / child::section`



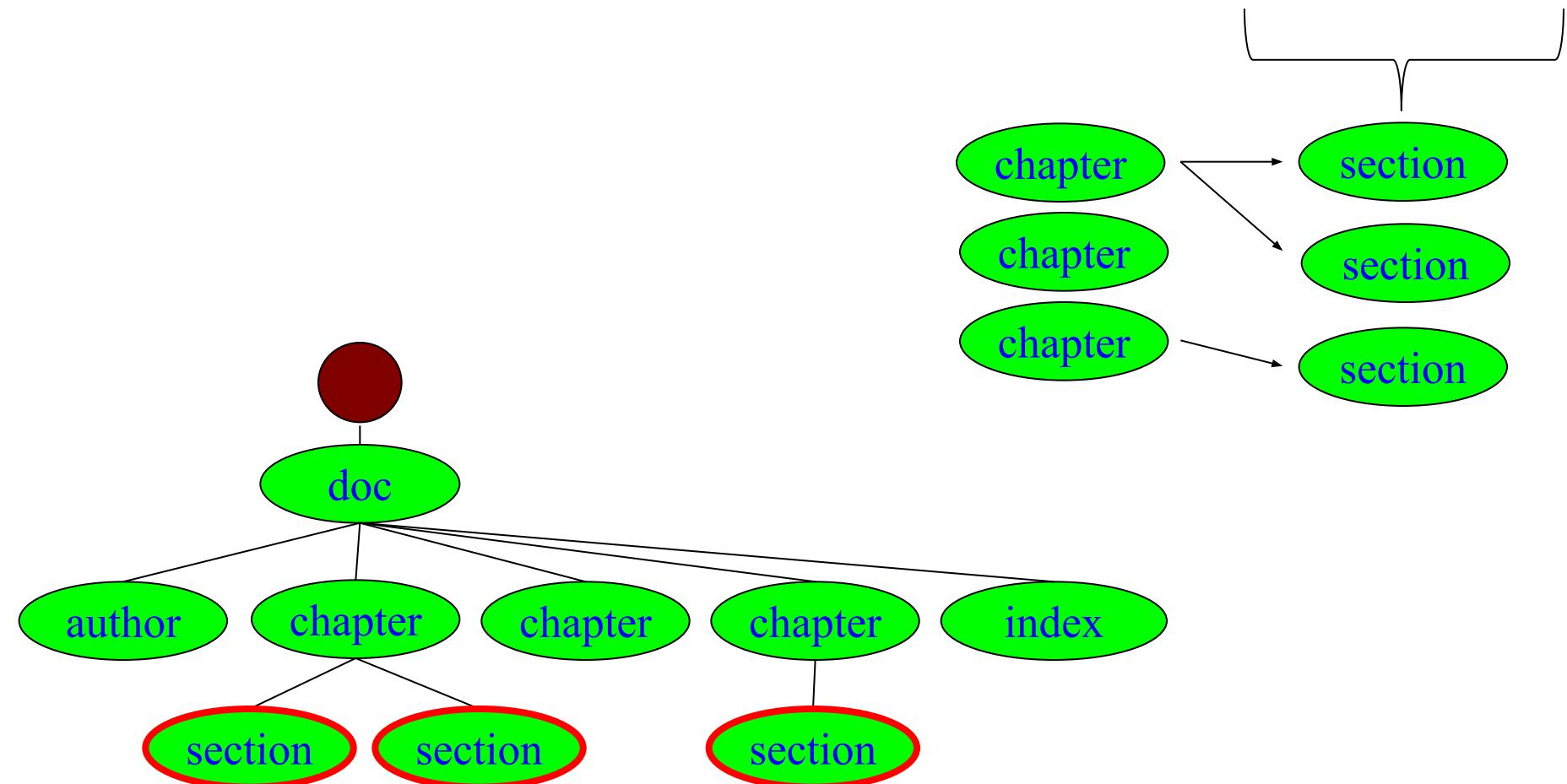
Example of evaluation (step 3)

`fn:doc('example.xml') / child::doc / child::chapter / child::section`



Example of evaluation (step 4)

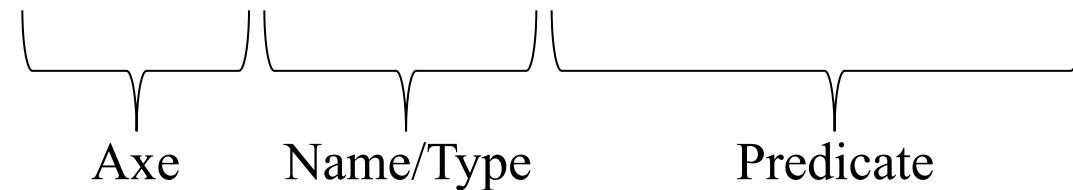
`fn:doc('example.xml') / child::doc / child::chapter / child::section`



Step structure

- The step of a Path Expression can be made of three main parts:
 1. An **axes**, that select nodes depending on its position w.r.t. the context node
 - in the example, the children – **child::**
 2. A **test** that filters these nodes depending on their name and their type
 - in the example, the name – **section**
 3. One or more **predicates**, which further filter the nodes depending on more generic criteria
 - in the example, the fact of **not being the first child**

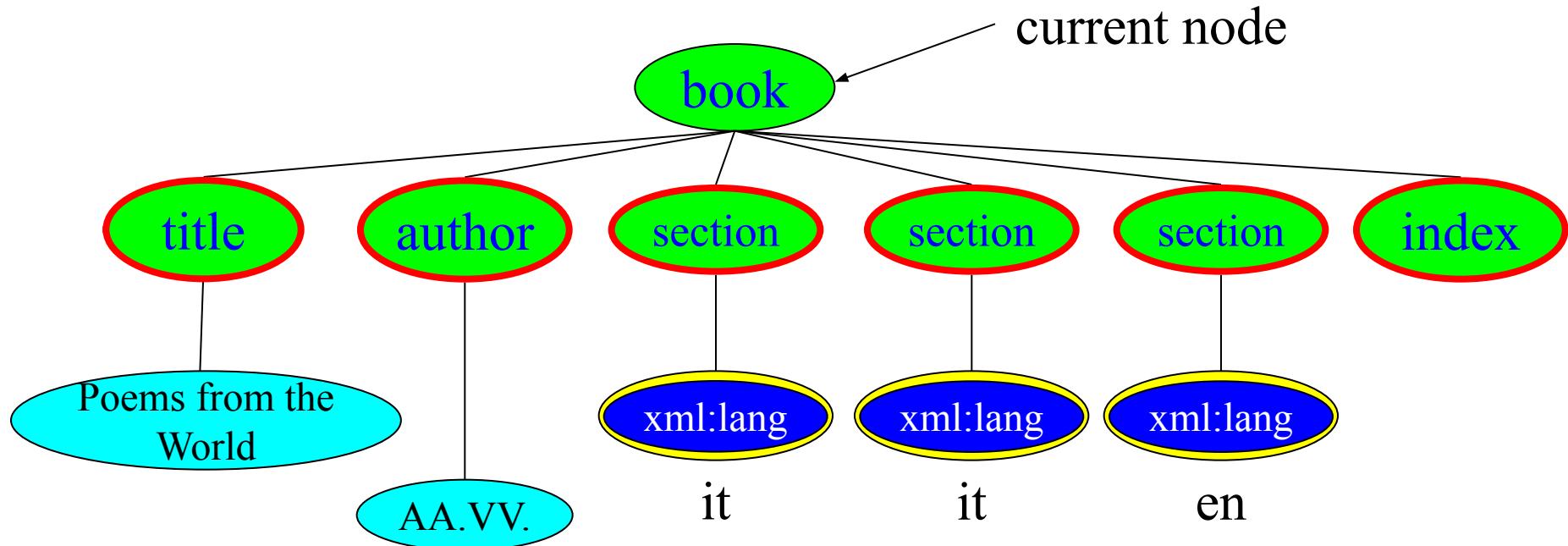
```
child::section[position()>1]
```





Example of evaluation in one step (1)

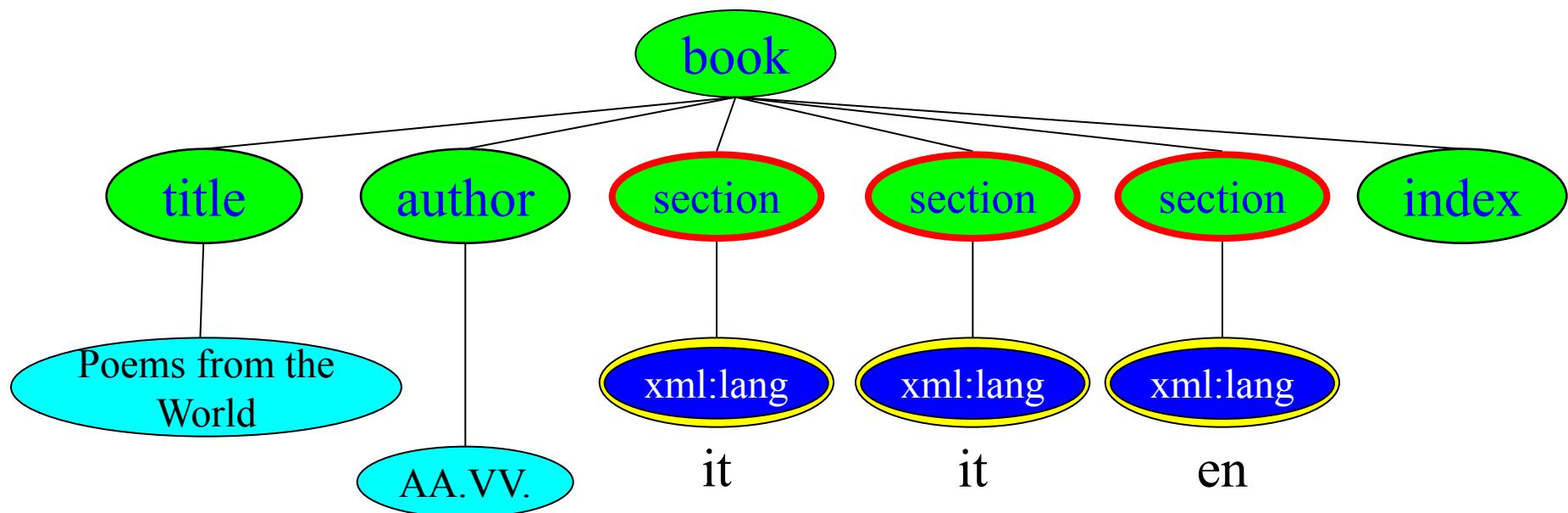
`child::section[attribute::xml:lang = 'it']`





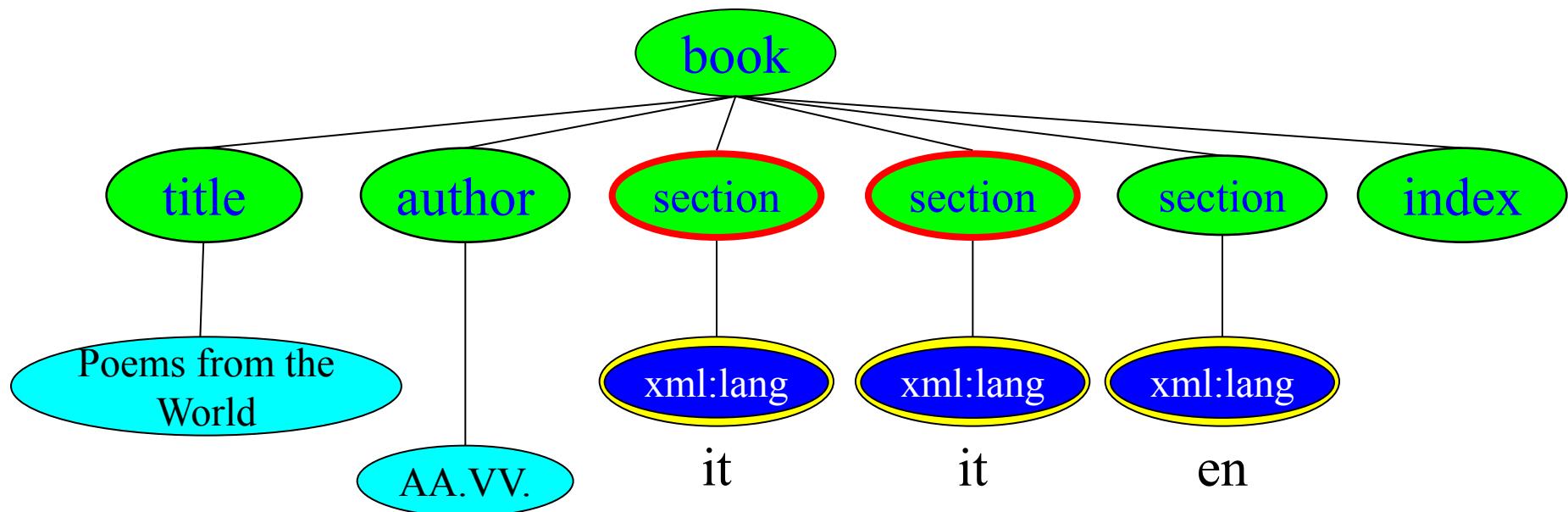
Example of evaluation in one step (2)

`child::section[attribute::xml:lang = 'it']`



Example of evaluation in one step (3)

child::section[attribute::xml:lang = 'it']

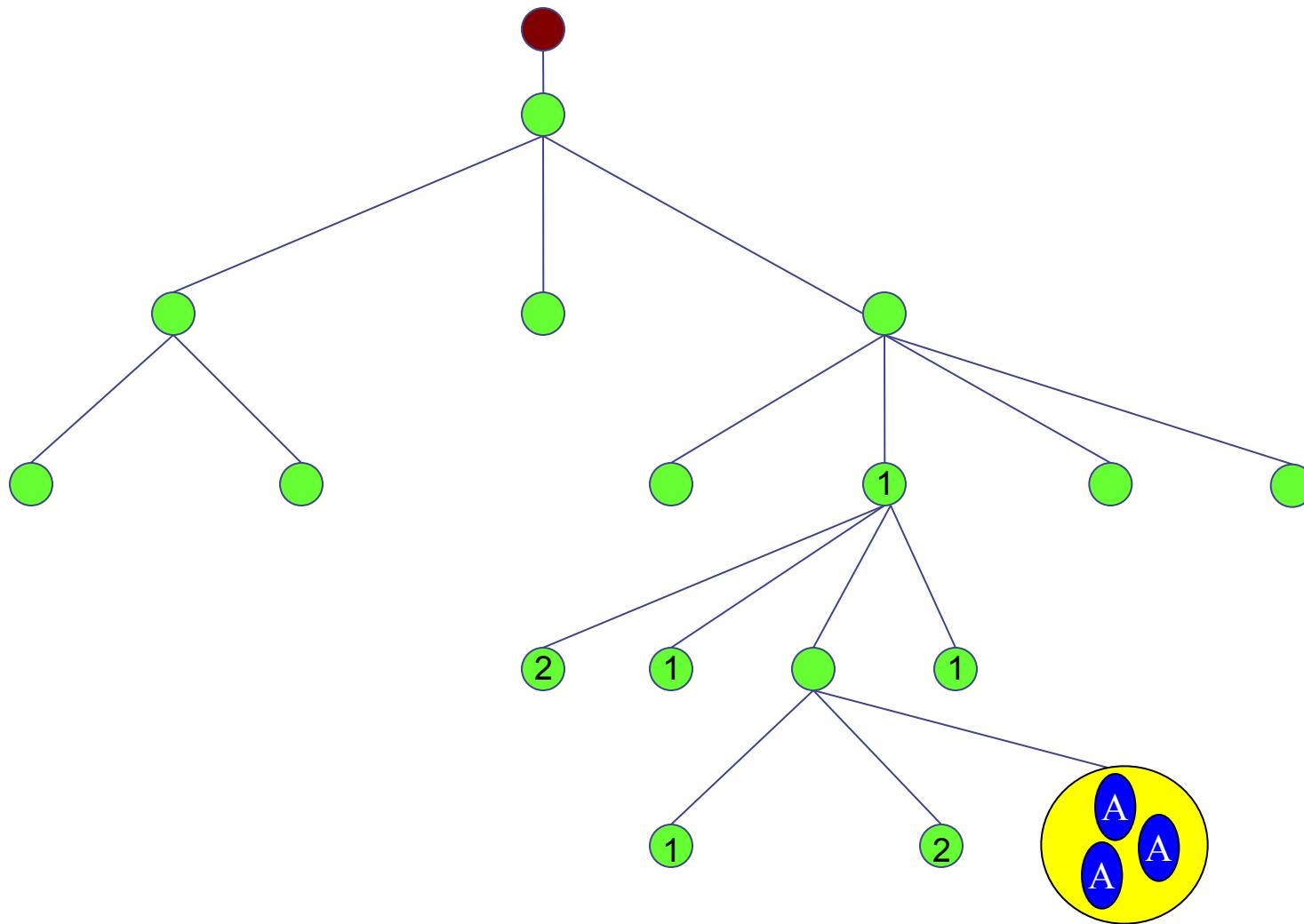


Axes

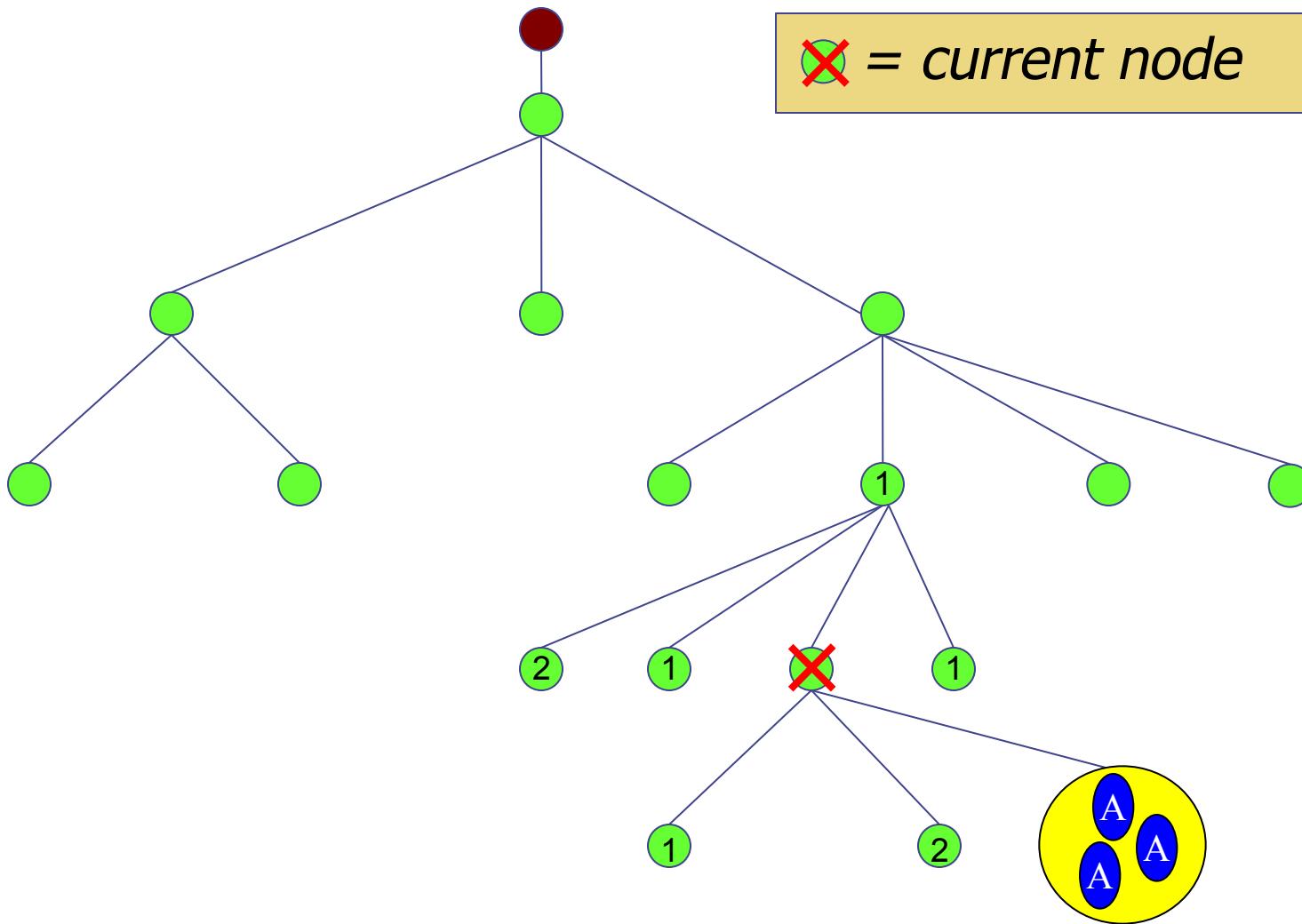
- The main axes defined in XQuery (and XPath) are the following, exemplified in the next slides:
 - self::
 - child::
 - parent::
 - ancestor::
 - descendant::
 - following-sibling::
 - preceding-sibling::
 - attribute::
- In addition, there are combined axes:
 - descendant-or-self::
 - ancestor-or-self::



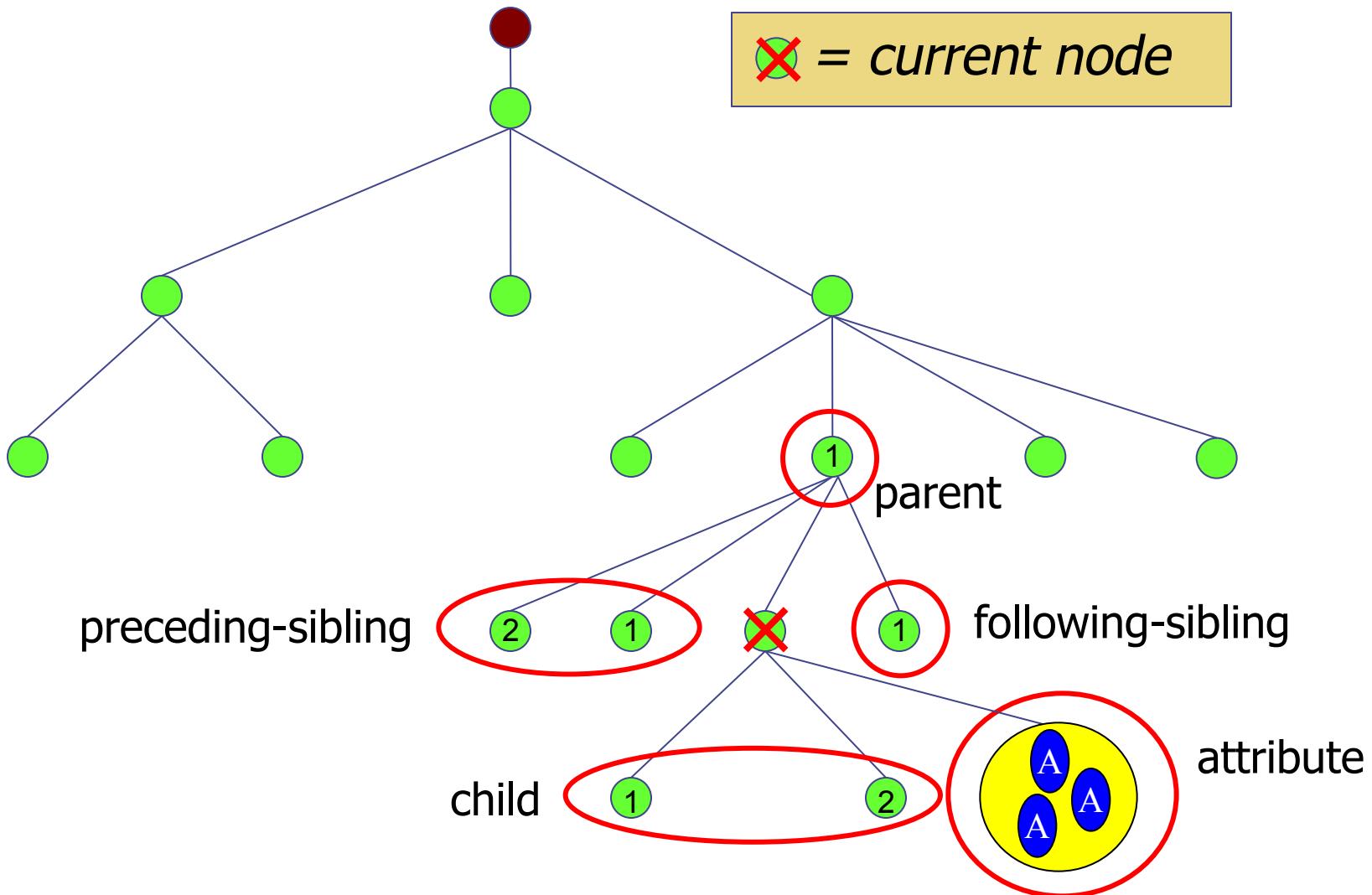
Example of Axes 1/4



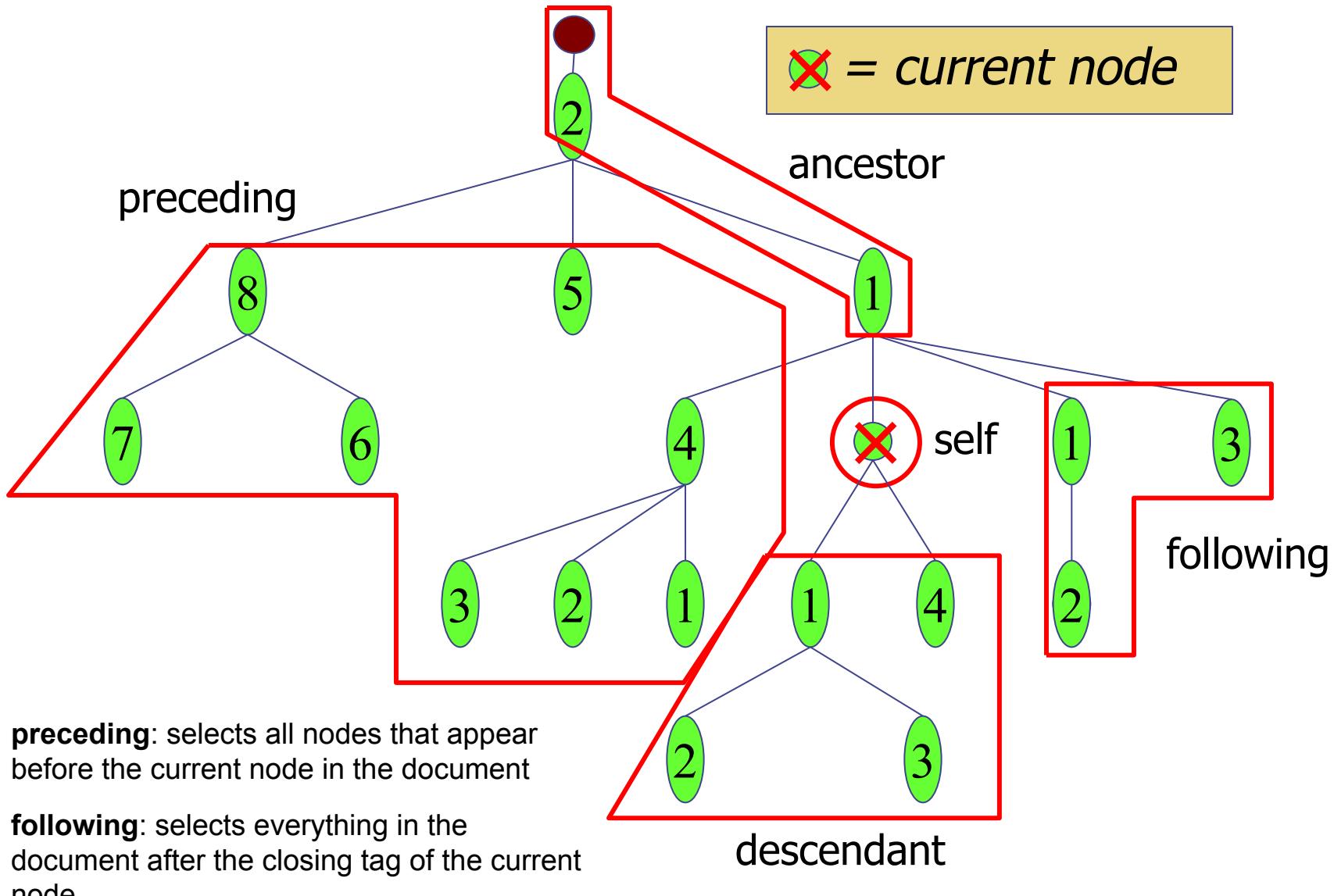
Example of Axes 2/4



Example of Axes 3/4

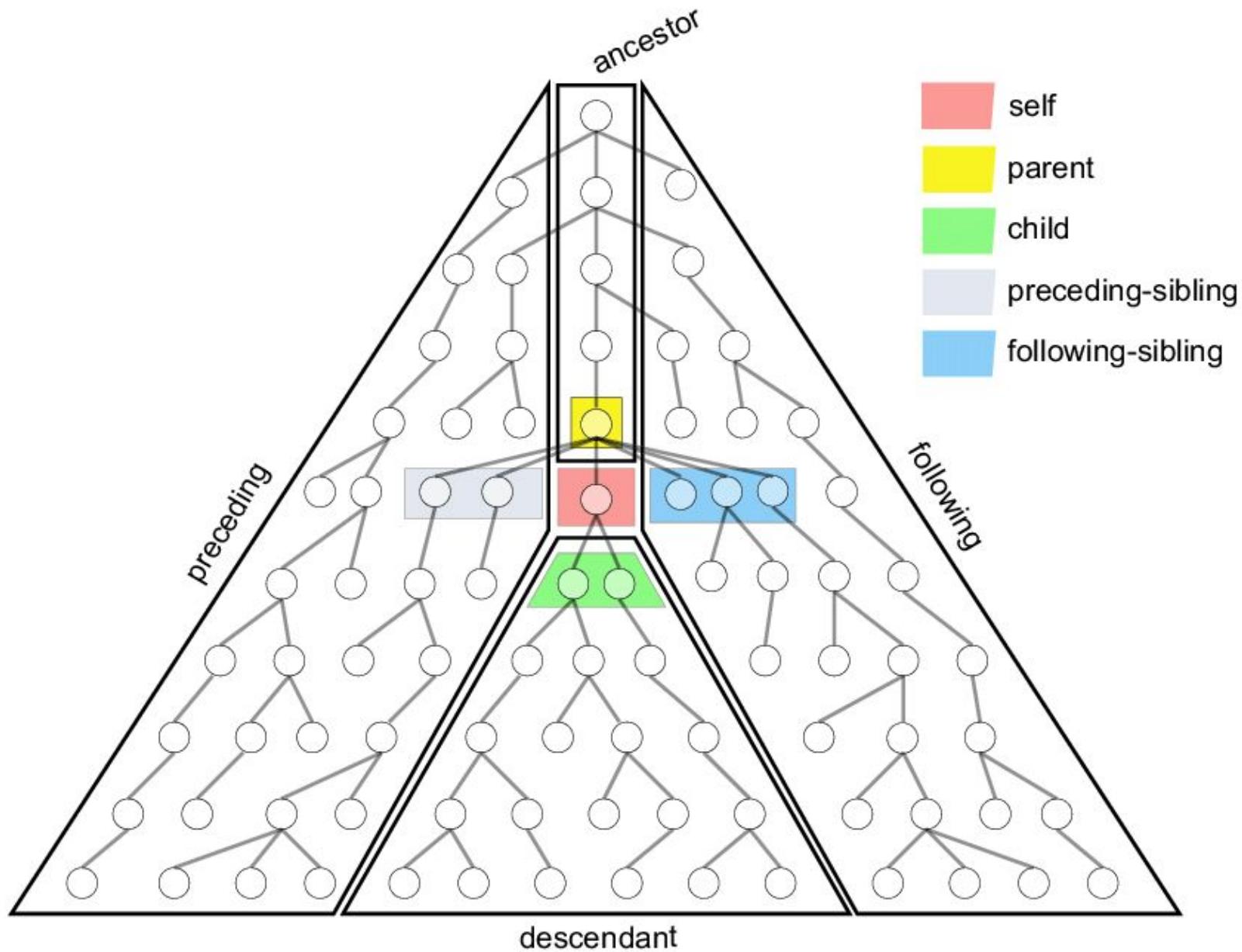


Example of Axes 4/4





Kinship in a Nutshell

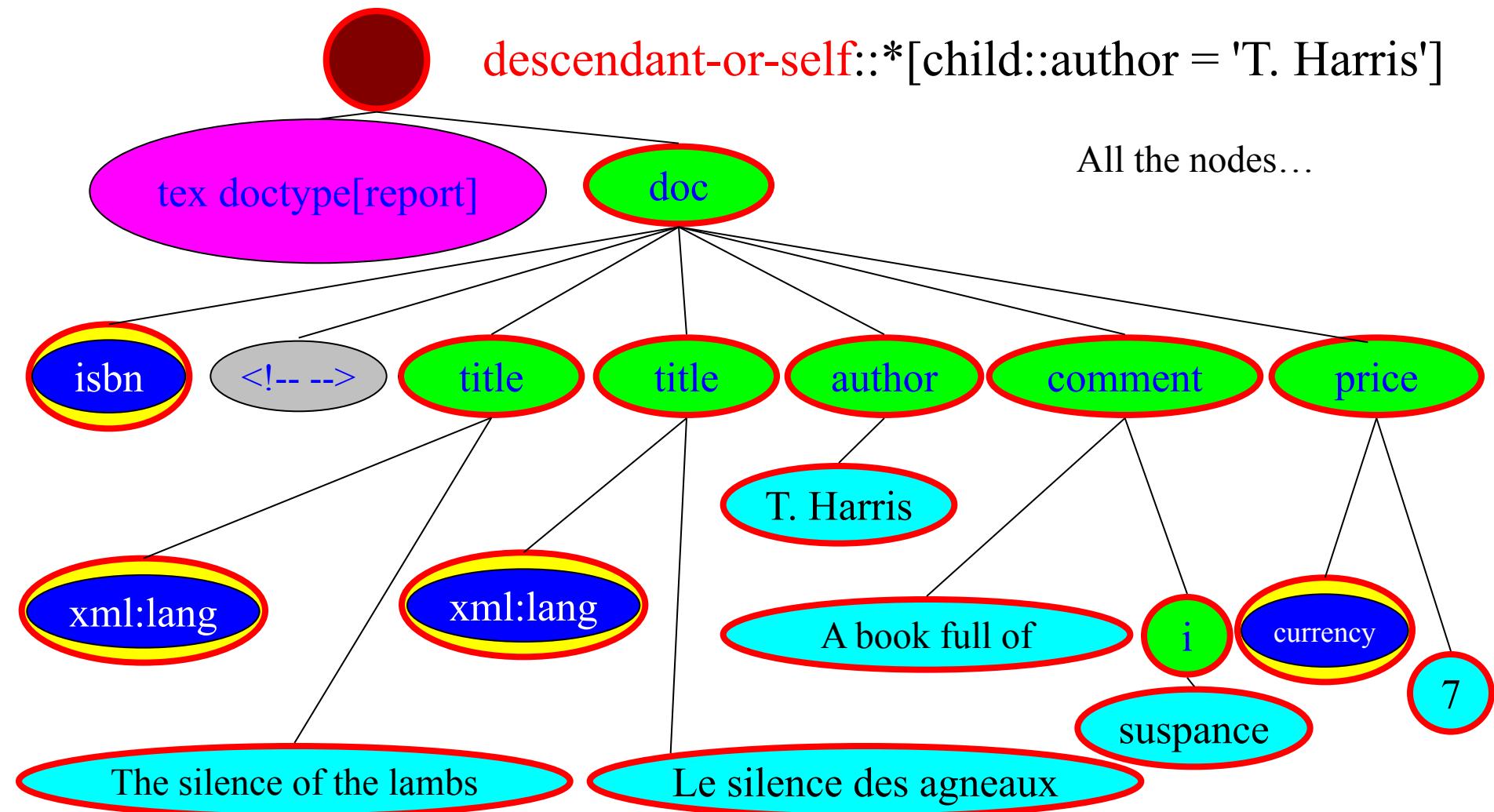




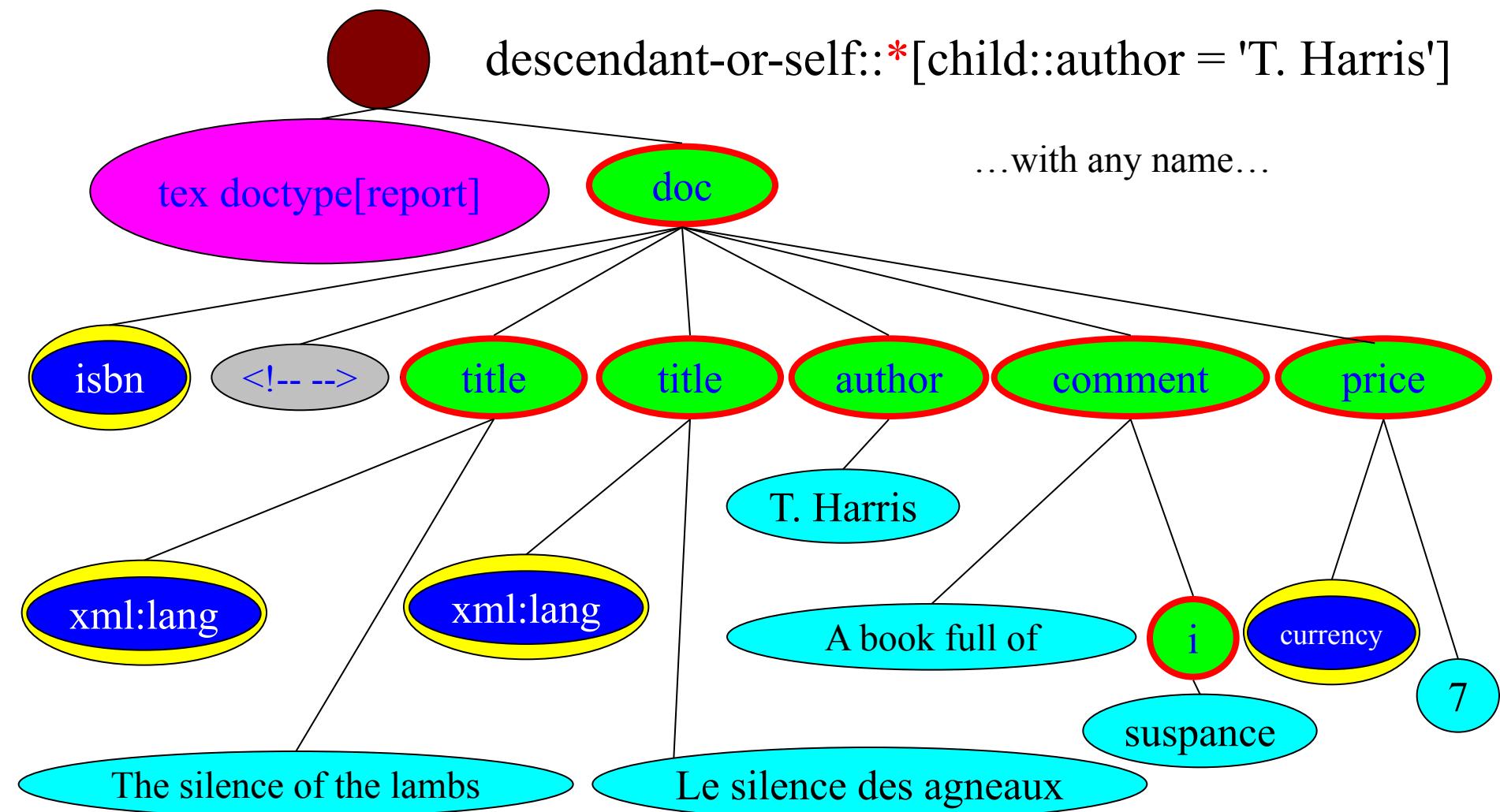
Tests on names/types: examples

- The second component of a navigation step, filters the nodes selected by the axe, verifying:
 - the **name**
 - **child::section** returns only the child elements with tag `<section>`
 - **child::*** returns all the child elements
 - **attribute::xml:lang** returns the attribute `xml:lang`
 - or the **type**
 - **descendant::node()** returns all the descendants nodes
 - **descendant::text()** returns all the descendants nodes of type text
 - **descendant::element()** returns all the descendants elements nodes
 - or **both**
 - **descendant::element(person, xs:decimal)** returns person elements of decimal type

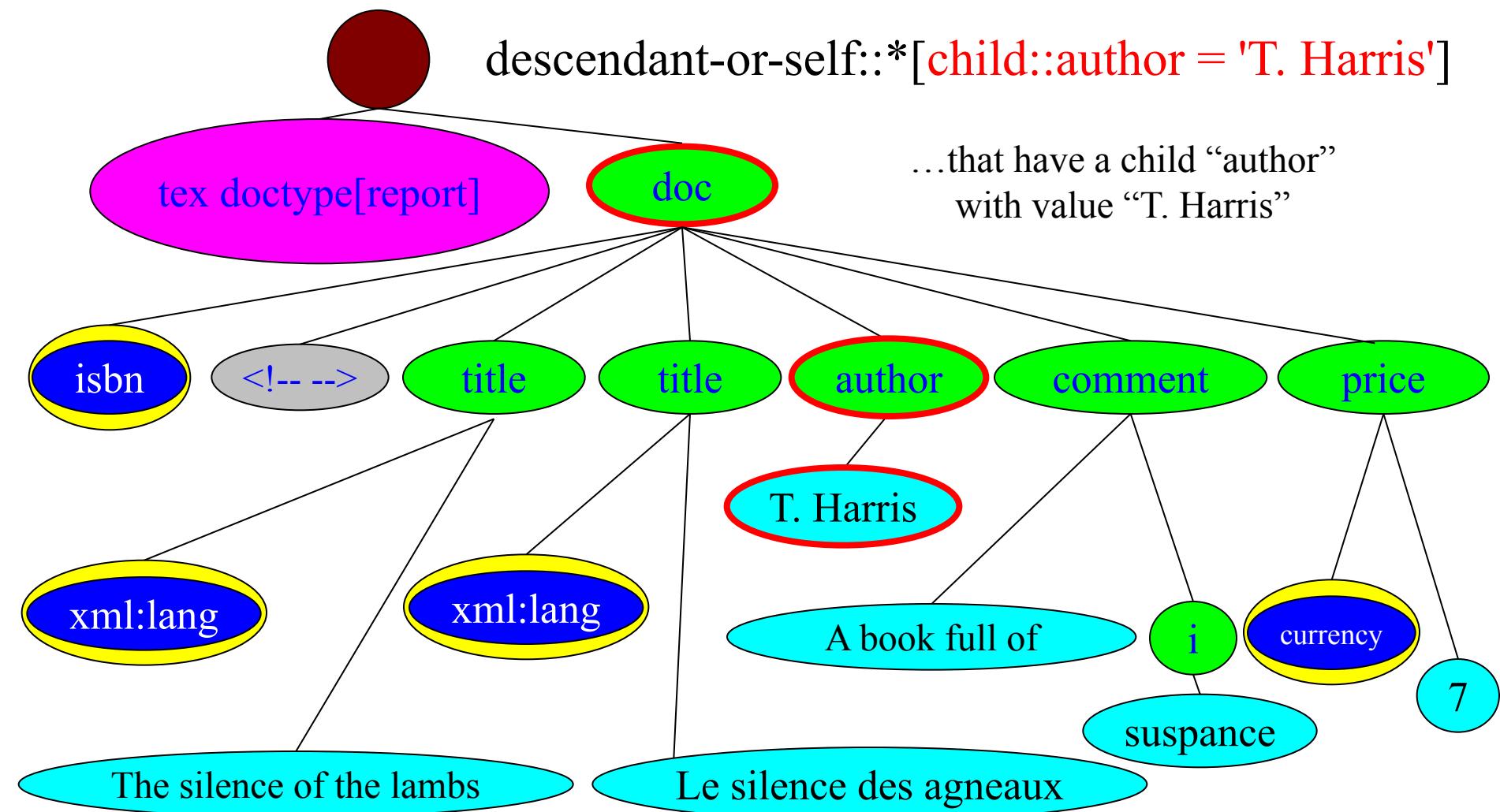
Another example of a step 1/3



Another example of a step 2/3



Another example of a step 3/3



Predicates

- Every step can end with one or more predicates (in conjunction with *and*), included between “[” and “]”, to further filter the nodes selected using axes and tests on names/types

An integer i denotes that we want to extract the i -th node

This predicate requires the extraction of only the nodes with an attribute `xml:lang` with value “it”

```
child::section[1][attribute::xml:lang="it"]
```

Predicate 1

Predicate 2



Evaluation of Predicates

- If the predicate expression is an atomic value of a numerical data type, the expression is evaluated to true in case the position indication of the element corresponds to the numerical value of the predicate
 - **child::chapter[2]** returns only the second child with the tag <chapter>
- If the expression returns a empty sequence, the predicate is false, conversely if the first item is a node it returns true
 - **child::chapter[child::title]** returns all the children with tag <chapter> that have at least a child with tag <title>
- Otherwise, the standard practices for predicates will be followed:
 - **child::chapter[attribute::xml:lang = “it”]** returns all the children with tag <chapter> that have an attribute xml:lang with value “it”



Complete Path Expressions

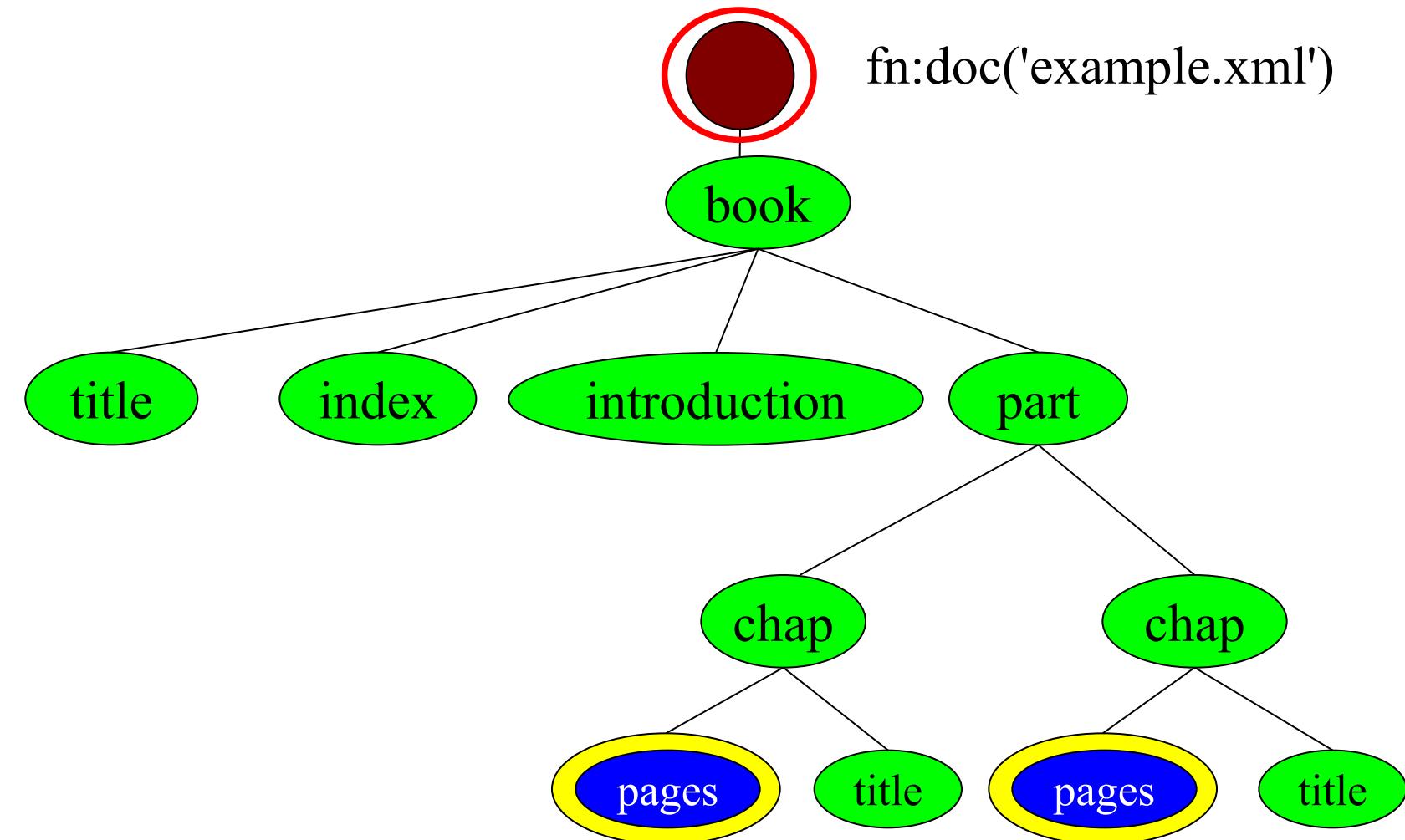
- A path expression can also begin with the following **prefixes**:
 - With the **/ character**: it corresponds to an input sequence of the expression that contains the **tree's root**
 - With the **// characters**: it corresponds to an input sequence of the expression that contains **all the nodes in the document**
- Some examples of complete path expressions:
 - **/descendant::figure[fn:position() = 42]**
Select the 42th figure of the document
 - **/child::book / child::chapter[5] / child::section[2]**
Select the second section of the fifth chapter
 - **//self::chapter[child::title]**
Select all the chapters that have at least one child <title>



Short syntax

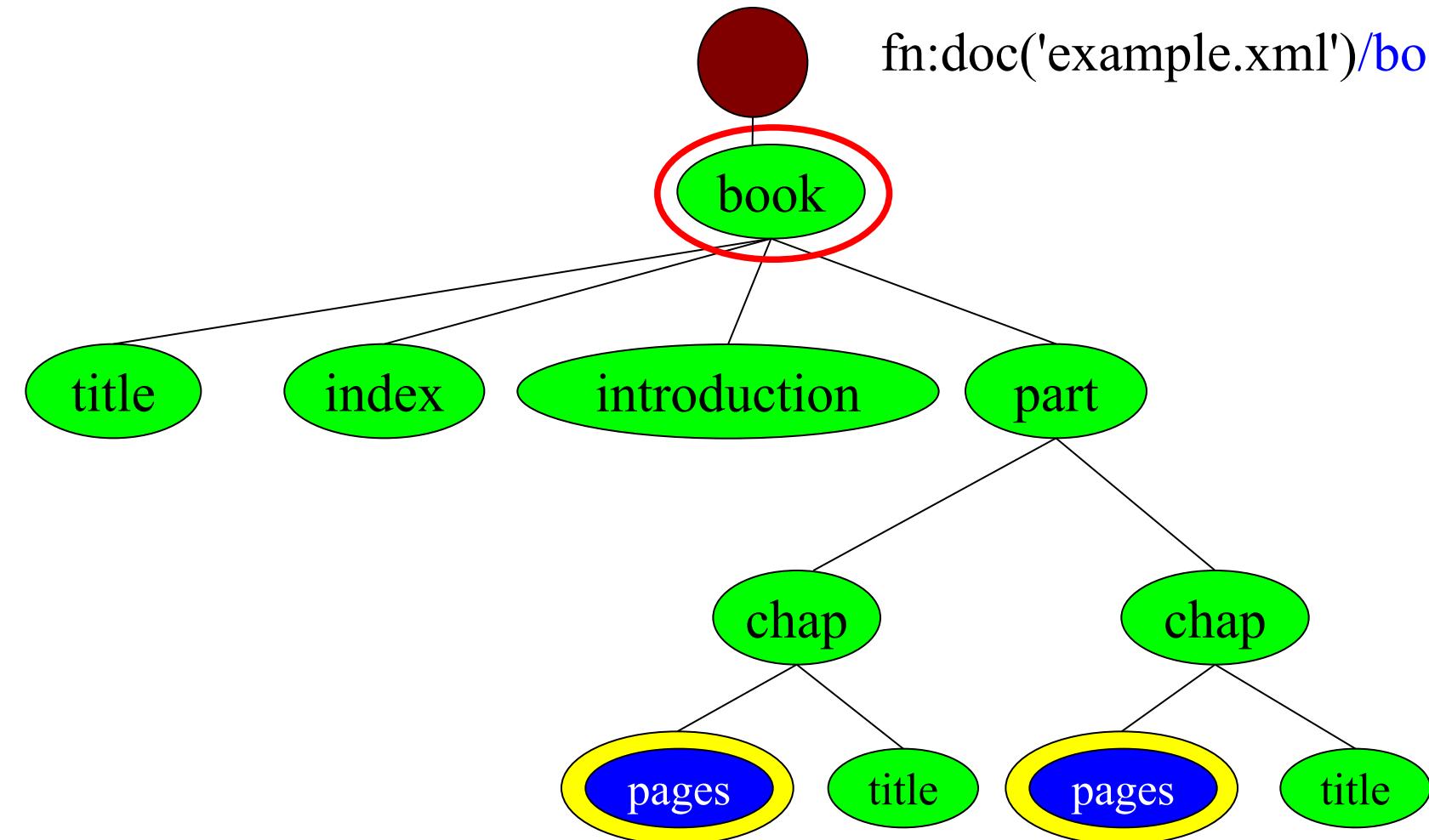
- In order to write more compact expressions, it is possible to use some shortcuts:
 - Omission of the child:: axe:
`child::section/child::paragraph` → `section/paragraph`
 - Substitution of the axe attribute:: with the character @:
`para[attribute::type="warning"]` → `para[@type="warning"]`
 - Substitution of descendant-or-self::node() with double slash (//):
`div/descendant-or-self::node()/child::paragraph` → `div//paragraph`
 - Substitution di self::node() with a dot (.):
`self::node()/descendant-or-self::node()/child::para` → `.//para`
 - Substitution of parent::node() with two dots (..):
`parent::node()/child::section` → `../section`

Examples 1/6



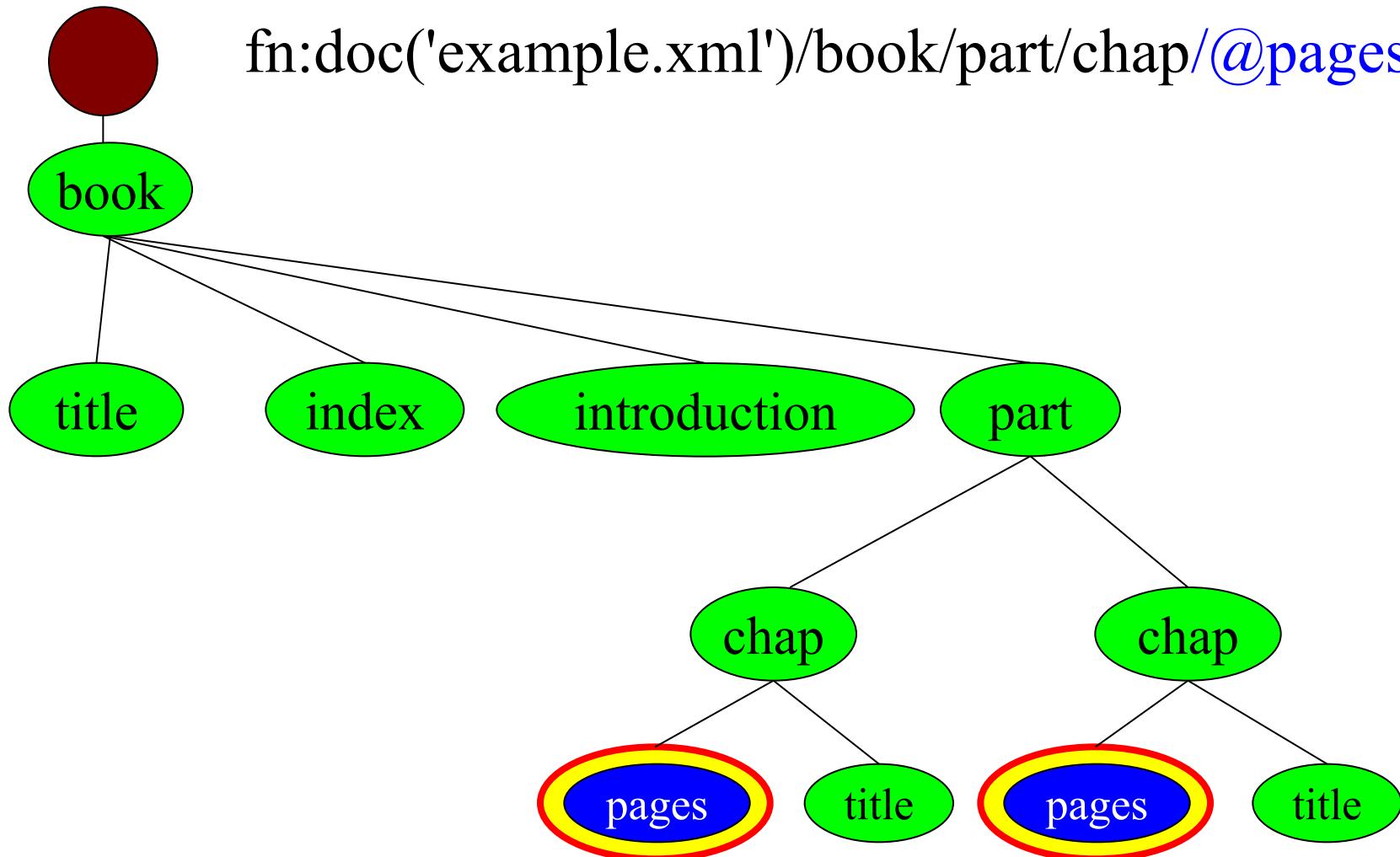
Examples 2/6

`fn:doc('example.xml')/book`



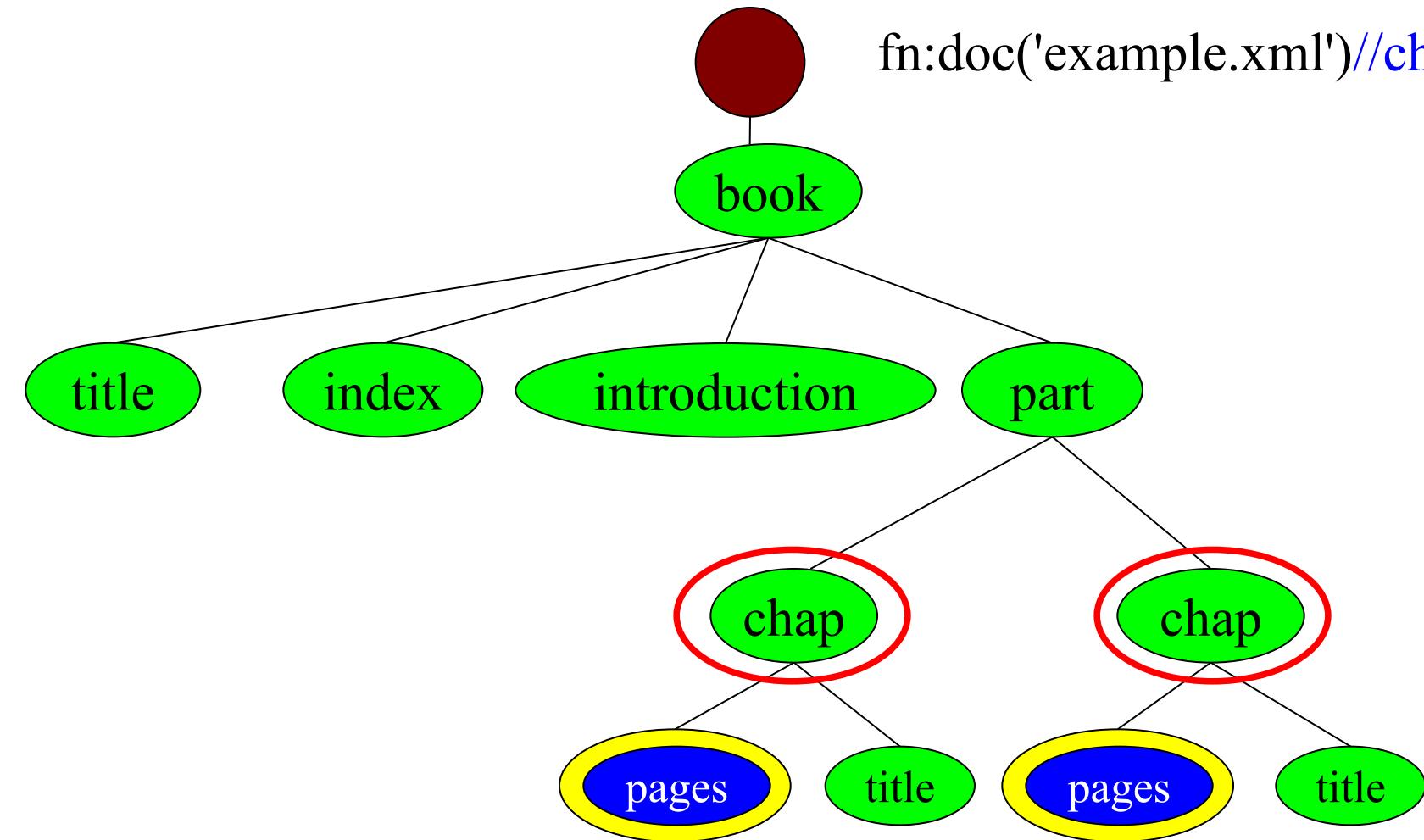
Examples 3/6

`fn:doc('example.xml')/book/part/chap/@pages`



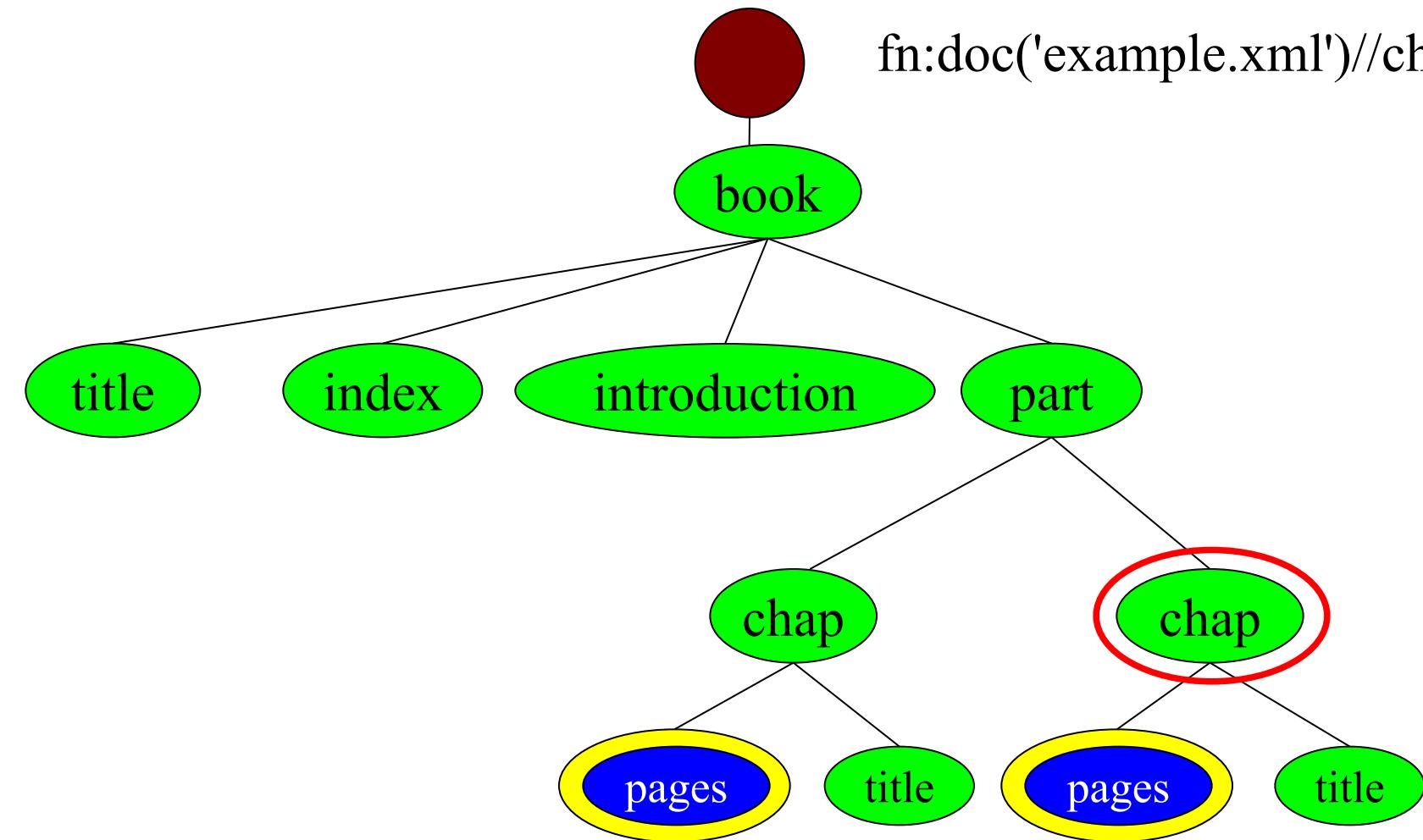
Examples 4/6

`fn:doc('example.xml')//chap`



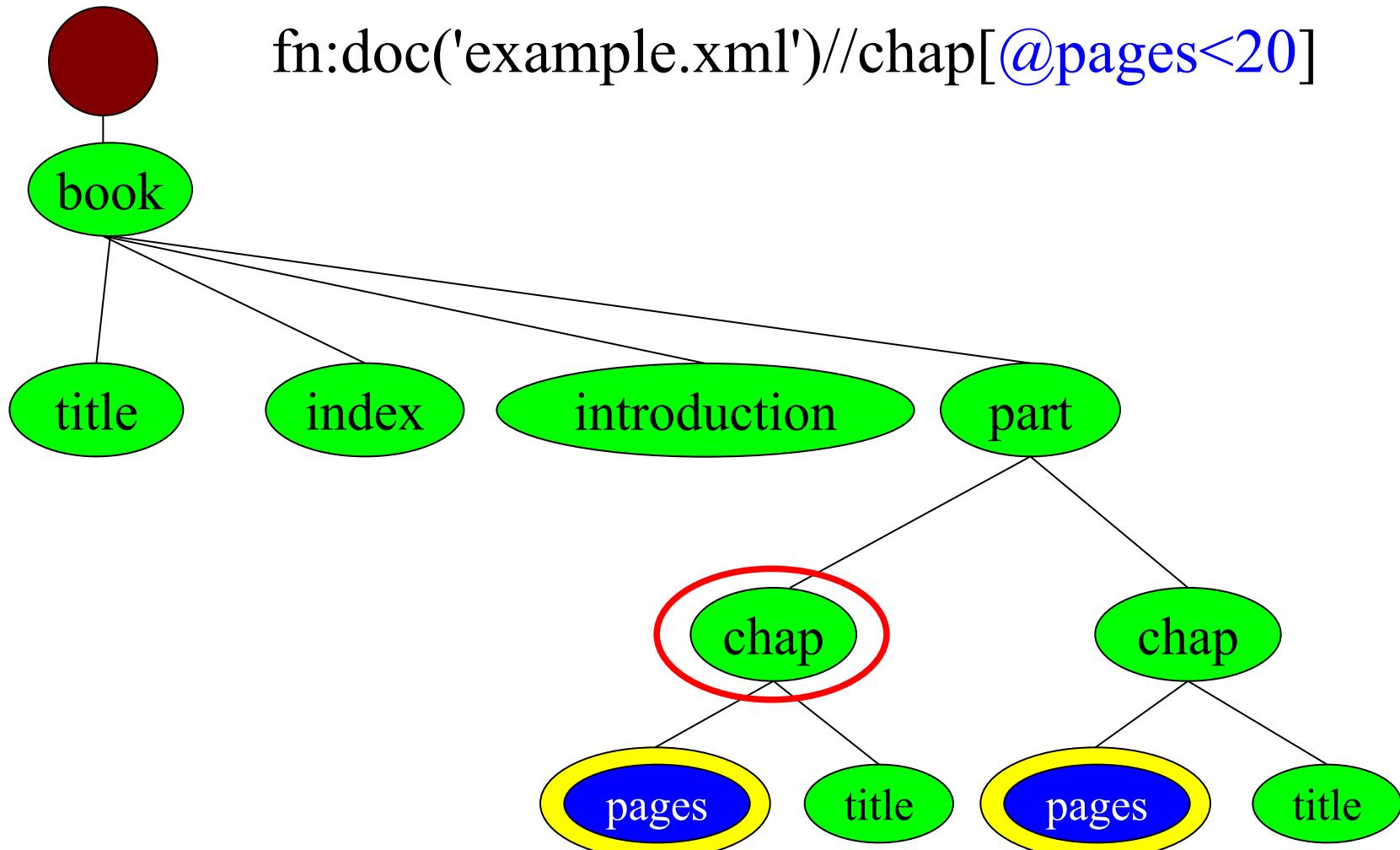
Examples 5/6

`fn:doc('example.xml')//chap[2]`



Examples 6/6

`fn:doc('example.xml')//chap[@pages<20]`





FLWOR Expressions

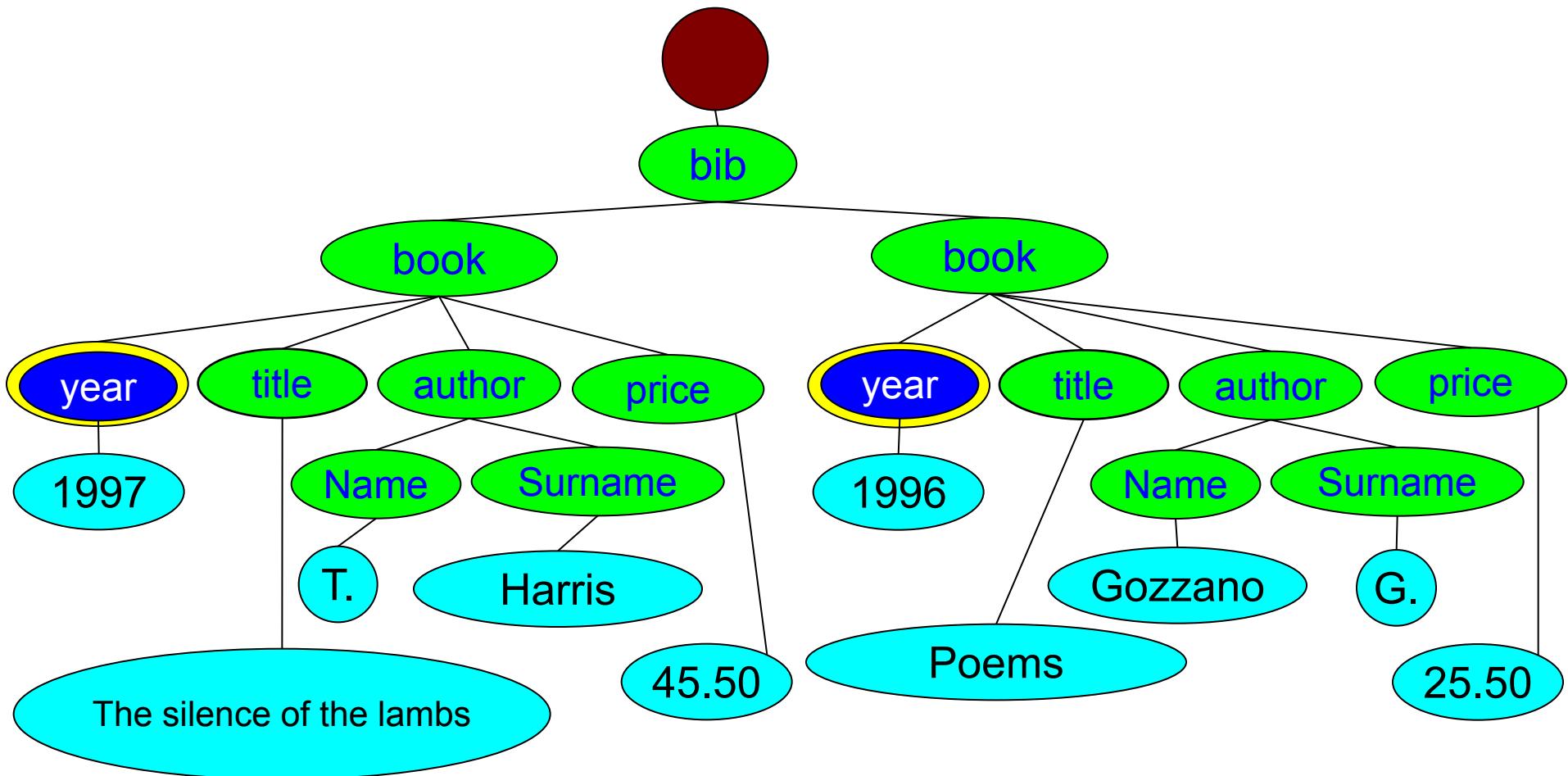


FLWOR Expressions

- A FLWOR expression (pronounced “flower”) is similar to an SQL statement Select-From-Where, however, it is defined in terms of variables binding. It’s made of 5 parts, some of them are optional:
 - **For**: associate one or more variables to expressions
 - **Let**: create an alias of the entire result of an expression
 - For and Let create a list with **all the possible associations**, also called “tuples” in the XQuery specification
 - **Where**: filters the list of associations on the basis of a condition
 - **Order by**: sorts the list of associations
 - **Return**: create the result of the FLWOR expression
- These expressions, as any other XQuery expression, can be commented using the symbols (: and :)
`(: This is a comment (: this is a nested comment... :) :)`

Iterations of elements – for clause 1/5

- For each book, list the year and the title



Iterations of elements – for clause 2/5

- First we select all the books:

doc("example.xml")/bib/book

- Then **for each book**, that we associate to the variable **\$b**,

for \$b in doc("example.xml")/bib/book

- We write the result:

return

<book year="*{Extraction of the year of \$b, in XQuery}*">

{Extraction of the title of \$b, in XQuery}

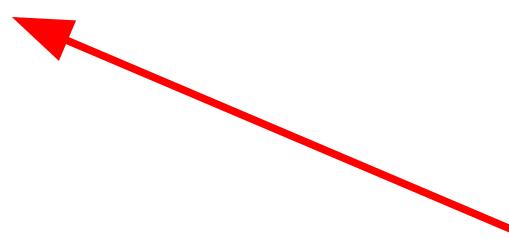
</book>



Iterations of elements – for clause 3/5

```
for $b in doc("example.xml")/bib/book  
return  
<book year="{ $b/@year }">  
  { $b/title }  
</book>
```

The curly brackets delimit an XQuery expression, which must be evaluated to create the result



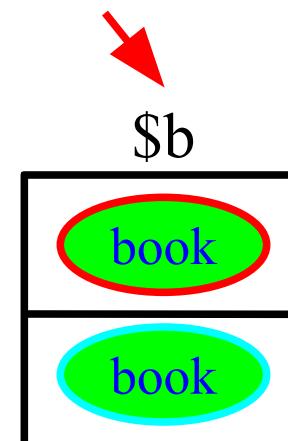
Inside the RETURN clause there can be specified any XQuery expression. In this case, we are using **constructors** that generates XML code, and are written in XML syntax



Iterations of elements – for clause 4/5

```
for $b in doc("example.xml")/bib/book  
return  
<book year="{ $b/@year }">  
  { $b/title }  
</book>
```

Evaluating this expression, we obtain a sequence of nodes (that are associated with the \$b variable):



Iterations of elements – for clause 5/5

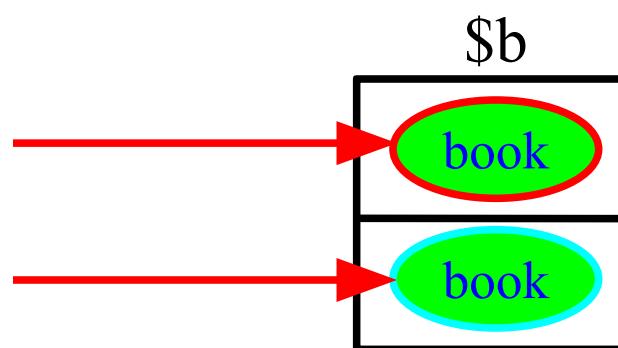
```
for $b in doc("example.xml")/bib/book
```

```
return  
<book year="{ $b/@year }">  
  { $b/title }  
</book>
```

For each association (\$b)
we evaluate this part of
the expression:

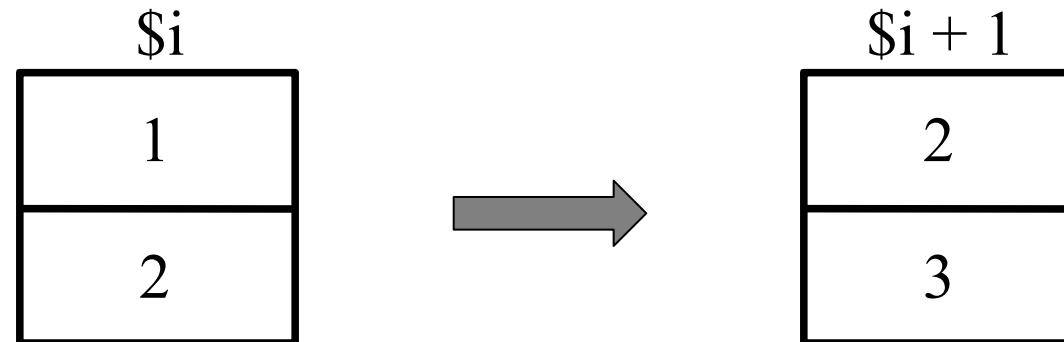
```
<book year="1997">Dubliners</book>
```

```
<book year="1996">Poems</book>
```



Other examples (1)

- for \$i in (1, 2) return \$i + 1
 - The sequence in input is composed of two elements: 1 and 2
 - For each \$i, (\$i + 1) is evaluated
 - The result is (2, 3)





Other examples (2)

- More than one variable can be used in a single expression
for $\$i$ in (10, 20), $\$j$ in (1, 2) return $(\$i + \$j)$
returns (11, 12, 21, 22)

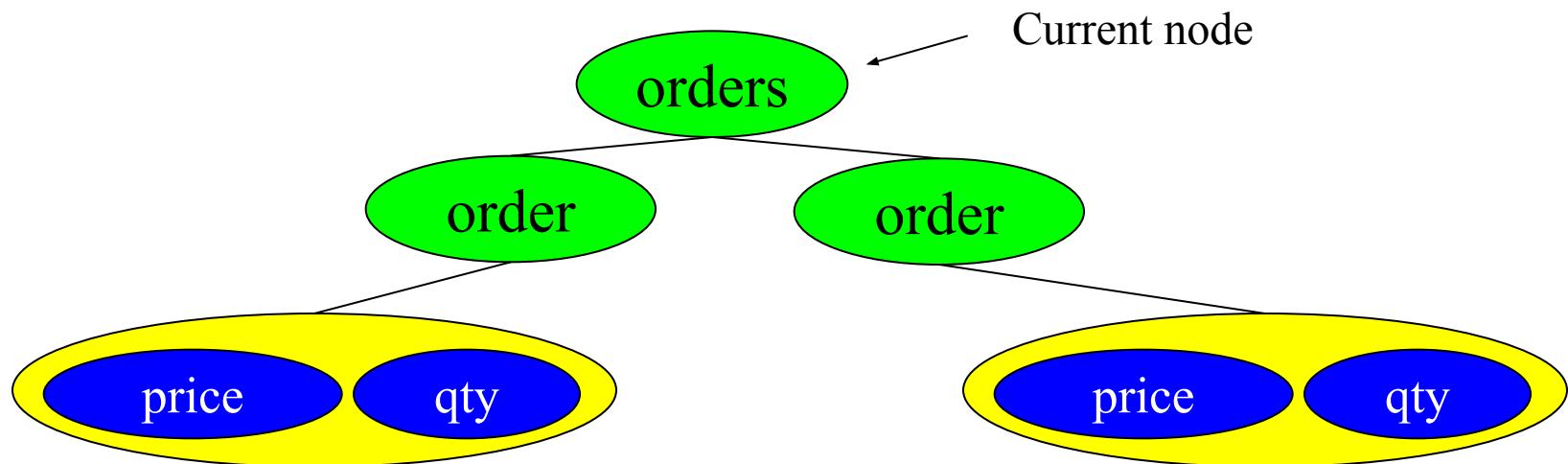
$\$i$	$\$j$
10	1
10	2
20	1
20	2

The list of associations is given by
the cartesian product of all the
possible values of the variables

Other examples (3)

- Compute the total price of orders:

sum(for \$i in order return \$i/@price * \$i/@qty)



Other examples (4)

- For each author, lists its books (distinct-values is equivalent to the distinct clause of SQL):

```
for $a in distinct-values(//author)
```

```
return ($a, for $b in //book[author = $a] return $b/title)
```

```
<book>
  <title>XPath</title>
  <author>John</author>
</book>
<book>
  <title>XQuery</title>
  <author>John</author>
  <author>Matt</author>
</book>
```



```
<author>John</author>
<title>XPath</title>
<title>XQuery</title>
<author>Matt</author>
<title>XQuery</title>
```



Iterations with filter - **where** clause

- Find the books published by Anchor Books after the 1991

```
for $b in doc("example.xml")/bib/book
```

```
  where $b/publisher = "Anchor Books"
```

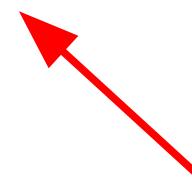
```
    and $b/@year > 1991
```

```
return
```

```
  <book>
```

```
    { $b/title }
```

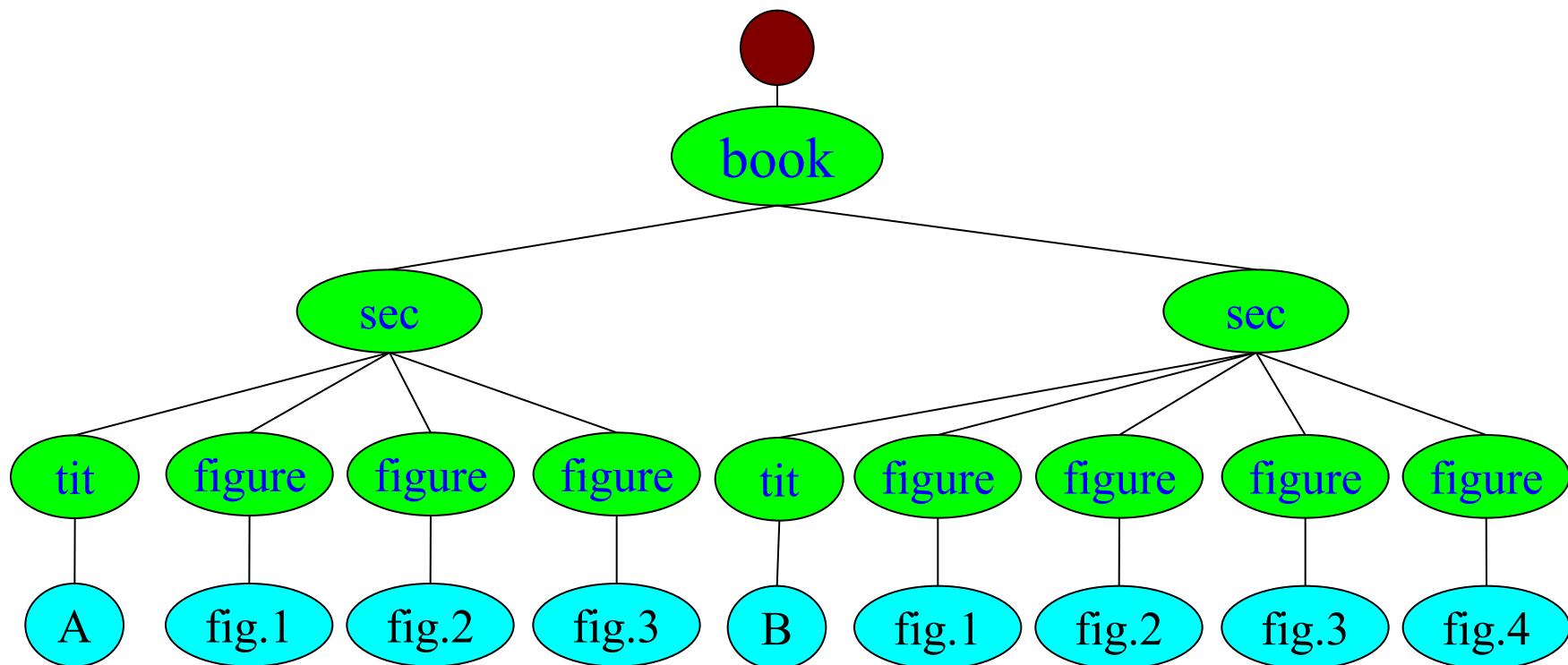
```
  </book>
```



With respect to the previous example, we added a filter

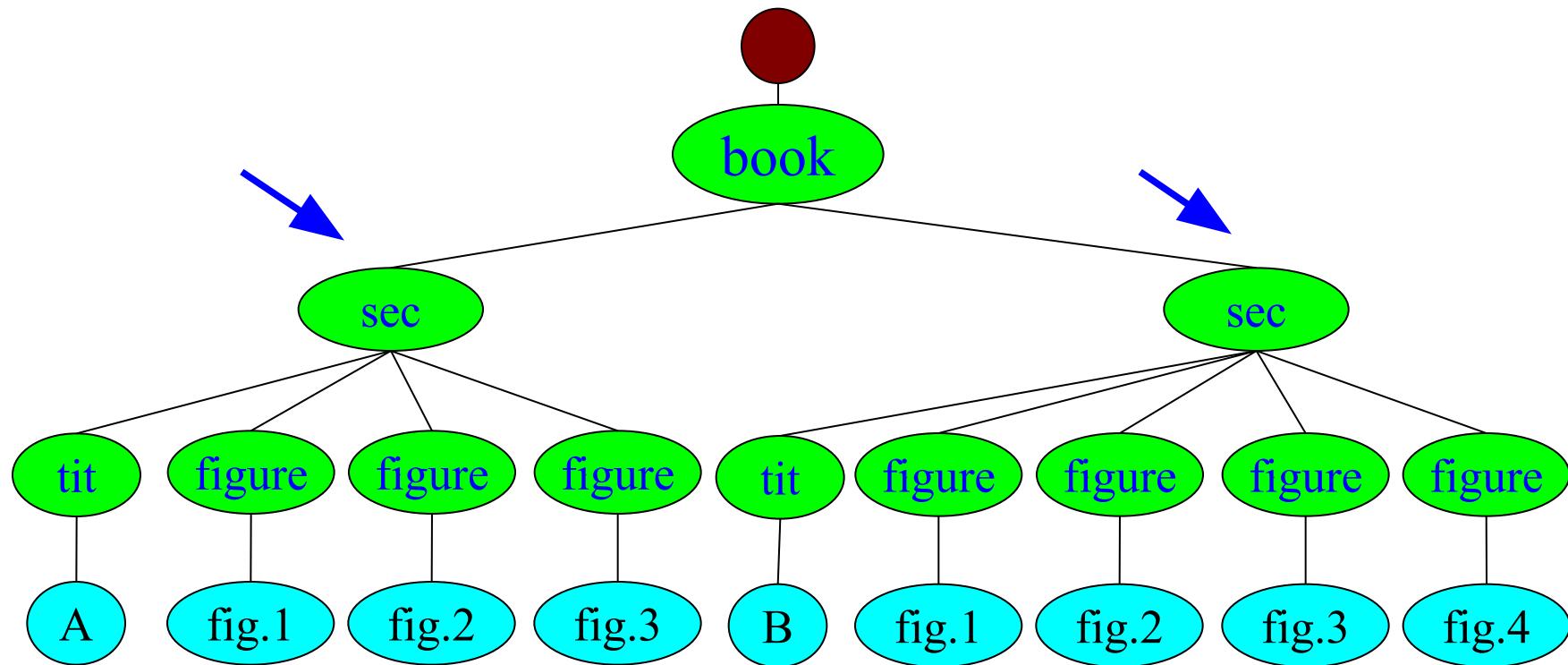
Expressions with aggregated functions – let 1/8

- Sometimes it's necessary to group elements together
- This is needed to count them, or to compute their average value, the minimum or the maximum (if we are dealing with numbers)
 - For example, we want to list for each section the title and the number of figures



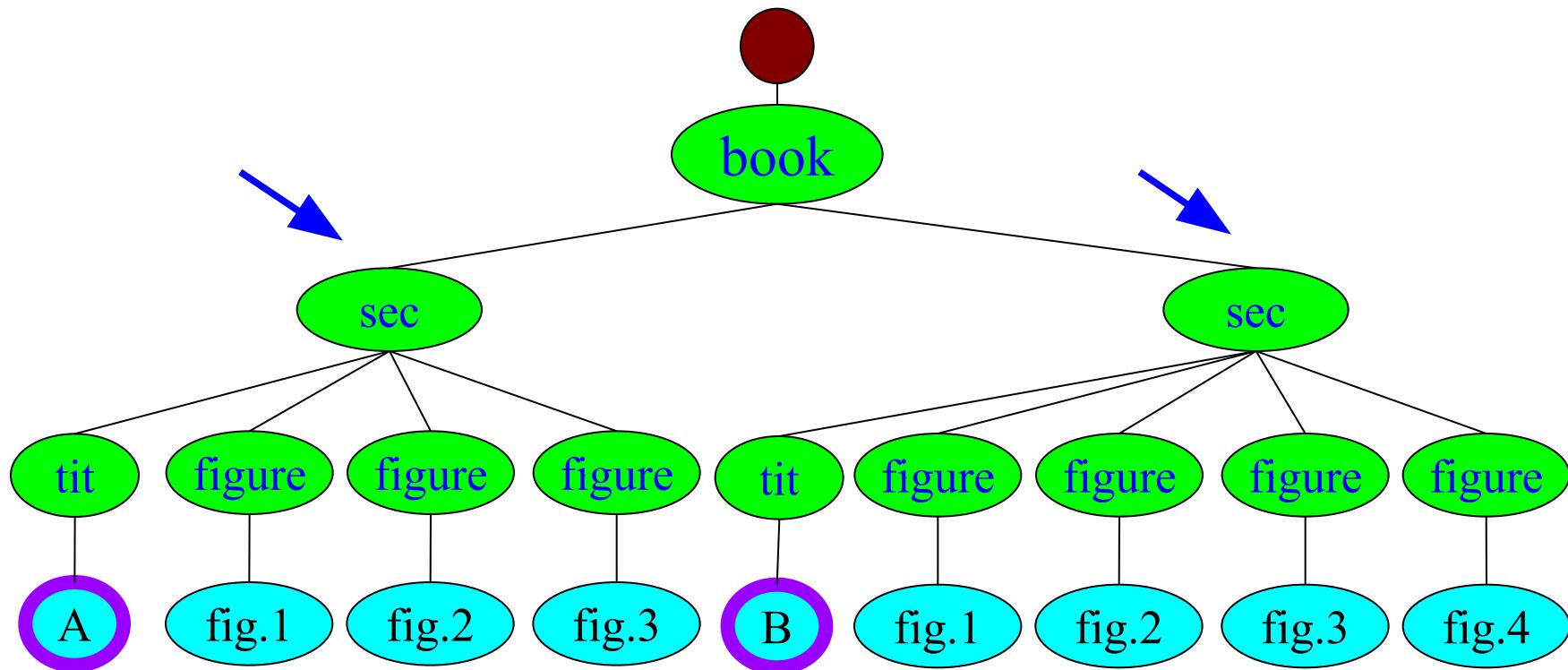
Expressions with aggregated functions – let 2/8

- List, **for each section**, the title and the number of figures



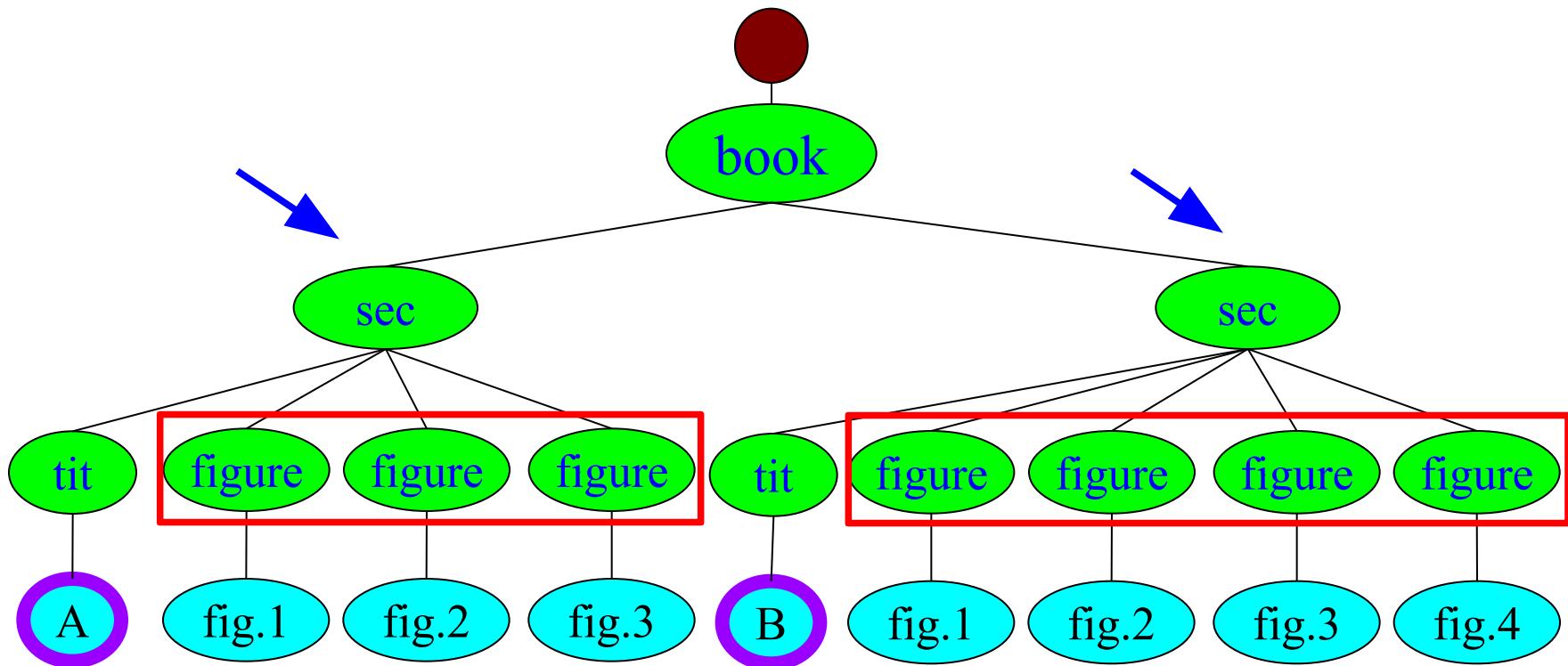
Expressions with aggregated functions – let 3/8

- List, **for each section**, the **title** and the number of figures



Expressions with aggregated functions – let 4/8

- List, **for each section**, the **title** and the **number of figures**





Expressions with aggregated functions – let 5/8

```
<result>
  {
    for $s in ./section
    let $f := $s/figure
    return
      <section title="{ $s/title/text() }"
              numfig="{ fn:count($f) }"/>
  }
</result>
```

// select all the descendants
of the current node

The `text()` function
returns the text value of
the node

fn: it's a namespace that denotes
that `count()` is a standard function
of XQuery

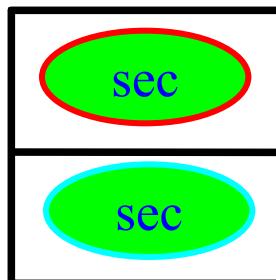
Expressions with aggregated functions – let 6/8

```
<result>
```

```
{  
for $s in ./section  
let $f := $s/figure  
return  
    <section title="{ $s/title/text() }"  
        numfig="{ fn:count($f) }"/>  
}
```

```
</result>
```

\$s

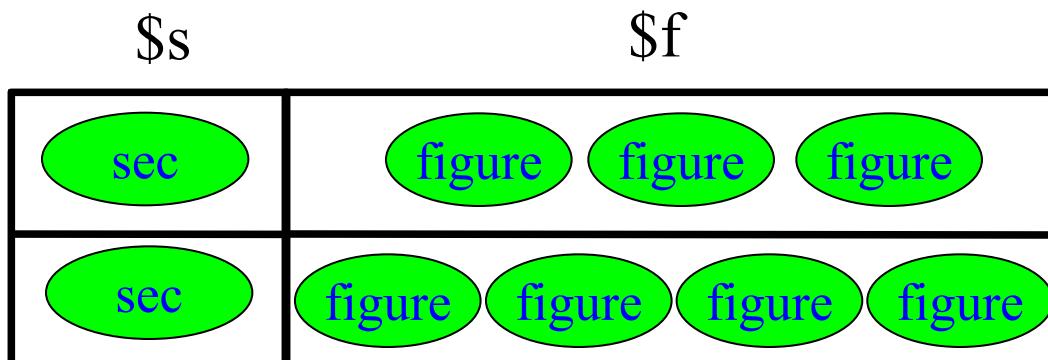


Each node section is associated with \$s

Expressions with aggregated functions – let 7/8

```
<result>
{
  for $s in ./section
    let $f := $s/figure
  return
    <section title="{ $s/title/text() }"
      numfig="{ count($f) }"/>
}
</result>
```

For each section $\$s$, $\$f$ is equal to the **set** of the elements *figure*

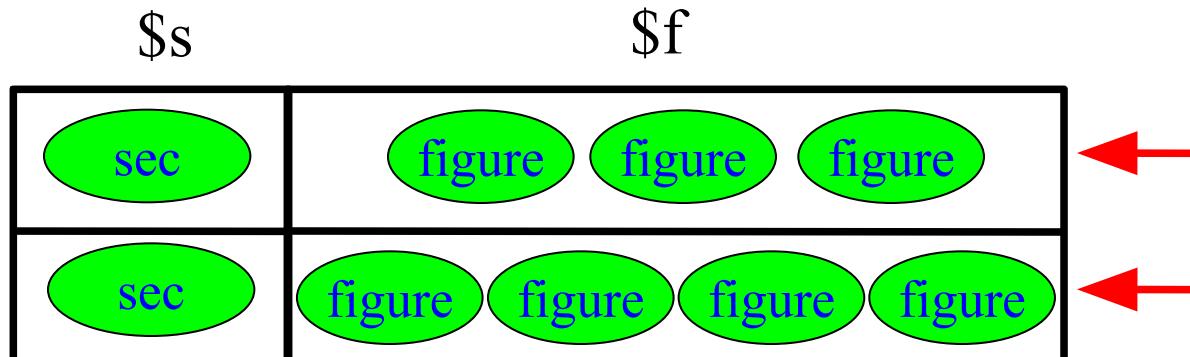


Expressions with aggregated functions – let 8/8

```
<result>
{
  for $s in ./section
  let $f := $s/figure
  return
    <section title="{ $s/title/text() }"
      numfig="{ count($f) }"/>
}
</result>
```

This part of the expression
is evaluated one time for
each possible association

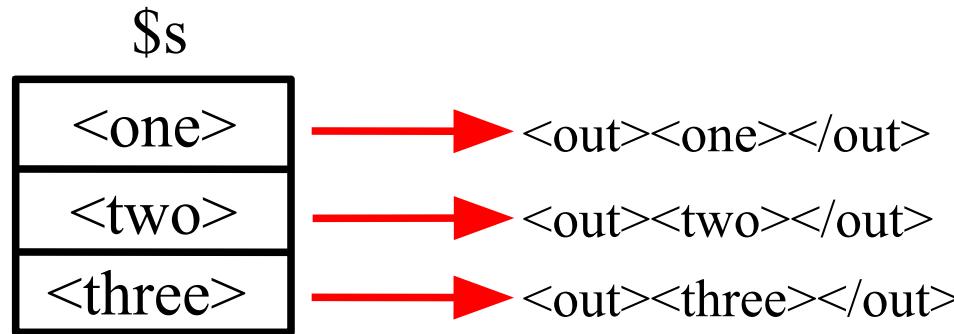
```
<result>
<section title="A" numfig="3"/>
<section title="B" numfig="4"/>
</result>
```



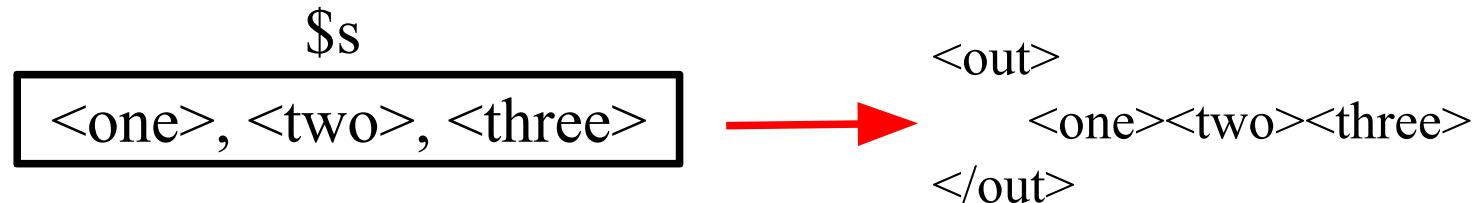


Beware of the difference between FOR and LET

```
for $s in (<one>, <two>, <three>)  
return <out>{$s}</out>
```



```
let $s := (<one>, <two>, <three>)  
return <out>{$s}</out>
```





Join in XQuery (1)

- Let's assume having two XML files, one describes the program of a concert, the other describes the composers
- In both files the composer has a common identifier
- We want to produce a screening schedule, which lists the songs, with the author and the birth and death dates



Join in XQuery (2)

conc.xml

```
<concert>
  <song>
    <title>Ciaccona</title>
    <author>B001</author>
  </song>
  <song>
    <title>Polacca</title>
    <author>C001</author>
  </song>
  <song>
    <title>Notturno</title>
    <author>C001</author>
  </song>
</concert>
```

comp.xml

```
<comp>
  <composer>
    <name>J.S.Bach</name>
    <bio>1685-1750</bio>
    <id>B001</id>
  </composer>
  <composer>
    <name>F.Chopin</name>
    <bio>1810-1849</bio>
    <id>C001</id>
  </composer>
</comp>
```



Join in XQuery (3)

```
<program>{
    for $comp in fn:doc('comp.xml')/comp/composer,
        $song in fn:doc('conc.xml')/concert/song
    where $comp/id eq $song/author
    return
        <song>
            <title>{$song/title}</title>
            <of>{$comp/name}</of>
            <date>{$comp/bio}</date>
        </song>
}</program>
```



Join in XQuery (4)

```
<program>{
    for $comp in fn:doc('comp.xml')/comp
        $song in fn:doc('conc.xml')/conc
    where $comp/id eq $song/author
    return
        <song>
            <title>{$song/title}</title>
            <of>{$comp/name}</of>
            <date>{$comp/bio}</date>
        </song>
}</program>
```



Join in XQuery (5)

```
<program>{
    for $comp in fn:doc('comp.xml')/comp/composer,
        $song in fn:doc('conc.xml')/concert/song
    where $comp/id eq $song/author
    return
        <song>
            <title>{$song/title}</title>
            <of>{$comp/name}</of>
            <date>{$comp/bio}</date>
        </song>
}</program>
```



Result of the query

```
<program>
  <song>
    <title>Ciaccona</title>
    <of>J.S.Bach</of>
    <date>1685-1750</date>
  </song>
  <song>
    <title>Polacca</title>
    <of>F.Chopin</of>
    <date>1810-1849</date>
  </song>
  ...

```



Sorting – **order by** clause 1/4

- As in SQL, XQuery provides a clause to sort the result of a FLWOR expression
- ORDER BY can be specified along with several parameters to alter its semantic
- We will see only the most common use of this clause, through some examples



Sorting – **order by** clause 2/4

- The employees associated with the variable \$employees are returned from the expression, sorted by their salaries

```
for $i in $employees  
order by $i/salary  
return $i/surname
```



Sorting – **order by** clause 3/4

- The employees associated with the variable \$employees are returned from this expression in order of salary, from the higher to the lower

```
for $i in $employees  
order by $i/salary descending  
return $i/surname
```



Sorting – **order by** clause 4/4

- If we want to order the result of a query, which otherwise would be written with a simple navigation expression, we must use a FLWOR expression

```
for $i in $books//book[price < 50]
```

```
order by $i/title
```

```
return $i
```



Other expressions



Conditional expressions

- Imperative programming languages (as the C programming language), provide conditional expression in the **if-then-else** form

- In XQuery, we can also use similar constructs
 - This way it is easy to express queries like:

Add an element <plateNumber>

if the status of a car for sale is “registered”



Conditional expressions: example 1/3

- The evaluation of this expression returns the value of the variable `$productX` that contains the highest price

```
if ($product1/price < $product2/price)
then $product2
else $product1
```



Conditional expressions: example 2/3

- This query verify the existence of an attribute `@discounted`, then choose consequently the elements to be selected

```
if ($product/@discounted)
```

```
then $product/wholesale
```

```
else $product/retail
```



Conditional expressions: example 3/3

- Retrieve the titles of those books with a price value that is greater than 30

```
<result>
{
    if(not(doc("books.xml"))) then (
        <error>
            <message>books.xml does not exist</message>
        </error>
    )
    else (
        for $x in doc("books.xml")/books/book
        where $x/price>30
        return $x/title
    )
}
</result>
```



Comparison operators 1/3

- The following operators act on **sequences** made of several items:
 - `=, !=, <, <=, >, >=`
 - For example,
`$book1/author = "Joyce"`
returns true if **at least one of the selected nodes** author has a text value equal to "Joyce"
- These operators must be used with extreme caution, as shown in the following examples



Comparison operators 2/3

- The fact that `=` and `!=` are true if **at least one** of the elements of the sequences in comparison satisfies the predicate, produces some counterintuitive behavior:
 - These operators are **not transitive**:
 - $(1, 2) = (2, 3)$
 - $(2, 3) = (3, 4)$
 - $(1, 2) \neq (3, 4)$
 - **Both** the following expressions return **true**:
 - $(1, 2) = (2, 3)$
 - $(1, 2) \neq (2, 3)$



Comparison operators 3/3

- XPath 2.0 introduces new operators to compare sequences made of a **single** item:
 - eq, ne, lt, le, gt, ge
- `$book1/author eq "Joyce"` is true only if a **single author node** has been selected
 - Otherwise, an error is reported



Logical and arithmetic operators

- XQuery provides the following **logical operators**, plus a standard function for logical negation
 - `or`, `and`
 - `fn:not()`
- And the following **arithmetic operators**:
 - `+`, `-`, `*`, `div`, `idiv` (integer division), `mod` (remainder of the division)
- The result of a logical expression is either a Boolean value or an error
- **For example**
 - The following expressions return true:
 - `1 eq 1 and 2 eq 2`
 - `1 eq 1 or 2 eq 3`
 - The following expression might return either false or an error:
 - `1 eq 2 and 3 idiv 0 = 1`
 - The following expression might return either true or an error:
 - `1 eq 1 or 3 idiv 0 = 1`



Expressions with quantifiers 1/3

- In XQuery, a variable can be associated with a single value
For instance, \$price can have an integer value of 12000
- However, often the variables are associated to sets of objects
For example, the expression:

`for $lib in doc(books.xml)/books/book`
could associate the variable `$lib` to several `<book>` elements
- Specific operators are needed to **verify the properties of sets of objects**



Expressions with quantifiers 2/3

- Let's introduce two operators of XQuery that serve this purpose, with some examples:
`some $emp in //employee
satisfies ($emp/salary > 13000)`
- This expression is true if **at least one** employee receives a salary greater than 13000
- This is the **existential quantifier**



Expressions with quantifiers 3/3

- The opposite result is obtained using the other quantifier:
`every $emp in //employee
 satisfies ($emp/salary > 13000)`
- This expression is true if **all the** employees receive a salary greater than 13000
- This is the **universal quantifier**



Expressions with quantifiers: examples

some \$x in (1, 2, 3), \$y in (2, 3, 4)

satisfies $$x + $y = 4$

Returns TRUE

every \$x in (1, 2, 3), \$y in (2, 3, 4)

satisfies $$x + $y = 4$

Returns FALSE



Some standard functions



Input functions

- They are needed in order to obtain XML document to be queried
- They are used to access a single document (**doc**) or a sequence of documents (**collection**), provided by the manager system
 - `fn:doc('bib.xml')`
 - `fn:collection('composers')`

<https://www.w3.org/TR/xpath-functions-31/>



Functions on sequences of nodes

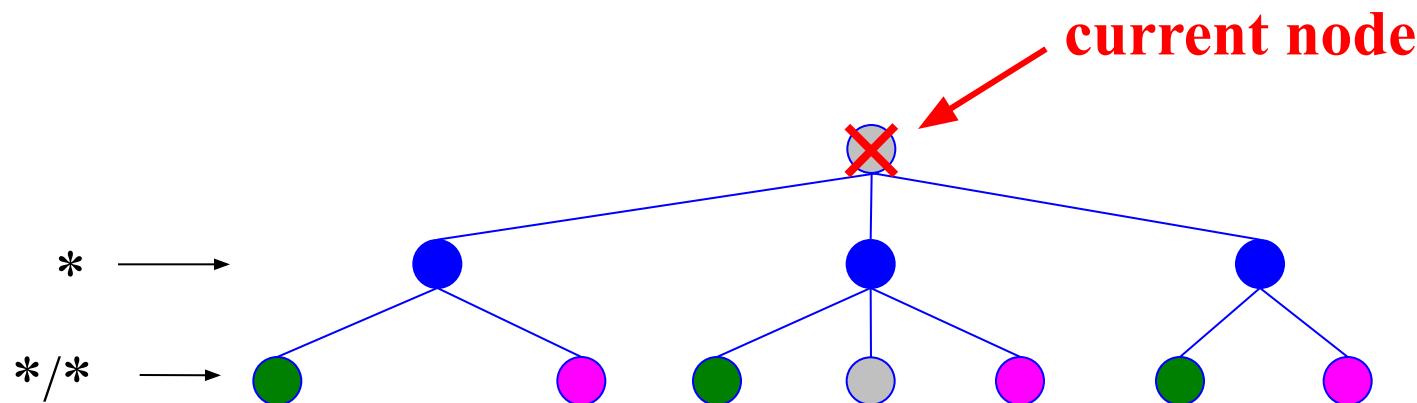
- **fn:position()**
- Position of current node

- **fn:last()**
- Returns an integer representing the number of items in the current context. It is most often used to retrieve the last item

- **fn:count(\$elements)**
- Cardinality of the sequence of nodes in the argument (\$elements)

Example

- Each color shows the nodes selected using the corresponding expression:
 - `fn:count(*)` returns 3
 - `*/*[fn:position()=1]`
 - `*/*[fn:position()=fn:last()]`





Aggregated functions

- The following functions are usually used together with the **let** construct, as we have already seen:
 - `count`
 - `avg`
 - `max`
 - `min`
 - `sum`



Aggregated functions: Examples

- \$seq3 = (3, 4, 5)
 - count(\$seq3) returns 3
 - avg(\$seq3) returns 4
 - max(\$seq3) returns 5
 - min(\$seq3) returns 3
 - sum(\$seq3) returns 12



References

- XQuery specifications:
 - <http://www.w3.org/XML/Query/>
- DBMS supporting XQuery:
 - Oracle database server
 - DB2
 - SQL Server
- List of all the main implementations of XQuery:
 - <http://www.w3.org/XML/Query/#implementations>



Big Data and Data Mining

xQuery in DB2, Oracle and SQL Server

Flavio Bertini

flavio.bertini@unipr.it



Introduction

- We will look at the XQuery standard support provided by most known DBMSs, highlighting any difference between the standard and its implementations
- We will take in consideration the following DBMS:
 - DB2 Express-C (Version 10.1)
 - Oracle XML DB (Version 11.2)
 - MS SQL Server 2012
- We will show some queries executed on these three systems, highlighting the differences. In addition, we will look in particular at the unique features of the three systems, and the techniques that they provide to integrate XML files with relational databases
- Finally we will make some references to Oracle Berkeley XML, a set of libraries dedicated to the management of XML databases



Example documents

- We will show several XQuery queries executed on different DBMSs
- The documents that we are going to query are **Employees.xml** and **Departments.xml**, which represent respectively the employees of a company and the departments where they work
- The two documents are ‘linked’ by the attribute *deptno*, that will be used to carry out eventual join operations



Example document: Departments

Departments.xml

```
<dept>
  <dept deptno="10" dname="Administration"/>
  <dept deptno="20" dname="Marketing"/>
  <dept deptno="30" dname="Purchasing"/>
  <dept deptno="40" dname="Publishing"/>
  <dept deptno="50" dname="Transport"/>
</dept>
```



Example document: Employees

Employees.xml

```
<emps>
    <emp empno="1" deptno="10" ename="John" salary="21000"/>
    <emp empno="2" deptno="10" ename="Jack" salary="310000"/>
    <emp empno="3" deptno="20" ename="Jill" salary="100001"/>
    <emp empno="4" deptno="30" ename="Andrew" salary="50000"/>
    <emp empno="5" deptno="30" ename="Smith" salary="123000"/>
    <emp empno="6" deptno="30" ename="James" salary="34000"/>
    <emp empno="7" deptno="40" ename="Tom" salary="210000"/>
    <emp empno="8" deptno="40" ename="Derek" salary="223000"/>
    <emp empno="9" deptno="50" ename="Alice" salary="120000"/>
    <emp empno="10" deptno="50" ename="Sandra" salary="12000"/>
</emps>
```



XQuery on DB2

DB2, XML and XQuery

- DB2 allows you to insert columns of **XML** type into relational tables, where you can then enter data using normal INSERT commands combined with **XMLEPARSE** function, which takes as input a string and (if possible), converts it into an XML fragment that DBMS can handle
 - Because you can use the XML type, the database must use the **UTF-8** encoding (you can choose this option to its creation)
- XQuery queries are then inserted in the SELECT clause of SQL queries, using the **XMLQUERY** function, and will return XML fragments
- It is also possible to use other functions to combine relational query and XQuery queries: **XMLEXISTS** can be used in the WHERE clause, and **XMLTABLE** is used in the FROM clause



DB2: Path expression

DB2 provides full support for all axes of the path expression, all node types and all tests required by the XPath standard

Examples:

- **/descendant-or-self::book[attribute::year>1992]**
select all the elements 'book' belonging to the axe 'descendant-or-self' that have an attribute 'year' with value greater than 1992

- **/child::bib/child::book/child::text()**
select all the elements of type text children of an element 'book' which in turn is child of an element 'bib'



DB2: Creation of an XML table

To start our series of examples, we have to **create the tables** able to contain XML data. To do this, we simply specify **XML** as a data type of a column. In our case, we want two tables, each of which will contain one of the documents seen previously

Database examples

```
CREATE TABLE employees (data XML)
CREATE TABLE departments (data XML)
```



DB2: Inserting XML data in a table

In order to insert data in the tables we just built, we will use **the function XMLPARSE** within a normal INSERT clause, passing a string representing the XML document we want to insert in the table. It is possible to choose whether or not to keep the whitespace

Insert example

```
INSERT INTO departments(data) VALUES
(XMLPARSE
(document cast
 ('<depts>
  <dept deptno="10" dname="Administration"/>
  ....
  <dept deptno="50" dname="Transport"/>
 </depts>' as clob)
preserve whitespace)
)
```



DB2: XQuery in SELECT

Now that we've entered the data in the tables, we begin to make some queries. In this first example, we want to extract **the names of all the departments** and insert them in **XML elements <deptName>**. We use the function XMLQUERY to enter the XQuery query in an SQL expression

XQuery 1

```
SELECT XMLQUERY
  ('for $d in $list//dept
   return <deptName>{ $d/@dname }</deptName>'
    passing dtable.data as "list")
  FROM departments dtable
```

The **passing** clause is used to specify which XML fragment we are working on. In this case we **pass the content of the departments table as "list"**: in this way **a variable \$list will be created** that we can use within the query to refer to the XML fragment that we have previously inserted into the table



DB2: XQuery with **where** and **order by** in SELECT

Let's complicate the query a little bit: now we want to extract the names and the salaries of all employees with a salary greater than 50.000. In addition, we want the results to be inserted into <empSalary> elements and sorted by salary

XQuery 2

```
SELECT XMLQUERY
  ('for $e in $list//emp
   where $e/@salary > 50000
   order by $e/@salary
   return <empSalary>{$e/@ename} {$e/@salary}</empSalary>'
   passing etable.data as "list")
FROM employees etable
```



DB2: XQuery with **let** and aggregation operators in SELECT

We want to extract for each employee his name, his salary and **the average salary of the department where he works**

XQuery 3

```
SELECT XMLQUERY
  ('for $e in $list//emp
   let $avgsal:=avg($list//emp[@deptno=$e/@deptno]/@salary)
   return <averages>{$e/@ename} {$e/@salary} {$avgsal}</averages>'
   passing etable.data as "list")
FROM employees etable
```

The **let** clause will be executed at each for cycle, computing for each **<emp>** the average of **@salary** value of all the **<emp>** elements with the same **@deptno**



DB2: Conditional operators

In this example we want to create a <salaries> element that is empty if the employee John earn more than the employee Smith, and that contains the salary of Smith otherwise

XQuery 4

```
SELECT XMLQUERY
  ('let $john := $list//emp[@ename="John"]
   let $smith := $list//emp[@ename="Smith"]
   return
     if ($john/@salary > $smith/@salary)
     then <salaries/>
     else <salaries>{$smith/@salary}</salaries>'
  passing etable.data as "list")
FROM employees etable
```



DB2: Sets operators

In this example, we want to extract the names of all the employees who work in the department with @deptno 30, or of those who earn less than 100.000

XQuery 5

```
SELECT XMLQUERY
  ('let $emps30 := $list//emp[@deptno = 30]
   let $empspoor := $list//emp[@salary < 100000]
   for $e in ($emps30 union $empspoor)
   return <empName>{$e/@ename}</empName>'
  passing etable.data as "list")
FROM employees etable
```

In this case, the **let** clause is outside the **for** clause, therefore it's computed just one time



DB2: Quantifiers

In this example, we want to extract the names and the salaries of all the employees, but only if there is at least one employee who has a salary greater than 100000

XQuery 6

```
SELECT XMLQUERY
  ('if
    (some $emp in $list//emp
      satisfies ($emp/@salary > 100000))
  then
    <results>
      {for $e in $list//emp
        return <emp>{$e/@ename} {$e/@salary}</emp>}
    </results>
  else
    <results/>'
  passing etable.data as "list")
FROM employees etable
```



DB2: Join

In this example, we want to select the name of all the employees
who work in the department named "Purchasing"

XQuery 7

```
SELECT XMLQUERY
  ('let $purchasingdep := $dlist//dept[@dname = "Purchasing"]
   for $e in $elist//emp
   where $e/@deptno = $purchasingdep/@deptno
   return <empName>{$e/@ename}</empName>'
   passing etable.data as "elist", dtable.data as "dlist")
FROM employees etable, departments dtable
```

In order for the join to work, we use the passing clause to pass both
the **employees content** and **departments content**



DB2: Join, let clause and aggregation operators

Now we want to select, **for each department**, its name and **the sum of the salaries of the employees who works in it**

XQuery 8

```
SELECT XMLQUERY
  ('for $d in $dlist//dept
   let $sumsalary:=sum($elist//emp[@deptno=$d/@deptno]/@salary)
   return <deptCost>{$d/@dname} {$sumsalary}</deptCost>'
  passing etable.data as "elist", dtable.data as "dlist")
FROM employees etable, departments dtable
```



DB2: Last example of XQuery in SELECT

We want to select for each department **the number of its employees** and **their average salary**. We want the results to be **sorted by the total cost (the sum of the salaries) of each department**, and that **only the departments with more than one employee must be included in the answer**

XQuery 9

```
SELECT XMLQUERY
  ('for $d in $dlist//dept
   let $emps := $elist//emp[@deptno = $d/@deptno]
   where count($emps) > 1
   order by sum($emps/@salary) descending
   return
     <big-dept>
       {$d/@dname}
       <headcount> {count($emps)}</headcount>
       <avgsal>{avg($emps/@salary)}</avgsal>
     </big-dept>'
   passing etable.data as "elist", dtable.data as "dlist")
FROM employees etable, departments dtable
```



XQuery in Oracle DB



Oracle DB and XQuery

- Oracle DB provide full support to XQuery, in a way very similar to DB2: as we will see in the examples, most functions are the same
- Oracle XML DB has been designed with particular focus on the coexistence between XQuery and SQL

Oracle DB: Unsupported features

Even if Oracle DB implements all the main features of XQuery, some features of the standard are not supported:

- **version encoding**: it is not possible to specify the encoding used inside an XQuery expression
- **xml:id**: if this feature is used, an error is thrown
- **xs:duration**: this type of data is not supported; it is possible to use xs:yearMonthDuration or xs:DayTimeDuration as an alternative

The XQuery standard specifies that some features are optional for a given implementation. The following optional XQuery features are not supported by Oracle XML DB:

- **schema validation feature** (optional in the standard)
- **module feature** (optional in the standard)



Oracle DB e XPath 2.0

OracleDB provides full support to the functions and operators included in XPath 2.0 (as the operations between sets and quantifiers we've already seen for DB2), with some exceptions:

- You can not use the standard functions for the definition of **regular expressions**, but you have to use those **built-in** inside the **DBMS**
- The functions **fn:id** and **fn:idref** are not supported
- The function **fn:collection** without arguments is not supported



Oracle DB: Creation of an XML table

As in DB2, in Oracle DB we can **enter columns of XML type in relational tables**, using the keyword **XMLType**. Moreover, it is possible to **create tables which are of XML type themselves**

Examples:

Creation of a relational table with an XML column:
CREATE TABLE departments

(id NUMBER(4), data XMLType)

Creation of an XML table:

CREATE TABLE employees OF XMLType

In the following examples we will assume that both *employees* and *departments* are tables of XML type

Oracle DB: Implementations of XMLType

The XMLType is an abstract construct, which can be implemented by the DBMS in two ways:

- Inside a **LOB (Large OBject)**, that is, like a simple string
- In a **Structured Storage**: choosing this option, the DBMS will build automatically tables and views that follows the XML schema of the document, and it will use them to contain the single nodes that the document contains, keeping the compliance to its *DOM (Document Object Model)*

- The choice of the implementation has no consequence on the methods the user will use to handle the *XMLTypes*; it has however consequences on the **performance of the queries**.
- It is not possible to determine which storage method is “superior”: in order to obtain the best performance you need to try case by case



Oracle DB: Entering XML data in a table

In order to enter data inside an XML table or column, we use the function **XMLType** within an INSERT command, passing **the XML fragment we want to insert**

Example:

```
INSERT INTO departments VALUES
(XMLType
( '
```



Oracle DB: Example of query and differences with DB2

We show now a query we already ran in DB2: the differences are really minimal. In this example, we want to extract **the name and the salary** of all the employees with a salary greater than 50000, ordered from the less paid to the most payed

Example of XQuery

```
SELECT XMLQUERY
  ('for $e in //emp
   where $e/@salary > 50000
   order by $e/@salary
   return <empSalary>{ $e/@ename } {$e/@salary}</empSalary>'
   passing etable.object_value
   returning content)
  FROM employees etable
```

Apart from some differences in the **passing** clause and the adding of the clause **returning content**, the query system is basically the same of DB2



Oracle DB: Example of join

We show a query with a join between two XML tables: even in this case, the syntax is extremely similar to the one used in DB2. The aim of the query is to extract all the employees who work in the department named “Purchasing”

Example of XQuery with join

```
SELECT XMLQUERY
  ('for $e in $elist//emp
   let $d := $dlist//dept[@dname = "Purchasing"]
   where $e/@deptno = $d/@deptno
   return $e'
   passing etable.object_value as "elist",
         dtable.object_value as "dlist"
   returning content)
FROM employees etable, departments dtable
```



Oracle DB: working directly on XML documents

If we don't want to enter data in the tables, it's possible to use the **doc() function** to launch queries directly on the XML files.

Let's see an example of the same query we've just seen, but this time applied to the document [employees.xml](#)

```
SELECT XMLQUERY
  ('for $e in doc("employees.xml") //emp
   where $e/@salary > 50000
   order by $e/@salary
   return <empSalary>{ $e/@ename } { $e/@salary}</empSalary>' 
   returning content)
FROM DUAL
```

The **DUAL** table is present by default in all Oracle databases, and it's empty: it only serves to compensate for the fact that the **FROM** clause is mandatory in all SQL query, even in the case in which the data are taken by other means (such as, in this case, the function **doc()**)



XQuery in SQL Server 2012



SQL Server: Supported XQuery version

- Microsoft SQL Server supports only a subset of the XQuery features, being based on the 2004 Working Draft of the standard
- We will see now which features are not implemented in this DBMS



SQL Server: Sequences and Path Expressions

For the **Sequences**, the following limitations are to be reported:

- The sequences must be homogeneous, so they can only be composed of **nodes** or **atomic values**
- The construct to build the sequences **to** is not supported
- You can not use the operators **union**, **intersect**, **except** for combining sequences formed by nodes

The **Path Expressions** are partially supported:

- It is possible to use the axes **child**, **descendant**, **parent**, **attribute**, **self**, **descendant-or-self**
- Only the *node types* **comment()**, **node()**, **processing-instruction()**, **text()** are supported



SQL Server: FLWOR expressions

The **FLWOR** expressions are completely supported

- It is important to note that in SQL Server 2019 the expressions assigned to a variable through the LET clause will be inserted in the query **every time that the variable has been mentioned in it**

- This means that the expression will be not executed one time only, instead **it will be recomputed each time that there is a reference to that variable**



SQL Server: other features

Other supported operators are:

- Conditional constructs **if-then-else**
- Comparison operators
- Logical operators (only **and** and **or**: **not** is not available as a function)
- All the operators **with the exception of idiv**
- All the **aggregation functions**
- Expressions with existential and universal quantifiers
(some/satisfies and every/satisfies)



SQL Server: Creation of an XML table

As for DB2, in order to handle XML data we first need to create a table to contain it. To do so, we just need to insert an XML column in it

Example of Tables

```
CREATE TABLE employees (
    col1 int primary key,
    data XML)
```

```
CREATE TABLE departments (
    col1 int primary key,
    data XML)
```



SQL Server: Inserting XML data in a table

In order to enter data in the table we just created, it is sufficient to enter a string containing the XML fragment in the INSERT operation

Example of INSERT

```
INSERT INTO departments VALUES  
(1,  
 ('<depts>  
  <dept deptno="10" dname="Administration"/>  
  <dept deptno="20" dname="Marketing"/>  
  <dept deptno="30" dname="Purchasing"/>  
  <dept deptno="40" dname="Publishing"/>  
  <dept deptno="50" dname="Transport"/>  
 </depts>'))
```



SQL Server: differences from DB2 and Oracle DB

We show the same query we had done for DB2 and Oracle DB: we want to extract the name and salary of employees earning more than 50,000, ranked by salary

Example of XQuery:

```
SELECT data.query
  ('for $e in //emp
   where $e/@salary > 50000
   order by $e/@salary
   return <employee>{ $e/@ename } {$e/@salary}</employee>')
AS result
FROM employees
```

Apart from a few variations in passing the data (instead of using a passing clause, apply directly **the query () function to the XML column "data"** contained **in the employees table**), the syntax is the same as always



XQuery in Oracle Berkeley DB XML



Oracle Berkeley: a set of libraries

- Oracle Berkeley it's not a real DBMS, but more of a set of libraries that can be used from within most common programming languages, to handle XML databases with simplicity
- Oracle Berkeley does not provide any support for relational databases
- It is possible to operate also from a text terminal interface which directly evaluate the commands (as in the examples we will see in the following slides)



Oracle Berkeley: Main operations

- The most fundamental object, in Oracle Berkeley, is the **container** (roughly comparable to the tables of relational DBMS)
- Once a container is created, it is possible to **insert** XML documents in it
- It is possible to **recover** XML fragments from a container by executing an **XQuery query**
- It is possible to use XML Schema to insert some **constraints** on the documents
- It is also possible to create indices that increase the performances



Oracle Berkeley: Creating a container

We can choose between two different ways of storing the data:

- **by document**: the documents are stored exactly as they have been given to the system
- **by nodes**: the documents are decomposed in the nodes they are made of, which then are stored separately

Storing by nodes allows to modify the document in the future, and also grant better performances

Example (through the console):

```
dbxml> createContainer example.dbxml
```



Oracle Berkeley: Inserting a document

To insert documents in the current container, we use the operation
“`putDocument <namePrefix> <string> [f|s|q]`”

- Putting “f” as third parameter denotes that **the string contains the path of the XML file** we want to insert
- Putting “s” as third parameter denotes that **the string contains the XML fragment** we want to pass

Examples (through the console):

```
dbxml> putDocument book1
'<book><title>Title1</title><author>Author1</aut
hor><pages>100</pages></book>' s
dbxml> putDocument book1 'Document.xml' f
```



Oracle Berkeley: query

In order to retrieve documents inserted in a container, XQuery queries are used. Oracle Berkley will run a search on all the documents contained in the container

Examples (through the console):

Retrieve the whole content of the collection:

```
dbxml> query 'collection("esempio.dbxml")'
```

Retrieve the title of all the books that have John as author:

```
dbxml> query  
'collection("esempio.dbxml") /book[author="John"] /title'
```

Retrieve the title of all the books with a price greater than 100:

```
dbxml> query  
'for $book in collection("esempio.dbxml") /book  
where $book/price > 100  
return $book/title'
```



Oracle Berkeley: constraints

- During the creation of a container, it is possible to bind a schema to an XML document; this way, only XML documents which satisfies the schema validation will be allowed
- It is also possible to specify different constraints for different documents, within the same container



Appendix: Install DBMS



Install DB2

- From the website <http://www-01.ibm.com/software/data/db2/> it is possible to download the software needed to execute DB2
- The free version of the DBMS is called "**DB2 Express-C**"
- Through the following link
<http://www.ibm.com/developerworks/downloads/im/data/> it is possible to download **IBM Data Studio**, a free graphic interface extremely useful to execute any kind of operations



Install Oracle XML

- From the website www.oracle.com, in the Downloads section, it is possible to freely download **Oracle Database 11g**, which also contains Oracle XML

- In this case too a graphic user interface is needed to easily interact with the DBMS: this is called **Oracle SQL Developer**, and this is also freely available in the Downloads section of www.oracle.com



Install Microsoft SQL Server 2012

- From the website <http://www.microsoft.com/sqlserver> it is possible to download **MS SQL Server 2012 Express**, a free version of the DBMS

- By selecting the download package **Express with Tools**, it will also install a graphic user interface named **SQL Server Management Studio Express**



Big Data and Data Mining

NoSQL

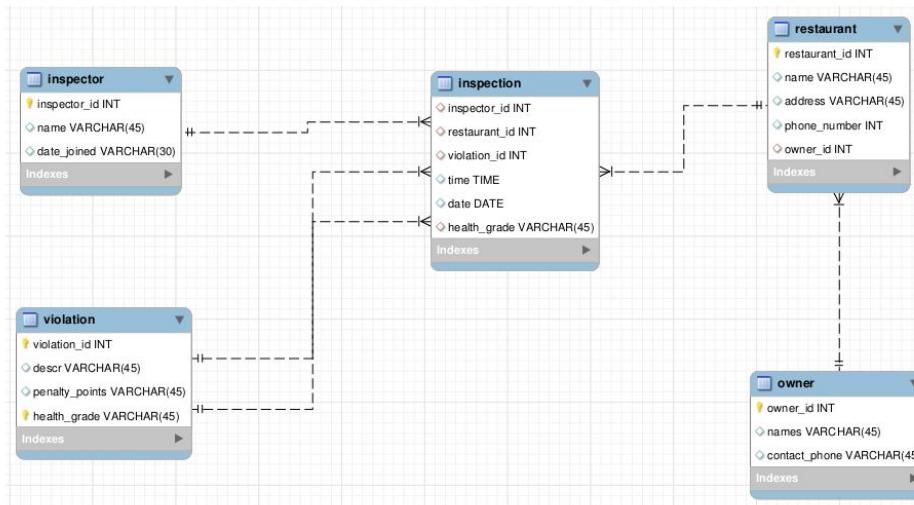
Flavio Bertini

flavio.bertini@unipr.it



Relational DBMS: Properties

- **Strong foundation:** Relational Model
- **Highly Structured:** rows, columns, data types
- **Structured Query Language:** standardized
- **ACID properties:** all or nothing
- **Joins:** new views from relationships

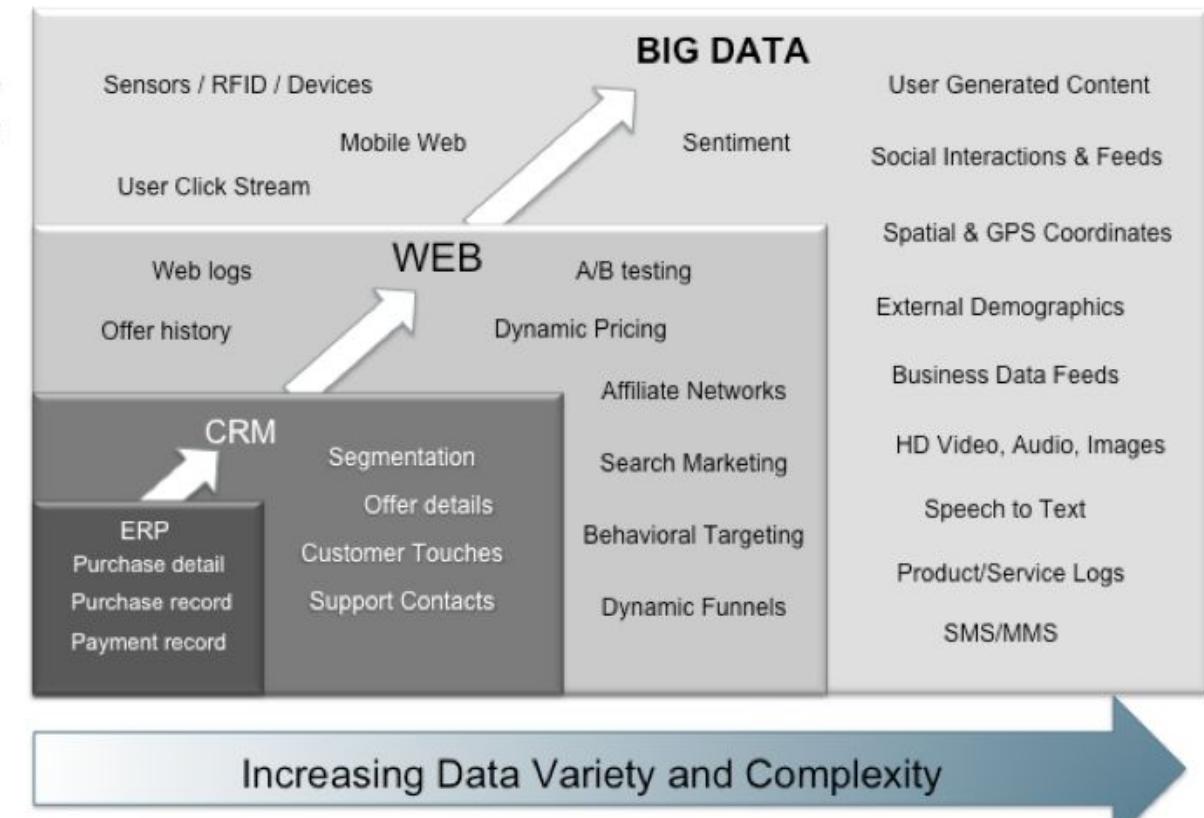
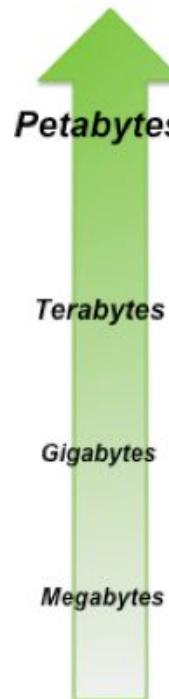


A C I D
ATOMICITY CONSISTENCY ISOLATION DURABILITY

Changes in scenario: Big Data

- The five (six) V(s) of Big Data:

- Volume
- Velocity
- Variety
- Variability
- Veracity
- (■ Value)



- Needs for:

- Flexible schema for variability and veracity **or no schema at all!**
- Distributed data for volume
- High availability for velocity



A real example

- A vehicle at the **lower end of the autonomous spectrum** will produce about 3 Gbit/s of data, which amounts to about **1.4 terabytes every hour**
- At **higher levels of autonomy**, the total sensor bandwidth will be closer to 40 Gbit/s and approximately **19 terabytes per hour**
- Over a whole year, a vehicle could generate **434 TB for lower level autonomy** or up to **5,894 TB of data for higher-level autonomy**

[Siemens, The Data Deluge: What do we do with the data generated by AVs?](#)

Relational DBMS: Weakness

- **Joins:** not scalable
- **Transactions:** read & write operations will be slow because of locking resources
- **Fixed definitions (schema):** difficult to work with highly variable data
- **Document integration:** difficult to create reports based on structured & unstructured data





NoSQL: Why, What and When

- In 2004 Google and Amazon built their own **non-relational** (do not feature primary / foreign keys, JOINs, or relational calculus of any type) databases designed to scale to **petabyte** of data across **thousands of machines** (Google BigTable and Amazon Dynamo)
- In 2008, Facebook releases its own non-relational database, with a design similar to Google BigTable (Cloud NoSQL database service)
- **Other (very) important reasons:** SQL licenses costs for hundreds of thousands machines!
- **#NoSQL was a twitter hashtag** for a conference in 2009
- The name refers to "non SQL", "non relational" or "not only SQL", but it does not indicate its characteristics
- There is **no strict definition** for NoSQL



NoSQL DBMSs

- There are currently more than 225 NoSQL databases systems!

[source:
<https://hostingdata.co.uk/nosql-database/>]



- We will see now the main differences from relational DBMSs (one or both the following):
 - NoSQL systems are **BASE**: they don't follow ACID properties
 - NoSQL systems are **schema-less**: they have a non-relational data model (e.g., key-value, document, graph)



**NoSQL systems are
BASE**



Recall ACID

Atomicity

Each transaction is “all or nothing”

Consistency

Data should be valid according to all defined rules

Isolation

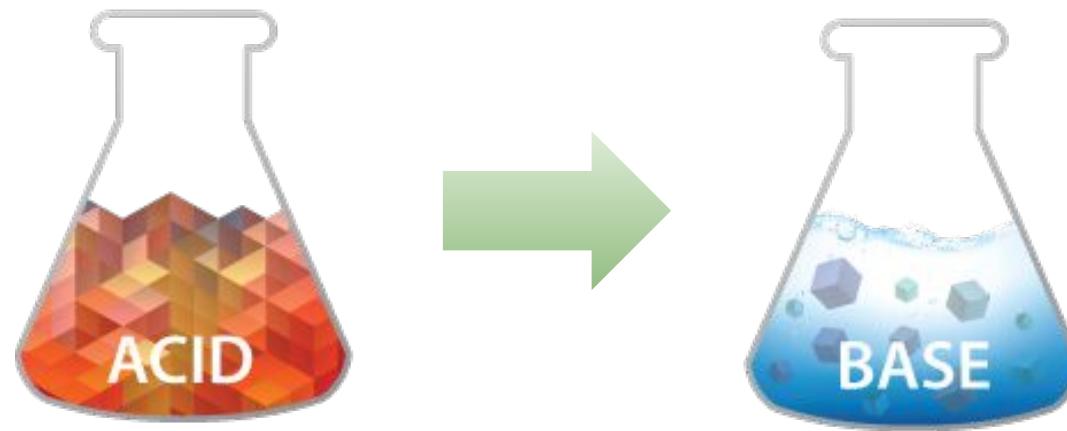
Transactions do not affect each other

Durability

Committed data would not be lost, even after power failure.



SQL (ACID properties) vs NoSQL (BASE properties)



- **Basic Availability:** there will be a response to any request (can be failure too)
- **Soft State:** the state of the system could change over time
- **Eventual Consistency:** may be inconsistent in short term, consistent in long term
 - **It's OK to use stale data**
 - **it's OK to give approximate answers**



Basically Available

■ Basically Available

- The system does guarantee the availability of the data as regards [CAP Theorem](#) (a.k.a. Brewer's theorem)
- There will be a response to any request but:
 - That response could still be 'failure' to obtain the requested data
 - The data may be in an inconsistent or changing state



Soft state

■ **Soft state**

- The state of the system could change over time
- Even during times without input there may be changes going on due to 'eventual' consistency
- The state of the system is always 'soft'



Eventual consistency

■ Eventual consistency

- The system will *eventually* become consistent once it stops receiving input
- The data will propagate to everywhere it should sooner or later
- The system will continue to receive input in the meantime
- The system does not check the consistency of every transaction before it moves onto the next one



ACID vs BASE

■ ACID

- Strong consistency
- Less availability
- Pessimistic concurrency
- Complex

■ BASE

- Availability is the most important thing! Willing to sacrifice other properties for this (like consistency)
 - Weaker consistency (Eventual)
 - Best effort
 - Simple and fast
 - Optimistic
-
- Why can't we have both together?
 - A tradeoff exists between consistency, availability, and partition tolerance → **CAP Theorem**



Another view: CAP trade off

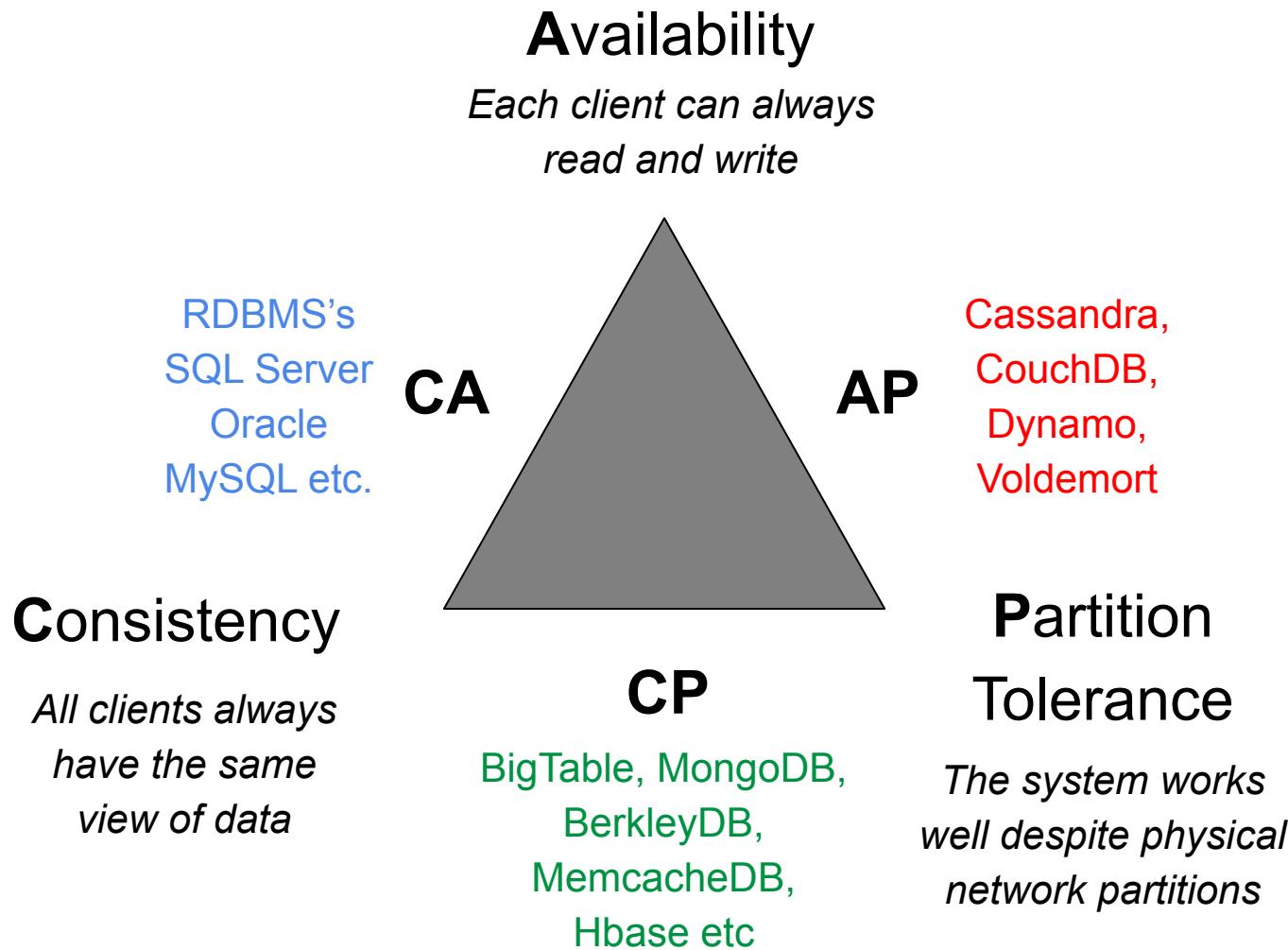
- **Consistency** refers to whether a system operates fully or not. Do all nodes within a cluster see all the data they are supposed to? This is the same idea presented in ACID
- **Availability** means just as it sounds. Is the given service or system available when requested? Does each request get a response outside of failure or success?
- **Partition Tolerance** represents the fact that a given system continues to operate even under circumstances of data loss or system failure. A single node failure should not cause the entire system to collapse

- In large scale, distributed, non relational systems, they need availability and partition tolerance, so consistency suffers and ACID collapses



CAP Theorem (or triangle)

Only **two properties out of three** can be satisfied in the same data model! **Pick any two**





**NoSQL systems are
schema-less**

Relational vs Schema-less 1/2

- In relational Databases:
 - You can't add a record which does not fit the schema
 - You need to add NULLs to unused items in a row
 - We should consider the data types, i.e. you can't add a string to an integer field
 - You can't add multiple items in a field (you have to create another table: primary-key, foreign key, joins, normalization, ... !!!)

```
create table customers (id int, firstname text, lastname text)
```

```
insert into customers (firstname, middlename, lastname) values (...)
```



Relational vs Schema-less 2/2

- In NoSQL Databases:
 - There is no schema to consider
 - There is no unused cell
 - There is no datatype (implicit)
 - Most of considerations are done at the *application layer*
 - We gather all items in an aggregate (document)

Relational Model



Document Model

Collection ("Things")



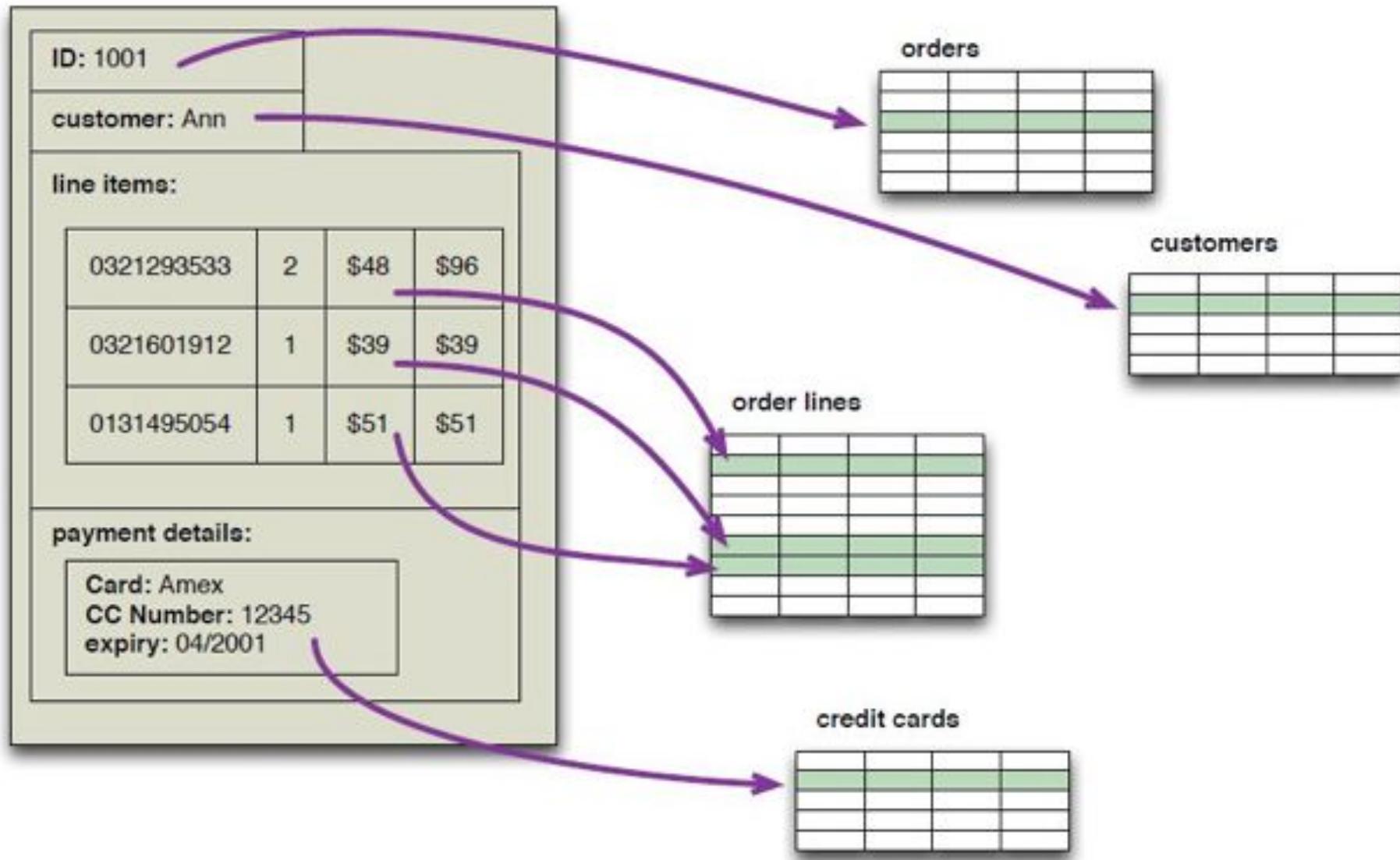


Aggregation

- The term comes from Domain-Driven Design
- An aggregate is a cluster of domain objects that can be treated as a single unit
 - For example, a web document is an aggregate with a title, a body, an image inside the body with its caption etc.
- Aggregates are the basic element of transfer of data storage: you request to load or save whole aggregates
- Transactions should not cross aggregate boundaries
- This mechanism reduces the join operations to a minimal level
 - Related things are already together



Aggregated vs Relational





Different types of NoSQL (data models)

 mongoDB

 CouchDB
relax

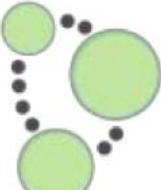
 RAVEN DB

DOCUMENT

APACHE
 HBASE

 Cassandra

COLUMN

 Neo4j

GRAPH

KEY-VALUE

ORACLE
NoSQL

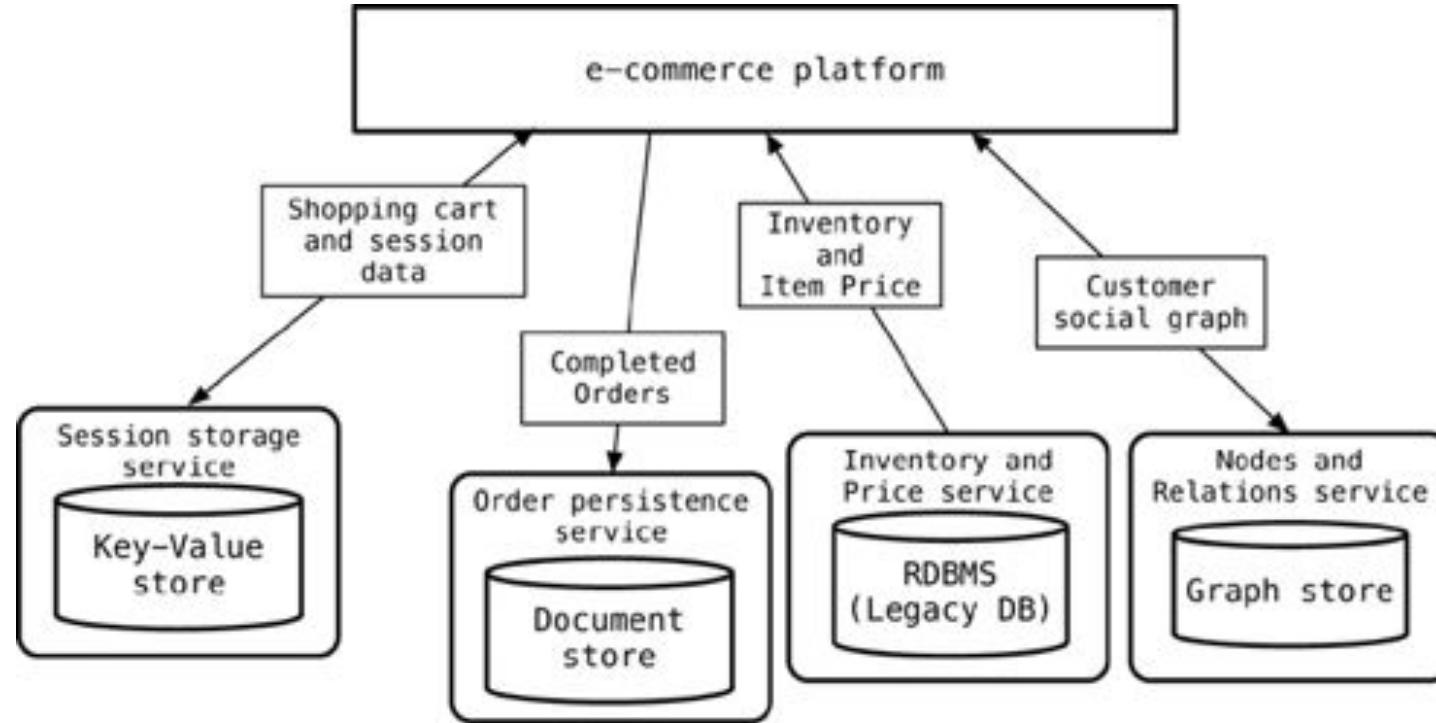
Project Voldemort
A distributed database.

 basho

 redis

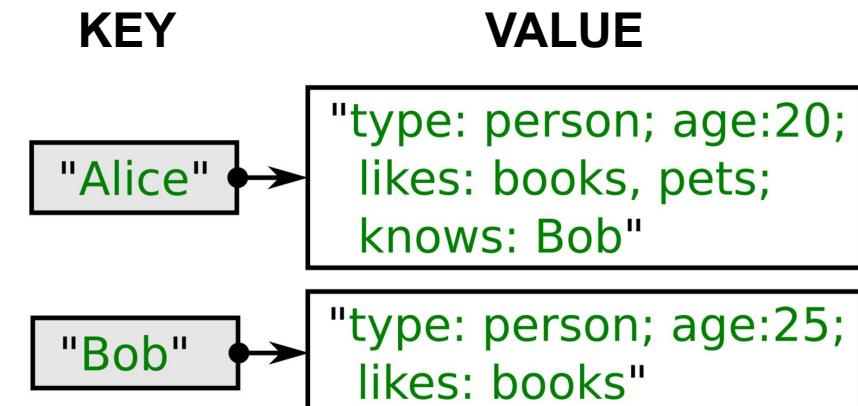
Why many models?

- Different data models are designed to solve different problems
 - Different kind of data
 - Different needs of access (temporary, very fast, historical)
- Using single database engine for all the requirements leads to non-performant solutions
- The solution is **polyglot persistence**: a hybrid approach to data persistence



Key-value data model

- **Key-value:** A very simple structure. Sets of named keys and their value(s), typically an uninterpreted chunk of data
 - Sometimes that simple value may in fact be a JSON or binary document
- Can be in memory only, or be backed by disk persistence
- Designed to handle massive data loads
- Supports versioning
- Examples:
 - Voldemort (LinkedIn)
 - Amazon SimpleDB
 - Redis
 - Memcache
 - BerkleyDB
 - Oracle NoSQL





Key-value main idea

- Data model: (key, value) pairs
- Access data (values) by strings called keys
- The main idea is the use of a hash table
- Data has no required format – data may have any format
- Basic Operations:
 - *Insert(key,value)*
 - *Fetch(key)*
 - *Update(key,value)*
 - *Delete(key)*

Car	
Key	Attributes
1	Make: Nissan Model: Pathfinder Color: Green Year: 2003
2	Make: Nissan Model: Pathfinder Color: Blue Color: Green Year: 2005 Transmission: Auto

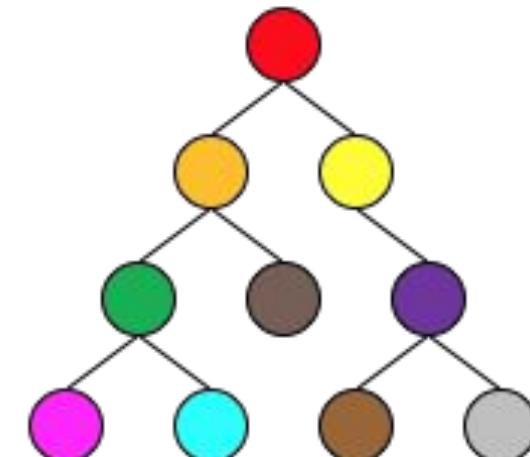
Key-value store in practice

- “Value” is stored as a “blob”
 - Without caring or knowing what is inside, or how long it is
 - **Application is responsible for understanding the data**
- Main observation from Amazon (using Dynamo)
 - “There are many services on Amazon’s platform that only need primary-key access to a data store.”
 - e.g., Best seller lists, shopping carts, customer preferences, session management, sales rank, product catalog



Document data model

- **Document:** XML, JSON, text, or binary blob
- Any treelike structure can be represented as an XML or JSON document, including things such as an order that includes a delivery address, billing details, and a list of products and quantities
- Similar to Key-value, except value is a document!
- Examples:
 - Couchbase
 - MongoDB
 - RavenDB
 - ArangoDB
 - MarkLogic
 - OrientDB
 - Redis
 - RethinkDB



Document is a tree-like
hierarchical structure



Data model: Relational to Document

Relational

Person:

Pers_ID	Surname	First_Name	City
0	Miller	Paul	London
1	Ortega	Alvaro	Valencia
2	Huber	Urs	Zurich
3	Blanc	Gaston	Paris
4	Bertolini	Fabrizio	Rom

Car:

Car_ID	Model	Year	Value	Pers_ID
101	Bentley	1973	100000	0
102	Rolls Royce	1965	330000	0
103	Peugeot	1993	500	3
104	Ferrari	2005	150000	4
105	Renault	1998	2000	3
106	Renault	2001	7000	3
107	Smart	1999	2000	2



MongoDB Document

```
{  
  first_name: 'Paul',  
  surname: 'Miller'  
  city: 'London',  
  location: [45.123, 47.232],  
  cars: [  
    { model: 'Bentley',  
      year: 1973,  
      value: 100000, ... },  
    { model: 'Rolls Royce',  
      year: 1965,  
      value: 330000, ... }  
  ]  
}
```



Example: SQL vs MongoDB

RDBMS		MongoDB
Database	↔	Database
Table	↔	Collection
Row	↔	Document
Index	↔	Index
JOIN	↔	Embedded / Reference



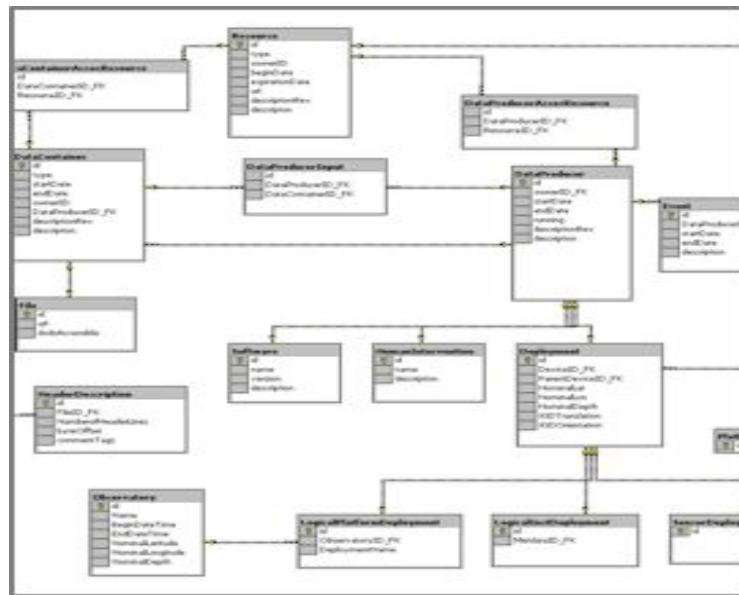
Document Model Benefits

- Rich data model, natural data representation
 - Embed related data in sub-documents & arrays
 - Support indexes and rich queries against any element
- Data aggregated to a single structure (pre-JOINed)
 - Programming becomes simple
 - Performance can be delivered at scale
- Dynamic schema
 - Data models can evolve easily
 - Adapt to changes quickly: agile methodology

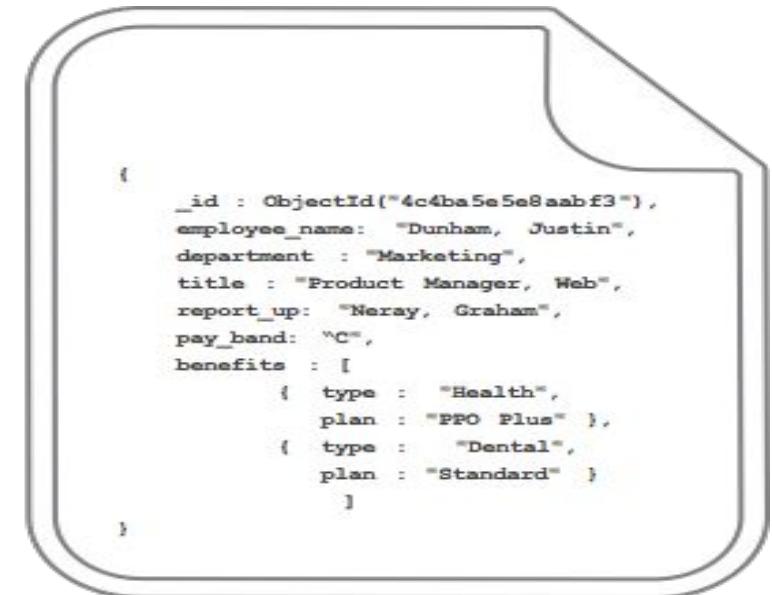
Join vs Aggregate

- Complex objects are maintained in a single place, not across tables

Relational



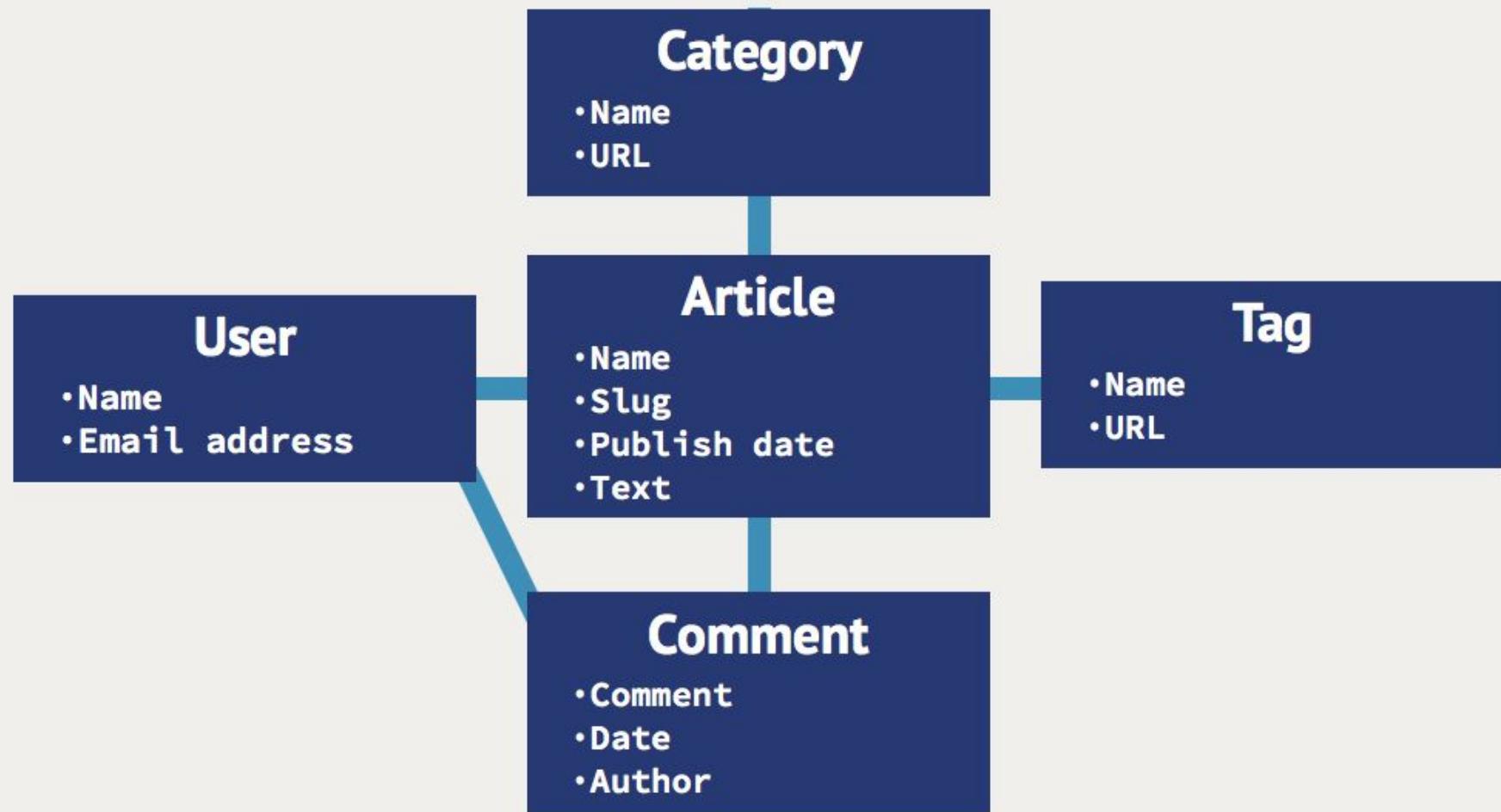
Document Model



- Schema-less models do not need beforehand design!



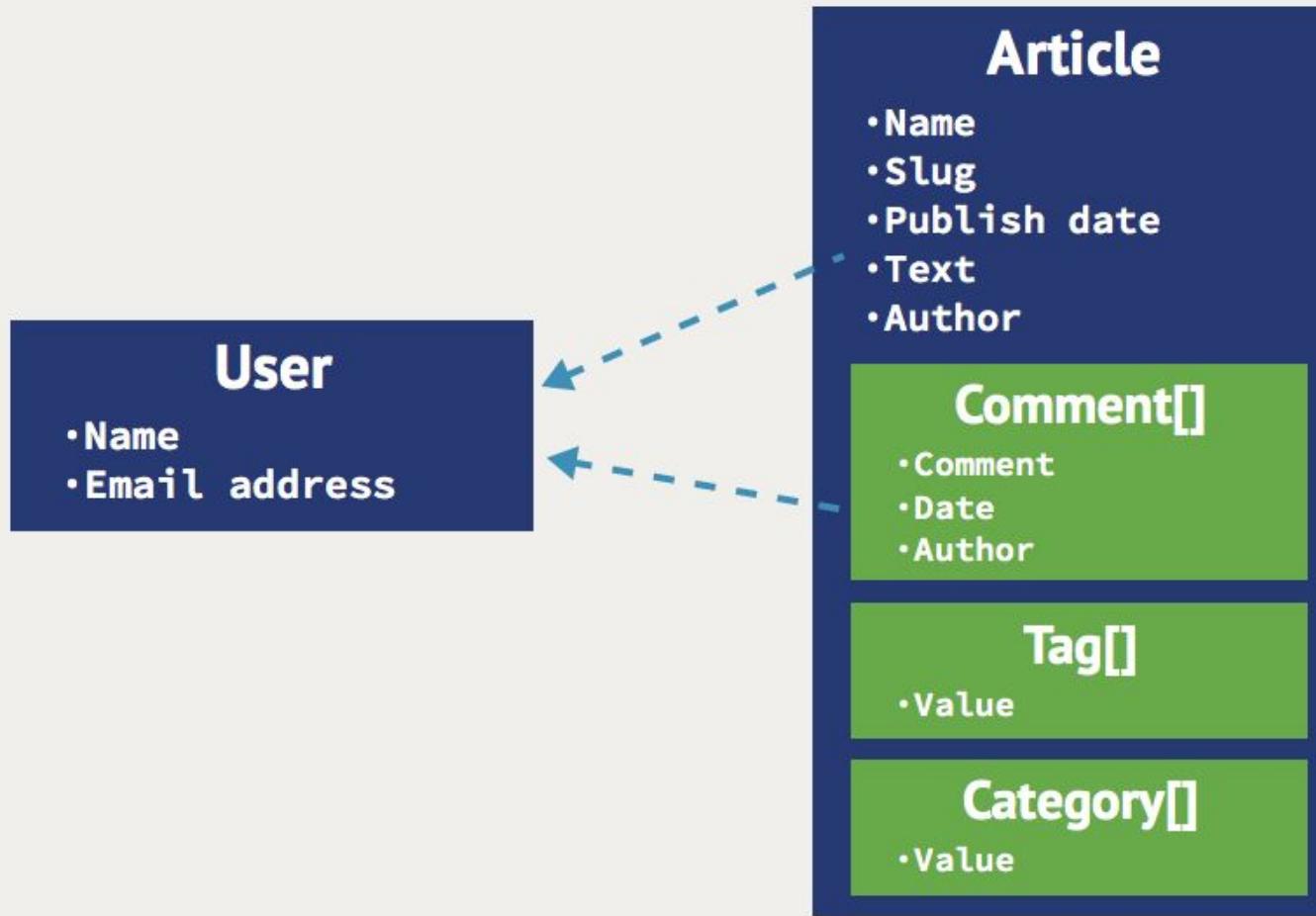
Example: Blogging Platform (Relational)



- Join between 5 table is needed!



Example: Blogging Platform (Document model)



- Higher Performance: Data Locality



Relational vs Document model: Operations

Application	Relational model Action	Document model Action
Create Product Record	INSERT to (n) tables (product description, price, manufacturer, etc.)	insert() to 1 document with sub-documents, arrays
Display Product Record	SELECT and JOIN (n) product tables	find() aggregated document
Add Product Review	INSERT to “review” table, foreign key to product record	insert() to “review” collection, reference to product document



Columnar data model 1/3

- Columnar data models store tables by columns of data instead of by rows of data

ID	Last	First	Bonus
1	Doe	John	8000
2	Smith	Jane	4000
3	Beck	Sam	1000

ROW-BASED

1, Doe, John, 8000;
2, Smith, Jane, 4000;
3, Beck, Sam, 1000;

COLUMN-BASED

1, 2, 3;
Doe, Smith, Beck;
John, Jane, Sam;
8000, 4000, 1000;



Columnar data model 2/3

- **Columnar:** extension to traditional table structures
- Supports variable sets of columns (column families) and is optimized for column-wide operations (such as count, sum, and mean average)
- Compression: column data is of uniform type
- Multiple values (columns) per key
- Examples:
 - Cassandra
 - Hbase
 - Amazon Redshift
 - HP Vertica
 - Teradata



Columnar data model 3/3

- The column is lowest/smallest instance of data
- It is a tuple that contains a key, a value and a timestamp

ColumnFamily: Authors											
Key	Value										
"Eric Long"	<table border="1"><thead><tr><th colspan="2">Columns</th></tr><tr><th>Name</th><th>Value</th></tr></thead><tbody><tr><td>"email"</td><td>"eric (at) long.com"</td></tr><tr><td>"country"</td><td>"United Kingdom"</td></tr><tr><td>"registeredSince"</td><td>"01/01/2002"</td></tr></tbody></table>	Columns		Name	Value	"email"	"eric (at) long.com"	"country"	"United Kingdom"	"registeredSince"	"01/01/2002"
Columns											
Name	Value										
"email"	"eric (at) long.com"										
"country"	"United Kingdom"										
"registeredSince"	"01/01/2002"										
"John Steward"	<table border="1"><thead><tr><th colspan="2">Columns</th></tr><tr><th>Name</th><th>Value</th></tr></thead><tbody><tr><td>"email"</td><td>"john.steward (at) somedomain.com"</td></tr><tr><td>"country"</td><td>"Australia"</td></tr><tr><td>"registeredSince"</td><td>"01/01/2009"</td></tr></tbody></table>	Columns		Name	Value	"email"	"john.steward (at) somedomain.com"	"country"	"Australia"	"registeredSince"	"01/01/2009"
Columns											
Name	Value										
"email"	"john.steward (at) somedomain.com"										
"country"	"Australia"										
"registeredSince"	"01/01/2009"										
"Ronald Mathies"	<table border="1"><thead><tr><th colspan="2">Columns</th></tr><tr><th>Name</th><th>Value</th></tr></thead><tbody><tr><td>"email"</td><td>"ronald (at) sodeso.nl"</td></tr><tr><td>"country"</td><td>"Netherlands, The"</td></tr><tr><td>"registeredSince"</td><td>"01/01/2010"</td></tr></tbody></table>	Columns		Name	Value	"email"	"ronald (at) sodeso.nl"	"country"	"Netherlands, The"	"registeredSince"	"01/01/2010"
Columns											
Name	Value										
"email"	"ronald (at) sodeso.nl"										
"country"	"Netherlands, The"										
"registeredSince"	"01/01/2010"										



Columnar data model: Performance

- Some statistics about Facebook Search (using Cassandra)

MySQL > 50 GB Data

Writes average: ~300 ms

Reads average: ~350 ms



Rewritten with Cassandra > 50 GB Data

Writes average: 0.12 ms

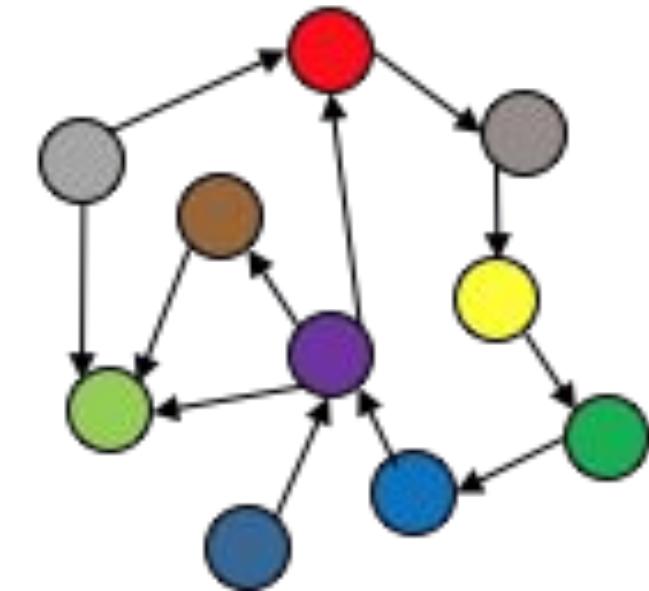
Reads average: 15 ms



Cassandra is 2500x faster!

Graph data model

- **Graph:** structure of data is graph based
- Uses Property Graph data model (Nodes, Relationships, properties)
- Based on Graph Theory
- Scale vertically, no clustering
- You can use graph algorithms easily
- ACID-compliant transactions
- Examples:
 - Neo4j
 - InfiniteGraph
 - OrientDB
 - Titan GraphDB





References

- [Making Sense of NoSQL \(2013\) - Dan McCreary and Ann Kelly](#)
- [NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence \(2012\) - Pramod J. Sadalage and Martin Fowler](#)
- [Data Access for Highly-Scalable Solutions: Using SQL, NoSQL, and Polyglot Persistence \(2013\) - John Sharp, Douglas McMurtry, Andrew Oakley, Mani Subramanian, Hanzhong Zhang](#)
- [Distributed Systems: Concepts and Design \(5th Edition\) \(2011\) - Coulouris, George; Jean Dollimore; Tim Kindberg; Gordon Blair. Boston: Addison-Wesley. ISBN 0-132-14301-1](#)



Big Data and Data Mining

Information Retrieval

Flavio Bertini

flavio.bertini@unipr.it

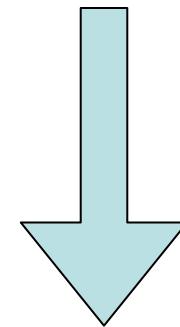


Information Retrieval

- Information Retrieval (IR) is **finding material** (usually documents) of an **unstructured nature** (usually text) that satisfies an **information need** from within **large collections** (usually stored on computers)
- These days we frequently think first of **web search**, but there are many other cases:
 - E-mail/social media search
 - Searching your laptop
 - Corporate knowledge bases
 - Domain specific search (i.e., Legal information retrieval)
 - Digital Libraries
 - News

Basic Assumptions

- **Document:** unstructured file
- **Collection:** a set of documents
 - Assume it is a static, non-hypertext collection for the moment
- **Query:** set of keywords to express an information need
- **Goal:** retrieve documents with information that is relevant to the user's information need and helps the user complete a task





Misconception/Misformulation

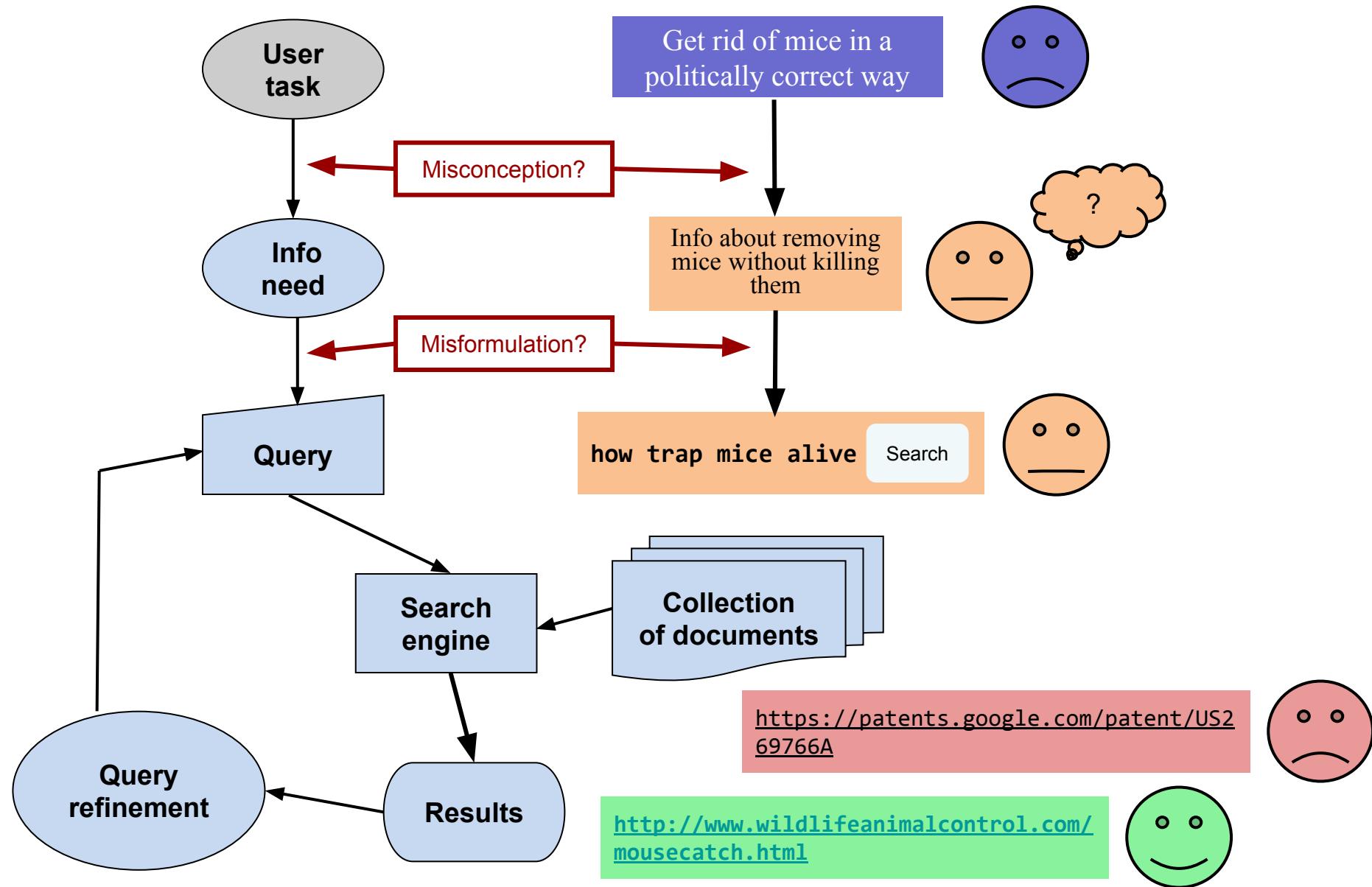


how do I get twelve-years-old girls excited





Classic Search Model





What is a document?

- Examples

- web pages, email, books, news stories, scholarly papers, text messages, Word, Powerpoint, PDF, forum postings, patents, instant messaging sessions, etc.

- Common properties

- Significant text content
- Mostly unstructured
- Some structure (e.g., title, author, date for papers, subject, sender, destination for email)





Data Retrieval vs IR (1/2)

- A Data Retrieval system (DBMS) deals with data of a **well defined structure (database schema)**
- An Information Retrieval system deals with texts written in **natural language**, often non structured and ambiguous (**text documents**)

Database record	Text document
Content is structured in typed fields	Content is unstructured
Values consistent with their defined data types	Can contain any kind of data (codes, dates, prices) all in text format
Data is stored in pre-defined formats	Information is expressed in natural language



Data Retrieval vs IR (2/2)

- A Data Retrieval system (DBMS) retrieve data using a formally defined **query language** (i.e., SQL, relational algebra, etc)
- An Information Retrieval system retrieve documents by means of a set of keywords (**query** or **more properly search**) expressed in natural language

Query language	Search language
Based on a formal grammar	Based on keywords from natural language
The answer is not dependent on the system implementation	Can be interpreted differently from different IR systems
Produces as an answer a predictable set of records which are true under the query	Produces a ranked list of results based on the relevance to the query



Comparing text

- **Comparing** the query text to the document text and determining what is a good match is the core issue of information retrieval
- Exact matching of words is not enough
 - Many different ways to write the same thing in a “natural language” like English
 - e.g., does a news story containing the text “bank director in Amherst steals funds” match the query?
 - Some stories will be better matches than others
- A **relevance** notion is needed instead of exact matching
- Different **retrieval models** are based on different relevance notions



Retrieval Models

- **Classic models**
 - Boolean retrieval
 - Vector Space Model
- **Probabilistic Models**
 - BM25 (Best Match 25)
 - Language models
- **Combining evidence**
 - Inference networks
 - Learning to Rank
- Progress in retrieval models has corresponded with improvements in effectiveness



Boolean Retrieval Model (1/2)

- The Boolean retrieval model is being able to ask a query that is a Boolean expression:
 - Boolean Search are queries using AND, OR and NOT to combine search terms
 - Views each document as a **set of words (Bag-of-words)**
 - Is precise: document **matches condition or not**
 - Perhaps the simplest model to build an IR system
- Primary commercial retrieval tool for three decades until the early 1990s (arrival of World Wide Web)
- Many search systems you use still are Boolean:
 - Email, library catalog, Mac OS X Spotlight

Boolean Retrieval Model (2/2)

- Which plays of Shakespeare contain the words *Brutus AND Caesar BUT NOT Calpurnia?*
- One could “grep” all of Shakespeare’s plays for Brutus and Caesar, then strip out lines containing Calpurnia?
- Why is that not the answer?
 - Slow (for large corpora)
 - NOT Calpurnia is non-trivial
- We need a way to represent documents as **sets of words**



William Shakespeare



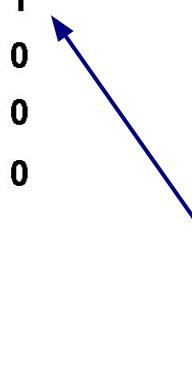
Term-document incidence matrix

Words

	Documents					
	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Remember our search:

***Brutus AND Caesar BUT NOT
Calpurnia***



1 if play contains
word, 0 otherwise



Incidence vectors

Brutus AND Caesar BUT NOT Calpurnia

- So we have a 0/1 vector for each word
- To **answer** search: take the vectors for Brutus, Caesar and Calpurnia (the last complemented!) → bitwise AND

$$110100 \text{ AND } 110111 \text{ AND } 101111 = 100100$$

Answer:	1 Antony and Cleopatra	0 Julius Caesar	0 The Tempest	1 Hamlet	0 Othello	0 Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0



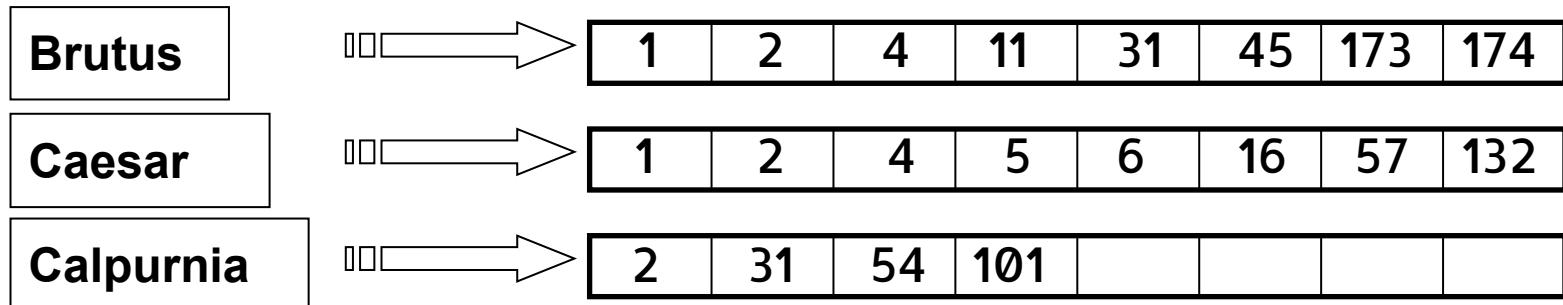
Matrix dimension

- Consider $N = 1$ million documents, each with about 1000 words
- Avg 6 bytes/word including spaces/punctuation
 - 6GB of data in the documents
- Say there are $M = 500K$ *distinct* terms among these
- $M \times N = 500K \times 1M$ matrix has half-a-trillion 0's and 1's
- Actually, it has no more than one billion 1's
 - matrix is **extremely sparse**
- What's a better representation?
 - **We only record the 1 positions**



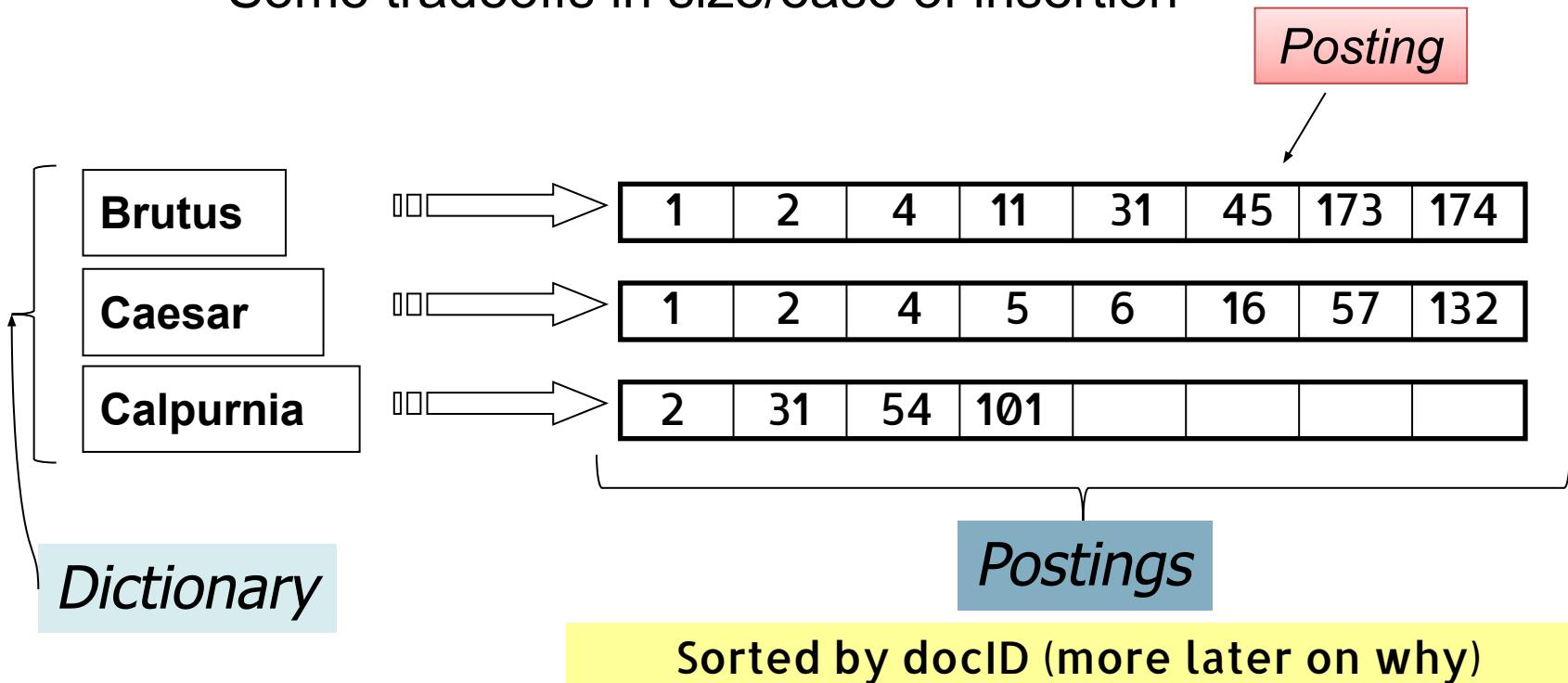
Inverted Index

- For each term t , we must store a list of all documents that contain t
 - Identify each doc by a **docID**, a document serial number
- Can we use fixed-size arrays for this?
- What happens if the word Caesar is added to document 14?



Posting-lists

- We need variable-size postings lists
 - On disk, a continuous run of postings is normal and best
 - In main memory, can use linked lists or variable length arrays
 - Some tradeoffs in size/ease of insertion



Inverted index construction

Documents to be indexed



Friends, Romans, countrymen.

⋮

Tokenizer

Token stream

Friends

Romans

Countrymen

Linguistic modules

Modified tokens

friend

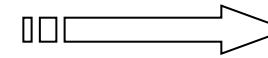
roman

countryman

Indexer

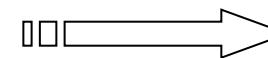
Inverted index

friend



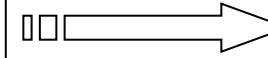
2 → 4

roman



1 → 2

countryman



13 → 16



Text preprocessing

- Tokenization
 - Cut character sequence into word tokens
 - Deal with “***John’s***”, a ***state-of-the-art solution***
- Normalization
 - Map text and query term to same “normal” form
 - You want ***U.S.A.*** and ***USA*** to match
- Stemming & Lemmatization
 - We may wish different forms of a root to match
 - ***authorize***, ***authorization***
- Stop words
 - We may omit very common words (or not)
 - ***the, a, to, of***



Tokenization

- Sequence of <modified token, document ID> pairs

Doc 1

I did enact Julius
Caesar I was killed
i' the Capitol;
Brutus killed me.

Doc 2

So let it be with
Caesar. The noble
Brutus hath told you
Caesar was ambitious



Term	docID
I	1
did	1
enact	1
Julius	1
Caesar	1
I	1
was	1
killed	1
i	1
the	1
Capitol	1
Brutus	1
killed	1
me	1
So	2
let	2
it	2
be	2
with	2
Caesar	2
The	2
noble	2
Brutus	2
hath	2
told	2
you	2
Caesar	2
was	2
ambitious	2



Normalization

- We may need to “normalize” words in indexed text as well as query words into the same form
 - We want to match *U.S.A.* and *USA*
- Result is terms: a **term** is a (normalized) word type, which is an entry in our IR system dictionary
- We most commonly implicitly define equivalence classes of terms by, for example:
 - deleting periods to form a term
 - *U.S.A.*, *USA* → *USA* ??
 - deleting hyphens to form a term
 - *anti-discriminatory*, *antidiscriminatory* → *antidiscriminatory* ??



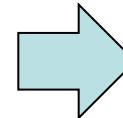
Normalization: Case folding

- Reduce all letters to lower case
 - Exception: upper case in mid-sentence?
 - e.g., General Motors
 - Fed vs. fed
 - SAIL vs. sail
 - Often best to lower case everything, since users will use lowercase regardless of ‘correct’ capitalization...
- Longstanding Google example: [fixed in 2011...]
 - Query C.A.T.
 - #1 result is for “cats” (well, Lolcats) not Caterpillar Inc.

Stemming

- Reduce terms to their “roots” before indexing
- “Stemming” suggests crude postfix chopping
 - language dependent
 - e.g., **automate(s), automatic, automation** all reduced to **automat**

*for example compressed
and compression are both
accepted as equivalent to
compress.*



for example compress
and compress are both
accept as equival to
compress



Lemmatization

- Reduce inflectional/variant forms to base form
 - For example:
 - *am, are, is* → *be*
 - *car, cars, car's, cars'* → *car*
 - *the boy's cars are different colors* → *the boy car be different color*
- Lemmatization implies doing “proper” reduction to dictionary headword form



Stop words

- With a stop list, you exclude from the dictionary entirely the commonest words. Intuition:
 - They have little semantic content: *the, a, and, to, be*
 - There are a lot of them: ~30% of total occurrences for top 30 words
- But the trend recently is away from doing this:
 - Good compression techniques means the space for including stop words in a system is very small
 - Good query optimization techniques mean you pay little at query time for including stop words
 - You need them for:
 - Phrase queries: “King of Denmark”
 - Various song titles, etc.: “Let it be”, “To be or not to be”
 - “Relational” queries: “flights to London”



Dictionaries

- The lexicon of the indexed corpus
 - e.g., all words on the web
 - All names, acronyms etc. (including the mis-spellings)
- Language-specific dictionary
 - Webster's English Dictionary
 - [**WordNet**](#) is a lexical database for the English language. It groups English words into sets of synonyms called synsets
- Domain-specific dictionary
 - The **Unified Medical Language System (UMLS)** is a compendium of many controlled vocabularies in the biomedical sciences
 - **MeSH** is a comprehensive controlled vocabulary for the purpose of indexing journal articles and books in the life sciences

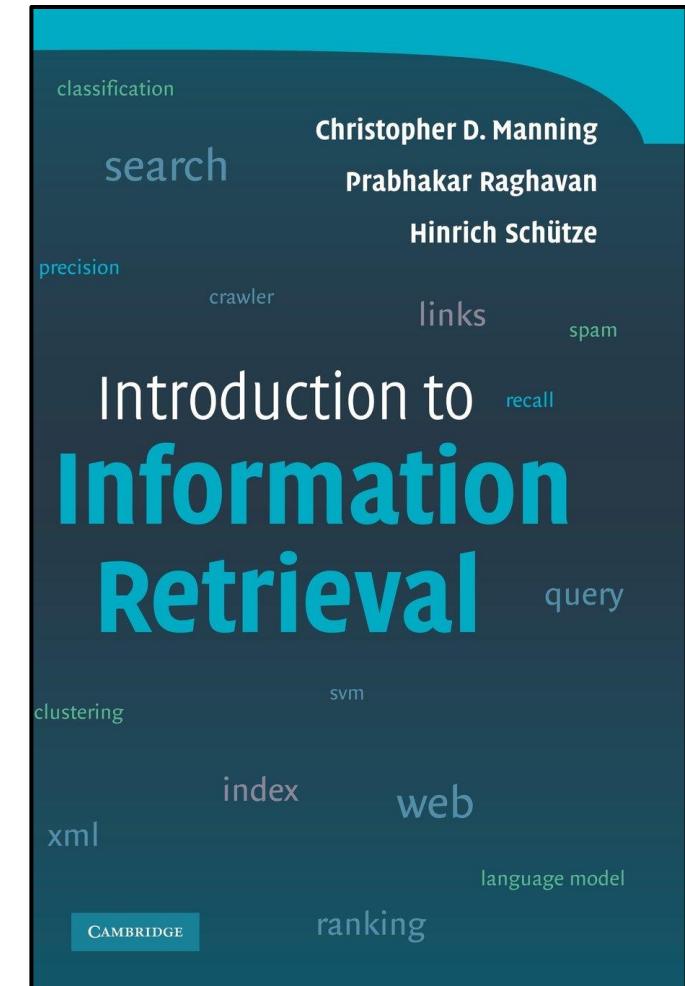


References

Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze
Introduction to Information Retrieval
Cambridge University Press. 2008

The book is also online for free:

- HTML edition (2009.04.07)
- PDF of the book for online viewing
(with nice hyperlink features,
2009.04.01)
- PDF of the book for printing
(2009.04.01)





Big Data and Data Mining

Ranking

Flavio Bertini

flavio.bertini@unipr.it



Boolean model limits

- Thus far, our queries have all been boolean
 - Documents **either match or don't**
- Good for **expert users** with precise understanding of their needs and knowledge of the collection
 - Also good for applications: applications can easily make use of 1000s of results for further processing
- Not good for the majority of users
 - Most users incapable of writing Boolean queries (or they are, but they think it's too much work)
 - Most users don't want to wade through 1000s of results
 - This is particularly true of web search!



Boolean model limits: example

- Boolean queries often result in either **too few** (=0) or **too many** (1000s) results
 - For example, suppose a user is having troubles with his network card:
 - Query 1: “*standard user dlink 650*” → 200,000 hits
 - Query 2: “*standard user dlink 650 no card found*”: 0 hits
 - It takes a lot of skill to come up with a query that produces a manageable number of hits
 - **AND gives too few**
 - **OR gives too many**

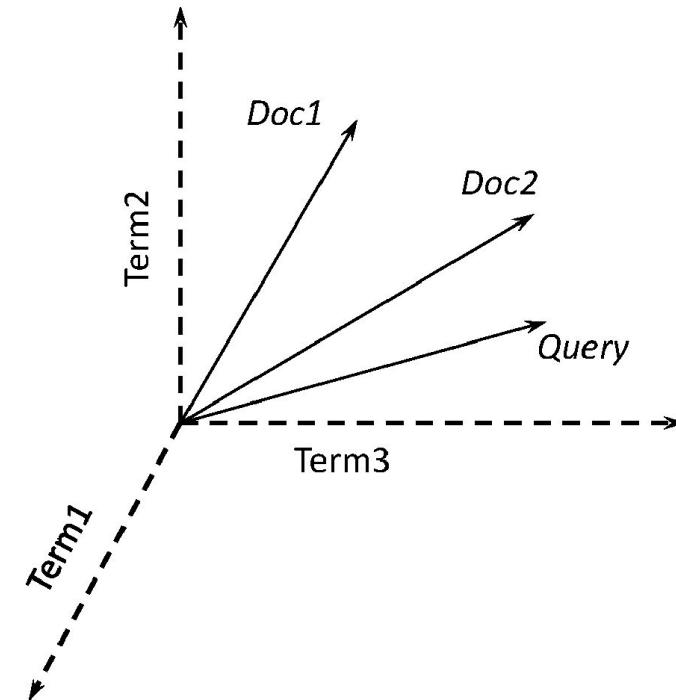


Ranked retrieval

- Rather than a set of documents satisfying a query expression, in **ranked retrieval**, the system returns an ordering over the **(top)** documents in the collection for a query
- When a system produces a ranked result set, large result sets are not an issue
 - We just show the **top k** (≈ 10) results
 - We don't overwhelm the user
 - Premise: the ranking model works
- Most known ranking model is Vector Space Model
 - Salton, G., Wong, A., & Yang, C. S. (1975). [A vector space model for automatic indexing](#). Communications of the ACM

Vector Space Model (VSM)

- Like in the bag-of-word model, documents are **represented by a vector** of “term weights”



- Collection represented by a matrix of **term weights**
 - d_{ij} is the weight of Doc_i for Term_j

$$D_i = (d_{i1}, d_{i2}, \dots, d_{it}) \quad Q = (q_1, q_2, \dots, q_t)$$

	<i>Term₁</i>	<i>Term₂</i>	...	<i>Term_t</i>
<i>Doc₁</i>	d_{11}	d_{12}	...	d_{1t}
<i>Doc₂</i>	d_{21}	d_{22}	...	d_{2t}
⋮	⋮			
<i>Doc_n</i>	d_{n1}	d_{n2}	...	d_{nt}

VSM: vector representation

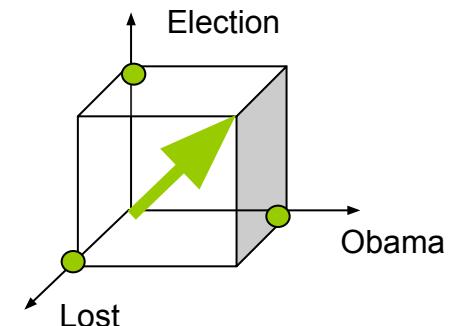
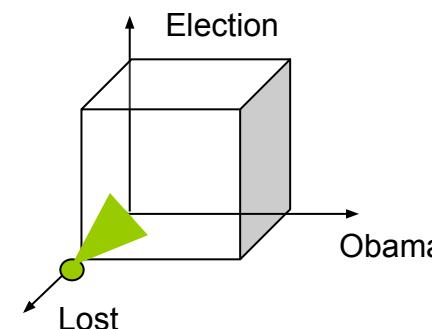
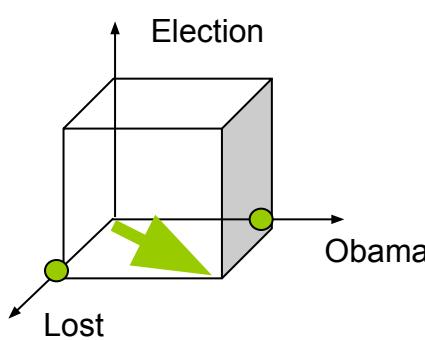
- An example with 3 terms (3-dimensional vector space) and 3 documents

D1 In last year election, Romney **lost** election against Obama

D2 I **lost** my faith in Clinton's ability to win

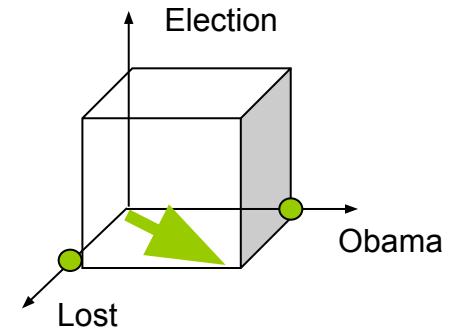
D3 **Obama** has **lost** control on Senate

We consider only
the terms in figure



Documents as Vectors

- So we have a $|V|$ -dimensional vector space
- Terms are axes of the space
- Documents are points or vectors in this space
- Very high-dimensional: tens of millions of dimensions when you apply this to a web search engine
- These are very sparse vectors - most entries are zero





Queries as Vectors

- **Key idea 1:** Do the same for queries representing them as vectors in the space
- **Key idea 2:** Rank documents according to their proximity to the query in this space
 - proximity = **similarity of vectors**
 - proximity \approx inverse of distance
- **Reminder:** We do this because we want to get away from the in-or-out Boolean model
- Instead: **rank more relevant documents** higher than less relevant documents



Recap

- In the boolean model, a document is either relevant or non-relevant
- Representing document and queries as vectors, we can compute the **similarity between the vector** of the document and the vector of the query:
 - This is an estimation of the relevance of the document with respect to the query
- We need to define the similarity between vectors but...
 - Before going on we must introduce the **term frequency**



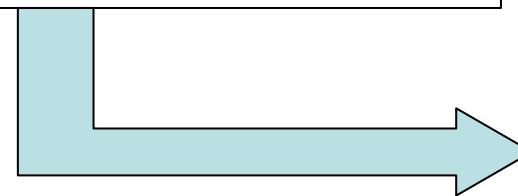
TF: Term Frequencies

- So far, vectors were binary, representing the **presence or absence** of terms in a document
- We would like to assign a **weight** for each term in a vector space:
 - The term frequency $\text{tf}_{t,d}$ of term t in document d is defined as the number of times that t occurs in d
 - We want to use term frequency (**tf**) when computing query-document match scores

Example: TF as term weight

Three dimensional pictures are useful, but can be misleading for high-dimensional space (more than 3 terms considered):

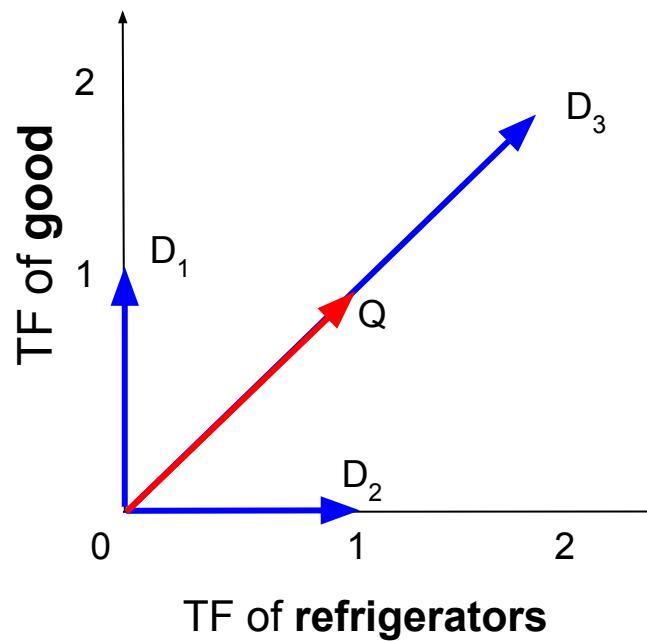
- D_1 Tropical Freshwater Aquarium Fish.
- D_2 Tropical Fish, Aquarium Care, Tank Setup.
- D_3 Keeping Tropical Fish and Goldfish in Aquariums, and Fish Bowls.
- D_4 The Tropical Tank Homepage - Tropical Fish and Aquariums.



Terms	Documents			
	D_1	D_2	D_3	D_4
Aquarium	1	1	1	1
Bowl	0	0	1	0
Care	0	1	0	0
Fish	1	1	2	1
Freshwater	1	0	0	0
Goldfish	0	0	1	0
Homepage	0	0	0	1
Keep	0	0	1	0
Setup	0	1	0	0
Tank	0	1	0	1
Tropical	1	1	1	2

Vector Space Distance

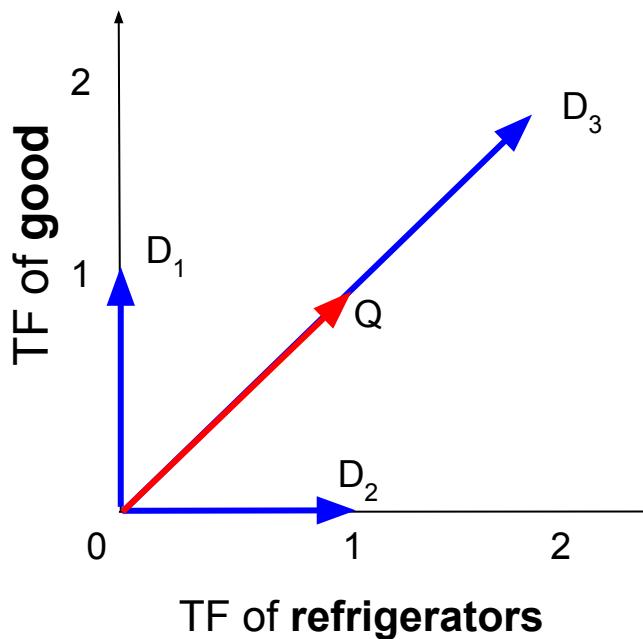
- How to formalize the proximity between two vectors when we have term frequencies?
- Example query Q: “Good refrigerators”
 - D_1 : “Good morning to all of you.”
 - D_2 : “Don’t put pizza in refrigerators”
 - D_3 : “Good Refrigerator Review: top five good refrigerators.”



Note: we are showing only the two interesting dimensions, “good” and “refrigerators”. There is a dimension for each indexed term (e.g., for pizza, morning, review...)

First cut: Euclidean Distance?

- We could consider the straight line distance between the endpoints, *i.e.*, the euclidean distance
 - Distance between Q and D_1 is 1
 - Distance between Q and D_2 is 1
 - Distance between Q and D_3 is $\sqrt{2} \approx 1.414$
- D_3 is the least similar, this doesn't seem right!



- Remember Q: “Good refrigerators”
 - D_1 : “Good morning to all of you.”
 - D_2 : “Don’t put pizza in refrigerators”
 - D_3 : “Good refrigerators review: top five good refrigerators.”

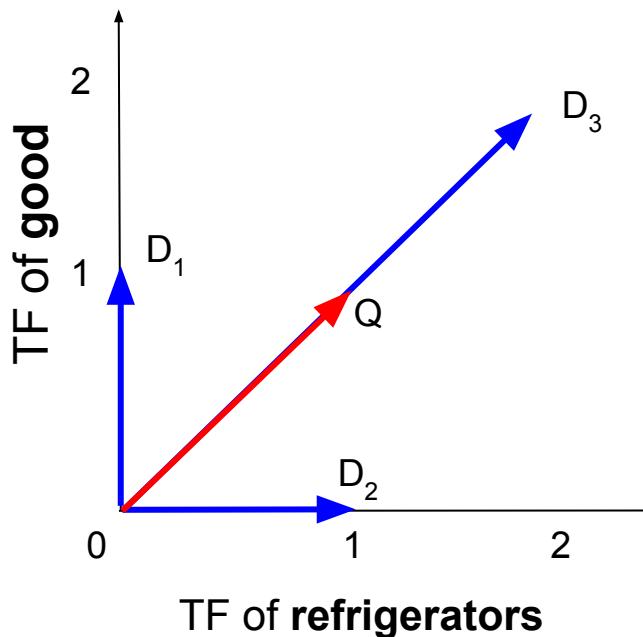


Angle instead of distance

- Thought experiment: take a document d and append it to itself many times. Call this longer document d'
- “Semantically” d and d' have the same content
- The Euclidean distance between the two documents can be quite large
- The angle between the two documents is 0, corresponding to maximal similarity
- **Key idea:** Rank documents according to angle with query

Example: Using angles

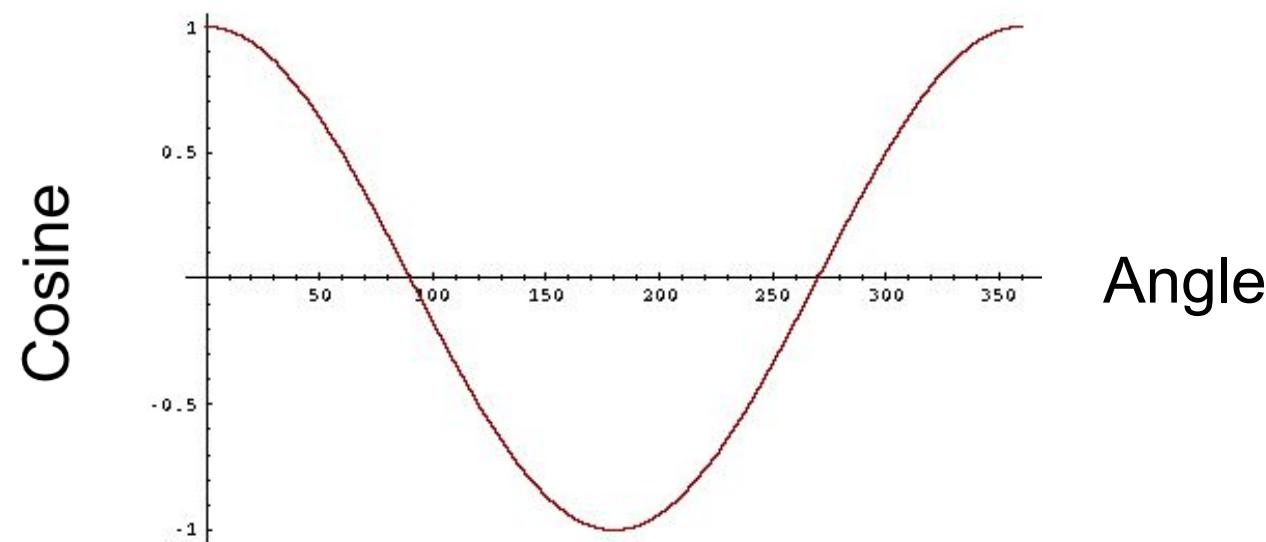
- We now consider the angle between vectors
 - Angle between Q and D_1 is 45°
 - Angle between Q and D_2 is 45°
 - Angle between Q and D_3 is 0°
- D_3 is the most similar, this is what we need!



- Remember Q: “Good refrigerators”
 - D_1 : “Good morning to all of you.”
 - D_2 : “Don’t put pizza in refrigerators”
 - D_3 : “Good refrigerators review: top five good refrigerators.”

From Angles to Cosines

- The following two notions are equivalent.
 - Rank documents in increasing order of the angle between query and document
 - Rank documents in decreasing order of *cosine(query, document)*
- Cosine is a monotonically **decreasing** function for the interval $[0^\circ, 180^\circ]$, meaning that it's inversely proportional of the angle





Cosine similarity

$$\cos(\vec{q}, \vec{d}) = \frac{\vec{q} \cdot \vec{d}}{|\vec{q}| |\vec{d}|} = \frac{\vec{q}}{|\vec{q}|} \cdot \frac{\vec{d}}{|\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2} \sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

Dot product **Unit vectors**

- q_i is the tf weight of term i in the query
- d_i is the tf weight of term i in the document

$\cos(\vec{q}, \vec{d})$ is the cosine similarity of \vec{q} and \vec{d} ... or,
equivalently, the cosine of the angle between \vec{q} and \vec{d} .



Example: Formula

	Election	Lost	Obama		Election	Lost	Obama		
D1 [1	1	1]	D2 [0	1	0]

	Election	Lost	Obama		Election	Lost	Obama		
D3 [0	1	1]	Q [0	0	1]

$$Sim(Q, D1) = \frac{\sum_{i=1}^3 Q_i D1_i}{\sqrt{\sum_{i=1}^3 Q_i^2} \sqrt{\sum_{i=1}^3 D1_i^2}}$$

$$Sim(Q, D2) = \frac{\sum_{i=1}^3 Q_i D2_i}{\sqrt{\sum_{i=1}^3 Q_i^2} \sqrt{\sum_{i=1}^3 D2_i^2}}$$

$$Sim(Q, D3) = \frac{\sum_{i=1}^3 Q_i D3_i}{\sqrt{\sum_{i=1}^3 Q_i^2} \sqrt{\sum_{i=1}^3 D3_i^2}}$$



Example: Instantiation

	Election	Lost	Obama		Election	Lost	Obama		
D1 [1	1	1]	D2 [0	1	0]

	Election	Lost	Obama		Election	Lost	Obama		
D3 [0	1	1]	Q [0	0	1]

$$\text{Sim}(Q, D1) = \frac{0*1 + 0*1 + 1*1}{\sqrt{1} * \sqrt{3}} = 0.577$$

$$\text{Sim}(Q, D2) = \frac{0*0 + 0*1 + 1*0}{\sqrt{1} * \sqrt{1}} = 0.000$$

$$\text{Sim}(Q, D3) = \frac{0*0 + 0*1 + 1*1}{\sqrt{1} * \sqrt{2}} = 0.707$$



Terms rarity

- Common terms are less informative than rare terms
- Consider a query like “*good refrigerators*”
 - With TF (term frequency) we consider the words frequency in each document
 - However, *good* is a very common word, that can be found in most documents, while *refrigerators* is less common and thus more informative for relevance
 - We **should weight more the rare words** than the common words
- We will use **inverse document frequency (idf)** to capture this, which in turn is based on document frequency (**df**)



DF: Document Frequency

- df_t is the document frequency of t : the number of documents that contain t
 - Note: despite its name is *document* frequency, it involves the **whole collection of documents!**
 - It measures **how common a term is in the whole collection of documents**
- df_t is an **inverse** measure of the informativeness of t
 - The **more common** is a word, the **less informative** will be for relevance (extreme case: stop words)
 - We use the **df** to formulate the **inverse** concept, in order to express the **rarity** of a word



IDF: Inverse Document Frequency

- We define the **inverse document frequency** (idf) of term t by

$$\text{idf}_t = \log_{10}(N/\text{df}_t)$$

- N is the number of documents in the collection
- We use **log** (N/df_t) instead of N/df_t to smooth the effect of idf, for example:
 - a common word like “*home*” can have a frequency **1 million bigger** than a rare word such as “*fenestration*”
 - we apply the logarithm to flatten this **exponential** phenomenon of linguistics (Zipf’s law), otherwise very common words would have an idf of zero



Effect of IDF on Ranking

- Does idf have an effect on ranking for one-term queries, like
 - iPhone?
- idf has no effect on ranking one term queries
 - idf affects the ranking of documents for queries with at least two terms
 - For the query **capricious person**, idf weighting makes occurrences of **capricious** count for much more in the final document ranking than occurrences of **person**
- IDF is never used alone, instead is used together with the TF for a better estimation of document relevance



TF-IDF 1/2

- The tf-idf weight of a term is the product of its tf weight and its idf weight

$$w_{t,d} = \text{tf}_{t,d} \times \log_{10}(N/\text{df}_t)$$

- Best known weighting scheme in information retrieval
 - Note: the “-” in tf-idf is a hyphen, not a minus sign!
 - Alternative names: tf.idf, **tf x idf**



TF-IDF 2/2

$$w_{t,d} = tf_{t,d} \times \log_{10}(N/df_t)$$

- The tf-idf weight of a term:
 - Increases with the number of occurrences within a document (**tf**)
 - Increases with the rarity of the term in the collection (**idf**)



How Search Engines Work



The user
inputs a query

Google

How search engines
how search engines work
how search engines work google
how search engines rank
how search engines see my site
how search engines work for dummies
how search engines operate
how search engines work pdf
how search engines work video
how search engines see your site
how search engines make money

Google Search I'm Feeling Lucky

The search engine search
the query terms in its very
large **index**



About 83,900,000 results (0.36 seconds)

How search engines work

Web Videos Images News Shopping More Search tools

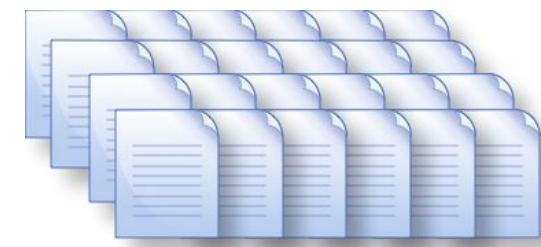
About 83,900,000 results (0.36 seconds)

You can find pages by following links from other pages but usually it is easier to **search** for things using a **search engine**. These are programs that **search** an index of the world wide web for keywords and display the results in order.

BBC Bitesize - How do search engines work?
www.bbc.co.uk/guides/ztbjq6f British Broadcasting Corporation

Millions of results
found are **ranked**
accordingly to a
notion of **relevance**
and are shown as a
sorted list

Automated programs (a.k.a.
spiders or **crawlers**) scan all
pages in the web to build an
index



How Search Engines Work - The Beginners Guide to SEO ...
moz.com/beginners-guide-to-seo/how-search-engines-operate Moz
Search engines have two major functions: crawling and building an index, and providing
search users with a ranked list of the websites they've determined are ...

How Internet Search Engines Work - HowStuffWorks
computer.howstuffworks.com/internet/basics/search-engine.htm
Internet search engines do your research for you. Learn how internet search engines
like Google work, how internet search engines build an index and what ...

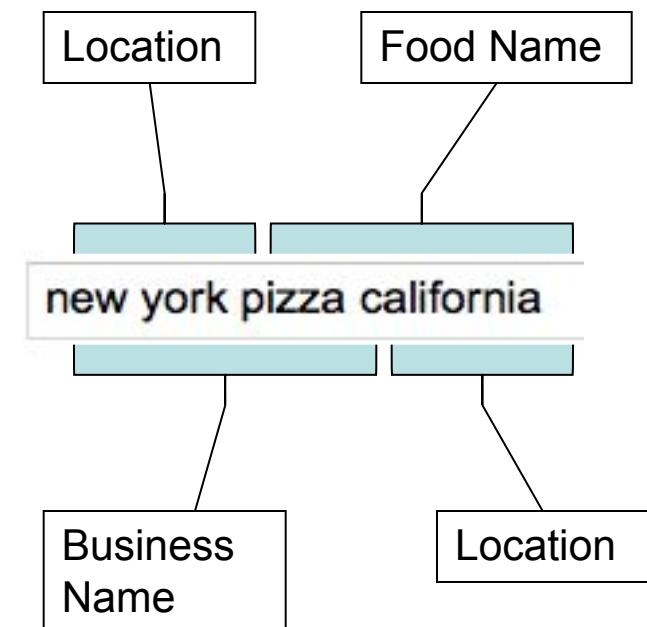
Images for How search engines work

Report images

What is Relevant?

- How can a search engine know what the user is looking for?
 - **Information need:** *a desire to locate and obtain information to satisfy a conscious or unconscious need* [Wikipedia]
 - **User query:** the set of consecutive terms formulated by a user to express his information need
 - **Query intent:** the task, goal or intent of a user, expressed by a query. The same query formulated by different users may be the result of different intents

Query intent can be ambiguous:



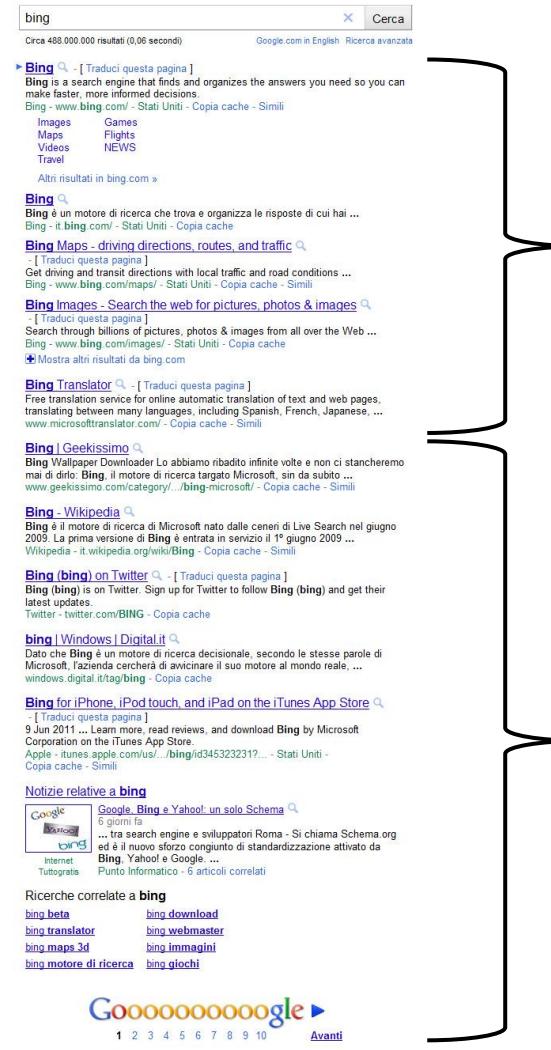


Ranking: Google PageRank

- A relevance notion based solely on term frequencies is not enough to rank billions of documents
- Complex measures are applied to evaluate the quality, reliability and authority of web pages:
 - Measures based **on topological network properties** (such as Google PageRank)
 - Measures based **on different field boosting** (Title, subtitle, body have different weight)
 - Measures based **on semantics and time-space properties** (freshness of a page)
- PageRank set up a **rich-get-richer loop**, whereby few sites dominate the top ranks

Search Engine Results Page

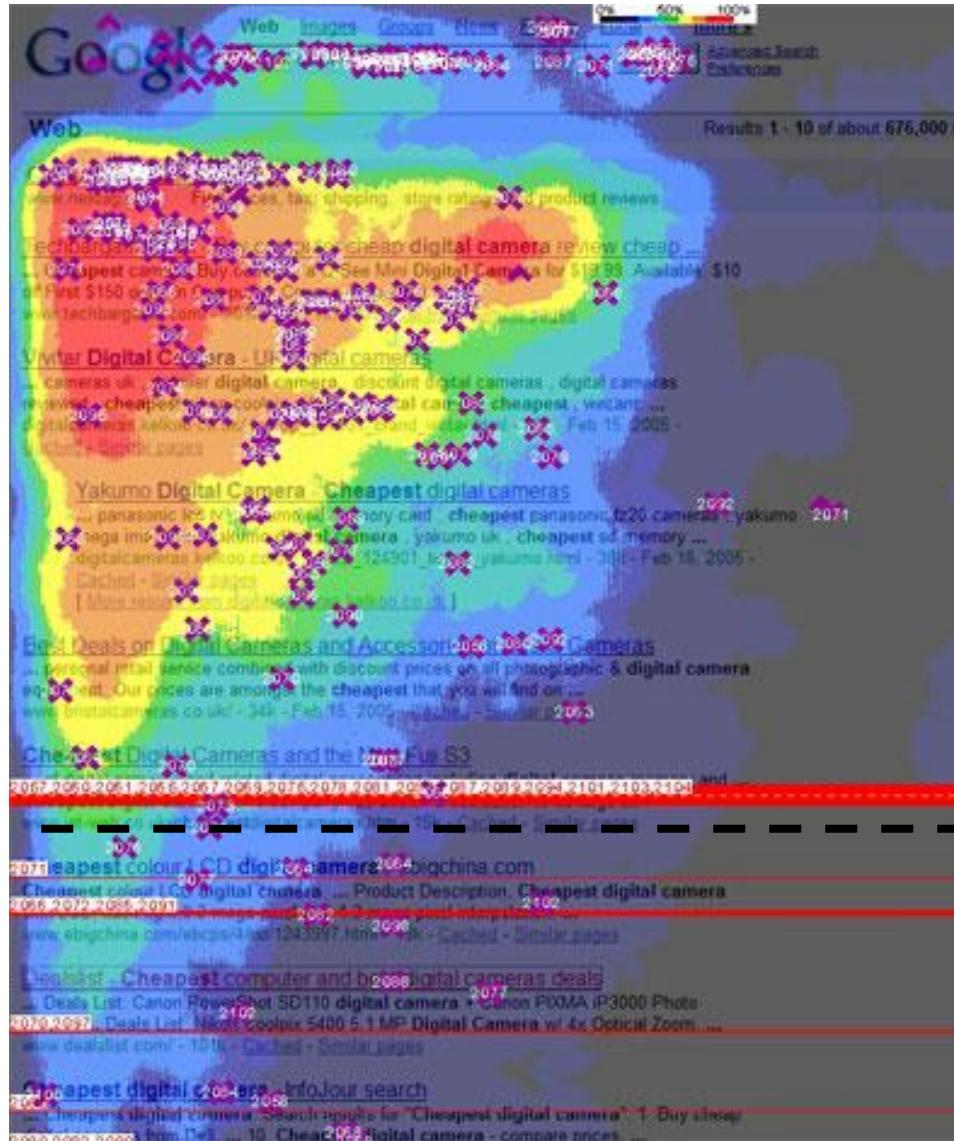
- A Search Engine Results Page (SERP) shows by default **10 results**
- Of these results, on average, the first **5 are visible** without scrolling down
- We will take in consideration the **user behavior on the first SERP** of search engines (the first 10 results)

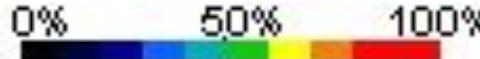


Visible
Area

Scroll
Area

Eye-tracking in 2005: Golden Triangle



- Color  represents percentage of time spent looking the area
- Purple **X** represents a mouse click on the page
- Dotted line - - - represents where the page breaks on the computer screen (Visible area/Scroll area)
- Red lines — indicate how far down the page scrolled before leaving the page

Results in 1st position are clicked more than 10 times
results on 6th position



Eye-tracking in 2014: F-shape Scan

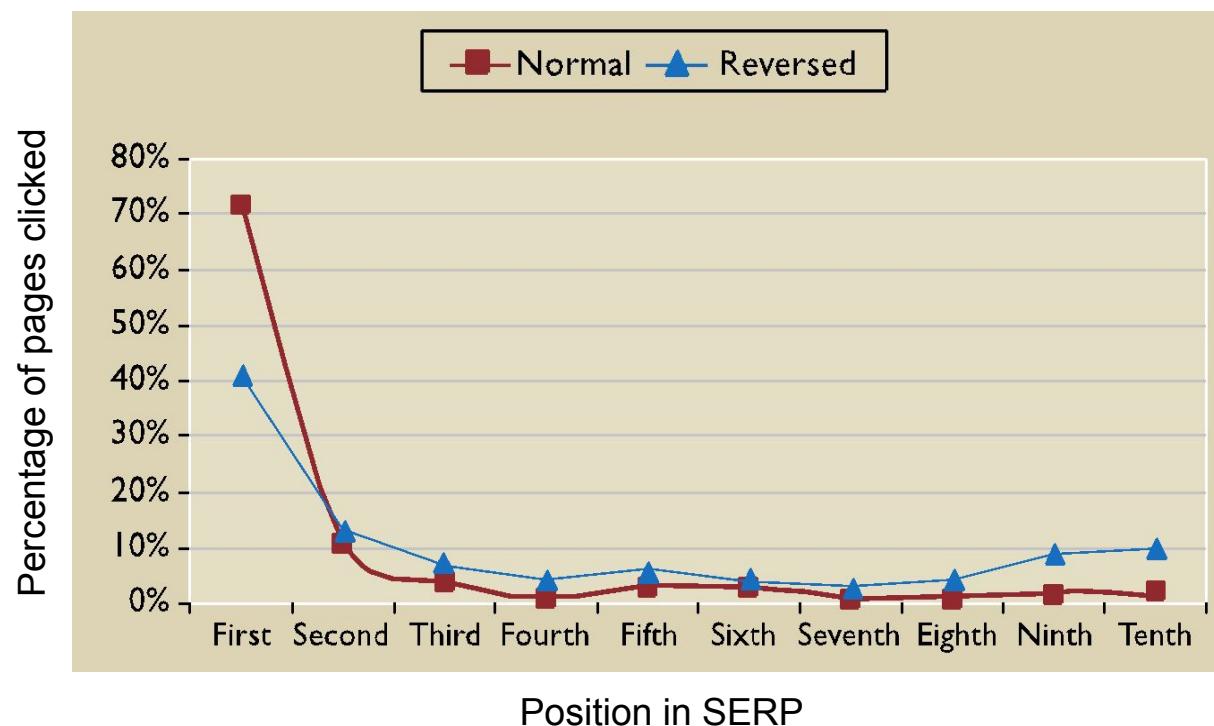


- In 2005, all results were links to a website. Now, we have a variety of results, and the results page layout varies from search to search.
- The first (vertical) glance allows to determine what categories Google has decided to show.
- Then, in a second (horizontal) glance, we go back and deep to the most relevant chunks of results.

The time spent scanning each result is halved: in 2005 was 2.5 seconds, in 2014 is 1.2 seconds.

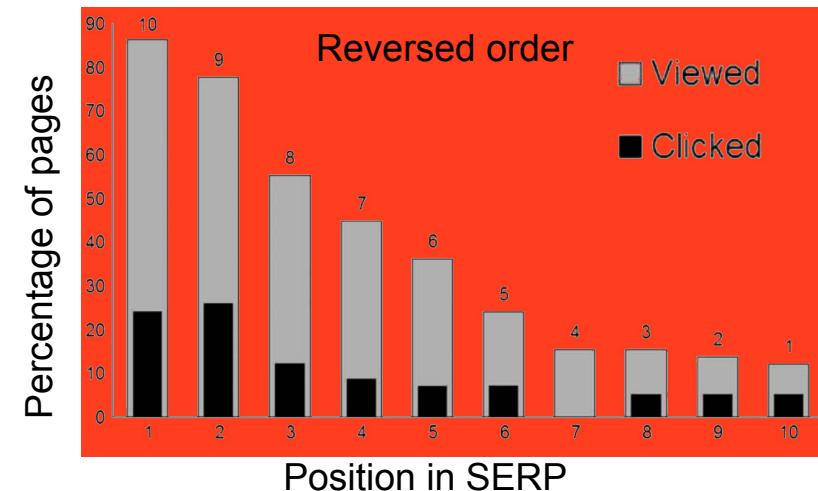
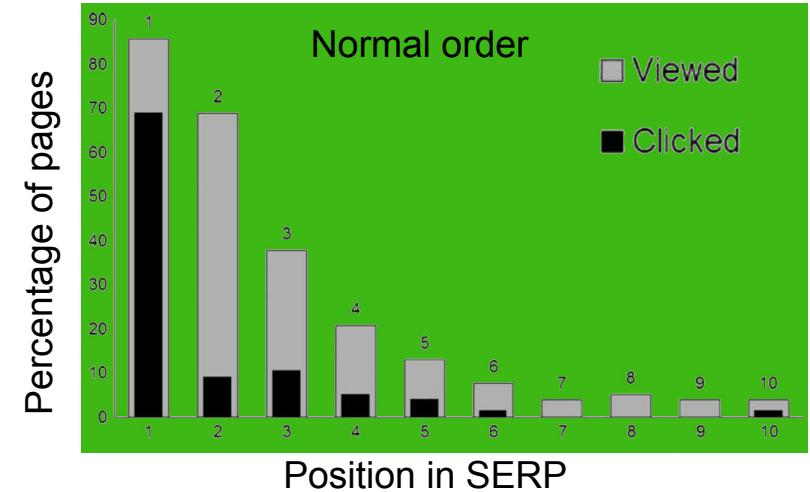
Result Positioning Bias

- When searching for information, the average searchers do not judge systematically all the results, instead they **simply click on top results**
- Studies have been conducted comparing the users' responses when they received results lists in **normal ordering versus a reversed order**



Click and Eye-tracking Bias

- View percentage and mouse clicks are compared in normal vs. reversed order of the first 10 results
- Even though results show some user awareness, **excessive trust in ranking** algorithms may negatively affect smaller positioned websites





Position Impact on Business

Web Images Videos Maps News Shopping Gmail more ▾

Google

shopping
About 958,000,000 results (0.21 seconds)

Everything
Images
Videos
Shopping
Blogs
More

Columbus, OH
Change location

All results
Related searches
Wonder wheel
Sites with images
More search tools

Something different
nightlife
art
price comparison
froogle

2006 AOL LEAKED DATA

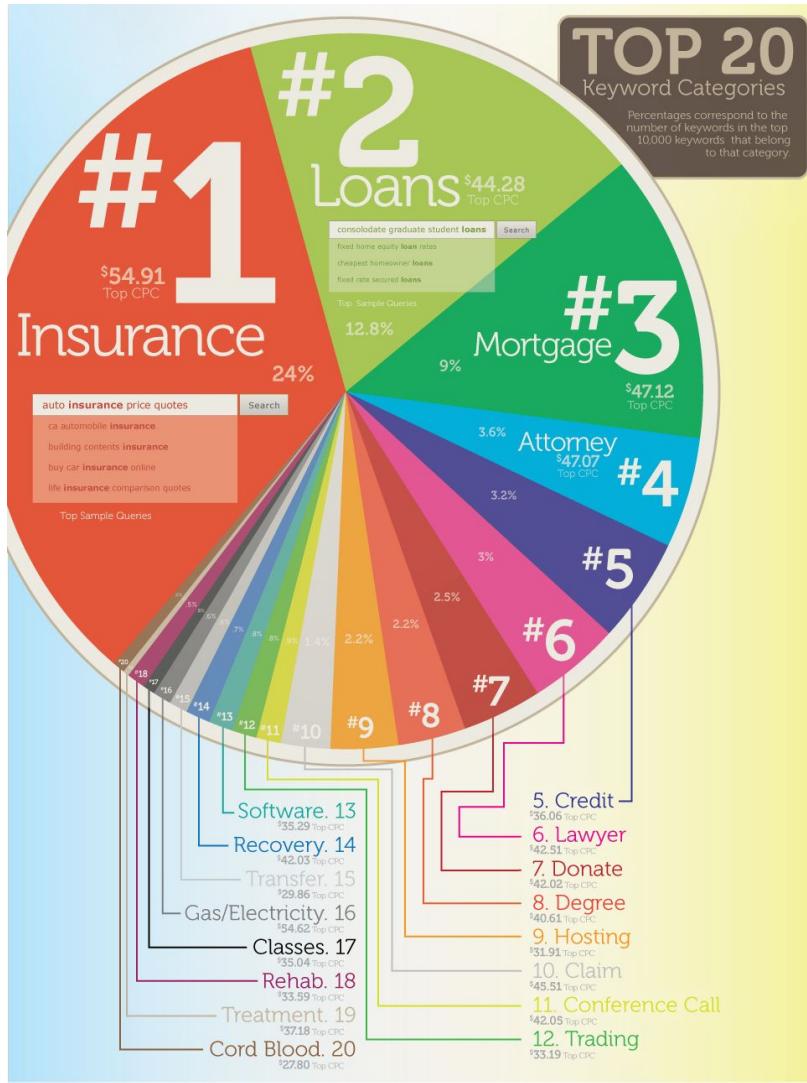


- Results in 1st position are clicked more than 10 times results on 6th position
- Suppose company A and company B are respectively on the 1st and 6th position for a valuable business keyword
- B should buy 10 times the clicks that A gets to obtain the same traffic!
- **How much does a click cost?**

[BMW given Google 'death penalty', 2006]



Click Economic Value



- In web advertising, the cost of a single click, also known as Cost-Per-Click (CPC), is **1\$ on average**. However...
 - On Google some keywords can be really expensive ([Google most expensive keywords 2011](#)):
 - Insurance: 54.91\$**
 - Mortgage: 47.12\$
 - Attorney: 47.07
 - Claim: 45.51\$
 - Loans: 44.28\$
 - Lawyer: 42.51\$
 - Bing is even more expensive ([Bing most expensive keywords 2015](#)):
 - Lawyers: 109.21\$**
 - Attorney: 101.77\$
 - Structured settlements: 78.39\$



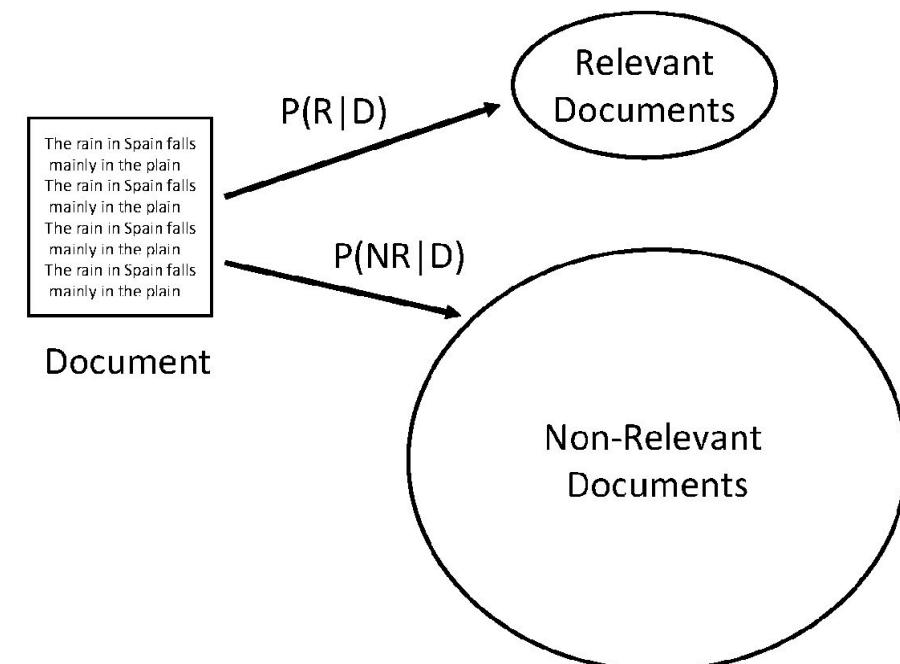
Retrieval Models

- **Classic models**
 - Boolean retrieval
 - Vector Space model
- **Probabilistic Models**
 - BM25
 - Language models
- Progress in retrieval models has corresponded with improvements in effectiveness

Probabilistic Models

- The Probability Ranking Principle in IR, [Robertson \(1977\)](#)
 - The ranking is given by a **probability of relevance**
 - Probability is estimated as accurately as possible using the available data → best effectiveness

“[...] If the **ranking of the documents is in order of decreasing probability of relevance** (where the probabilities are estimated as accurately as possible from the available data) **the overall effectiveness will be the best** that is obtainable on the basis of those data.”



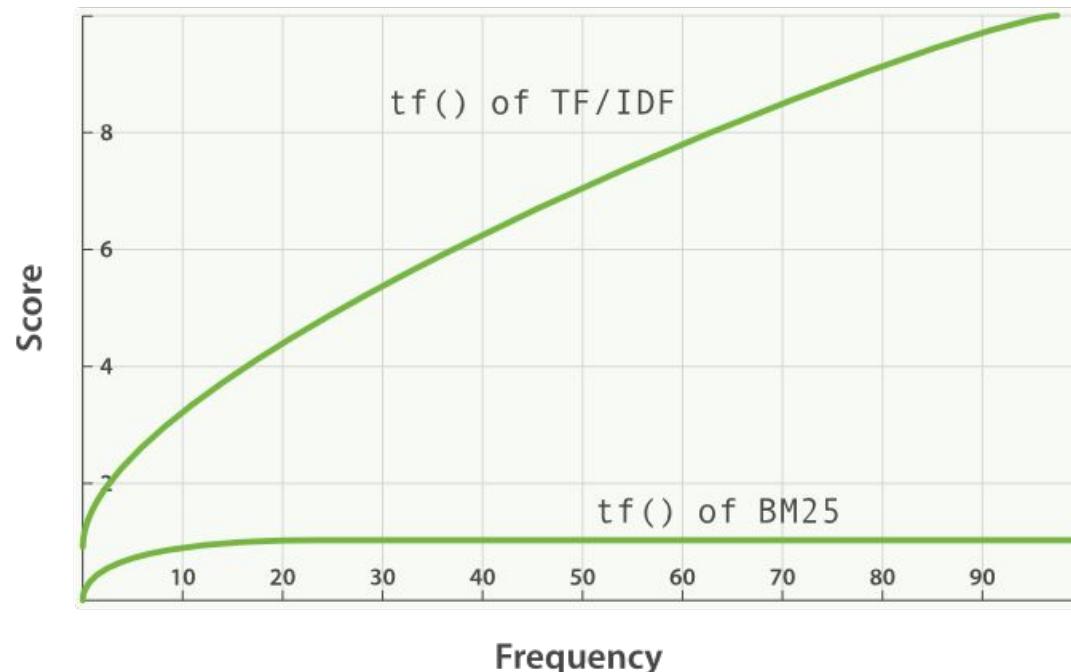


Okapi BM25

- The *BM25 weighting scheme*, often called *Okapi weighting*, after the system in which it was first implemented, was developed as a way of building a probabilistic model sensitive to
 - **term frequency**
 - **term rareness** in corpus (similarly to IDF)
 - **document length**
- BM25 became a popular and effective ranking algorithm while relying on probabilistic foundations
- The BM25 term weighting formulas have been used quite widely and quite successfully across a range of collections and search tasks

Okapi BM25: TF saturation

- In TF-IDF, a document similarity score could reach very high values if the term frequency of the query term is very high
- In Okapi BM25, the similarity score is always comprised between 0 and 1
 - At a certain point, highest values of term frequency do not affect the score





Okapi BM25: parameters

- Okapi BM25 has two parameters:
 - **k1**: this parameter controls **how quickly an increase in term frequency results in term-frequency saturation** (i.e., the slope of “ $tf()$ of BM25” curve)
 - The default value is 1.2
 - Lower values result in quicker saturation
 - Higher values in slower saturation
 - **b**: this parameter controls **how much effect document length normalization should have**
 - The default value is 0.75
 - A value of 0.0 disables normalization completely
 - A value of 1.0 normalizes fully



Language Model

- A statistical language model assigns a probability to a sequence of m words $P(w_1, \dots, w_m)$ by means of a probability distribution
- A separate language model is associated with each document in a collection. Documents are ranked based on the probability of the query Q in the document's language model $P(Q | M_d)$.
- Unigram language model
 - probability distribution over the words in a language
 - generation of text consists of pulling words out of a “bucket” according to the probability distribution and replacing them
- N-gram language model
 - some applications use bigram and trigram language models where probabilities depend on previous words



Efficiency Aspects of Ranking

- Computing the relevance score using non boolean models can be quite computationally-intensive:
 - Document is retrieved (has a relevance score) even if not all the query terms are found in it
 - Complex relevance formulas requires additional computations to compute score
- Strategies to control the efficiency:
 - The way we access documents affects computation time (one document at a time or one term at a time?)
 - Most of the time we are not interested in the least relevant documents



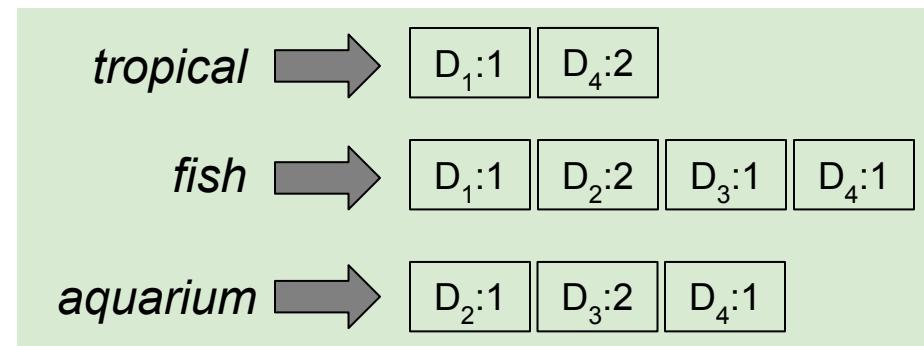
TAAT vs DAAT Techniques

- **TAAT** : “Term At A Time”
 - Scores for all docs computed concurrently, one query term at a time
- **DAAT** : “Document At A Time”
 - Total score for each doc (including all query terms) computed, before proceeding to the next
- Each has implications for how the retrieval **index** is structured and stored

TAAT: Example

Query: *tropical fish aquarium*

Inverted index:



Term frequency for each document:

$$D_1 = 1 + 1 = 2$$

$$D_2 = 2 + 1 = 3$$

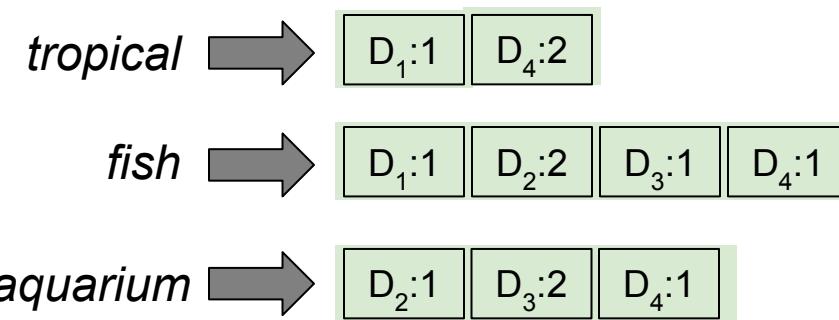
$$D_3 = 1 + 2 = 3$$

$$D_4 = 2 + 1 + 1 = 4$$

DAAT: Example

Query: *tropical fish aquarium*

Inverted index:



Term frequency for each document:

$$D_1 = 1 + 1 = 2$$

$$D_2 = 2 + 1 = 3$$

$$D_3 = 1 + 2 = 3$$

$$D_4 = 2 + 1 + 1 = 4$$



Efficient Scoring

- A large fraction of the CPU work for each search query is devoted just to **compute the score** of relevance
 - Generally, we have a tight budget on latency (e.g., 250ms) to answer the query, otherwise the user will not be happy
 - CPU provisioning doesn't permit exhaustively scoring every document for every query
- Basic idea: avoid scoring docs that won't make it into the top K



Safe Ranking

- The terminology “safe ranking” is used for methods that guarantee that the K docs returned are the **K absolute highest scoring documents**
 - (Not necessarily just under cosine similarity)
- Is it ok to be non-safe?
 - Yes, the results would be approximated but good enough to satisfy the user
- The goal is to run less computations while giving good results



Non-safe Ranking

- Non-safe ranking may be okay
 - Ranking function is only a proxy for user happiness
 - Documents close to top K may be just fine
- Non-safe: **Index “elimination”**
 - Only consider **high-idf (rare) query terms**, discard the common ones
 - Only consider **documents containing many query terms**, discard the ones with only few query terms
- Non-safe: **Champion lists**
 - For each term, the scores of top relevant documents are pre-computed (before any query is issued)

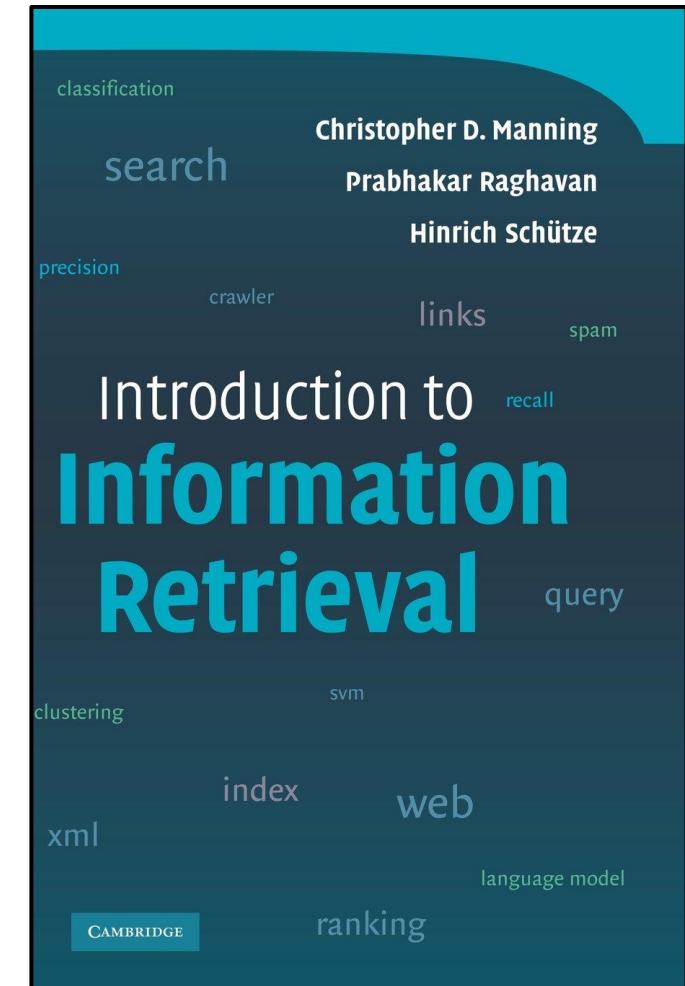


References

Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze
Introduction to Information Retrieval
Cambridge University Press. 2008

The book is also online for free:

- HTML edition (2009.04.07)
- PDF of the book for online viewing
(with nice hyperlink features,
2009.04.01)
- PDF of the book for printing
(2009.04.01)





Big Data and Data Mining

Web Information Retrieval

Flavio Bertini

flavio.bertini@unipr.it



Web Crawling

- Web crawling is the process by which **we gather pages from the Web**
- Goal: **quickly** and efficiently gather as many useful Web pages as possible, together with the **link structure** that interconnects them



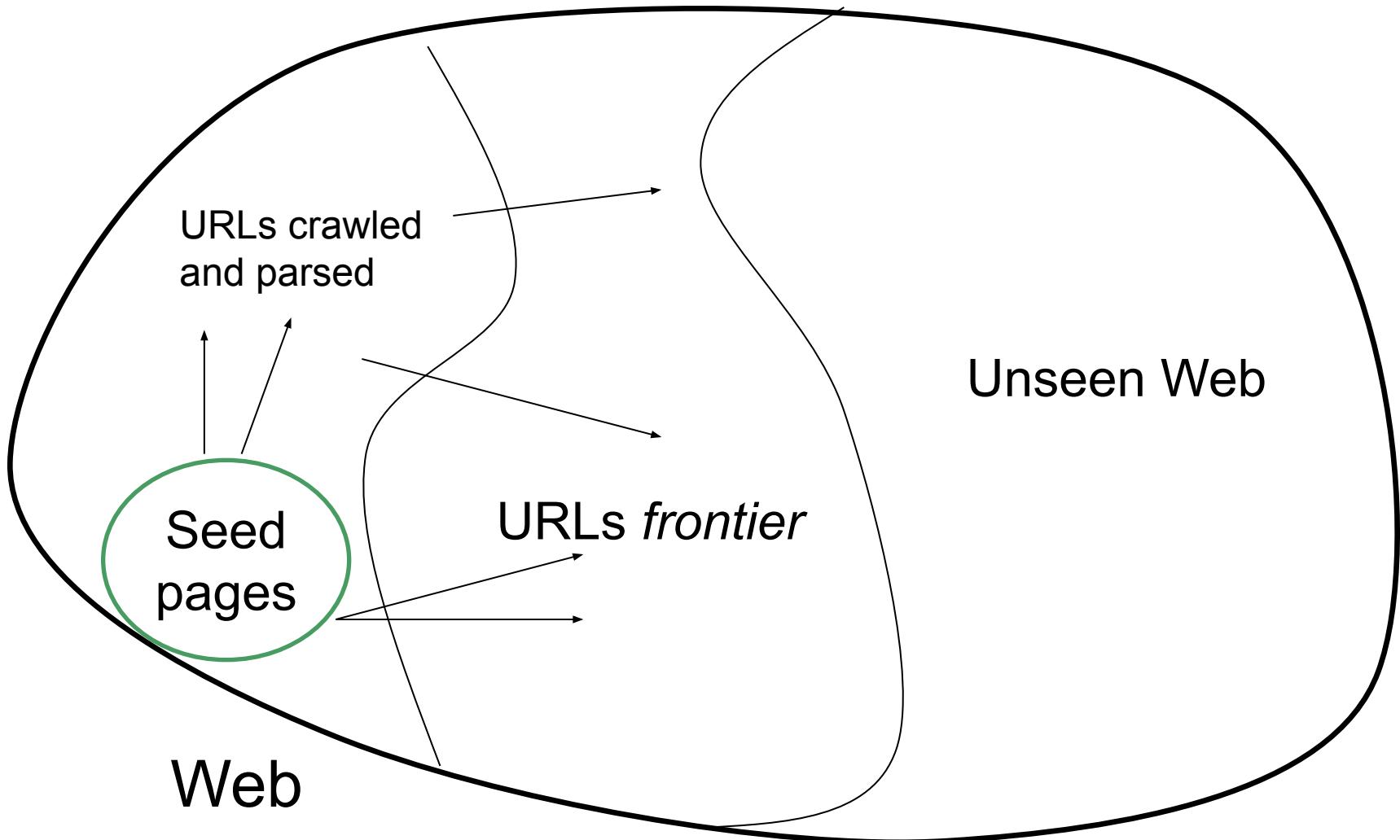


Basic Crawler Operation

- A crawler (a.k.a. spider)
 - Begin with known “seed” URLs
 - Fetch and parse them
 - Extract URLs they point to
 - Place the extracted URLs on a queue (the URLs *frontier*)
 - Fetch each URL on the frontier and repeat

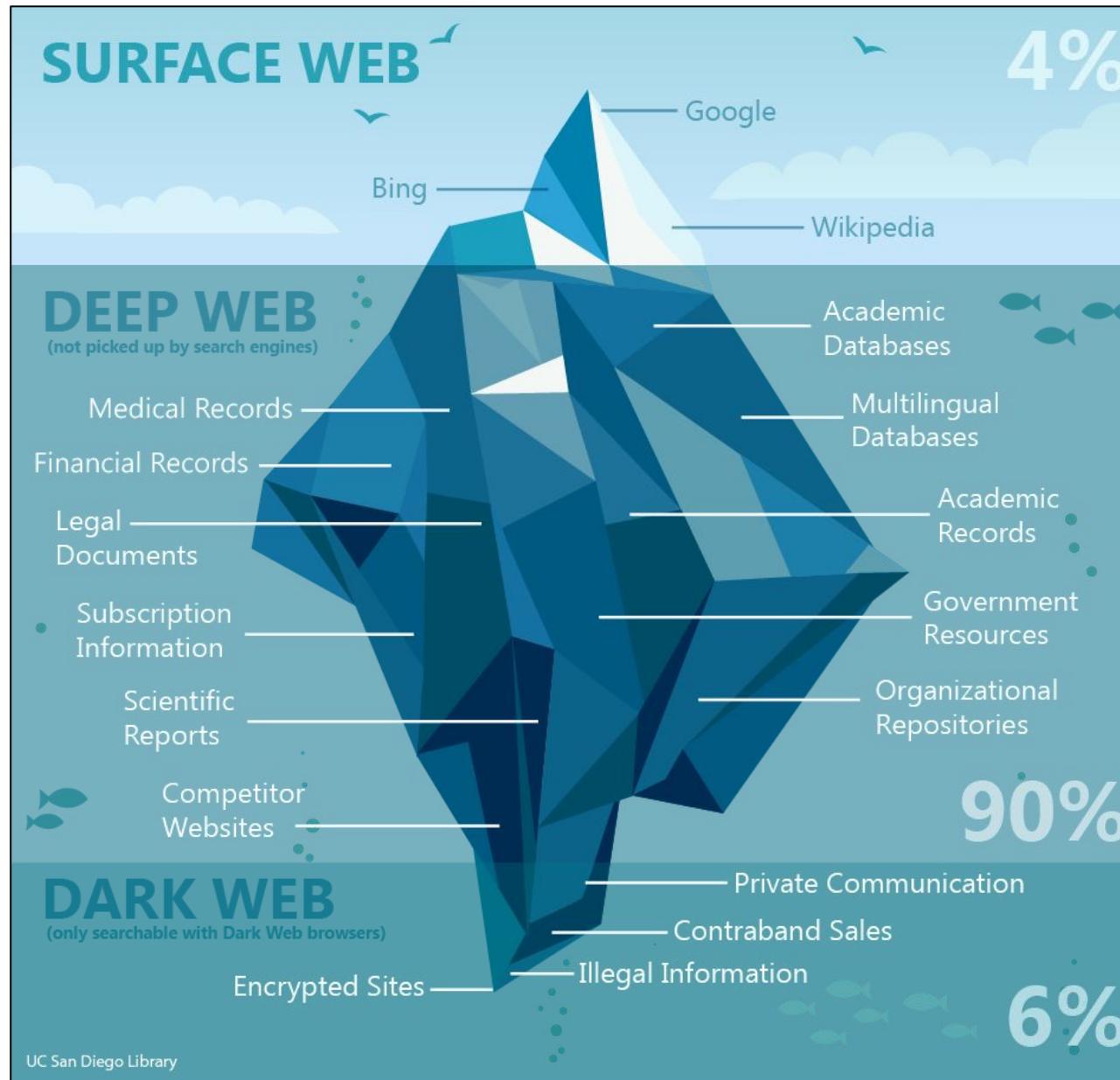


Crawling Picture





Web, Deep Web & Dark Web





Crawler requirements

- A crawler must respect some requirements:
 - **Robustness:** MUST avoid spider traps (fetching an infinite number of pages in a particular domain)
 - **Politeness:** MUST respect Web servers policies, regulating the rate at which crawlers can visit them



Robustness

- Web crawling isn't feasible with one machine
 - All of the above steps are distributed
- Malicious pages
 - Spam pages
 - Spider traps – including dynamically generated
- Even non-malicious pages pose challenges
 - Latency/bandwidth to remote servers vary
 - Webmasters specific guidelines
 - How “deep” should you crawl a site’s URL hierarchy?
 - Site mirrors and duplicate pages



Politeness

- **Explicit politeness:** specifications from webmasters on what portions of site can/cannot be crawled
 - robots.txt
- **Implicit politeness:** even with no specification, avoid hitting any site too often



Robots.txt

- Protocol for giving spiders (“robots”) limited access to a website, originally from 1994
 - www.robotstxt.org
- Website announces its request on what can (or cannot) be crawled
 - For a server, create a file named robots.txt
 - This file specifies access restrictions
- Robots.txt contains set of rules that **should** be followed by clients



Robots.txt: Example

- Example:
 - No robot should visit any URL starting with "/yoursite/temp/", except the robot called "searchengine":

User-agent: *

Disallow: /yoursite/temp/

} For all user-agents (client names) forbid access to directory /yoursite/temp

User-agent: searchengine

Disallow:

} For user-agents named "searchengine" do not forbid nothing, everything is thus accessible

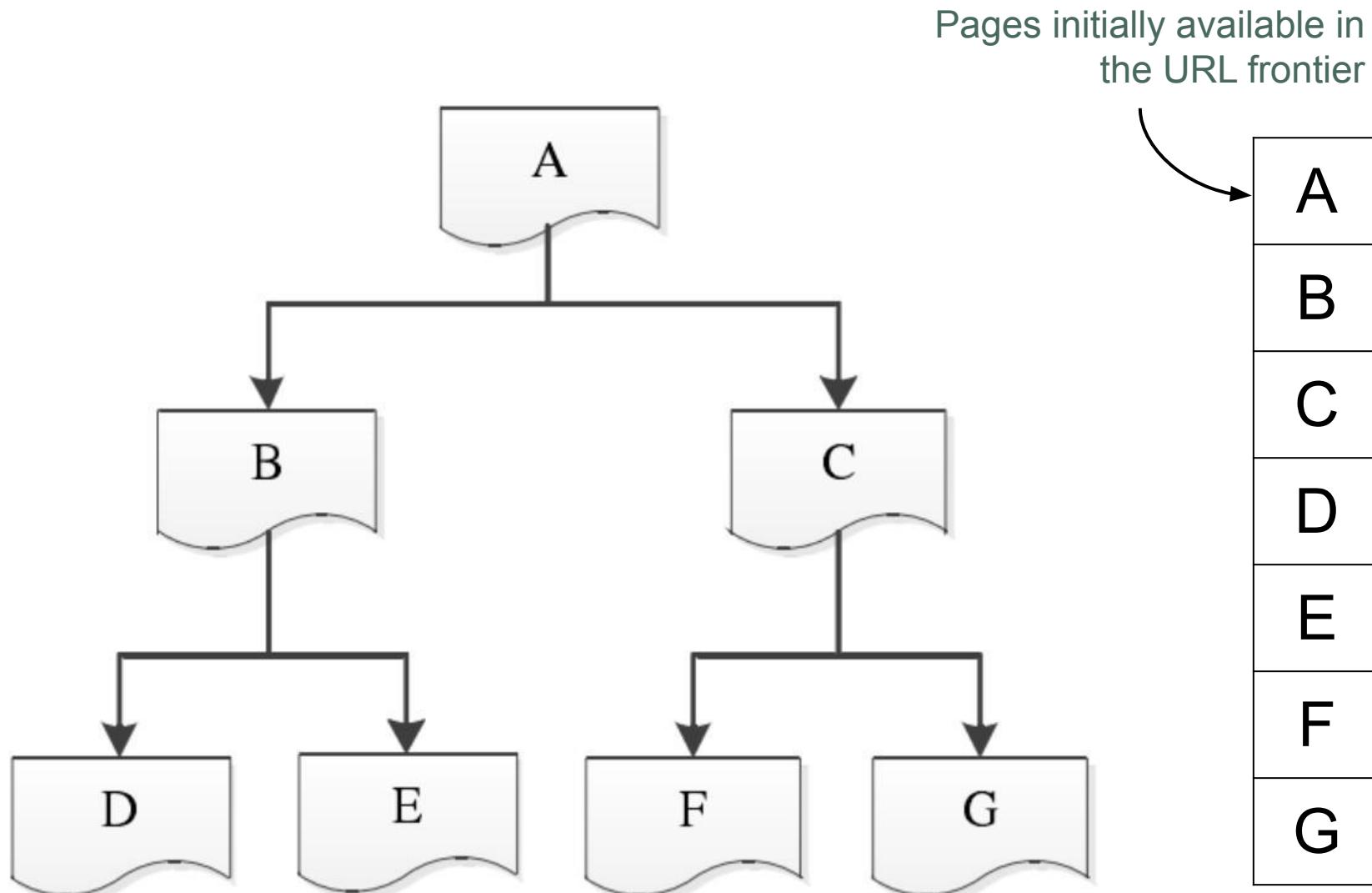


URL frontier

- Pages are **added** to the URL frontier according to the following strategies:
 - **Breadth first** strategy: given a Web page in the URL frontier, add **all pages linked by the current page**. Coverage is wide but superficial
 - **Depth first** strategy: given a Web page in the URL frontier, **follow the first link in the current page** until the first page without links



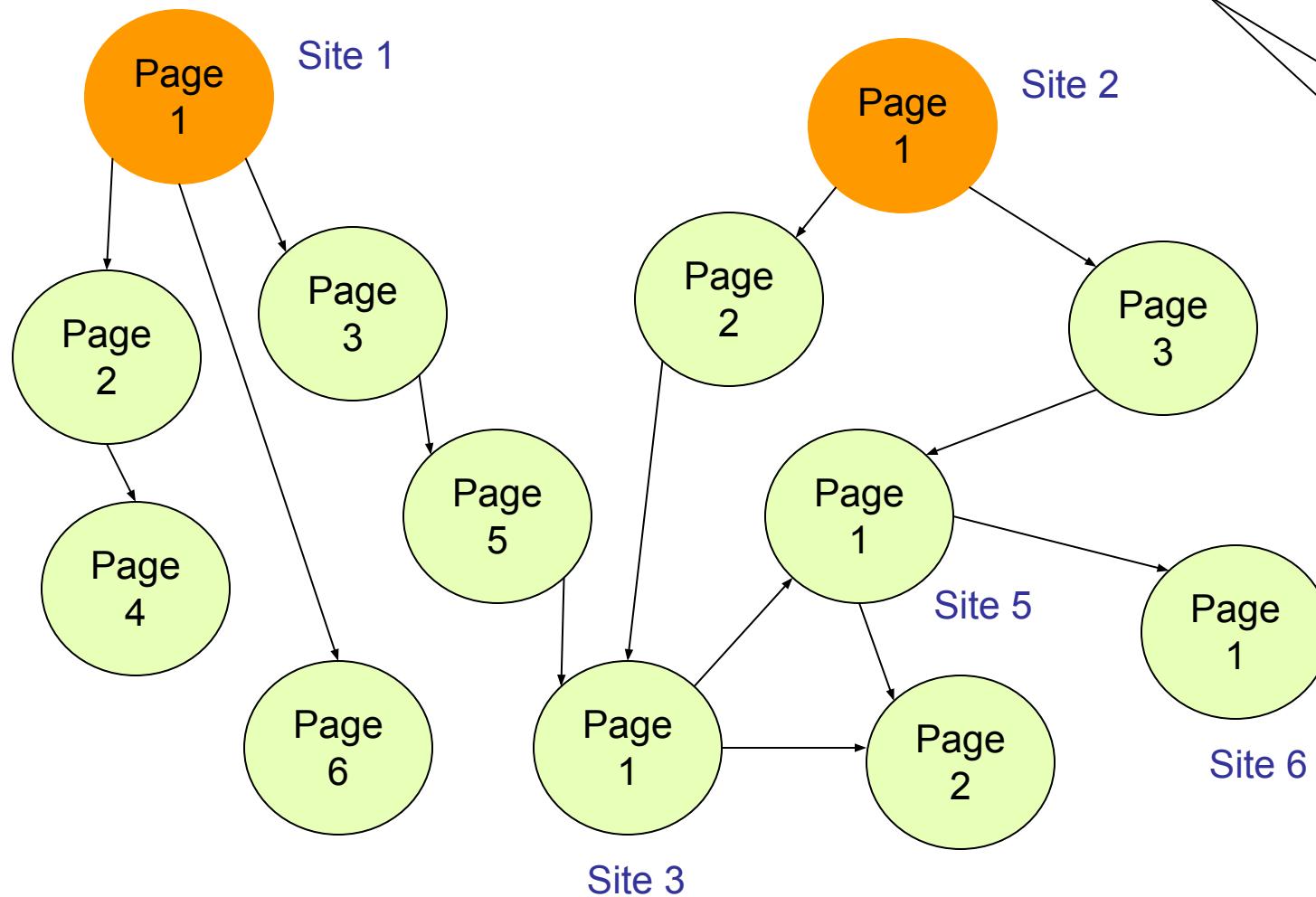
Breadth first strategy





URL frontier with BFS

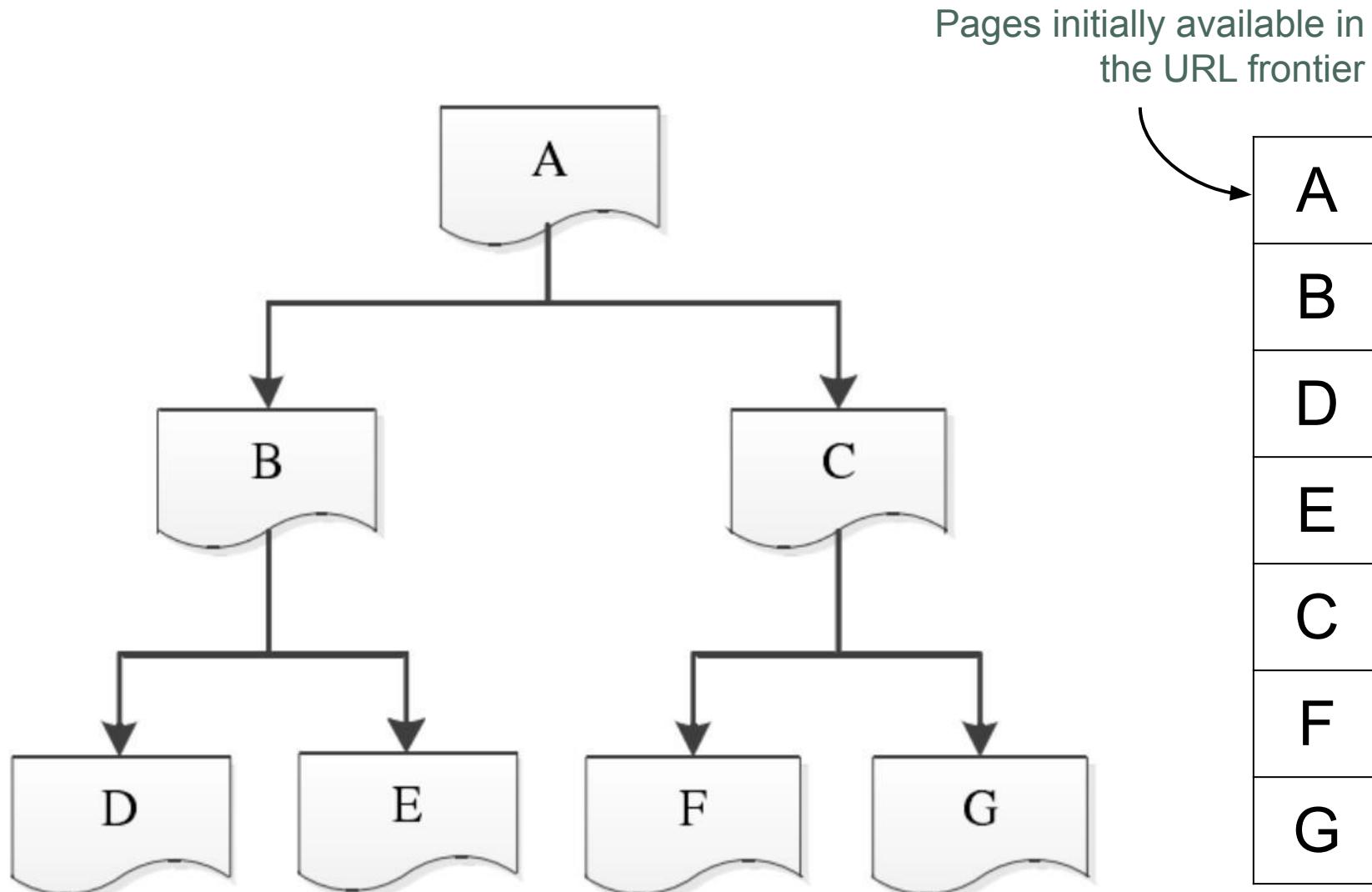
Pages initially available
in the URL frontier



Site	Page
1	1
2	1
1	2
1	6
1	3
2	2
2	3
1	4
1	5
3	1
5	1
5	2
6	1



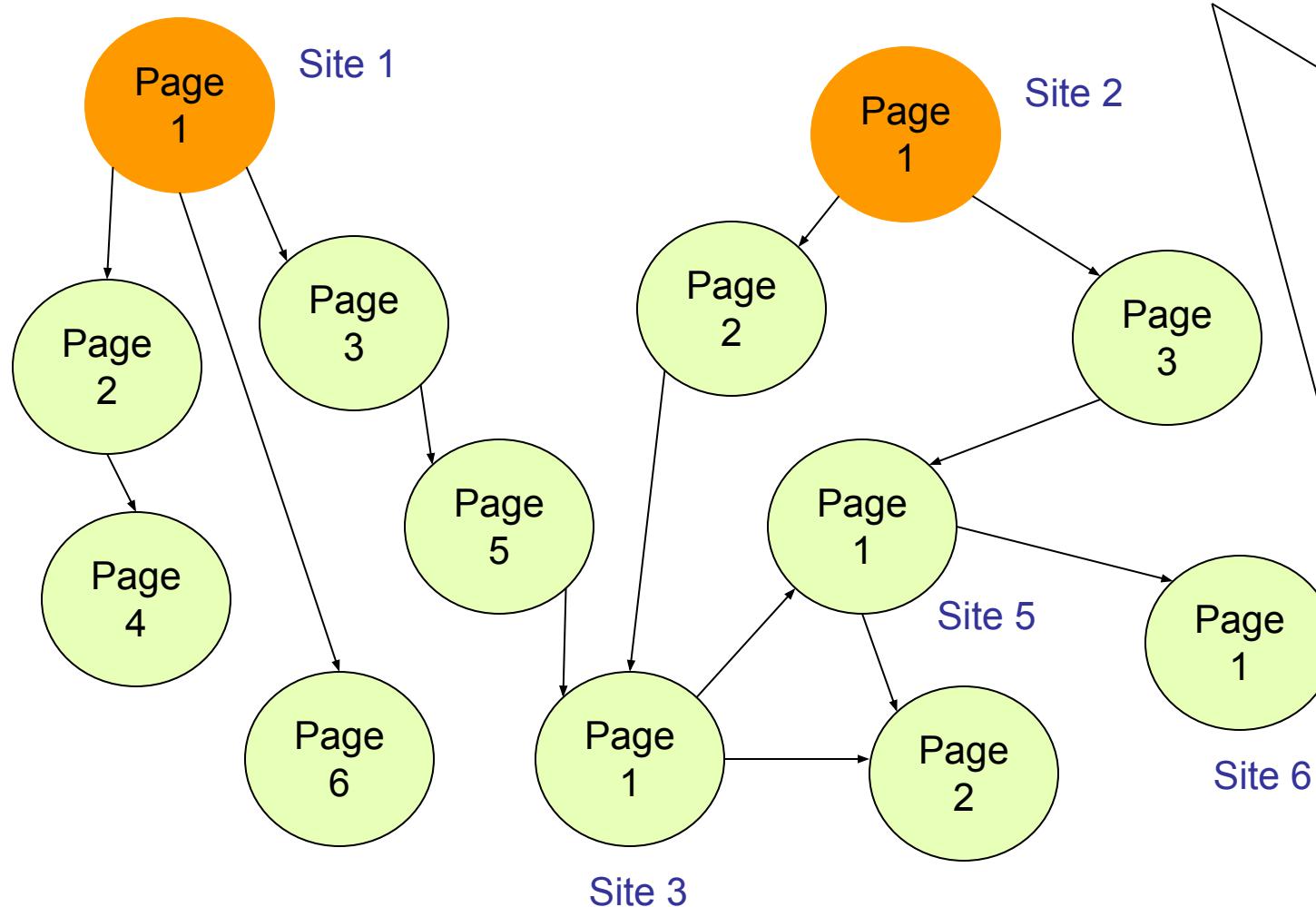
Depth first strategy





URL frontier with DFS

Pages initially available
in the URL frontier



Site	Page
1	1
1	2
1	4
1	6
1	3
1	5
3	1
5	2
5	1
6	1
2	1
2	2
2	3



BFS vs DFS

BFS frontier

Site	Page
1	1
2	1
1	2
1	6
1	3
2	2
2	3
1	4
1	5
3	1
5	1
5	2
6	1

DFS frontier

Site	Page
1	1
1	2
1	4
1	6
1	3
1	5
3	1
5	2
5	1
6	1
2	1
2	2
2	3



Searching the web

- There are thousands of billions of pages on the web, but most of them are not very interesting
- Suppose you have to visit the site for eBay and you don't know that www.ebay.com is the URL
 - There are millions of web pages that contain the term “eBay”
 - There can be websites with more frequency on the term “eBay” than eBay itself
- We need a notion of **popularity**, together with a notion of relevance

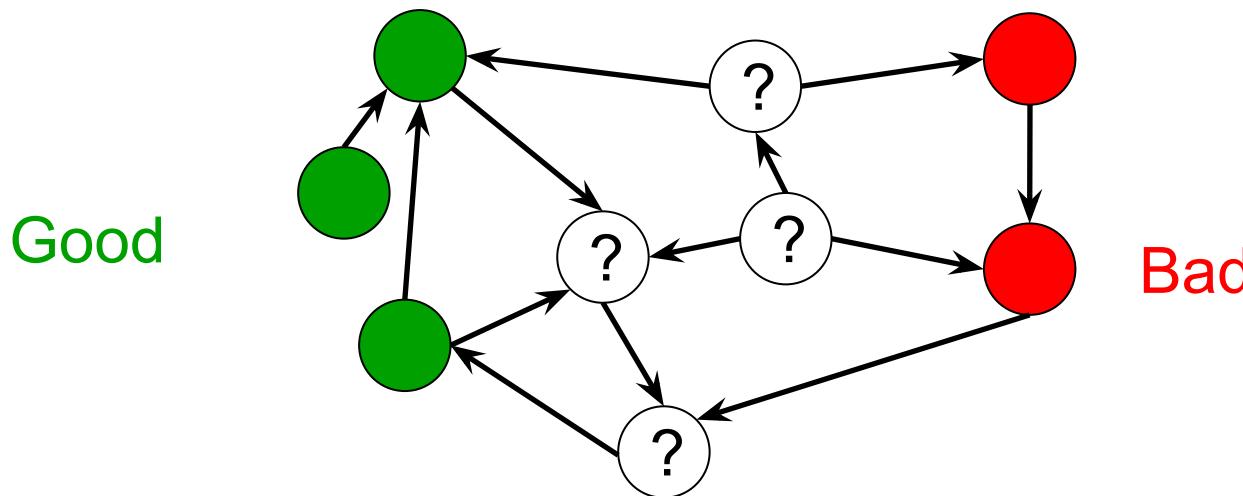


Web information retrieval

- With respect to traditional textual search engines, **Web information retrieval** systems build **ranking** by combining at least two evidences of relevance:
 - the degree of matching of a page: the **content score**
 - the degree of importance of a page: the **popularity score**
- While the **content score** can be calculated using one of the information retrieval models described so far
- The **popularity score** can be calculated from an analysis of the indexed pages' **hyperlink structure** using one or more ***link analysis*** models
 - Do the links represent a conferral of authority to some pages? Is this useful for ranking?

Simple link analysis

- Links are powerful sources of authenticity and authority
- The **Good**, The **Bad** and The **Unknown**, simple iterative logic
 - Good nodes won't point to Bad nodes
 - If you point to a Bad node, you're Bad
 - If a Good node points to you, you're Good





Citation Analysis

- Citation frequency is an estimation of a researcher popularity
- Bibliographic coupling frequency
 - Articles that co-cite the same articles are related
- Citation indexing: as a tool in journal evaluation
 - Who is this author cited by? ([Garfield 1972](#))
- PageRank preview: Pinski and Narin '70s*
 - Asked: which journals are authoritative?

*[Citation influence for journal aggregates of scientific publications: Theory, with application to the literature of physics](#)



PageRank

- The PageRank technique for link analysis assigns a **numerical score between 0 and 1** to every node in the web graph
- The PageRank score of a node depends on the **link structure** of the web graph
- Given a query, a web search engine computes a **composite score** for each web page that combines hundreds of features such as cosine similarity, together with the PageRank score
- This composite score is used to provide a **ranked list of results** for the query



The random surfer (1/3)

- Consider a *random surfer* Alice who begins a random walk on the web, starting from a page
 - Alice is extremely bored, she wanders aimlessly between web pages
 - Her browser has a special “**surprise me**” button at the top that will jump to a random web page when clicked
 - Each time a web page loads she chooses whether to
 - Click on a **random** link on the page
 - Click the surprise me button
 - Alice is sufficiently bored that she intends to keep browsing the Web like this forever



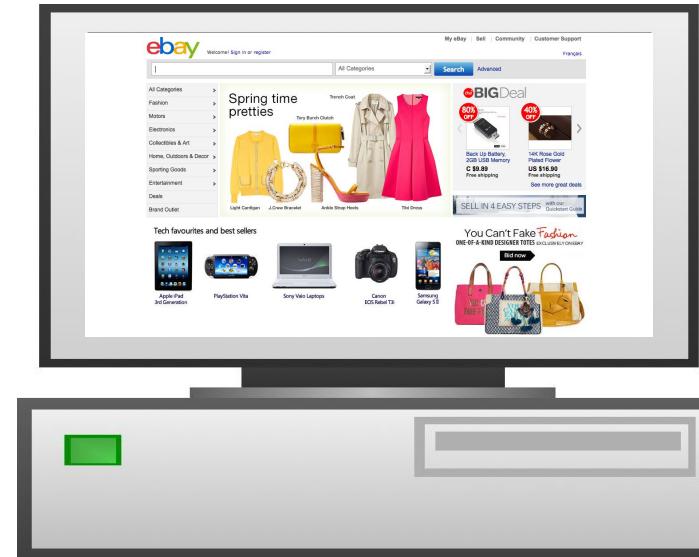
The random surfer (2/3)

- Let's provide a more formal definition: Alice browses the Web using this algorithm:
 1. Choose a random number r between 0 and 1
 2. If $r > \lambda$:
 ⇒ Click the “surprise me” button
 3. If $r \leq \lambda$:
 ⇒ Click a link at random on the current page
 4. Start again
- Because of Alice's special “surprise me” button, we can be guaranteed that eventually she will reach every page on the Internet



The random surfer (3/3)

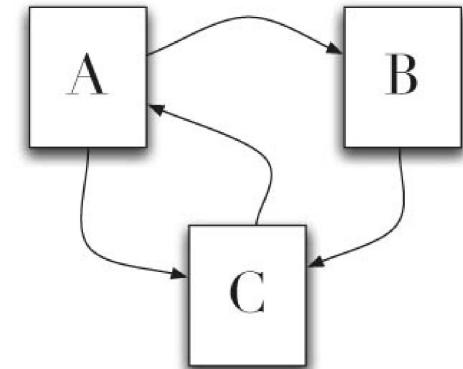
- Now suppose that while Alice is browsing, you walk in and glance at the web page on her screen. **What is the probability** that she will be looking at the eBay website?



- That probability is eBay's **PageRank**

PageRank calculation

- The PageRank calculation corresponds to finding the stationary probability distribution of a *random walk* on the graph of the Web. A random walk is a special case of a *Markov chain* in which the next state depends solely on the current state



- If the web consists of the 3 pages in figure (A,B,C), the PageRank of C depends on the PageRank of A and B:

$$PR(C) = \frac{PR(A)}{2} + \frac{PR(B)}{1}$$

- The PageRank conferred by an outbound link is equal to the document's own PageRank score divided by the number of outbound links
- We start by assuming that the PageRank values for all pages are the same, then we iterate the calculation. After few iterations, the PageRank values converge to the final values of
 - $PR(C) = 0.4$
 - $PR(A) = 0.4$
 - $PR(B) = 0.2$



Use of PageRank in Google

- PageRank is now **only one of the many** factors that determine the final score of a Web page in Google
- It is now a part of a much larger ranking system that it is believed to account for more than **200 different “signals”** (ranking variables):
 - **language features** (phrases, synonyms, spelling mistakes, etc.)
 - **query features** that relate to language features, trending terms/phrases
 - **time-related features** (e.g., “news” related queries might be best answered by recently indexed documents, while factual queries are better answered by more “resilient” pages)
 - **personalization features**, which relate to one’s search history, behavior, and social surrounding

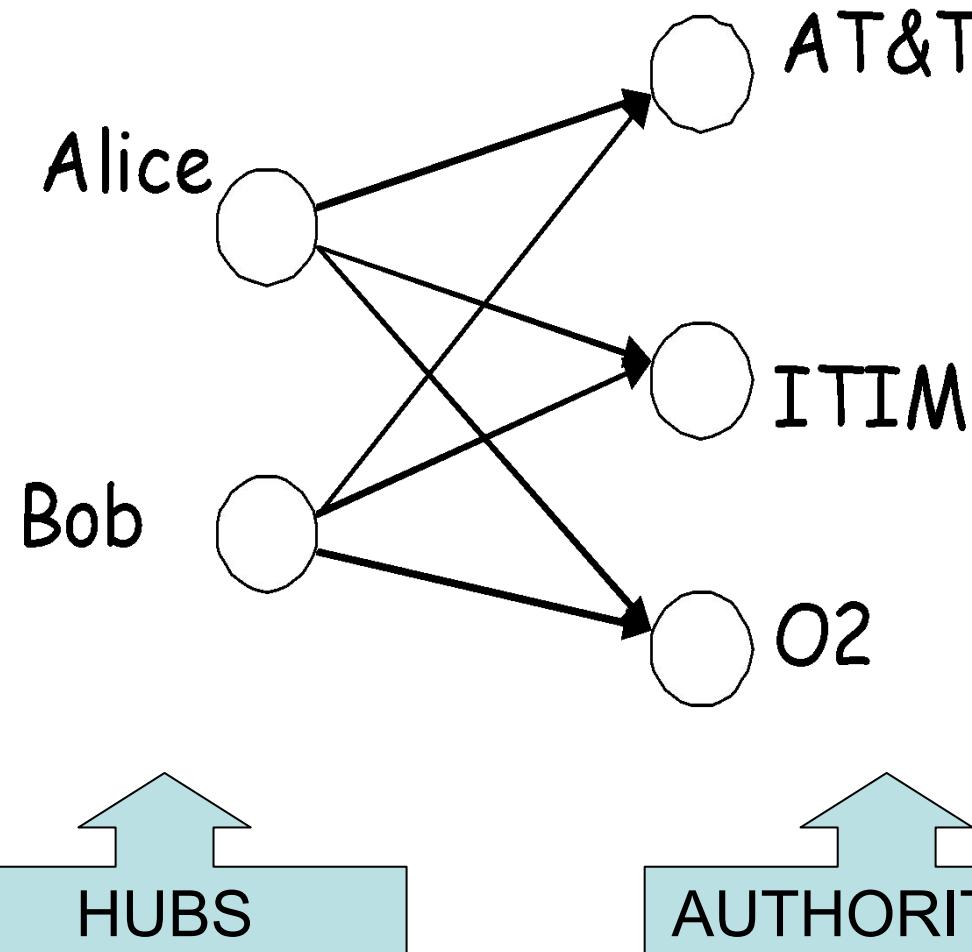


Hyperlink-Induced Topic Search (HITS)

- In response to a query, instead of an ordered list of pages each meeting the query, find **two sets of inter-related pages**:
 - ***Hub pages*** are good lists of links on a subject
 - e.g., “Bob’s list of cancer-related links.”
 - ***Authority pages*** occur recurrently on good hubs for the subject
- Best suited for “broad topic” queries rather than for page-finding queries
- Gets at a broader slice of common *opinion*

HITS Example

Query: “***Mobile telecom companies***”





Hubs and Authorities

- Thus, a **good hub page** for a topic *points* to many authoritative pages for that topic
- A **good authority page** for a topic is *pointed* to by many good hubs for that topic
- Circular definition - will turn this into an iterative computation



Semantic Search

- The name “information retrieval” is standard, but as traditionally practiced, it’s not really right
- All you get is **document retrieval**, and beyond that the job is up to you
- **Semantic Search:** doing graph search over structured knowledge rather than traditional text search:
 - Google Knowledge Graph
 - Facebook Graph Search
 - Bing’s Satori
 - Things like Wolfram Alpha





References

- Bruce Croft, Donald Metzler, Trevor Strohman **Search Engines: Information Retrieval in Practice** Pearson (2010)
- Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze **Introduction to Information Retrieval** Cambridge University Press. (2008)
- Stefano Ceri, Alessandro Bozzon **Web Information Retrieval** Springer. (2013)



Big Data and Data Mining

Information Retrieval Evaluation

Flavio Bertini

flavio.bertini@unipr.it



Measuring Relevance

- Methods pioneered by Cyril Cleverdon (a British librarian and computer scientist) in the *Cranfield Experiments* in the 1960s
- Three elements:
 1. A benchmark **document collection**
 2. A benchmark suite of **queries**
 3. A **human assessment** (relevance judgments) of either Relevant or Nonrelevant for each query and each document



Cyril Cleverdon



Assessments

- Note: **user need** is translated into a **query**
- Relevance is assessed relative to the **user need**, *not* the **query**, for example:
 - Information need: *My swimming pool bottom is becoming black and needs to be cleaned*
 - Query: **pool cleaner**
- Assess whether the doc addresses the underlying need, not whether it has these words



Relevance judgments

- Binary (relevant vs. non-relevant) in the simplest case, or more precisely (0, 1, 2, 3 ...) in others
- If, for each query, we consider all the set of documents to be judged, the relevance assessment can be huge and expensive
- The depth-**k** pooling solution:
 - Take in consideration the top-**k** (e.g. 100) documents of **N** (e.g. 100) different information retrieval systems
 - Humans must judge a “pool” of no more than $k \times N$ documents (e.g. 10'000), which is far less than the entire document collection (could be millions of documents)



Qualified Test Collections

<i>Collection</i>	<i>NDocs</i>	<i>NQrys</i>	<i>Size (MB)</i>	<i>Term/Doc</i>	<i>Q-D RelAss</i>
ADI	82	35			
AIT	2109	14	2	400	>10,000
CACM	3204	64	2	24.5	
CISI	1460	112	2	46.5	
Cranfield	1400	225	2	53.1	
LISA	5872	35	3		
Medline	1033	30	1		
NPL	11,429	93	3		
OSHMED	34,8566	106	400	250	16,140
Reuters	21,578	672	28	131	
GOV2	25 Mln	10,000	426,000	956	1,000

Typical
TREC



TREC Collections

Text REtrieval Conference (TREC)

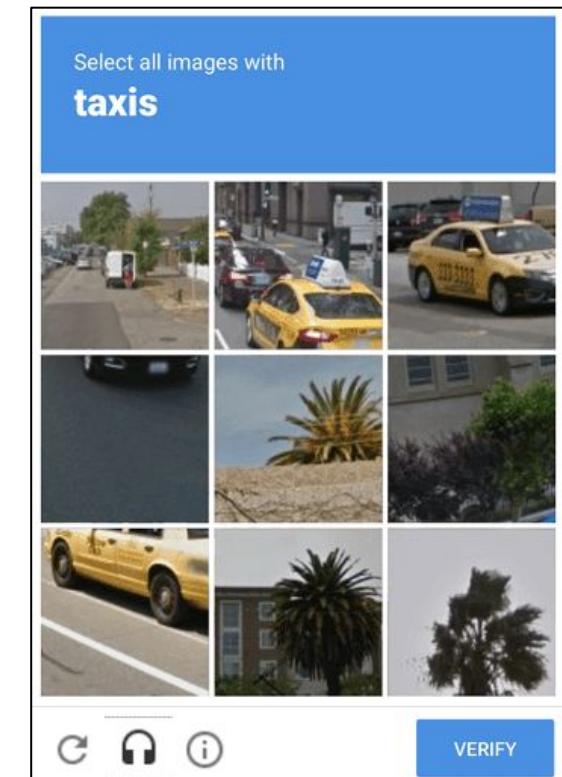
... to encourage research in information retrieval
from large text collections.

- The U.S. *National Institute of Standards and Technology* (NIST) has run a large IR test bed evaluation series since 1992. Within this framework, there have been **many tracks** over a range of **different test collections**
- TREC GOV2 is now the largest Web collection easily available for research purposes, including 25 million pages



Mechanical Turk

- Present query-document pairs to low-cost labor on online crowd-sourcing platforms
 - Hope that this is cheaper than hiring qualified assessors
[Amazon Mechanical Turk](#)
- Lots of literature on using crowd-sourcing for such tasks
 - From Carnegie Mellon OCR to Google login captcha
- Main takeaway – you get some signal, but the variance in the resulting judgments is very high
 - MIT Technology Review, [Death to captchas](#)





Confusion matrix

	True condition			
Total population	Condition positive	Condition negative	Prevalence $= \frac{\sum \text{Condition positive}}{\sum \text{Total population}}$	Accuracy (ACC) = $\frac{\sum \text{True positive} + \sum \text{True negative}}{\sum \text{Total population}}$
Predicted condition	True positive	False positive, Type I error	Positive predictive value (PPV), Precision = $\frac{\sum \text{True positive}}{\sum \text{Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\sum \text{False positive}}{\sum \text{Predicted condition positive}}$
Predicted condition positive	True positive	False positive, Type I error	Positive predictive value (PPV), Precision = $\frac{\sum \text{True positive}}{\sum \text{Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\sum \text{False positive}}{\sum \text{Predicted condition positive}}$
Predicted condition negative	False negative, Type II error	True negative	False omission rate (FOR) = $\frac{\sum \text{False negative}}{\sum \text{Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\sum \text{True negative}}{\sum \text{Predicted condition negative}}$
	True positive rate (TPR), Recall, Sensitivity, probability of detection, Power $= \frac{\sum \text{True positive}}{\sum \text{Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm $= \frac{\sum \text{False positive}}{\sum \text{Condition negative}}$	Positive likelihood ratio (LR+) $= \frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) $= \frac{\text{LR+}}{\text{LR-}}$
	False negative rate (FNR), Miss rate $= \frac{\sum \text{False negative}}{\sum \text{Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) $= \frac{\sum \text{True negative}}{\sum \text{Condition negative}}$	Negative likelihood ratio (LR-) $= \frac{\text{FNR}}{\text{TNR}}$	$F_1 \text{ score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$

[source: [Wikipedia](#)]

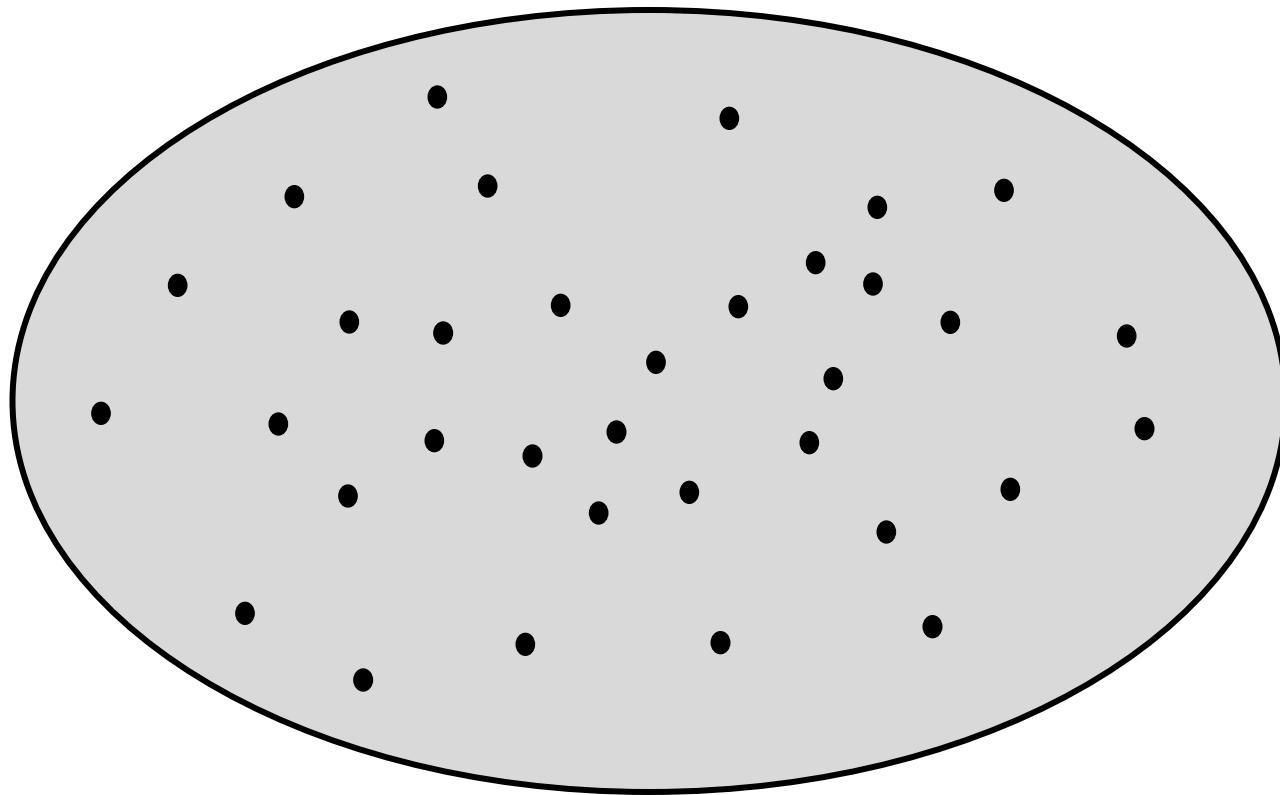


Effectiveness measures

- To assess the *effectiveness* of an IR system (the quality of its search results), there are two parameters about the system's returned results for a query:
 - **Precision:** What fraction of the **returned** documents are relevant to the information need?
 - **Recall:** What fraction of the **relevant** documents in the collection were returned by the system?

Collection of documents

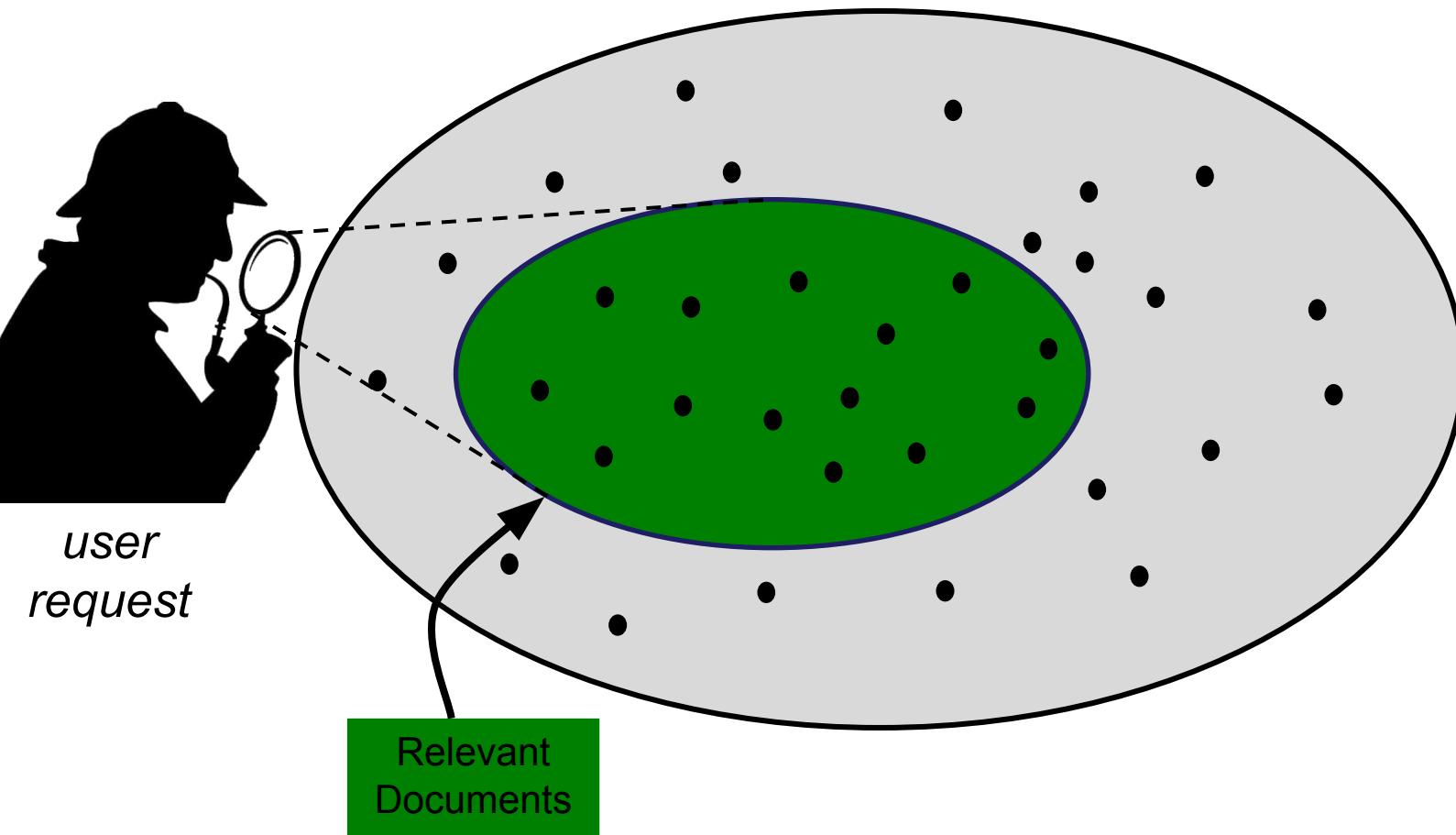
Each dot • is a document of the collection





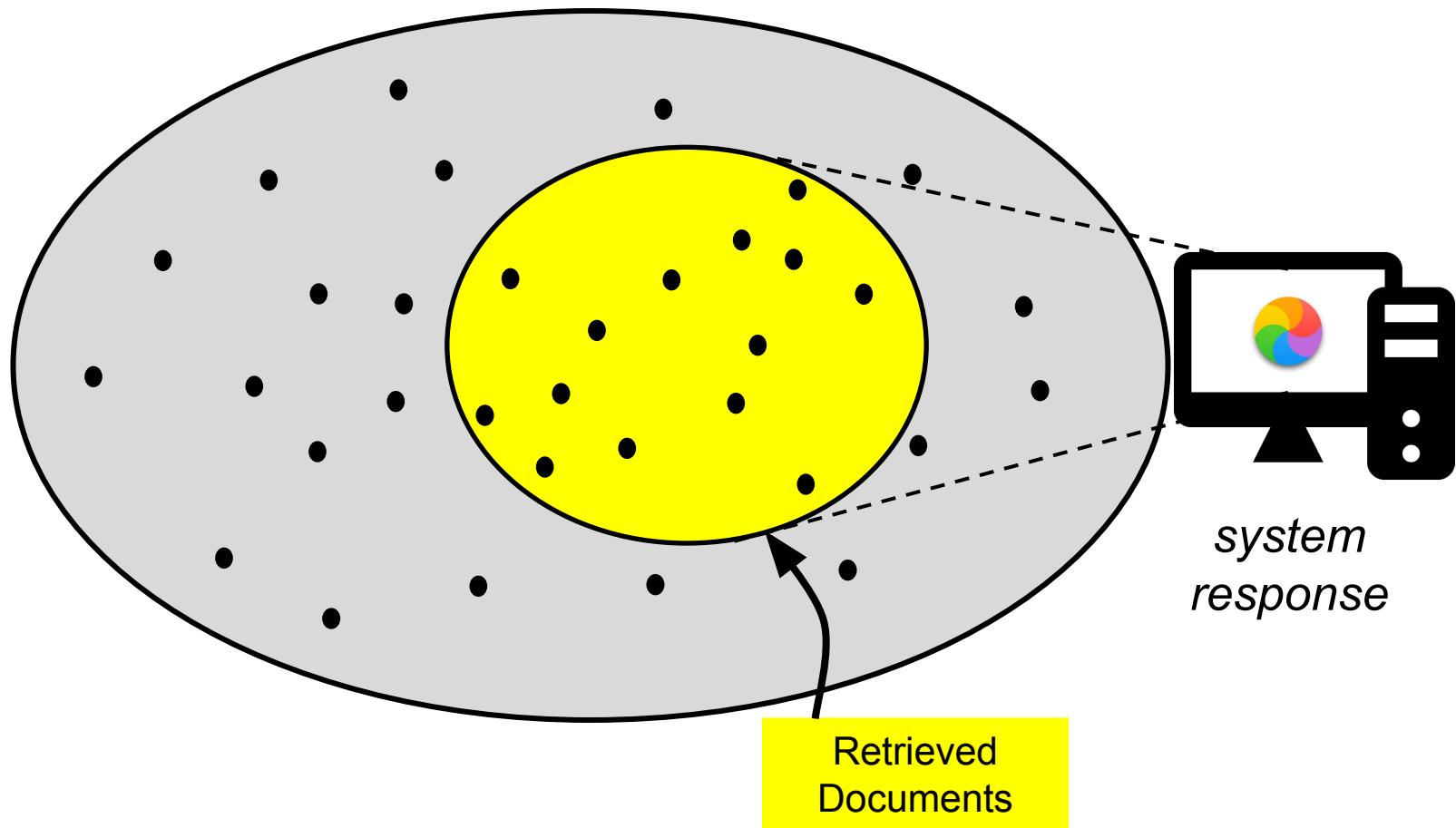
Relevant documents

Given a query, the relevant documents is the set of all the documents **really relevant** to the query



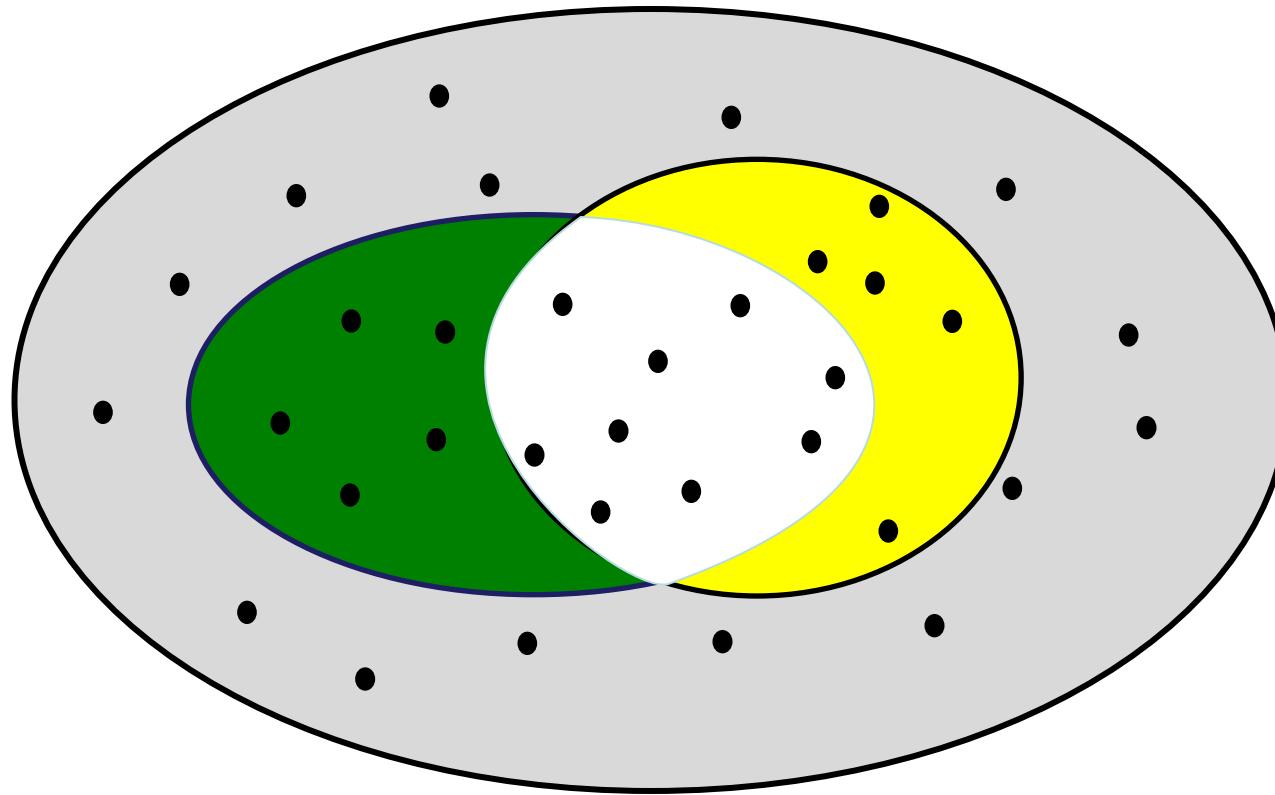
Retrieved documents

Given the same query, the Retrieved documents is the set of all the documents **returned by the system** we want to **evaluate**





Relevant retrieved documents



Relevant Retrieved
Documents

=

Relevant
Documents

∩

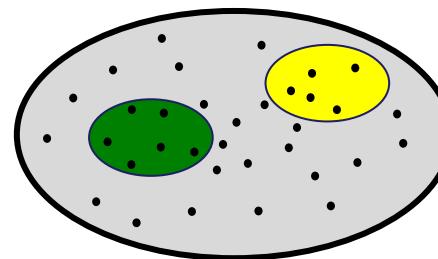
Retrieved
Documents

Precision

$$\text{Precision} = \frac{\text{Relevant Retrieved Documents}}{\text{Retrieved Documents}}$$

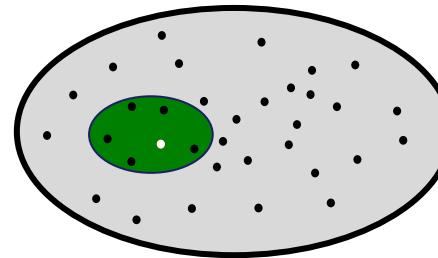
*The accuracy of positive predictions.
Range of values [0, 1]*

- Relevant Retrieved Documents = \emptyset relevant documents \Rightarrow Precision = 0



The worst IR system

- Retrieved Documents = One relevant document \Rightarrow Precision = 1



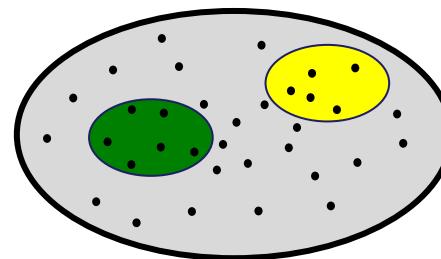
The miraculous IR system

Recall

$$\text{Recall} = \frac{\text{Relevant Retrieved Documents}}{\text{Relevant Documents}}$$

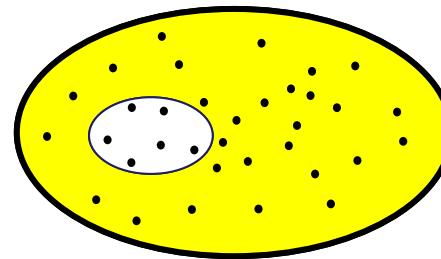
*The proportion of positives
that are correctly identified.
Range of values [0, 1]*

- Relevant Retrieved Documents = \emptyset relevant documents \Rightarrow Recall = 0



The worst IR system

- Retrieved Documents = All documents \Rightarrow Recall = 1

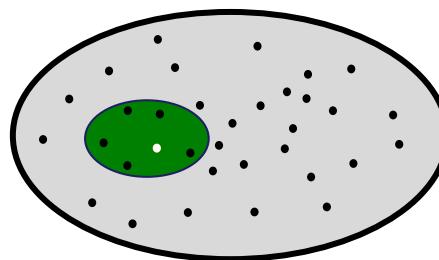


The naive IR system

Precision Recall Extremes

■ The miraculous IR system

Relevant Retrieved Documents = One relevant document



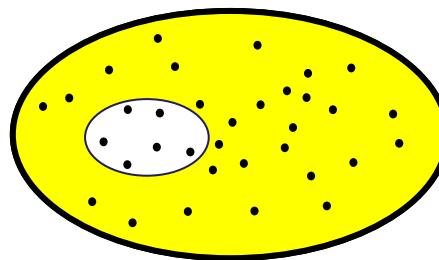
Precision = 1

Recall is very low!

Just one relevant document

■ The naive IR system

Retrieved Documents = All the documents



Recall = 1

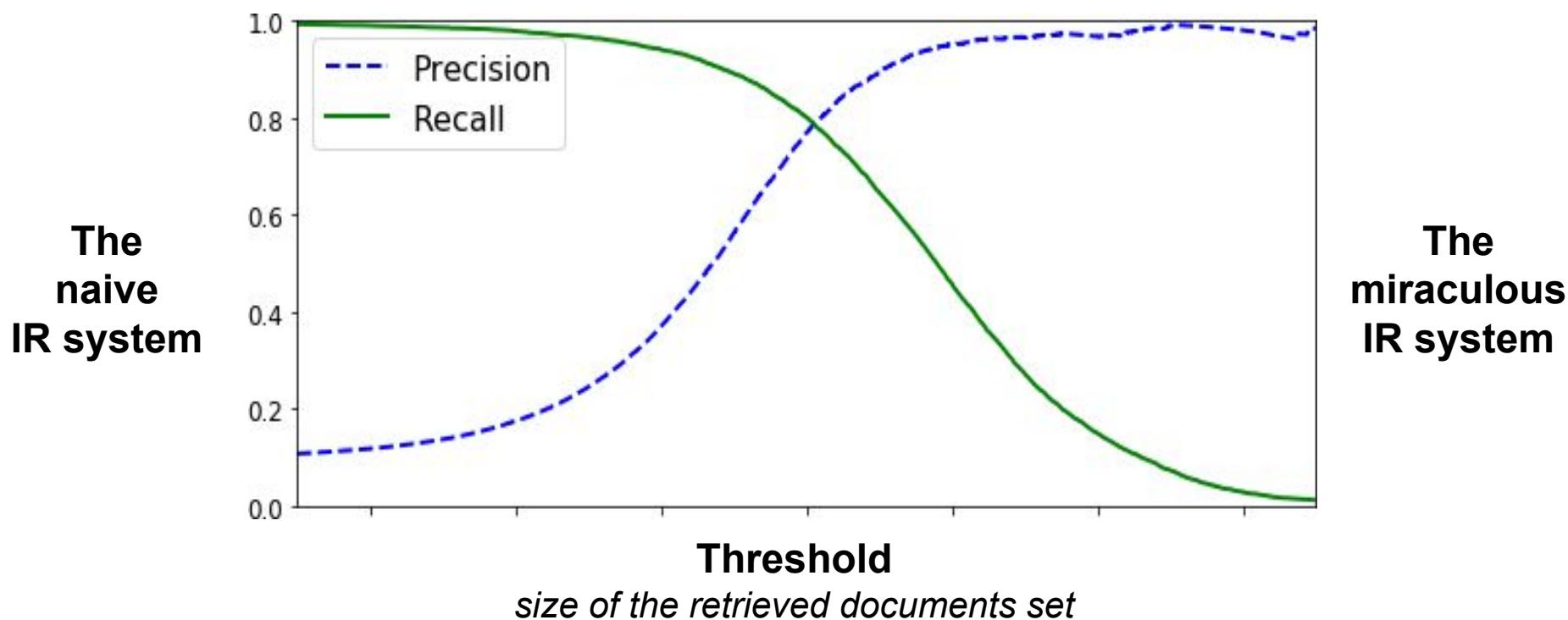
Precision is very low!

A lot of irrelevant documents

Precision Recall Tradeoff

In an **ideal scenario** where there is a perfectly separable data, both **precision and recall can get maximum value of 1**

Unfortunately, in most of the practical situations, you can't have both precision and recall high. **If you increase precision, it will reduce recall, and vice versa**





F-Measure

Alone, neither precision or recall tells the whole story. We can have excellent precision with terrible recall, or alternately, terrible precision with excellent recall

F-Measure combines both into a single measure that **provides a overall evaluation of the accuracy of the IR system.** The traditional F-Measure is calculated as follows:

$$\text{F-Measure} = \frac{(2 * \text{Precision} * \text{Recall})}{(\text{Precision} + \text{Recall})}$$

This is the **harmonic mean a.k.a. F-Score or the F1-Score**. The more generic F_β score applies additional weights (β), valuing one of precision or recall more than the other



Evaluation of Ranked Results

- **Precision, recall, and the F-measure** are set-based measures **for unordered sets of documents**
- **We need to extend these measures if we are to evaluate the ranked retrieval results** that are now standard with **search engines**, where what matters is how many **good results there are on the first page**
- In a ranked retrieval context, appropriate sets of retrieved documents are naturally given by the **top k retrieved documents**
- This leads to measuring precision and recall at a fixed low level of retrieved results, that is the k documents. This is referred to as **Precision @ K** and **Recall @ K** (a.k.a. P@K and R@K)



Precision @ K

- Set a rank threshold K: the number of retrieved documents
- Compute the proportion of the top k returned documents that are relevant
- Ignores documents ranked lower than K

Ranked results list

	#1 is relevant
	#2 is not relevant
	#3 is relevant
	#4 is not relevant
	#5 is relevant

Precision* at different K

P@1: 1/1 = 1.00
P@2: 1/2 = 0.50
P@3: 2/3 = 0.67
P@4: 2/4 ...
...

*Precision = Relevant Retrieved / Retrieved

Recall @ K

- Set a rank threshold K: the number of retrieved documents
- Compute the proportion of relevant items found in the top k returned documents
- Ignores documents ranked lower than K

Ranked results list

	#1 is relevant
	#2 is not relevant
	#3 is relevant
	#4 is not relevant
	#5 is relevant

Recall* at different K

R@1: 1/3 = 0.33
R@2: 1/3 = 0.33
R@3: 2/3 = 0.67
R@4: 2/3 ...
...

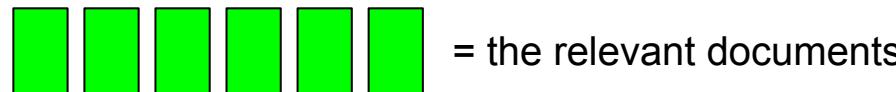
*Recall = Relevant Retrieved / Relevant



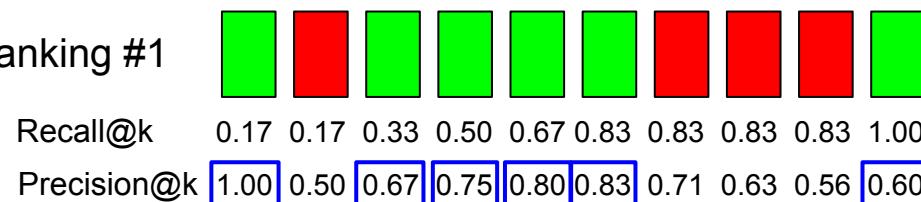
Average Precision (AP)

- As for the F-Measure, the need to use an aggregate value is even stronger with a variable value of K
- **Average Precision is an aggregated measure for ranked results**
- It is computed as follows:
 - Instead of setting an arbitrary K, we **stop only when all the relevant documents** are retrieved: that value for K for which **Recall @K is equal to 1**
 - We **compute the Precisions @K only for those K where relevant result is retrieved**
 - The **average** of this precision measures is the **Average Precision (AP)**

AP example

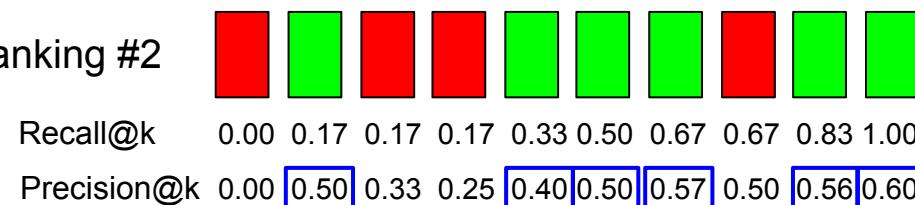


IR system 1 ⇒ Ranking #1



We stop
when
Recall@k=1

IR system 2 ⇒ Ranking #2



We stop
when
Recall@k=1

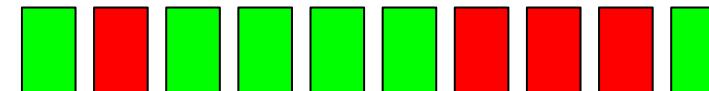
Compute AP using P@K only for those K where relevant result is retrieved

$$\text{Ranking } \#1: (1.00 + 0.67 + 0.75 + 0.80 + 0.83 + 0.60)/6 = 0.78$$

$$\text{Ranking } \#2: (0.50 + 0.40 + 0.50 + 0.57 + 0.56 + 0.60)/6 = 0.52$$

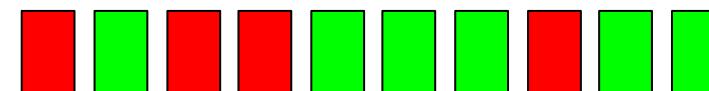
AP observations

IR system 1 ⇒ Ranking #1



1.00 = Recall @ 10
0.60 = Precision @ 10

IR system 2 ⇒ Ranking #2



1.00 = Recall @ 10
0.60 = Precision @ 10

- Recall @ 10 and Precision @ 10 is equal for the two rankings
- However, AP is able to capture that Ranking #1 is better, as it ranks **more relevant documents in higher positions**
 - AP IR system 1 = 0.78
 - AP IR system 2 = 0.52



Mean Average Precision (MAP)

- When evaluating a system we usually measure the effectiveness over **more than one query** (10,000 in GOV2 TREC collection)
- The **number of queries** is another **dimension of aggregation** of the measures to evaluate the IR system
- After computing the Average Precision of each query in the test collection, the **Mean Average Precision (MAP)** is the average of the Average Precision over all the queries

MAP example



= relevant documents for **Query 1**

Ranking #1



Recall@k 0.20 0.20 0.40 0.40 0.40 0.60 0.60 0.60 0.80 1.00

Precision@k 1.00 0.50 0.67 0.50 0.40 0.50 0.43 0.38 0.44 0.50



= relevant documents for **Query 2**

Ranking #2



Recall@k 0.00 0.33 0.33 0.33 0.67 0.67 0.67 1.00

Precision@k 0.00 0.50 0.33 0.25 0.40 0.33 0.43

Compute MAP as the average of the Average Precision over all the queries

AP query 1: $(1.00 + 0.67 + 0.50 + 0.44 + 0.50)/5 = 0.62$

AP query 2: $(0.50 + 0.40 + 0.43)/3 = 0.44$

MAP: $(0.62 + 0.44)/2 = 0.53$



MAP observations

- MAP assumes user is **interested in finding many relevant documents for each query**
 - If a relevant document never gets retrieved, we assume the precision corresponding to that relevant doc to be zero
- MAP is macro-averaging: **each query counts equally**
- There is normally **more agreement in MAP for an individual information need across systems** than for MAP scores for different information needs for the same system



Beyond binary relevance

- We assumed a **binary notion of relevance**:
 - either a document is relevant to the query or
 - it is non relevant to the query
- Some documents can be **less relevant** than others, but still relevant (non binary notion)
 - Specific measure with non-binary assessments: **DCG** (Discounted Cumulative Gain) or **NDCG** (Normalized Discounted Cumulative Gain)
- Binary relevance is still more common and provide a good estimation for IR evaluation

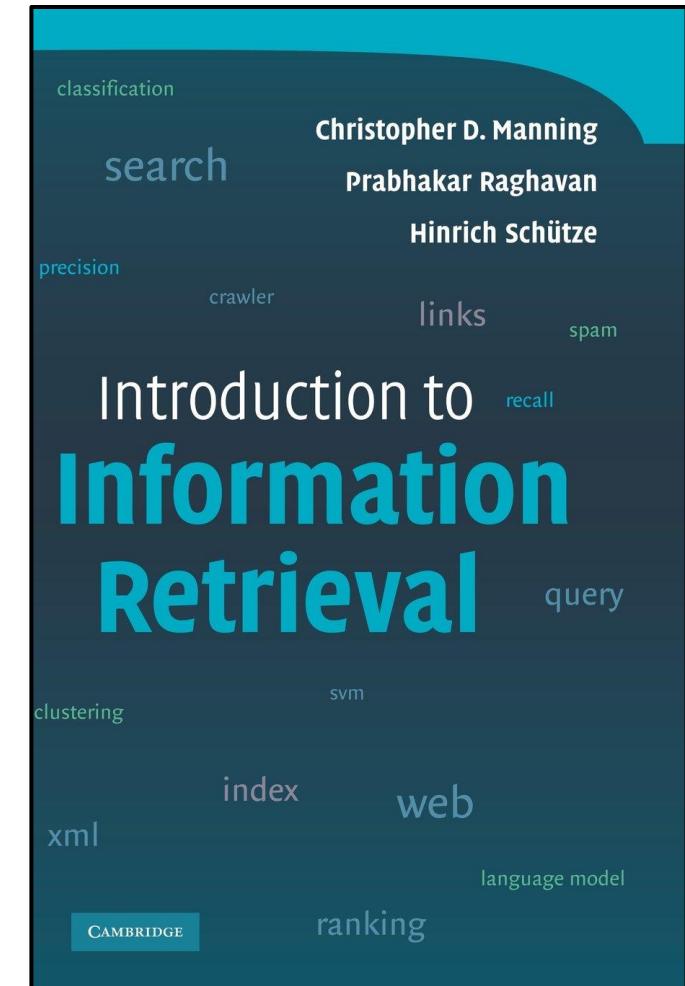


References

Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze
Introduction to Information Retrieval
Cambridge University Press. 2008

The book is also online for free:

- HTML edition (2009.04.07)
- PDF of the book for online viewing
(with nice hyperlink features,
2009.04.01)
- PDF of the book for printing
(2009.04.01)





Big Data and Data Mining

Advanced methods

Flavio Bertini

flavio.bertini@unipr.it



Synonymy

- In most collections, the same concept may be referred to using **different words**
- This issue, known as **synonymy**, has an impact on the **recall** of most information retrieval systems
 - For example, you would want a search for **aircraft** to also match the word **airplane**



Query expansion (1/3)

- Idea: we augment the query with keywords synonyms

User Query:

“car”

Expanded Query:

“car cars
automobile
automobiles
auto”



Query expansion (2/3)

- Idea: we augment the query with keywords, synonyms and related terms
- A variety of automatic or semi-automatic query expansion techniques have been developed
 - goal is to improve effectiveness by matching related terms
 - semi-automatic techniques require user interaction to select best expansion terms
- Query suggestion is a related technique
 - alternative queries, not necessarily more terms



Query expansion (3/3)

- Query expansion involves techniques such as:
 - Finding synonyms of words
 - Finding semantically related words
 - Finding all the various morphological forms of words by stemming each word in the search query
 - Fixing spelling errors and automatically searching for the corrected form or suggesting it in the results
 - Re-weighting the terms in the original query



Related terms

- Where to find terms related to a query, in order to expand it?
 - Controlled vocabularies
 - [WordNet](#) - *A Lexical Database for English* [Princeton University]
 - Text collection
 - **Co-occurring** terms
 - Terms from relevant documents
 - Terms from retrieved documents
 - Terms in an adjacent **window** (of relevant or retrieved documents)

Thesaurus query expansion

- Automatic expansion based on general controlled vocabulary (thesaurus) is **not much effective**
 - It does not take **context** into account:



Query: “tropical fish tanks”

Expanded query: “tropical fish tanks aquariums”



Query: “armor for tanks”

Expanded query: “armor for tanks aquariums”



Co-occurrence query expansion

- Instead of using a thesaurus, related keyword can be extracted from text collections
- Different measures of **co-occurrence** can be used to find related keywords:
 - Dice's coefficient
 - Mutual information
 - Expected mutual information
 - Pearson's Chi-squared (χ^2)
- Measures are based on **entire documents** or smaller **parts of documents** (sentences, paragraphs, windows). We will consider entire documents now, for simplicity



Dice's coefficient (1/2)

- Suppose we want to find words related to “fish”
- How to measure the “relatedness” of a second term to the word fish?
- A measure of co-occurrence:

How many times they appear **together**

How many times they appear **singularly**

- Idea: the higher this score, the more related should be the two words!



Dice's coefficient (2/2)

- Term association measure used since the earliest studies of **term similarity** and **automatic thesaurus construction** in the 1960s and 1970s
- Given two words **a** and **b**, it is formally defined as:

$$2n_{ab}/(n_a + n_b)$$

- n_{ab} is the number of documents containing **both** words **a** and **b**
- n_a is the number of documents containing word **a**
- n_b is the number of documents containing word **b**



Mutual information

- It has been used in a number of studies of word collocation
- Similar to Dice, based on probabilities
- For two words **a** and **b**, it is defined as

$$\log \frac{P(a, b)}{P(a)P(b)}$$

- **P(a, b)** is the probability that a and b occur in the same text window
- **P(a)** is the probability that word **a** occurs in a text window
- **P(b)** is the probability that word **b** occurs in a text window



Mutual information: problem

- A problem with mutual information is that it tends to favor low-frequency terms
- For example:
 - Consider two words **a** and **b**:
 - $n_a = n_b = 10$ and co-occur **half the time** $n_{ab} = 5$
 - Mutual information for these two terms is $5 \cdot 10^{-2}$
 - Consider two words **c** and **d**:
 - $n_c = n_d = 1000$ and co-occur **half the time** $n_{cd} = 500$
 - Mutual information for these two terms is $5 \cdot 10^{-4}$
- **Both pairs co-occur half of the time they occur.**
However, they have different mutual information: 0.05 vs 0.0005



Expected mutual information

- The *expected mutual information* addresses the **low-frequency problem** by weighting the mutual information value using the probability $P(a,b)$
- We are primarily interested in the case where both terms occur, giving the formula:

$$P(a,b) \cdot \log \frac{P(a,b)}{P(a)P(b)}$$

- In the previous case
 - $n_{ab} = 5$, $MI_{ab} = 5 \cdot 10^{-2} \rightarrow eMI_{ab} = 0.25$
 - $n_{cd} = 500$, $MI_{cd} = 5 \cdot 10^{-4} \rightarrow eMI_{cd} = 0.25$



Pearson's Chi-squared (χ^2)

- This measure
 - compares the number of co-occurrences of two words with the expected number of co-occurrences if the two words were independent
 - normalizes this comparison by the expected number

$$\frac{(n_{ab} - N \cdot \frac{n_a}{N} \cdot \frac{n_b}{N})^2}{N \cdot \frac{n_a}{N} \cdot \frac{n_b}{N}}$$

- N is the number of documents in a collection
- $N \cdot \frac{n_a}{N} \cdot \frac{n_b}{N}$ is the expected number of co-occurrences if the two terms occur independently



Query expansion: an example

- Using a **TREC news collection** the four co-occurrence measures are applied on a document level
- Top-5 related words are shown
- Word for which we are searching related terms is

fish



Query expansion results

Dice's coefficient	Mutual information	Expected mutual information	Pearson's Chi-squared
species	zoologico	water	arslq
wildlife	zapanta	species	happyman
fishery	wrint	wildlife	outerlimit
water	wpfmc	fishery	sportk
fisherman	wighout	sea	lingcod

- Mutual information favors very rare words (sometimes mistyped words!)
- Chi-squared also capture unusual words
- Dice's coefficient and Expected mutual information are more suitable for IR query expansion



Query expansion with relevance feedback

- Relevance feedback (RF) is a query expansion and refinement technique based on **user feedback**
- General idea:
 1. The user issues a (short, simple) query
 2. The system returns an initial set of results
 3. The user marks some returned documents as relevant (or non relevant)
 4. The system computes a better representation of the information need based on the user feedback
 5. The system displays a revised set of retrieval results



RF example (1/2)

Query: New space satellite applications

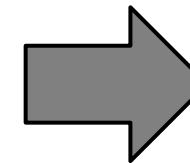
Rank	Document Title	User Feedback
1	NASA Hasn't Scrapped Imaging Spectrometer	YES
2	NASA Scratches Environment Gear From Satellite Plan	YES
3	Science Panel Backs NASA Satellite Plan, But Urges Launches of Smaller Probes	NO
4	A NASA Satellite Project Accomplishes Incredible Feat: Staying Within Budget	NO
5	Scientist Who Exposed Global Warming Proposes Satellites for Climate Research	NO
6	Report Provides Support for the Critics Of Using Big Satellites to Study Climate	NO
7	Arianespace Receives Satellite Launch Pact From Telesat Canada	NO
8	Telecommunications Tale of Two Companies	YES

RF example (2/2)

Query: *New space satellite applications*

Documents relevant
from the user feedback

#1	NASA Hasn't Scrapped Imaging Spectrometer
#2	NASA Scratches Environment Gear From Satellite Plan
#8	Telecommunications Tale of Two Companies



Recurring keywords in
relevant documents



new, space, satellite,
application,
nasa, eos, launch,
aster, instrument,
arianespace,
bundespost, ss,
rocket, scientist,
broadcast, earth,
oil, measure

Expanded query:

*new space satellite application + nasa eos launch aster
instrument arianespace bundespost ss rocket scientist
broadcast earth oil measure*



Pseudo RF

- *Pseudo relevance feedback*, also known as *blind relevance feedback*, provide a method for **automatic** relevance feedback
- It automates the manual part of RF, so that the user gets improved retrieval performance **without an extended interaction**
- The method involves the following:
 1. normal retrieval to find an initial set of most relevant documents
 2. assume that the **top k** ranked documents are **relevant**
 3. compute RF as before under this assumption



Machine Learning and IR: Why?

- Suppose we want to consider (**combining**) at the same time:
 - term frequency in the document body
 - term frequency in the document title
 - document length
 - document popularity (e.g. PageRank)
- ...as “features” to estimate the relevance, how we should **weight** each feature?
- A **learning to rank** model learn the weights from a training set of features and relevance judgements



Machine Learning and IR

- Idea: using machine learning (ML) to build a classifier that classify documents into **relevant and non-relevant classes**
- Although ML has been around for a long time, this good idea has been researched only recently
 - **Limited training data:** it was very hard to gather test collection queries and relevance judgments that are representative of real user needs
 - Traditional ranking functions in IR used a very **small number of features:**
 - Term frequency
 - Inverse document frequency
 - Document length



Learning to rank

- In the last 10 years things have changed
- Modern systems – especially on the Web – use **a great number of features**
 - Log frequency of query word in anchor text
 - Query word color on page
 - # of images on page
 - # of (in/out) links on page
 - PageRank of page
 - URL length
 - URL contains query terms
 - Page edit recency
 - Page length
 - ...
- Lot of **training data** is available from huge query logs that are collected from user interactions



Example: features

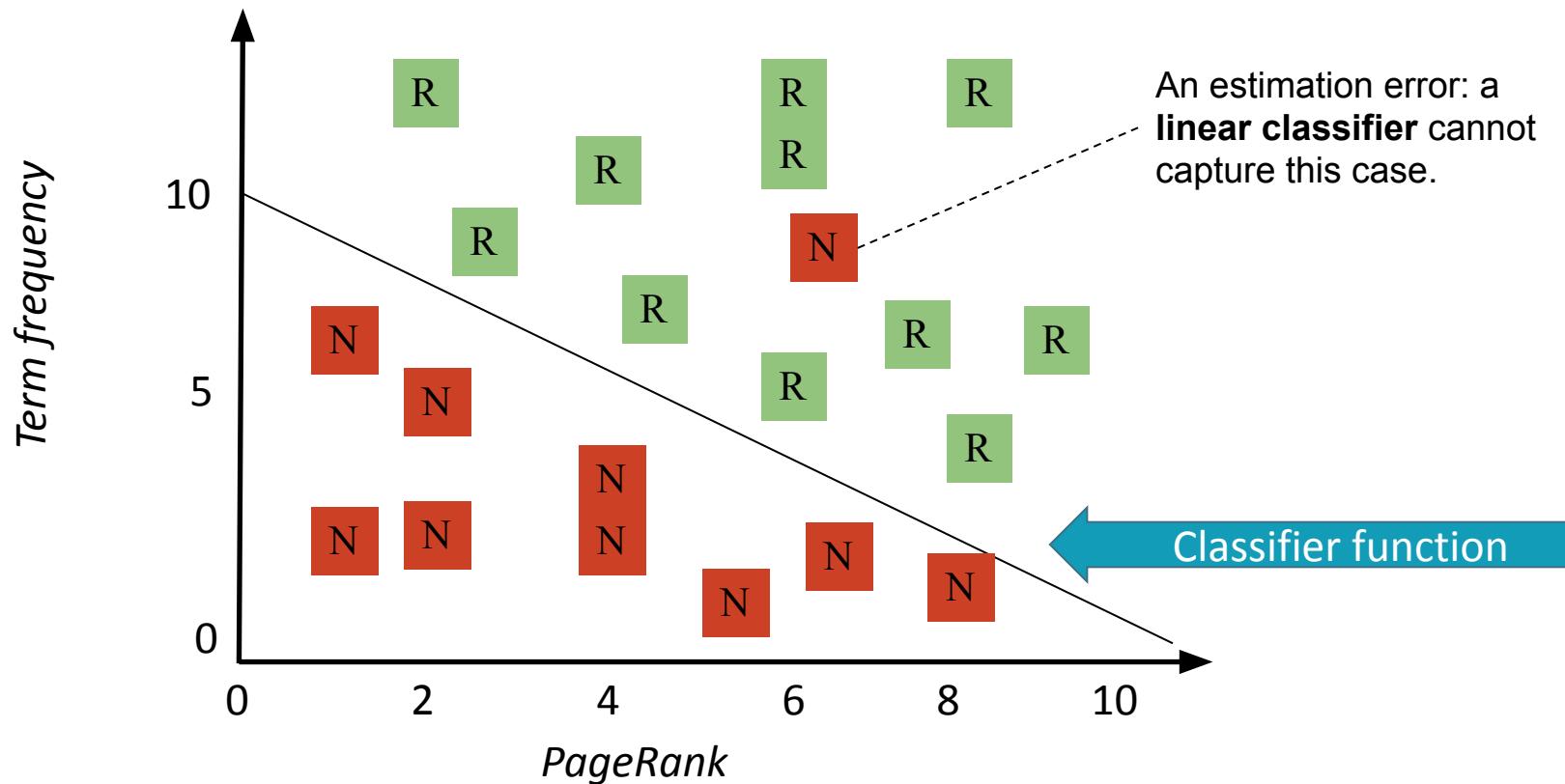
- Suppose we are considering two features in each document:
 - Term frequency ***tf***: how many times the query terms are found in the document
 - Pagerank ***pr***: popularity of the document
- Training set is made of (tf,pr) vectors each with a correspondent relevance judgment
- A learning to rank model given the document features as input (tf,pr), should output the estimated relevance



Example: training data

Query: “ <i>fish tank price</i> ”		INPUT		OUTPUT
Training sample	Doc ID	Term frequency	PageRank	judgement
001	37	11	3	1 (relevant)
002	38	0	8	0 (non-relevant)
003	238	8	2	1 (relevant)
004	248	1	2	0 (non-relevant)
005	1741	5	6	1 (relevant)
006	2094	18	1	1 (relevant)
...

Example: classifier



- The learned **weights** are the **coefficients** of a linear function
- The function represent the learned model that **separates** relevant (output 1) from non relevant (output 0) documents based on the two input variables



Relevance feedback and learning

- RF is a simple example of using supervised machine learning in information retrieval: **training data** (i.e., the identified relevant and non-relevant documents) is used to improve the system's performance
- In the last example, we have used the relevance judgements of a test collection to train a classifier: this is called **offline learning**
- Using relevance feedback to tune the classifier weights and improve its accuracy is an example of **online learning**

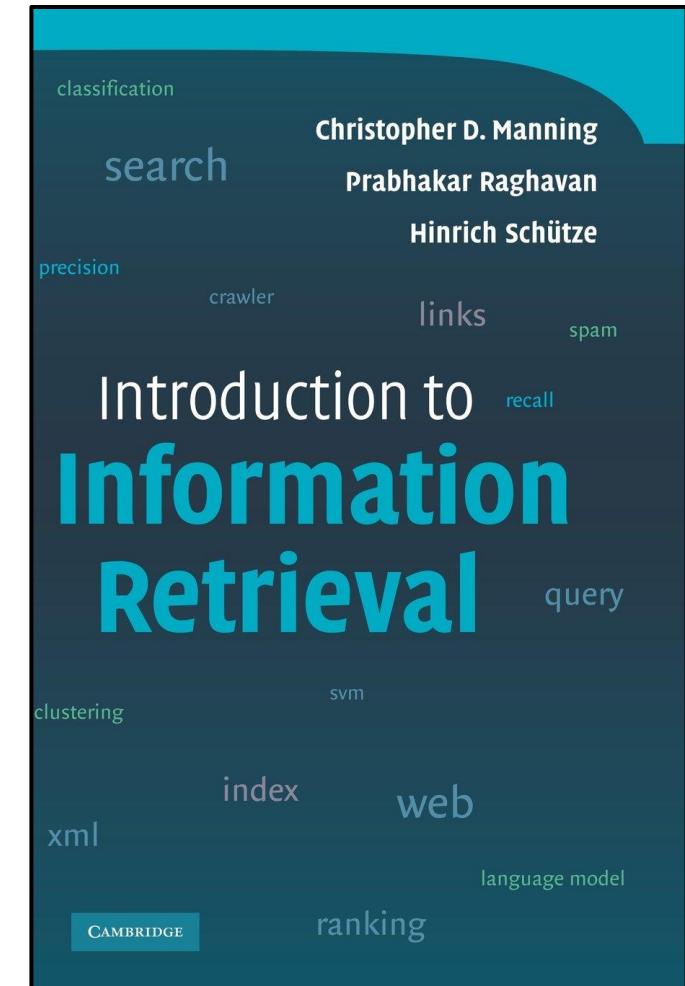


References

Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze
Introduction to Information Retrieval
Cambridge University Press. 2008

The book is also online for free:

- HTML edition (2009.04.07)
- PDF of the book for online viewing
(with nice hyperlink features,
2009.04.01)
- PDF of the book for printing
(2009.04.01)





Big Data and Data Mining

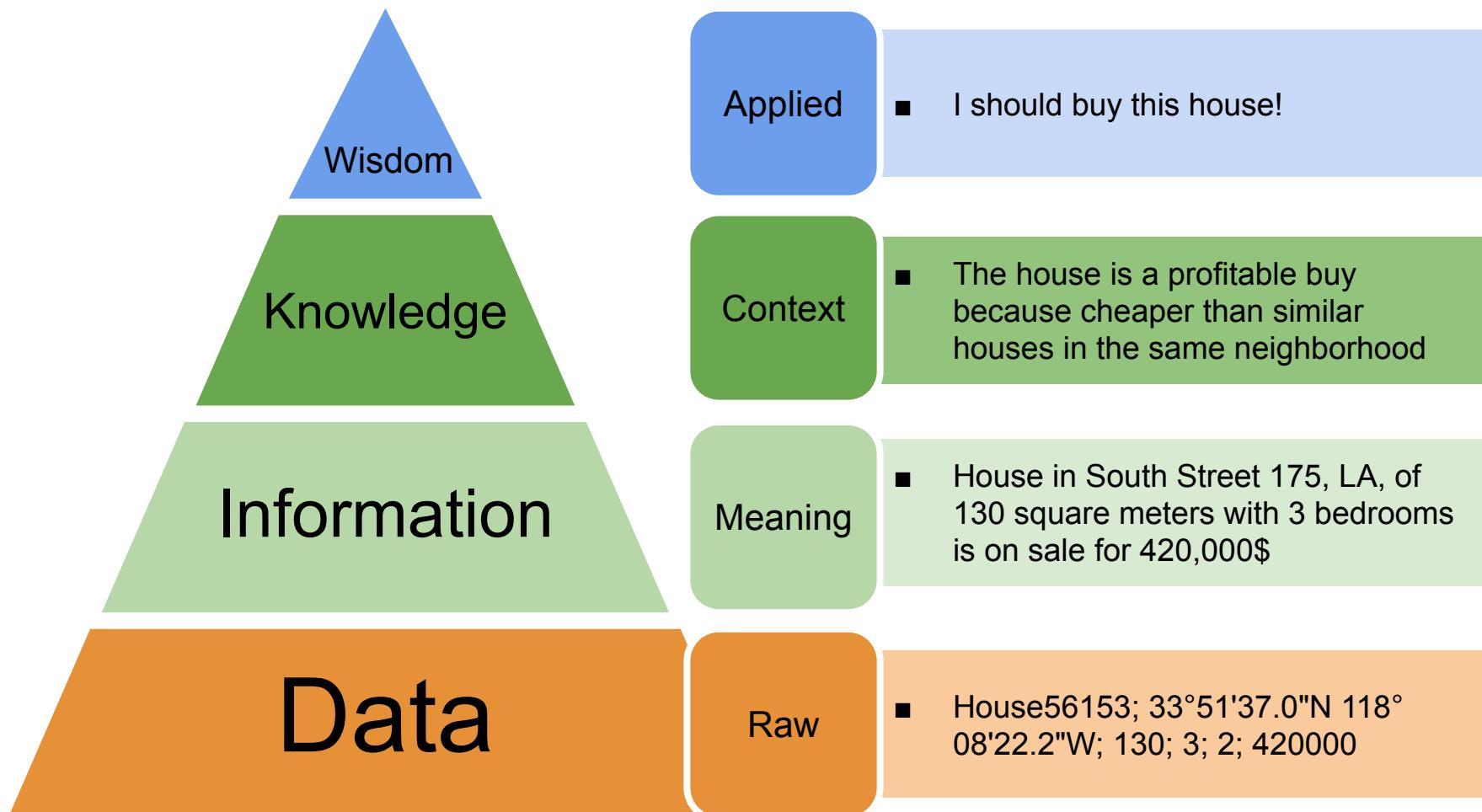
Data Analytics

Flavio Bertini

flavio.bertini@unipr.it

From data to wisdom

DIKW Pyramid: Typically information is defined in terms of data, knowledge in terms of information, and wisdom in terms of knowledge

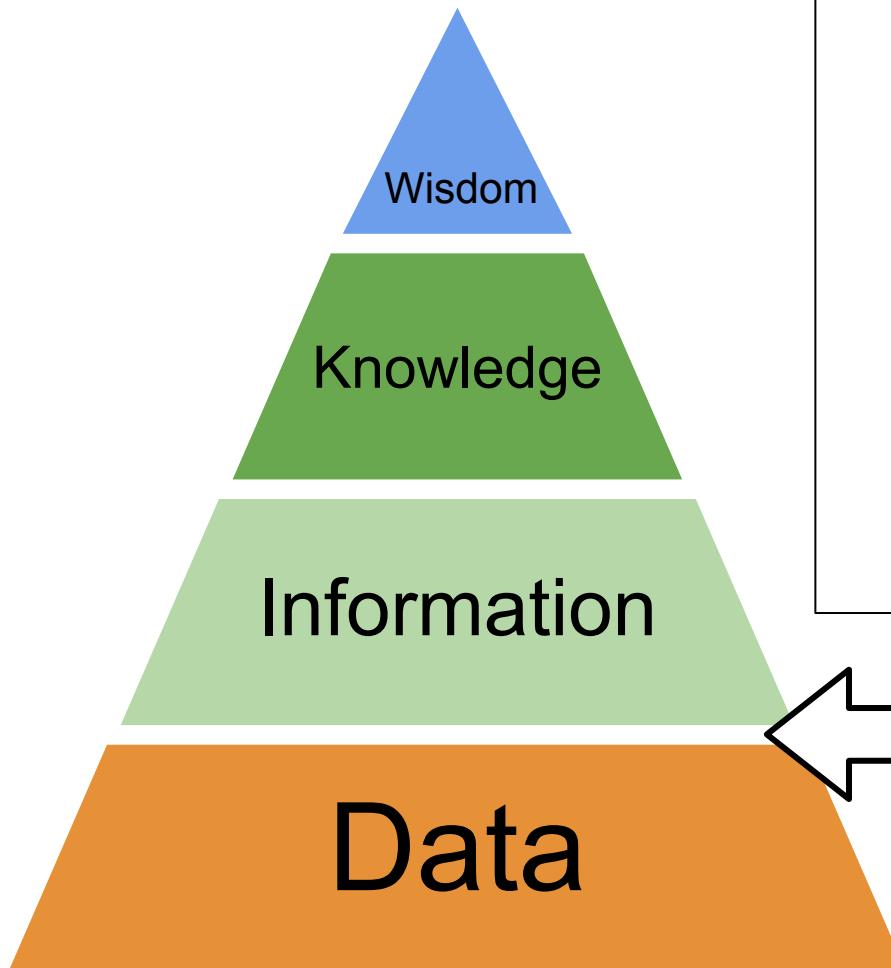




So far in this course

- **Semi-structured data**
 - Storing and querying data without having a rigid schema
 - How semi-structured data relates to structured data and can be queried using a query language for structured data (SQL)
- **Information retrieval**
 - Retrieve a subset of documents with respect to user's information need
 - Searching in the WWW
 - Ranking documents by their relevance to a text query and user's feedback

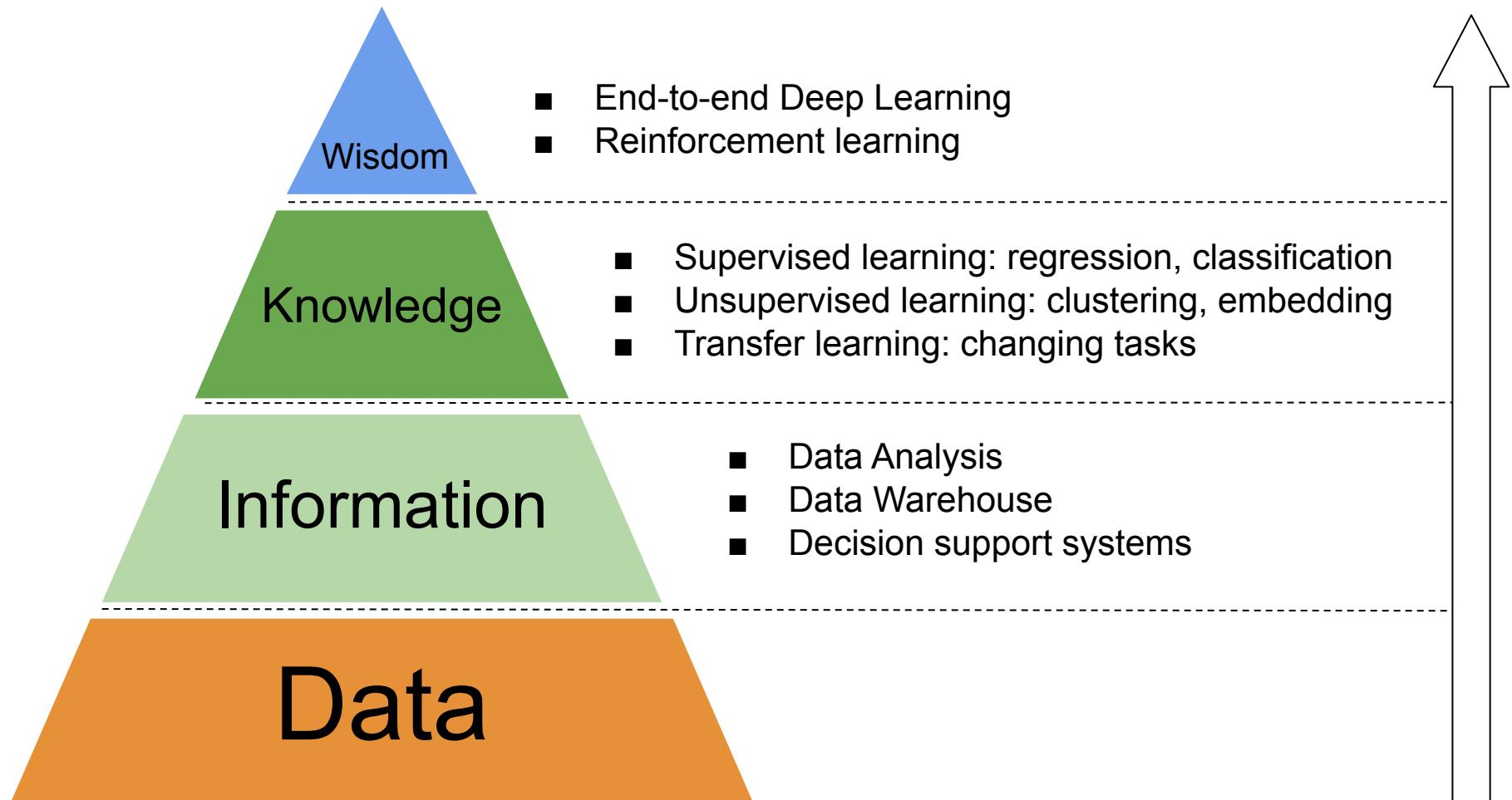
Where in the pyramid?



- We were still in a low level:
 - Semi-structured data can provide information only by manually defining complex queries
 - Information retrieval searches and ranks information without extracting it from data: documents are already “information” in natural language

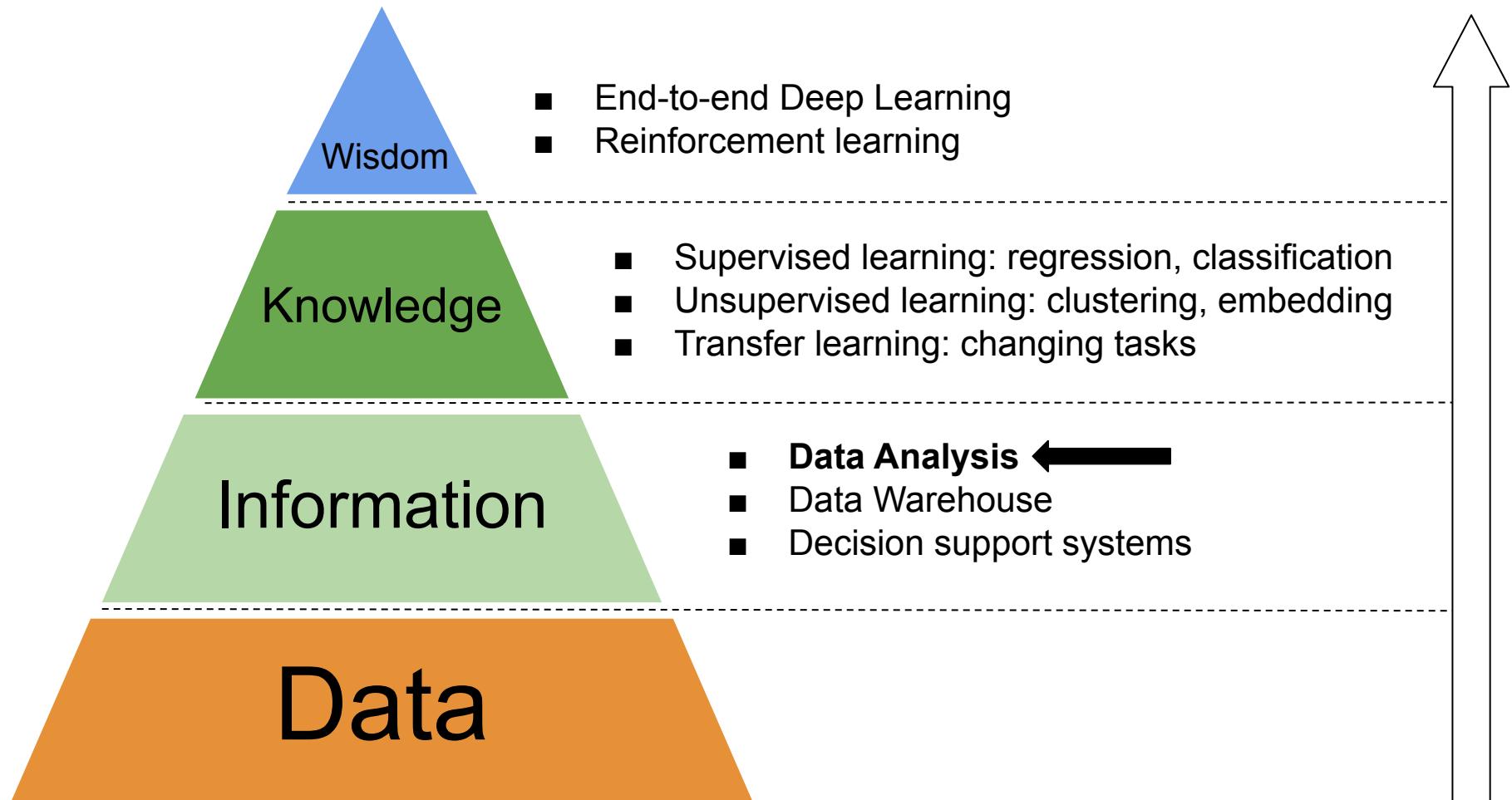
What we will see

Starting from the bottom:



What we will see

Starting from the bottom:



Data Analysis

- The scope of Data Analysis is to **extract** basic **information** from collections of data
- Extracted information can be:
 - Summarized information: e.g., the average from a set of numerical values
 - Association information: e.g., the relation between two sets of values (houses' price vs. square meters)
- Conceptual foundation is descriptive **statistics**

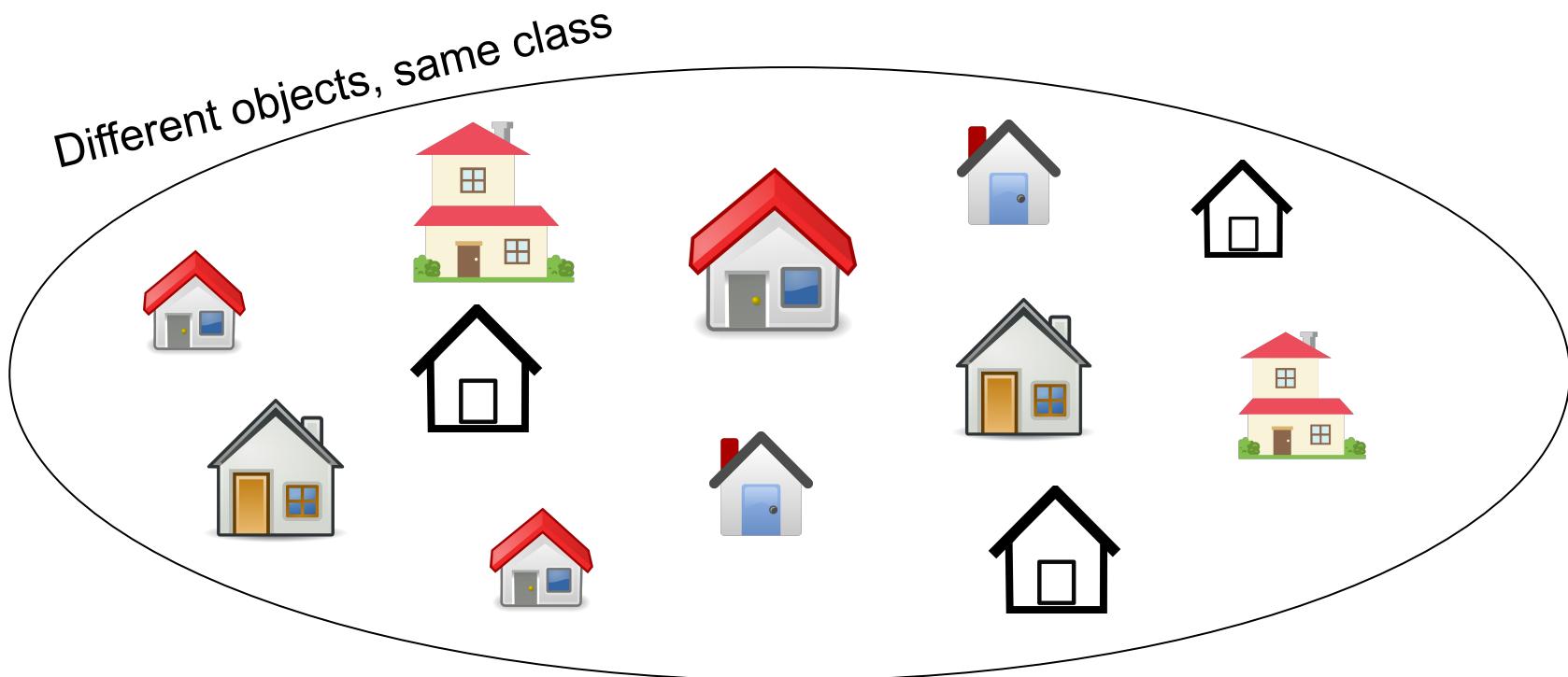
Data



Information

Concepts: Population

- A **population** is a collection of objects we are interested in, for example:
 - All the houses in Los Angeles
 - All the students in the university
 - All the receipts from a grocery shop



Concepts: Record

- A **record** (or observation, case) is a tuple of values that characterize an element of a population

City	Latitude	Longitude	Bedrooms	SquareMeters	Price
Los Angeles	33°51'37.0"N	118°08'22.2"W	3	130	420000

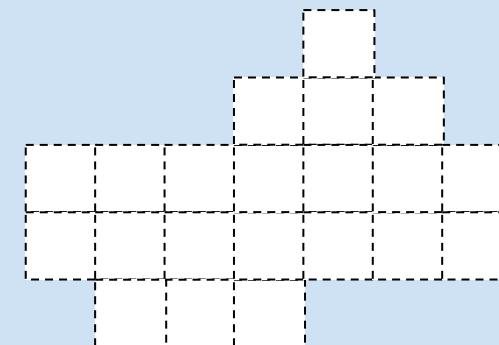


Concepts: Variable

- A **variable** (or field, feature) it's the name for a record's value and has a common meaning and type for all the records in the population



SquareMeters
(Area of the property,
Real value)





Concepts: type of variable 1/2

- We can classify variable depending on the type of the values they can take
- Most important distinction is between
 - **numerical** variables (*quantitative*): if we can apply arithmetic operations on them
 - **categorical** variables (*qualitative*): otherwise
- Example:
 - The price (e.g. 420000) is a **numerical** variable
 - The city (e.g. Los Angeles, New York, Rome) is a **categorical** variable



Concepts: type of variable 2/2

- Numerical variables can be:
 - **Discrete**, if values can be counted
 - **Continue**, if they are the results of a continuous measure
- Categorical variables can be:
 - **Ordinal**, if a natural order exists on the possible values (e.g., school grades: A, B, C, D)
 - **Nominal**, otherwise (e.g., colors)



A dataset

- Finally, the collection of records (a dataset) takes the form of a single table

City	Latitude	Longitude	Bedrooms	SquareMeters	Price
Los Angeles	33°51'37.0"N	118°08'22.2"W	3	130	420000
Los Angeles	33°50'17.7"N	118°09'12.6"W	2	60	380000
Los Angeles	33°49'32.3"N	118°08'44.1"W	5	230	2500000
...
Albuquerque	35°12'08.1"N	106°58'31.1"W	2	105	190000
Albuquerque	35°15'17.0"N	106°59'26.8"W	4	225	440000
Albuquerque	35°14'22.0"N	106°26'26.2"W	2	140	220000
Albuquerque	35°32'23.0"N	106°38'21.2"W	3	150	250000



Descriptive statistics

- Descriptive statistics provides synthesizing indicators to identify, with a single value, **statistical properties** of a population ...
- ... with respect to a **single variable**:
 - *Centrality indicators*: arithmetic mean, mode, median
 - *Variation indicator*: variance, standard deviation
- ... with respect to **multiple variables**:
 - *Covariance*
 - *Correlation*



Centrality: arithmetic mean

- Let X be a **numerical** variable of our dataset (we can't extract the mean from categorical values!)
- n is the **number of records** in our population
- X_i is the i -th record

$$mean = \frac{\sum_{i=1}^n X_i}{n}$$



Arithmetic mean: properties

- Suppose you have a record with a missing data (e.g., the price)

City	Latitude	Longitude	Bedrooms	SquareMeters	Price
Los Angeles	33°51'37.0"N	118°08'22.2"W	3	130	???

- You can keep the record without affecting the variable mean:
 - A solution is to **replace the missing data with the mean** for that variable
 - Adding a record with a mean value will not change the arithmetic mean for the whole dataset

Centrality: median

- Given a population of *sorted* values (e.g., the column “SquareMeters” sorted by its value):

(30,34,37,37, … ,91,91,91,91,91, … ,525,600,670)



x_1 $x_{n/2}$ x_n

- The *median* is the value in central position ($x_{n/2} = 91$)



Median: properties

- Median is a **robust** indicator: anomalies such as very large or very small values do not affects much the median value
- This was not true for mean, which is much more sensitive to anomalies (a.k.a. outlier)
 - Consider the following example:

$\{2,3,3,4,5,6,6\}$ Median=4 / Mean=4

$\{2,3,3,4,5,6,80\}$ Median=4 / Mean=14.6

- Median is still 4 (value in the central position) while mean has shifted from 4 to 14.6!



Centrality: mode

- Given a set of values of a variable (e.g., the column “bedrooms”):
$$(1,2,4,2,5,3,2,2,3,4,1,3,4,2,6,2,1,3,1,2)$$
- First we count the occurrences of a value, that is the **frequency** of that value
 - e.g., “1” is repeated 4 times, “2” is repeated 7 times, “3” is repeated 4 times ...
- The *mode* is the value with higher frequency in the set of observations (i.e. the value “2” in the above example)



Mode: properties

- Unlike mean and median, mode also makes sense on **categorical** data:
 $\text{mode}(\text{Rome}, \text{Rome}, \text{Los Angeles}, \text{Albuquerque})$: *Rome*
- In a **voting** system (e.g., a set of many different classifiers) the mode determines the winning final result
- While robust to **anomalies** like the median, it makes sense also when there is **no linear order** on the possible values (e.g., points in the plane)



Centrality: comparison (1)

- We consider the following set of observations (values) on the variable bedrooms:

$$(1, 2, 2, 3, 4, 7, 16)$$

- **Arithmetic mean** (sum of values of a data set divided by number of values):

$$(1+2+2+3+4+7+16)/7 = \mathbf{5}$$

- **Median** (middle value):

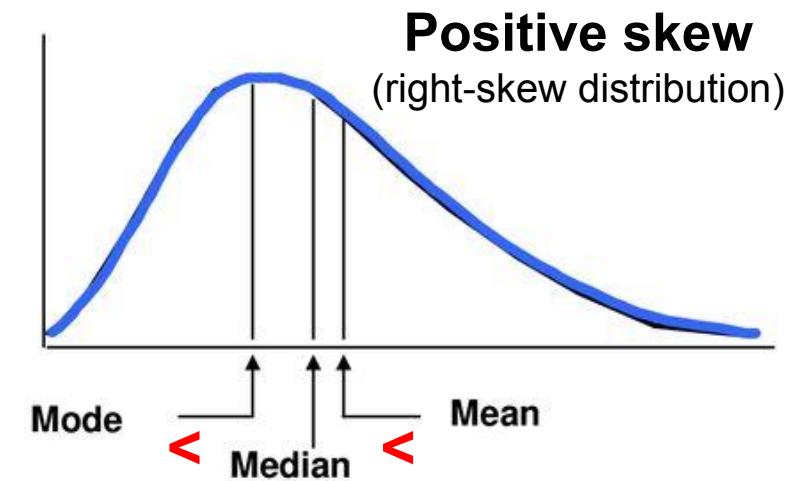
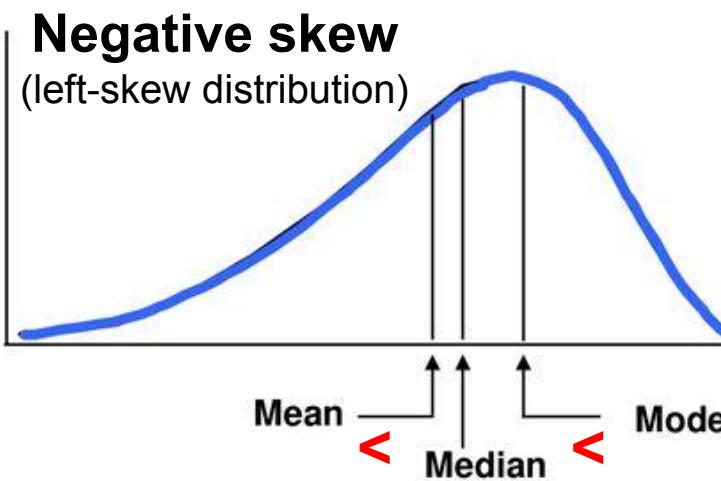
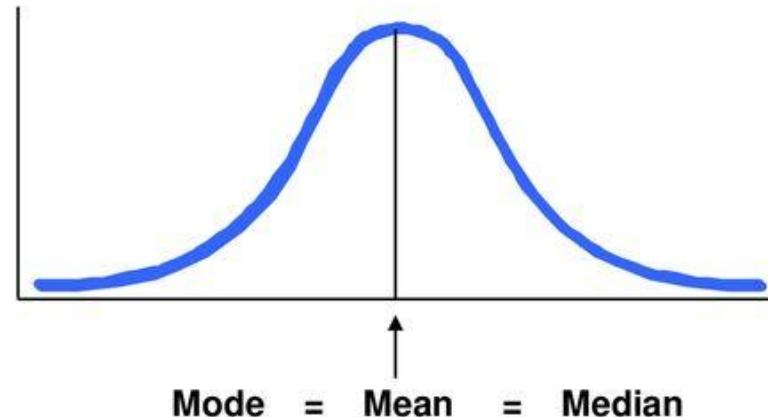
$$(1, 2, 2, \mathbf{3}, 4, 7, 16) = \mathbf{3}$$

- **Mode** (most frequent value):

$$(1, \mathbf{2}, \mathbf{2}, 3, 4, 7, 16) = \mathbf{2}$$

Centrality: comparison (2)

- **Skewness:** distortion or asymmetry that deviates from the symmetrical bell curve, or normal distribution, in a set of data





Centrality: in summary

- We have seen three **centrality** indicators: arithmetic mean, median and mode
- All these indicators provides a different way of “**summarizing**” a set of values into a single, synthesizing value
 - **Arithmetic mean** is also useful to replace missing or wrong data without changing its overall distribution, but its value it's not drawn from the available data and it's sensitive to anomalies
 - **Median** is an actual value from the observations and it's robust to anomalies but needs ordinal data
 - **Mode** is also an actual value, the most frequent one. Robust to anomalies, does not need ordinal data and can be applied on categorical variables



Variation: squared deviation

- Squared deviation: it measures the difference between each value x_i and the mean of the observations \bar{x}

$$dev = \sum_{i=1}^n (x_i - \bar{x})^2$$

- The more the values are far from the mean, the higher the deviation. In the following sample the mean is 15

$$dev(4,6,10,40) = \sum_{i=1}^n (x_i - 15)^2 =$$

$$\begin{aligned} &= (4-15)^2 + (6-15)^2 + (10-15)^2 + (40-15)^2 = 121 + 81 + 25 + 625 \\ &= 852 \end{aligned}$$

Variation: variance

- The squared deviation is **affected by the number of observations**: the more values we have, the higher the deviation tend to be
- The **variance** (often represented with s^2 , σ^2 , or Var) normalizes squared deviation by the number of observations:

$$s^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2 = \frac{1}{n} dev$$

- In the previous example: $s^2(4,6,10,40) = 852/4 = 213$



Variation: standard deviation

- Squared deviation and variance consider the **squared difference between values and the mean**, in order to have non-negative differences
- This leads to large values that do not reflect the estimated deviation from the mean
- **Standard deviation** (often represented with s , σ , or $Stdev$) is the **square root** of the variance

$$s = \sqrt{s^2} = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2}$$

- Unlike squared deviation and variance, standard deviation is expressed in **the same units** as the data



Variation: comparison

- We use the same example on the three variation indicators to observe their difference:

(4,6,10,40)

- **Square deviation:** $\text{dev}(4,6,10,40) = 852$
- **Variance:** $\text{Var}(4,6,10,40) = 213$
- **Standard deviation:** $\text{Stdev}(4,6,10,40) = 14.6$



Other single variable indicators

- **Minimum** (min): it's the minimum value in the observations
- **Maximum** (max): it's the maximum value of the observations
- **Range**: it's the difference between the maximum and the minimum value



Multiple variables indicators

- In descriptive statistics, the *association measures* allow to describe the **relation between two variables**, looking for associations
 - For example, they are useful to determine:
 - If the **price** of the houses is associated with the **square meters**
 - If the **smoke** is associated with heart diseases
 - If the **budget** on advertising is associated with the **number of sales**
- We will see two measures:
 - **Covariance**: classifies the type of the relationship between two variables
 - **Correlation**: measures the strength of the relation between -1 and 1



Association measures: covariance

- The **covariance** classifies the relationship between two variables X and Y
- More specifically, it is the mean of the products of the values deviations:

$$cov(X, Y) = \frac{\sum_{i=1}^n (X_i - \bar{X}) \times (Y_i - \bar{Y})}{n}$$

Association measures: covariance

- The **covariance** classifies the relationship between two variables X and Y
- More specifically, it is the **mean** of the **products** of the values **deviations**:

$$cov(X, Y) = \frac{\sum_{i=1}^n (X_i - \bar{X}) \times (Y_i - \bar{Y})}{n}$$

Deviation of x_i from the mean of x

Deviation of y_i from the mean of y

Arithmetic mean

Product is **high** if deviations are large at the same time (in absolute terms)



Meaning of covariance

- The idea behind: the sum of products of the two deviation is high if, every time X deviate from its mean, Y also deviate from its mean **accordingly**
- Why?
 - If Y does not deviate, its deviation is low and so will be the product
 - If Y deviates randomly (sometimes in a direction, sometimes in another direction) the summation will “cancel out”
- Note: a **positive** covariance means X and Y deviates in the **same direction** (directly proportional), a **negative** covariance means they deviates in **opposite directions** (inversely proportional)



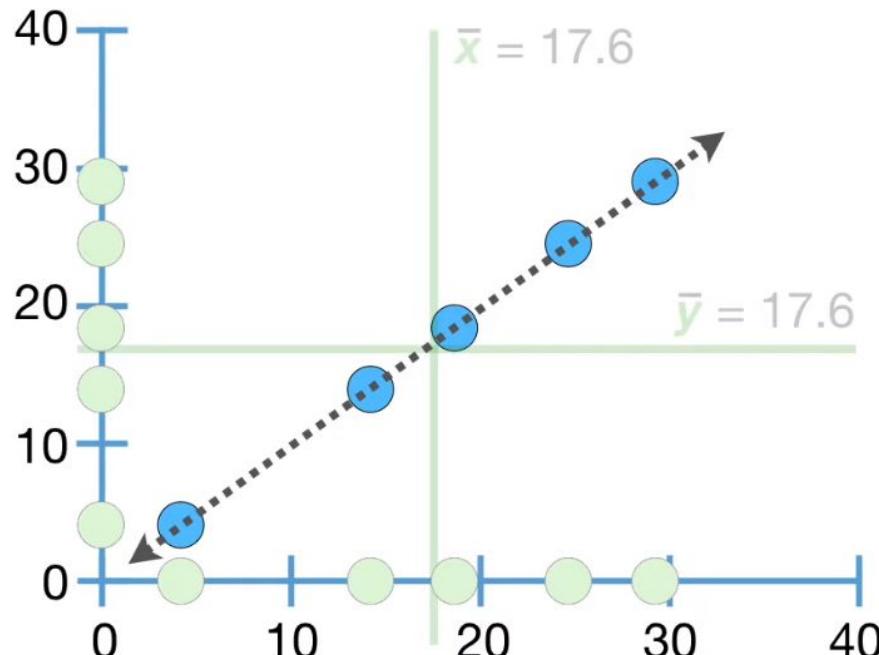
Covariance limit

- A problem with covariance is that its value is affected by the **unit of measure**:
 - If values are large, the covariance tends to be large (even if X and Y are not much related)
 - If values are small, the covariance tends to be small (even if X and Y are strongly related)
- For example, given **the same set** of prices, we can **decrease** the covariance by a factor 1000, by simply expressing the price as **thousands of \$**!

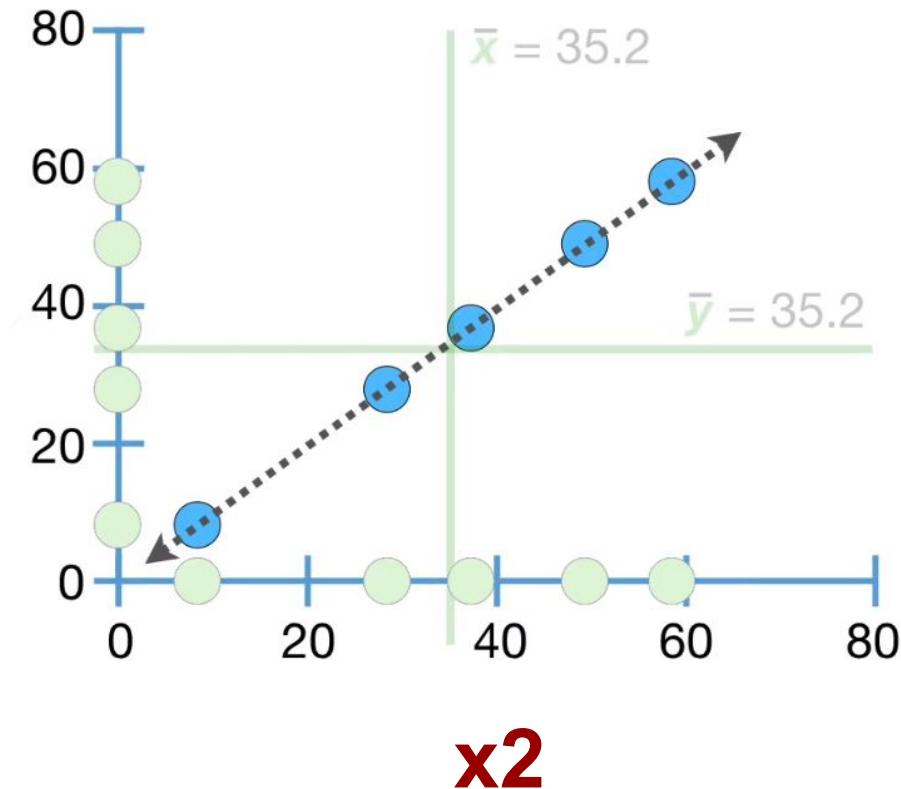


Covariance is hard to interpret 1/3

$$\text{cov}(X, Y) = 102$$



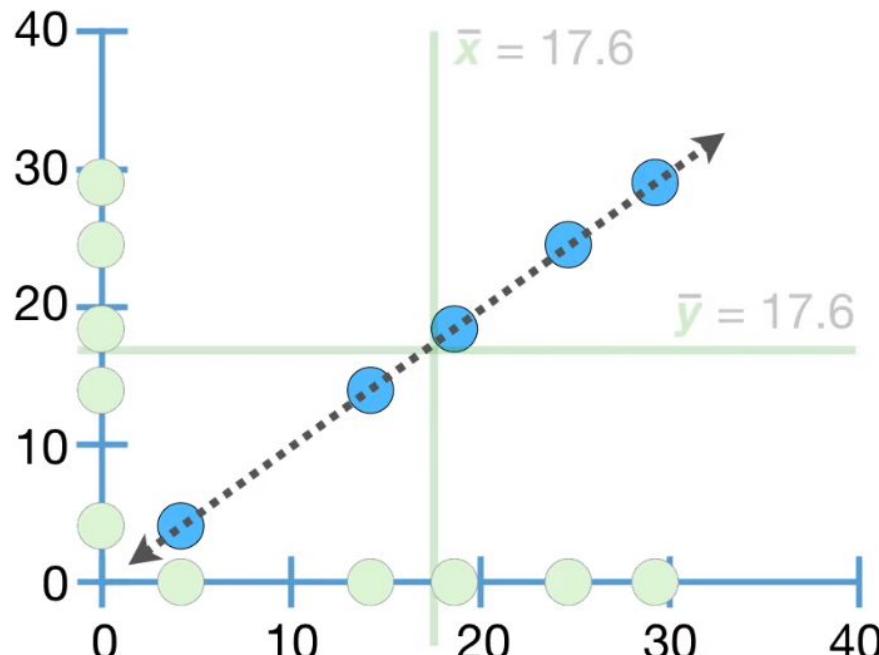
$$\text{cov}(X, Y) = 408$$



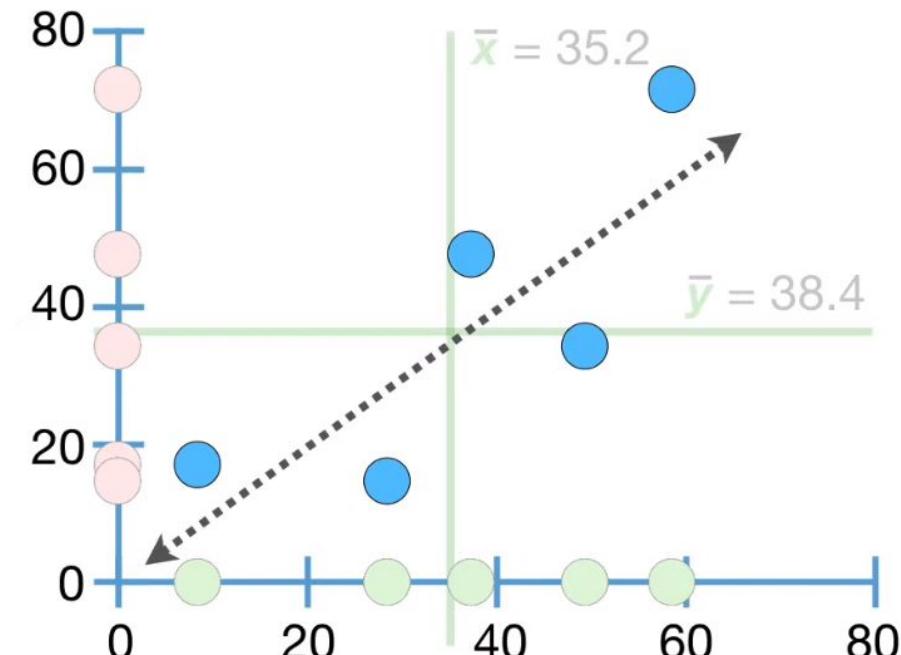


Covariance is hard to interpret 2/3

$$\text{cov}(X, Y) = 102$$



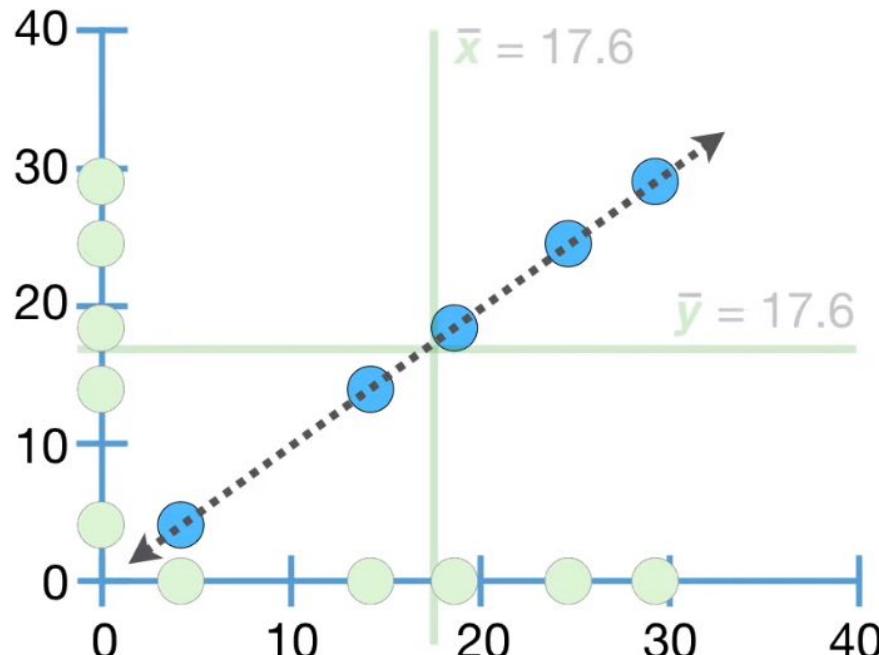
$$\text{cov}(X, Y) = 381$$



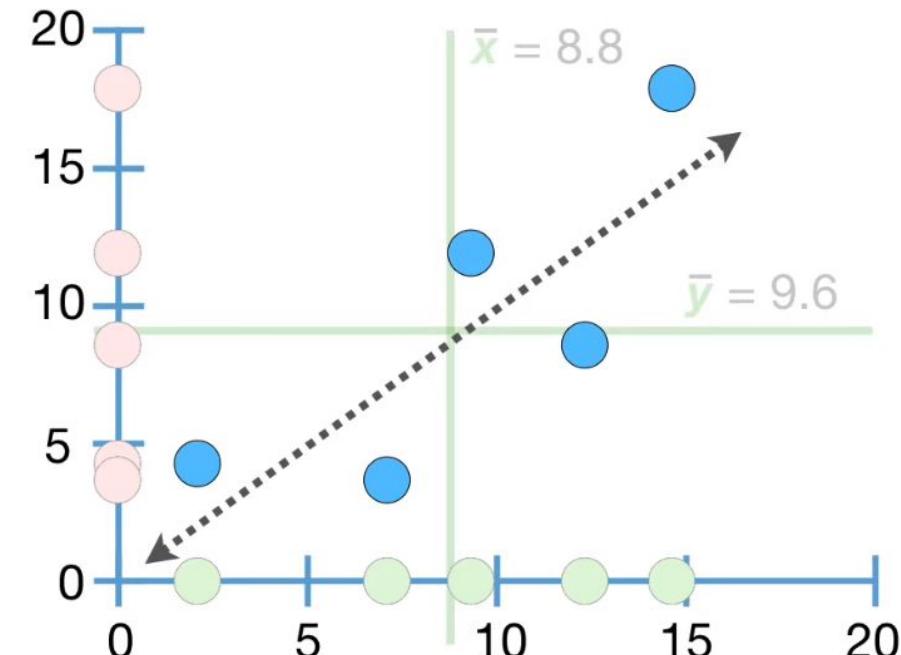


Covariance is hard to interpret 3/3

$$\text{cov}(X, Y) = 102$$



$$\text{cov}(X, Y) = 24$$



scaling



Correlation

- In order to overcome the problem of the unit measure, we use the **correlation**
- The correlation solve this problem producing a result which is independent from unit measure, because it takes into account the standard deviations of X and Y:

$$Corr(X, Y) = \frac{Cov(X, Y)}{Stdev(X) \times Stdev(Y)}$$

- Dividing the covariance by the product of the two standard deviations we ensure a value between **-1** and **1**

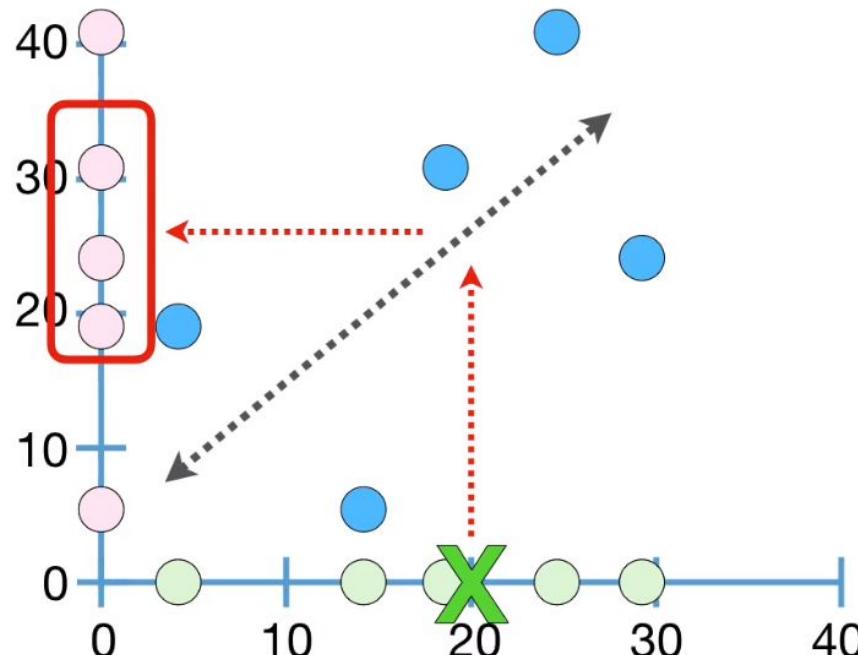


Meaning of correlation

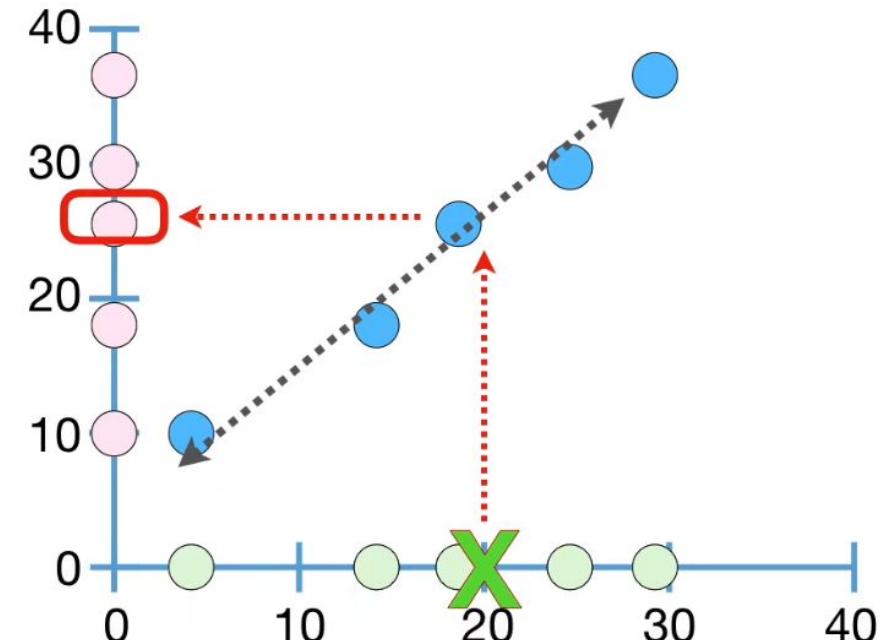
- A value of correlation is close to -1 if the two variables tend to vary in opposite direction (inversely proportional)
- A value of correlation is close to 1 if the two variables tend to vary in the same direction (directly proportional)
- A value of correlation is close to 0 if the two variables have independent variations (at least for **linear** relations!)

Correlation: an example

Weak relationship



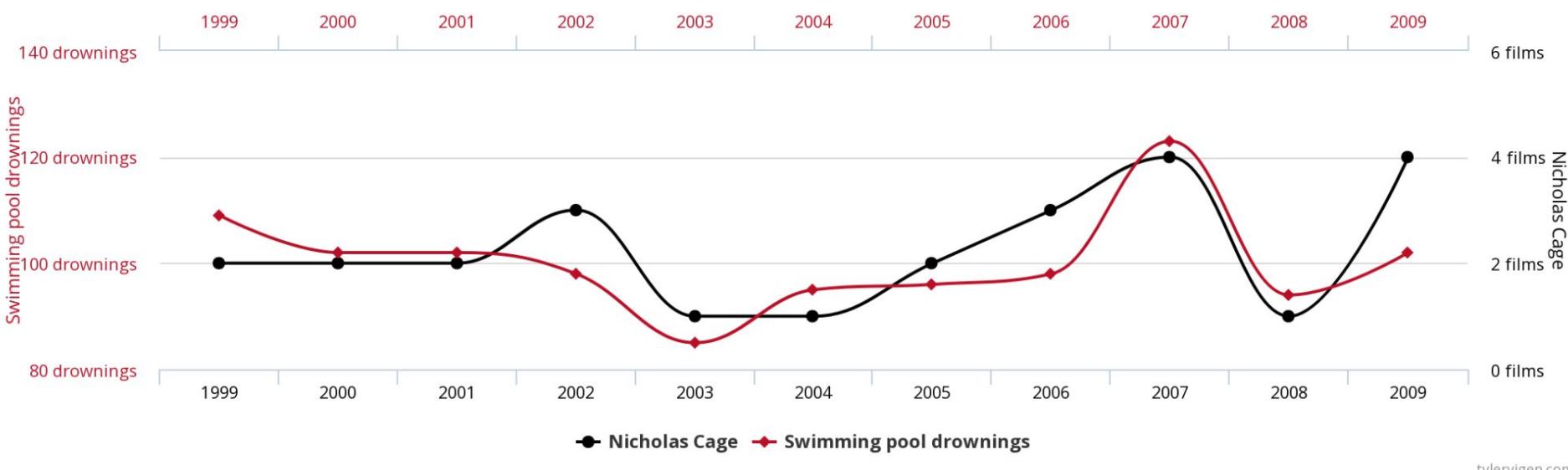
Strong relationship





Correlation is not Causation (1)

Number of people who drowned by falling into a pool
correlates with
Films Nicolas Cage appeared in



Correlation: 0.66

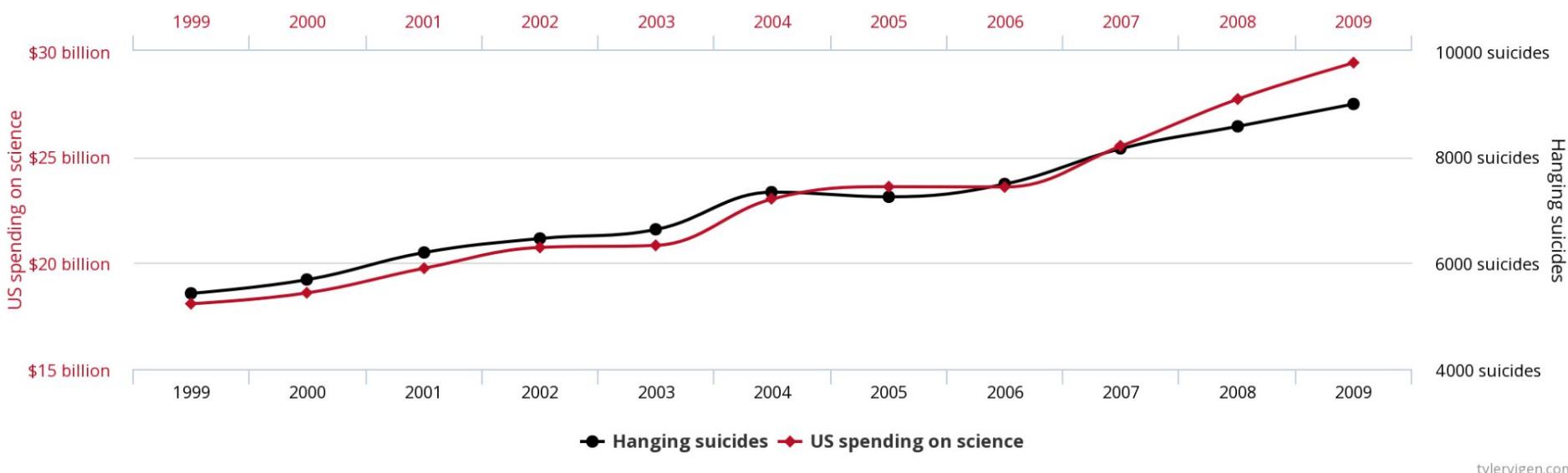
<https://www.tylervigen.com/spurious-correlations>

tylervigen.com



Correlation is not Causation (2)

US spending on science, space, and technology
correlates with
Suicides by hanging, strangulation and suffocation



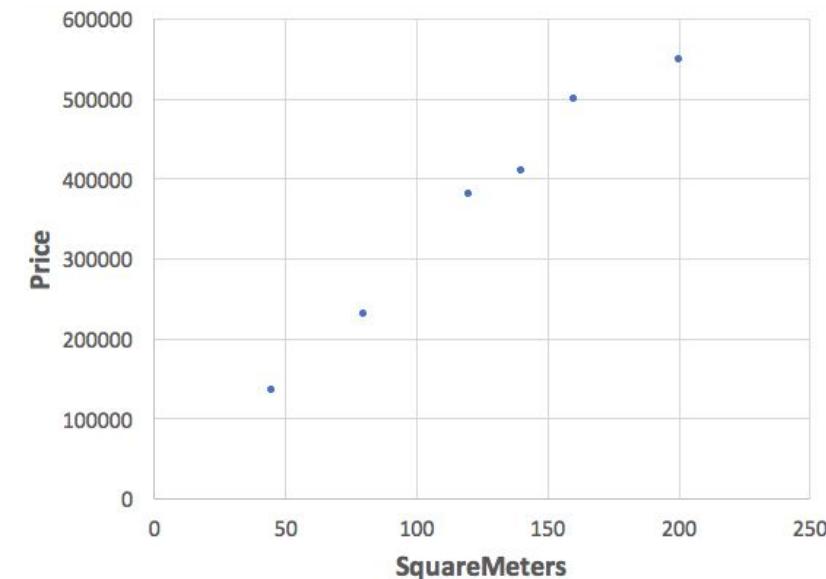
Correlation: 0.99

<https://www.tylervigen.com/spurious-correlations>

Scatterplot and Regression

- Apart from computing association measures, it is useful to visualize the pairs of values from the two variables in an XY plane:

SquareMeters	Price
120	380,000
200	550,000
80	230,000
160	500,000
45	135,000
140	410,000



- While association measures tell us **if** an association exists (correlation here is 0.99!!), **regression** models estimate the **actual function** that relates the two variable: in this case

$$\text{Price}(\text{SquareMeters}) = 3000 * \text{SquareMeters}$$

- But hold on! We will/have see regression models, in the machine learning section



Association measures: summary

- Looking at two different variables at the same time help us understand if there is any relation between the two:
 - Covariance tells us the type of relationship between X and Y
 - Correlation, in addition, it's not affected by the variables unit measures
- **Warning:** this association measures works only if the relation is **linear** (i.e. the points form a straight line in the XY plane)
 - Correlation can be 0 even if exists a strong but **non-linear** relation between X and Y



References

- Jackie Nicholas Introduction to Descriptive Statistics Mathematics Learning Centre University of Sydney 2010



Big Data and Data Mining

Data Warehouse

Flavio Bertini

flavio.bertini@unipr.it



Operational Database

- The kind of database we have seen so far is **“operational”**
 - It’s made to support **software applications**
 - It’s made to be **updated** in real time (add, change or delete data), not just for viewing
 - It needs **complex** queries to make data analysis and exploration (and many computationally-expensive **joins** operations to join data in a single table)
 - It needs **technical** knowledge of the database schema
 - It may not contain **historical data**, because it doesn’t need it for application to work!

Data-driven decision making

- Operational databases are more than fine for software data, but in a **business** context, data should also become a rich source of **information**:
 - It represents the business situation and behaviour
 - It can support **informed decisions** for the business decision makers



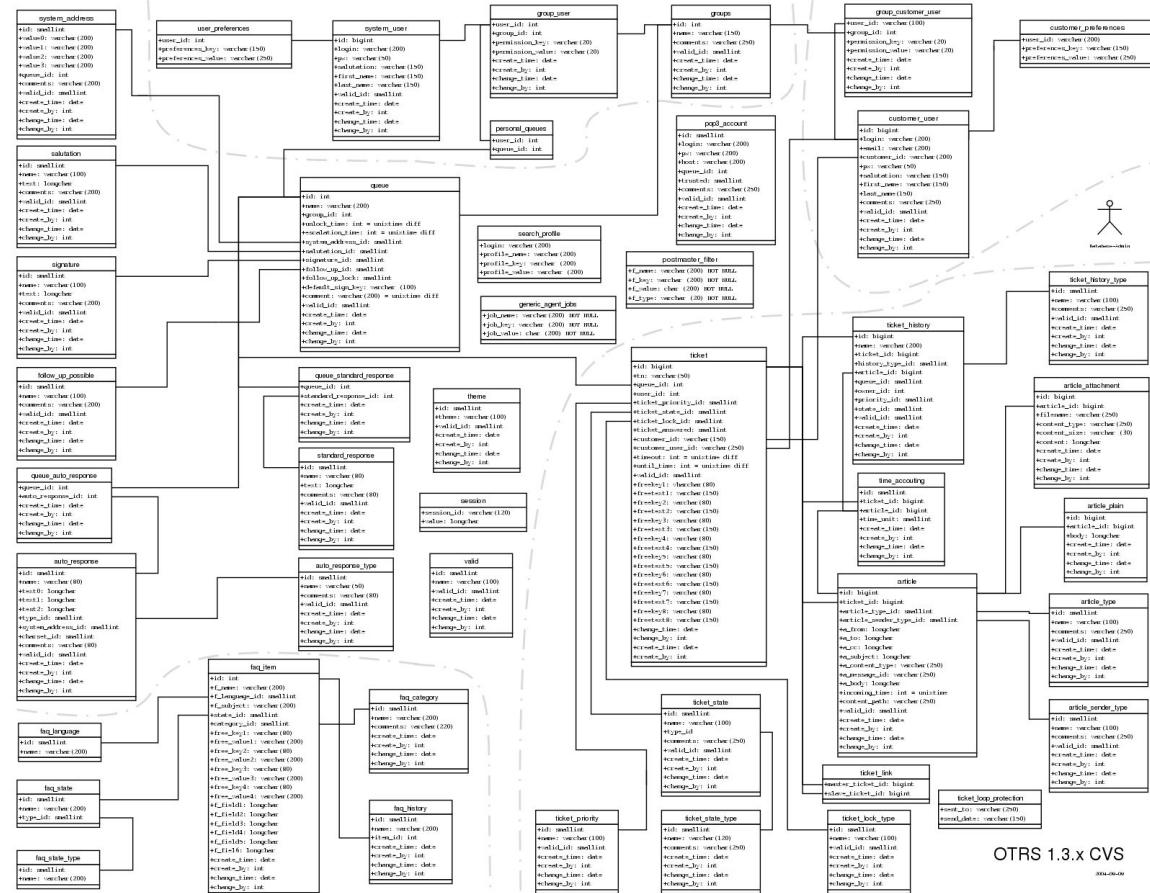


Decision making from a database

- Decision makers cannot directly **benefit** from a database's data without a deep and technical knowledge of the **data model**

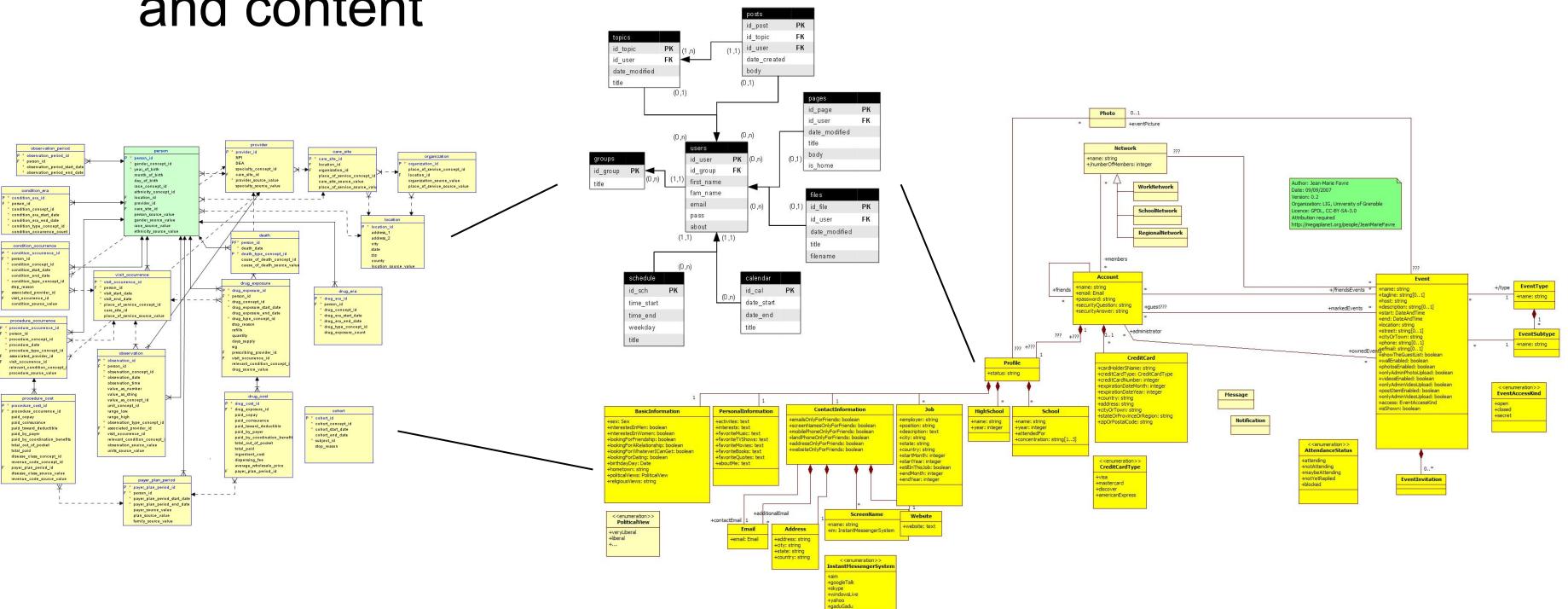


Decision maker



Data integration

- Often decision makers want to **put together** data from different operational databases
 - Example: *sales data with advertising data*
- Integrating different operational databases is **extremely** difficult: they have different data definitions and content

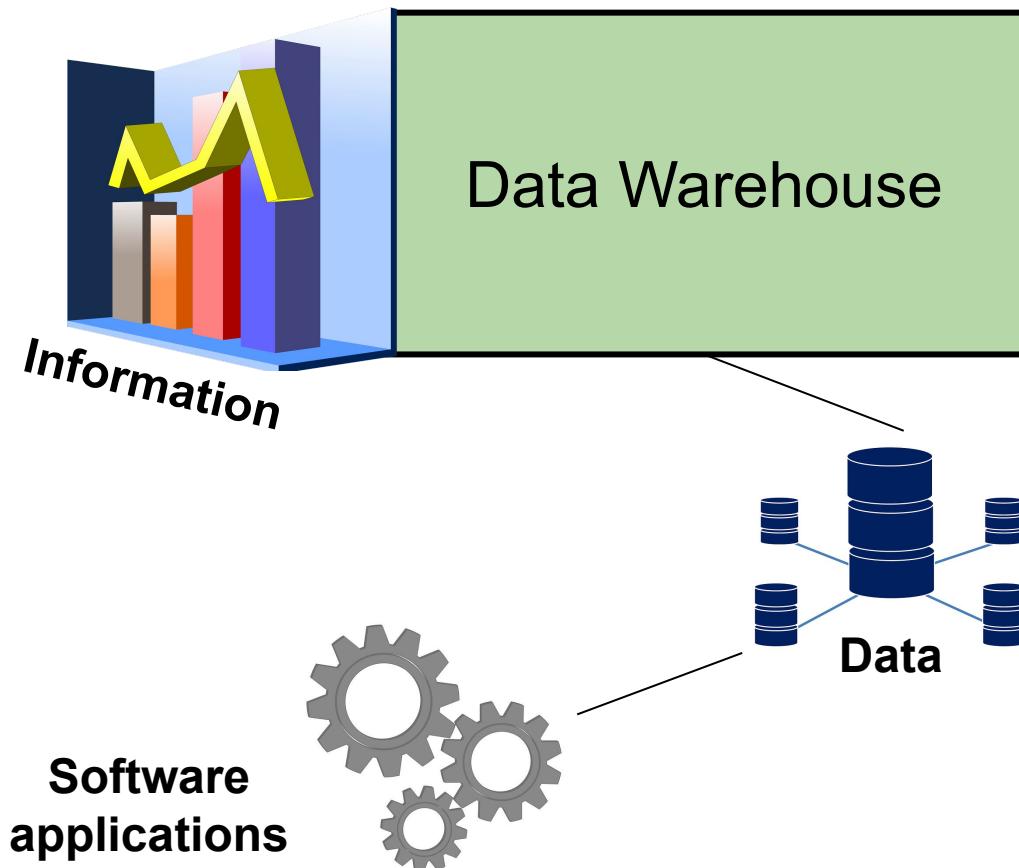


Data warehouse

Data warehouse (DW): **collection of subject-oriented, integrated, nonvolatile, and time-varying data to support management decisions**



Decision
maker





Data warehouse properties

More in details, a Data Warehouse is:

- **Subject-oriented**: a data warehouse targets one or several subjects of analysis according to the analytical requirements of managers at various levels of the decision-making process
- **Integrated**: the content of a data warehouse result from the integration of data from various operational and external systems
- **Nonvolatile**: a data warehouse **accumulates** data from operational systems for a long period of time → **modification and removal are not allowed** the only operation is the purging of no longer needed, obsolete data
- **Time-varying**: a data warehouse keeps track of how its data has **evolved over time**; for instance, it may allow one to know the evolution of sales or inventory over the last months/years



Design of databases

- Typically performed in four phases:
 - **Requirements specification:** needs of users are collected to create a database schema
 - **Conceptual design:** describes the data with entity-relationship (ER) model
 - **Logical design:** implementation paradigm for database applications → relational model
 - **Physical design:** specific implementation over a DBMS
- The final result is a **relational database:** highly normalized to guarantee consistency under frequent updates, usually achieved at a higher cost of querying (normalization produces multiple tables)



Design of a Data Warehouse

- The relational paradigm is **not appropriate** for data warehouses: we need good performance for complex query in data analysis
- We need a design technique that:
 - It's intended to support **end-users** (decision makers) queries
 - It's oriented around **understandability** of the results, without the need of knowing a complex data schema
 - It's oriented to **performance**, redundancy is accepted if brings a performance advantage, less degree of normalization is required
- The paradigm used to design a data warehouse is called **multidimensional modeling** (or simply dimensional modeling, DM)



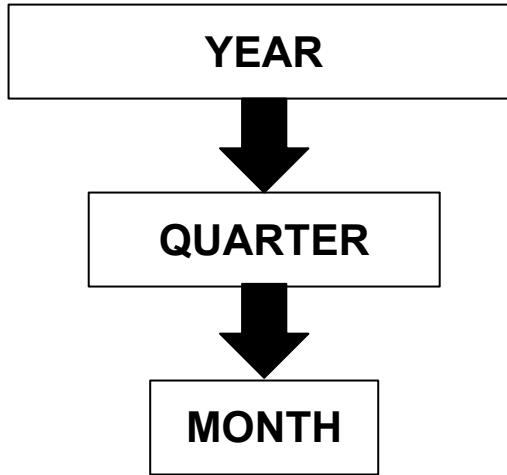
Multidimensional modeling 1/2

- The multidimensional modeling views data as consisting of **facts** linked to **dimensions**
- A **fact** represents the focus of analysis, the main object we are interested into (e.g., in the analysis of sales in stores, the fact is the sale)
- **Measures** quantify facts: usually measures are **numeric** values (e.g., the amount of sales, number of items, etc)
- **Dimensions** are used to view measures from several perspectives, for example:
 - **Time dimension**: to analyze changes in sales over specific periods of time
 - **Location dimension**: to analyze sales according to the geographic distribution of stores

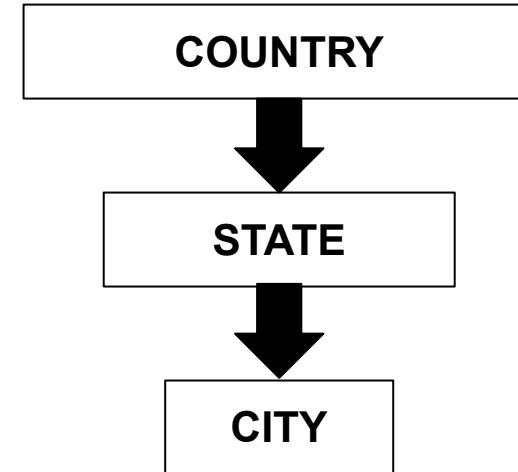
Multidimensional modeling 2/2

- Dimensions include attributes that form **hierarchies**, which allow decision making users to explore measures at various **levels of detail**, for example:

Time dimension:



Location dimension:



- Aggregation** of measures occurs every time a hierarchy is traversed, e.g., moving from month to year will aggregate values of sales for each year



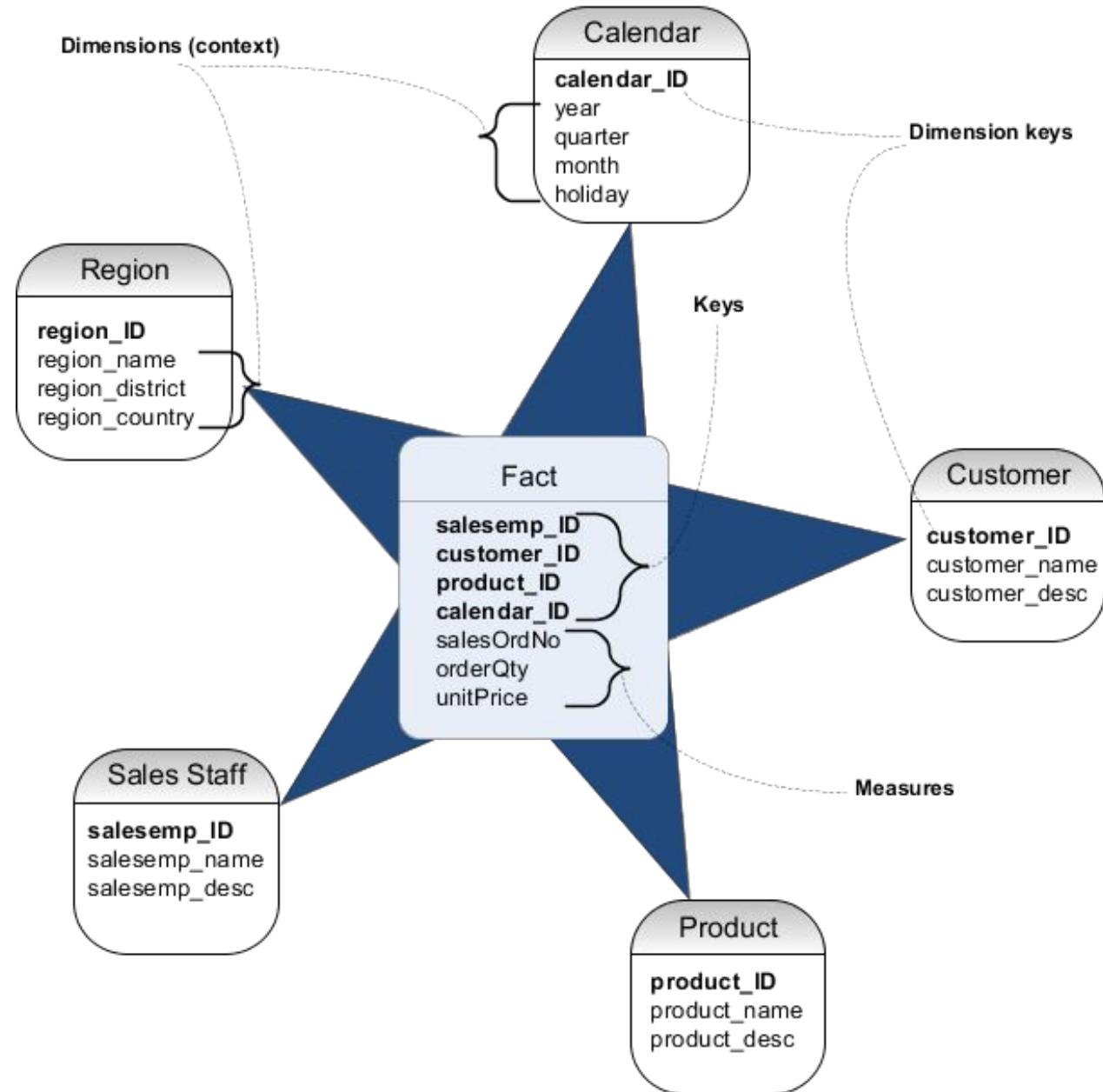
Star and snowflake schemas

- The multidimensional modeling is a **conceptual** model for data warehouses
- At the **logical** level, the multidimensional model is usually represented by relational tables organized in **star schemas** and **snowflake schemas**
- These schemas put the **fact table in the center**, linked to several dimension tables
 - **Star** schemas use a **unique** table for each dimension, even in the presence of hierarchies (**denormalized** table, because of redundancies)
 - **Snowflake** schemas use **multiple normalized** tables (dimensions with hierarchies can be decomposed into a snowflake structure)



Star schema

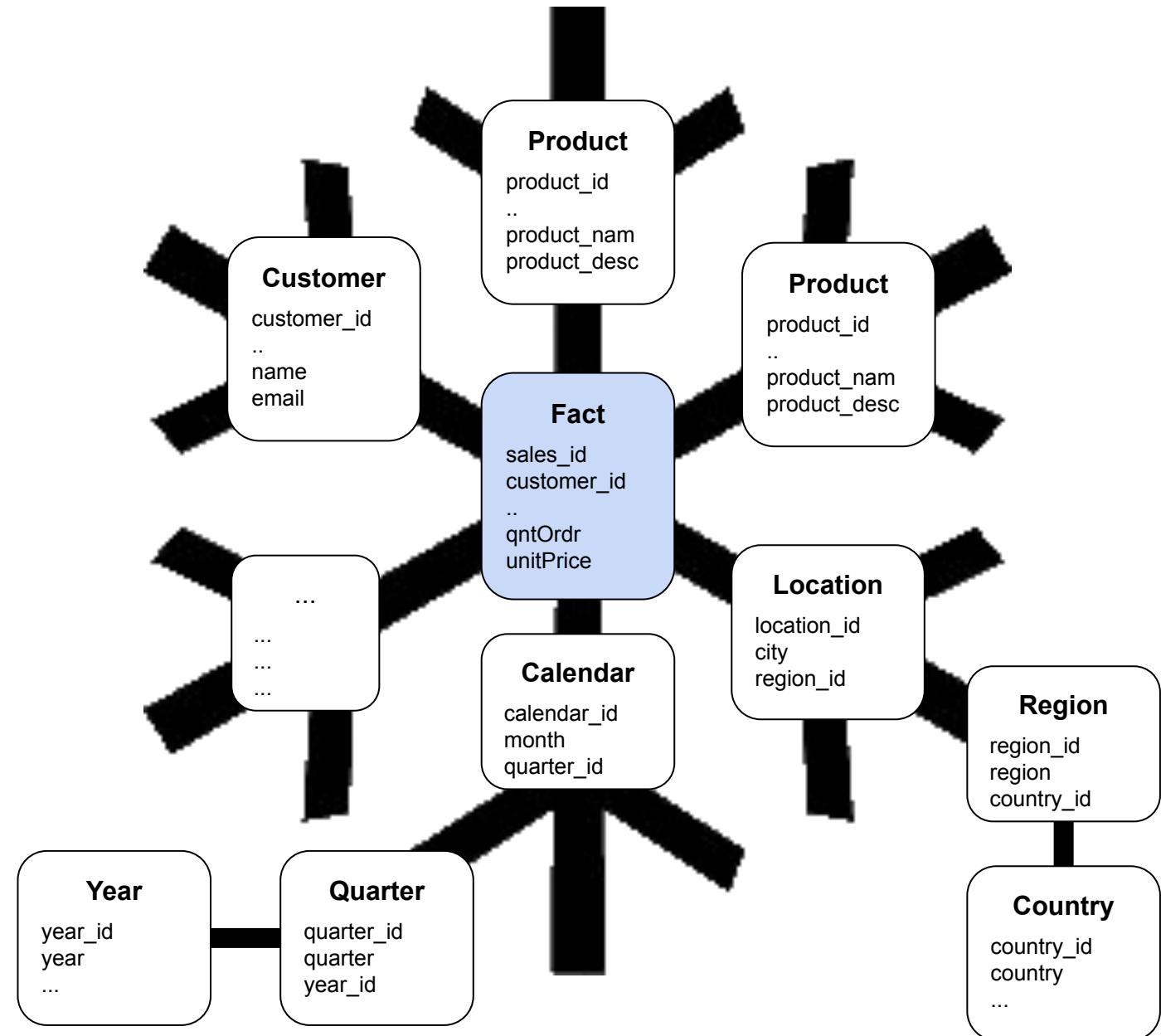
**Redundancy in star schema:
country will be
repeated for all
the regions in the
country**





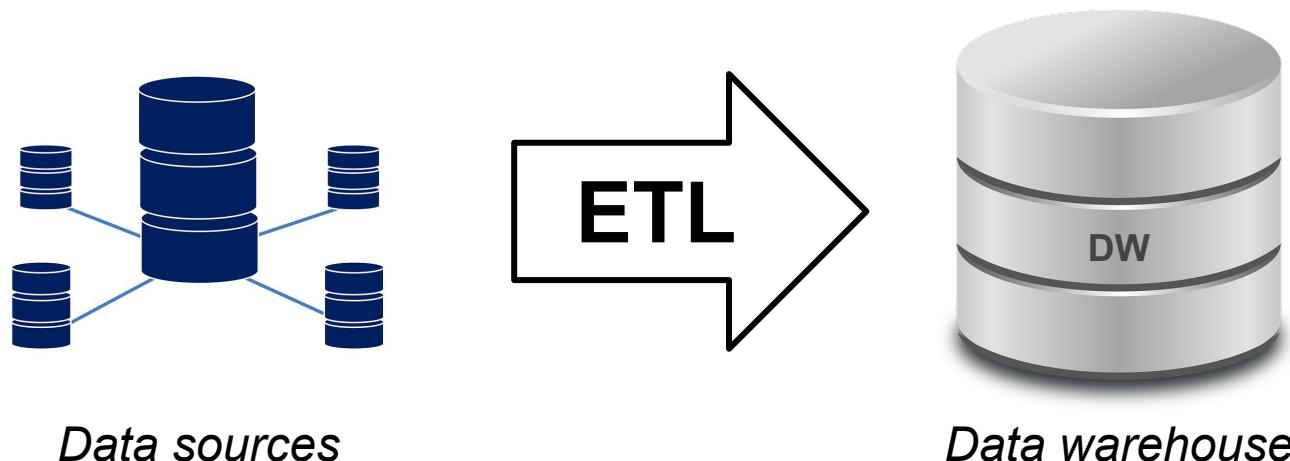
Snowflake schema

No redundancy:
country has
its own table



Populating a DW: ETL

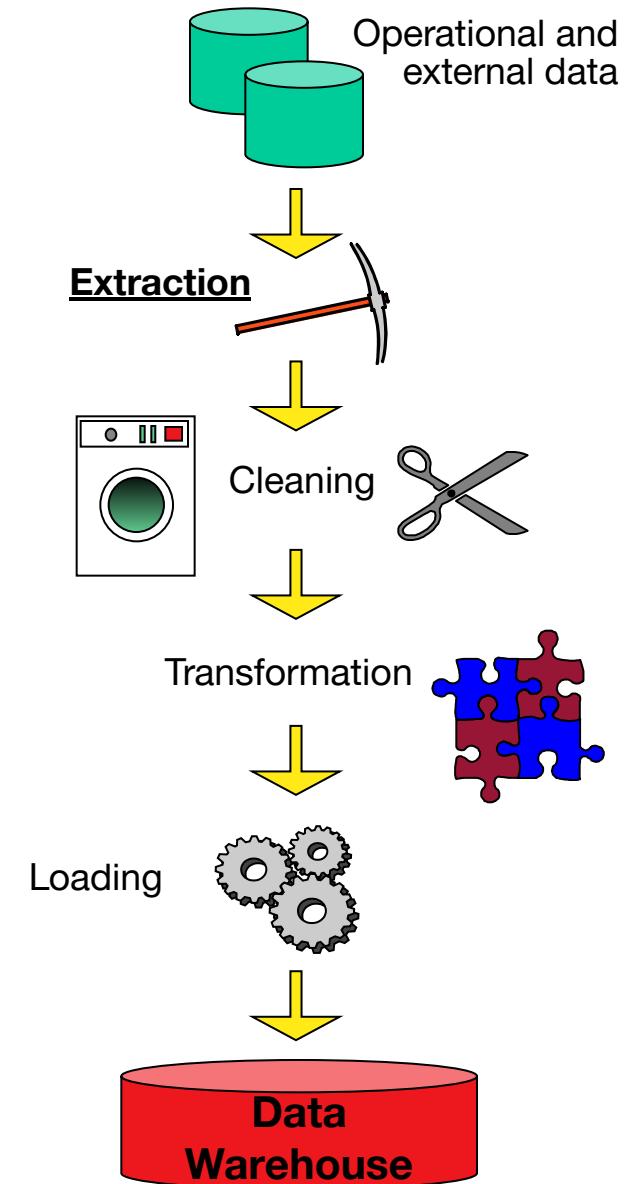
- ETL stands for **Extraction, Transformation** and **Loading**
- It's a crucial process for the success of a data warehousing project and it goes through 3 steps:
 - **Extract** data from several source systems
 - Clean and **Transform** data to fit the data warehouse model
 - **Load** transformed data into the data warehouse



- Requires great efforts: ETL is estimated to be the 80% of the total DW cost!!

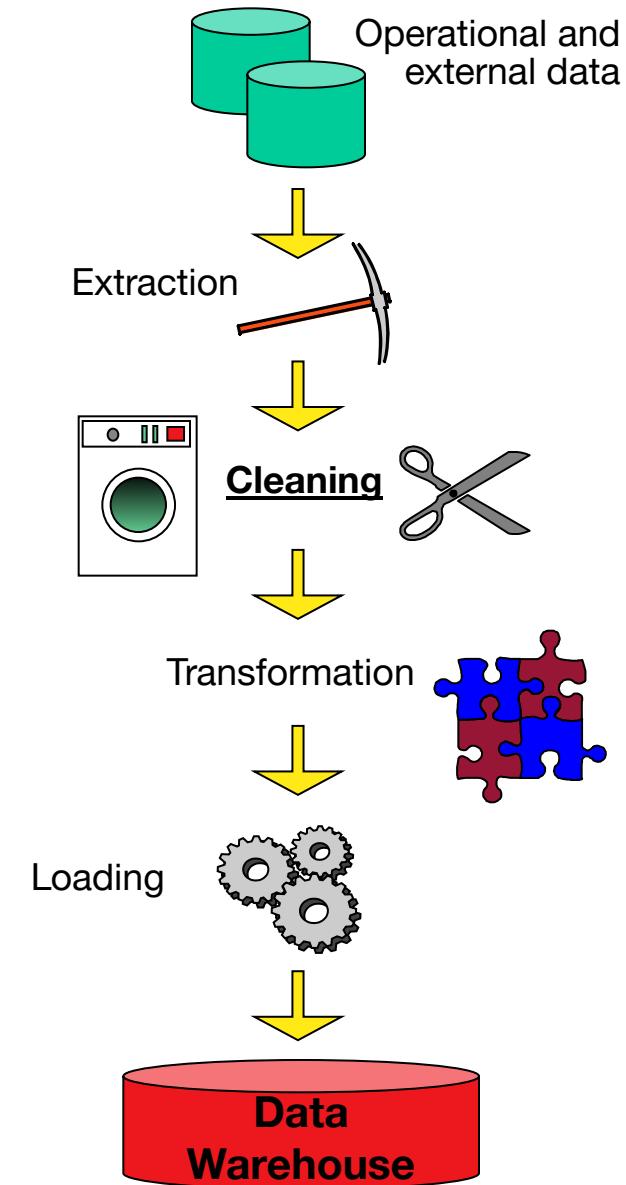
ETL: Extraction

- Relevant data is extracted from the source
 - **Static** extraction is carried out when the DW must be populated for the first time, it basically consists of making a “snapshot” of the operational data
 - **Incremental** extraction is used to update the DW, and it captures only the changes with respect to the last extraction
 - Based on the DBMS log
 - Based on timestamps
- The selection on what data should be extracted depends mainly on its quality



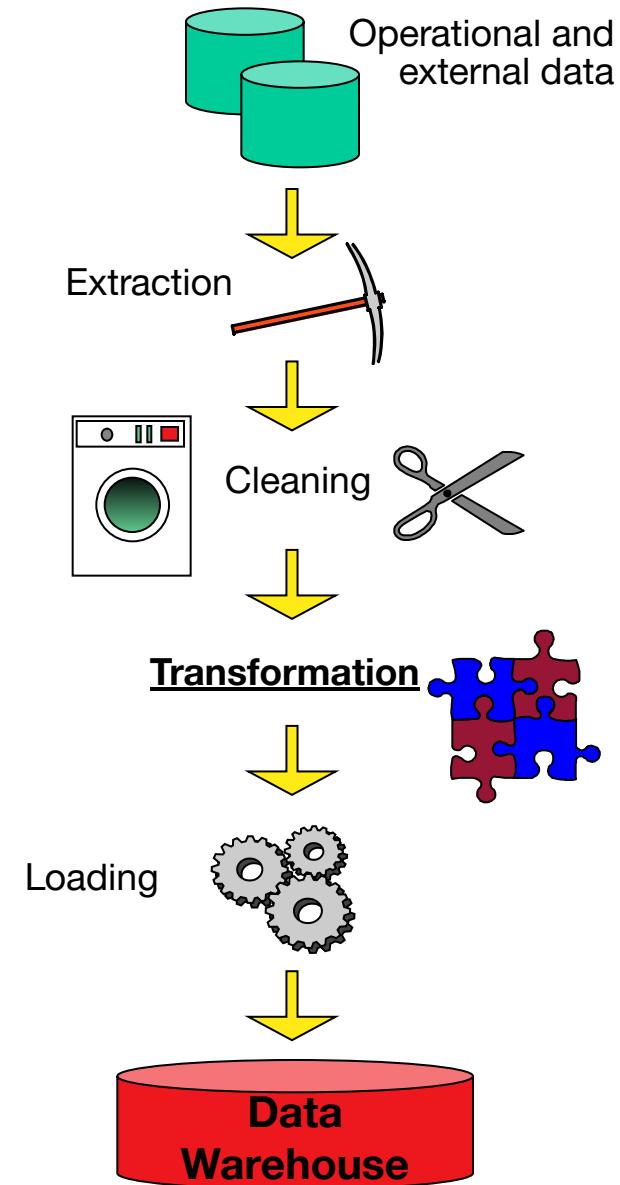
ETL: Cleaning

- The cleaning process has the goal of improving the data quality of the sources
 - Duplicate data
 - Inconsistency between associated values
 - Missing values
 - Misuse of a field
 - Impossible or erroneous values
 - Inconsistent values for the same entity because of different standards
 - Inconsistent values for the same entity given by typing errors



ETL: Transformation

- The process converts the data from the operational format to the DW format. The correspondence with the source level is made more complex by the heterogeneity of the different sources, that requires a complex integration step
 - Presence of free text data, hiding important information
 - Use of different formats for the same data

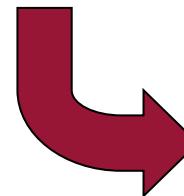




Cleaning & Transformation: Example

Carlo Bianchi
P.zza Grande 12
50126 Bologna (I)

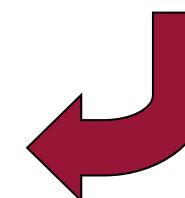
Normalization



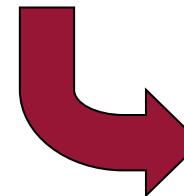
name: Carlo
surname: Bianchi
address: P.zza Grande 12
ZIP code: 50126
city: Bologna
country: I

name: Carlo
surname: Bianchi
address: Piazza Grande 12
ZIP code: 50126
city: Bologna
country: Italy

Standardization



Correction



name: Carlo
surname: Bianchi
address: Piazza Maggiore 12
ZIP code: 40126
city: Bologna
country: Italy



OLTP Systems

- **Traditional database** systems designed and tuned to support the day-to-day **operations**:
 - ensure fast, concurrent access to data
 - transaction processing and concurrency control
 - focus on online update data consistency
 - known as operational databases or *online transaction processing (OLTP)*
- **OLTP DB** data characteristics:
 - **Detailed** data
 - **Do not include historical** data
 - **Highly normalized**
 - Poor performance on complex queries including **joins** and aggregation
- **Data analysis** requires a **new paradigm**: *online analytical processing (OLAP)*
 - Typical OLTP query: “*pending orders for customer c1*”
 - Typical OLAP query: “*total sales amount by product and by customer*”



OLAP Systems

- The **online analytical processing (OLAP)** characteristics:
 - While OLTP paradigm focused on transactions, OLAP focused on **analytical queries**
 - Normalization not good for analytical queries, reconstructing data requires a high number of joins (denormalized tables in OLAP)
 - OLAP databases support a heavy query load
 - OLTP indexing techniques not efficient in OLAP: oriented to access few records
 - OLAP queries typically include aggregation
- The need for a different database model to support OLAP was clear: led to data warehouses



Comparison: OLTP vs OLAP

- On many aspects, there are several differences between OLTP and OLAP systems:

	OLTP	OLAP
User type	Operators, office employees	Managers, executives
Usage	Predictable, repetitive	Ad hoc, nonstructured
Data content	Current, detailed data	Historical, summarized data
Data organization	According to operational needs	According to analysis needs
Data structures	Optimized for small transactions	Optimized for complex queries
Access frequency	High	From medium to low
Access type	Read, insert, update, delete	Read, append only
Number of records per access	Few	Many
Response time	Short	Can be long
Concurrency level	High	Low
Lock utilization	Needed	Not needed
Update frequency	High	None
Data redundancy	Low (normalized tables)	High (denormalized tables)
Data modeling	UML, ER model	Multidimensional model



DW: Recap

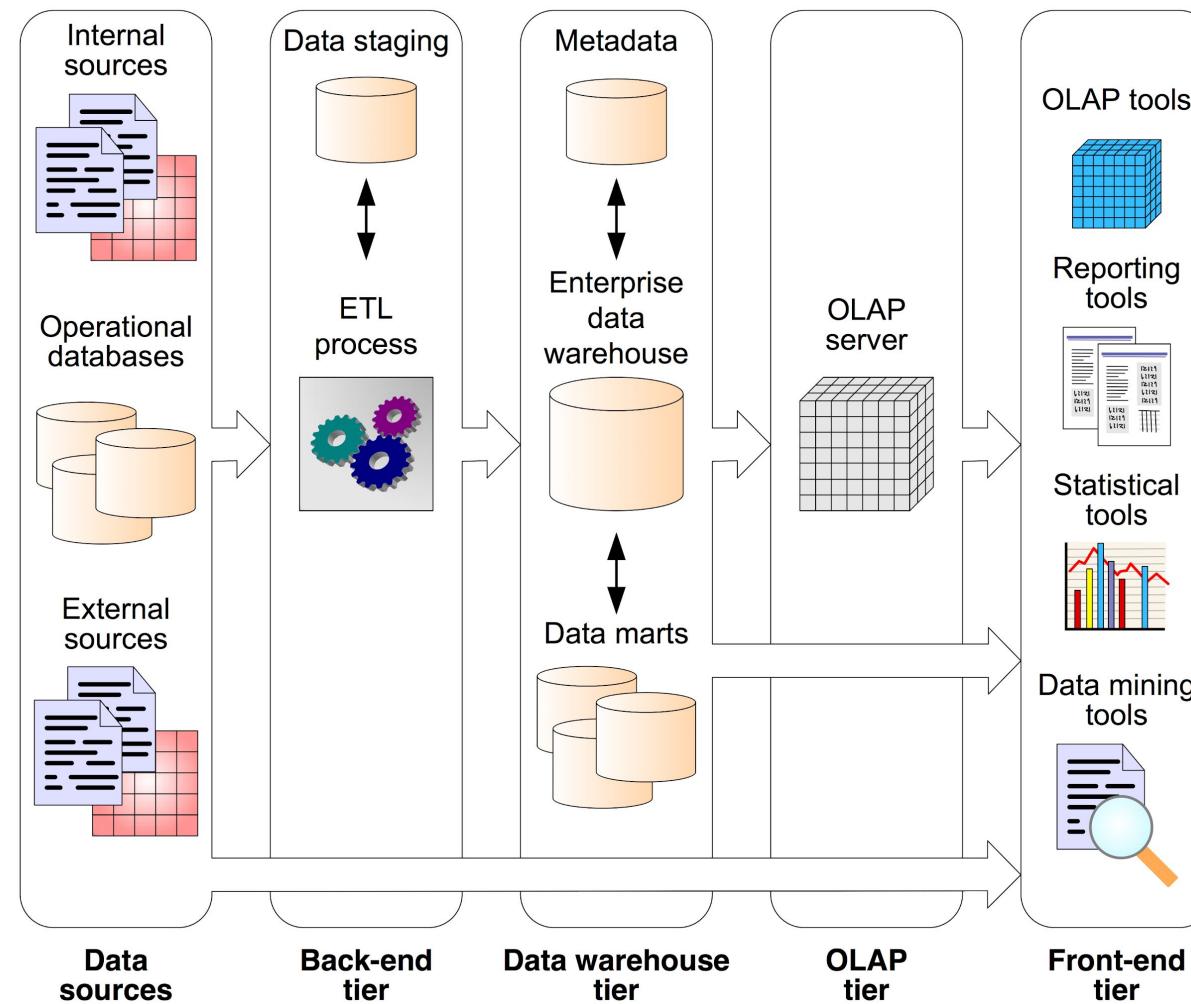
- Data warehouses:
 - large **repositories** that consolidate data from **different** sources (internal and external to the organization),
 - are updated **offline**,
 - follow the **multidimensional data model**,
 - implemented using **star** or **snowflake** schemas
 - to **efficiently support OLAP queries**
- We will now dive deeper into these concepts to better understand the **architecture**, **design**, and **querying** aspects of a data warehouse



Architecture: Multiple tiers

Tiers 1/3

- Despite many different architectures have been proposed, a common trait is the presence of **several tiers**





Tiers 2/3

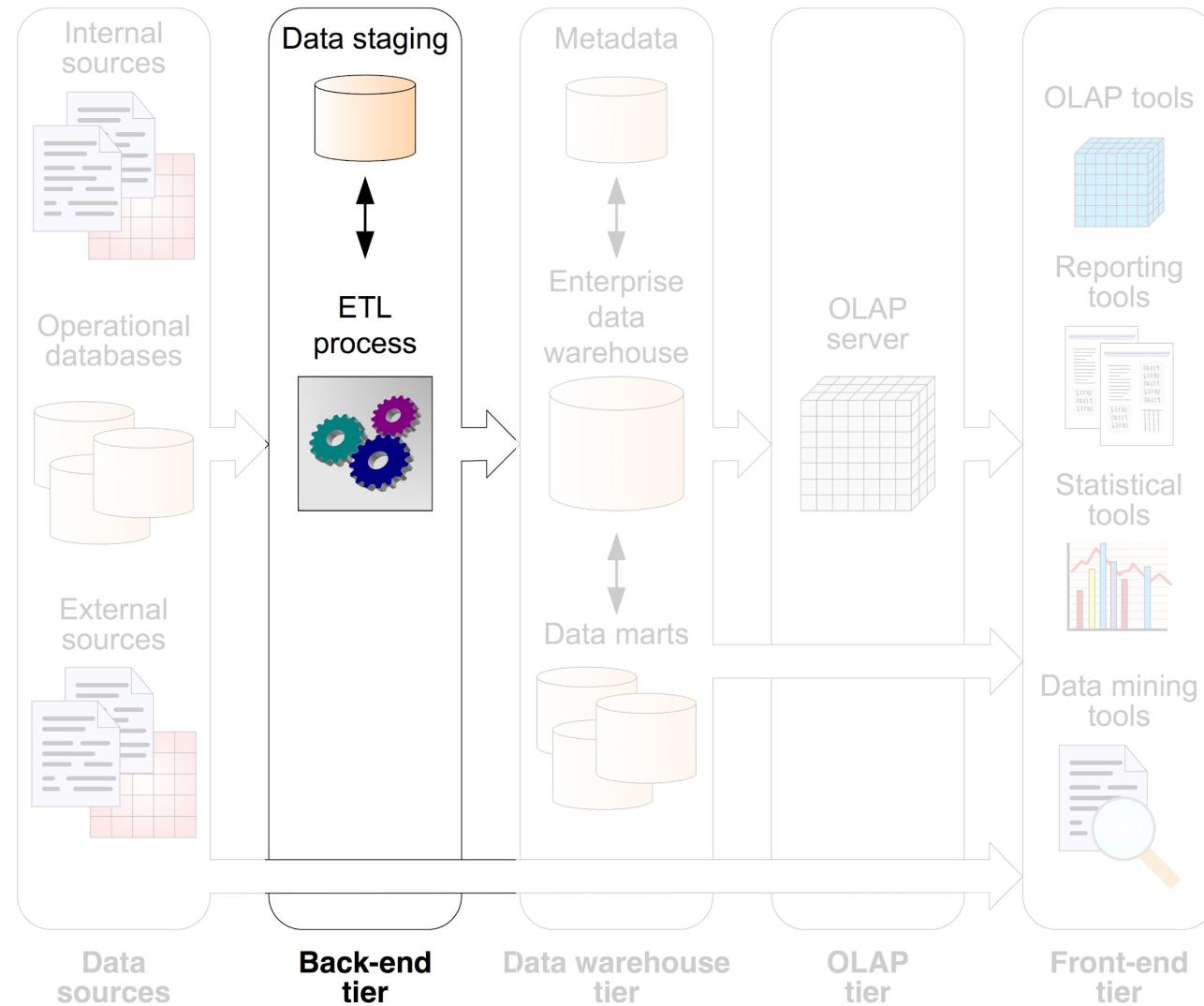
- **Data source** is not a tier of a data warehouse, but more of a collection of data of different kind and origins (sometimes referred as *data lake*)
- **Back-end** tier composed of:
 - The **extraction, transformation, and loading** (ETL) tools: feed data into the data warehouse from operational databases, internal and external data sources
 - The **data staging** area: an intermediate database where all the data integration and transformation processes are run prior to the loading of the data into the data warehouse



Tiers 3/3

- **Data warehouse** tier composed of:
 - An enterprise data warehouse and/or several data marts
 - A metadata repository storing information about the data warehouse and its contents
- **OLAP tier** composed of:
 - An OLAP server which provides a multidimensional view of the data, regardless the actual way in which data are stored
- **Front-end tier** is used for data analysis and visualization
 - Contains client tools such as OLAP tools, reporting tools, statistical tools, and data-mining tools

Back-end Tier 1/3





Back-end Tier 2/3

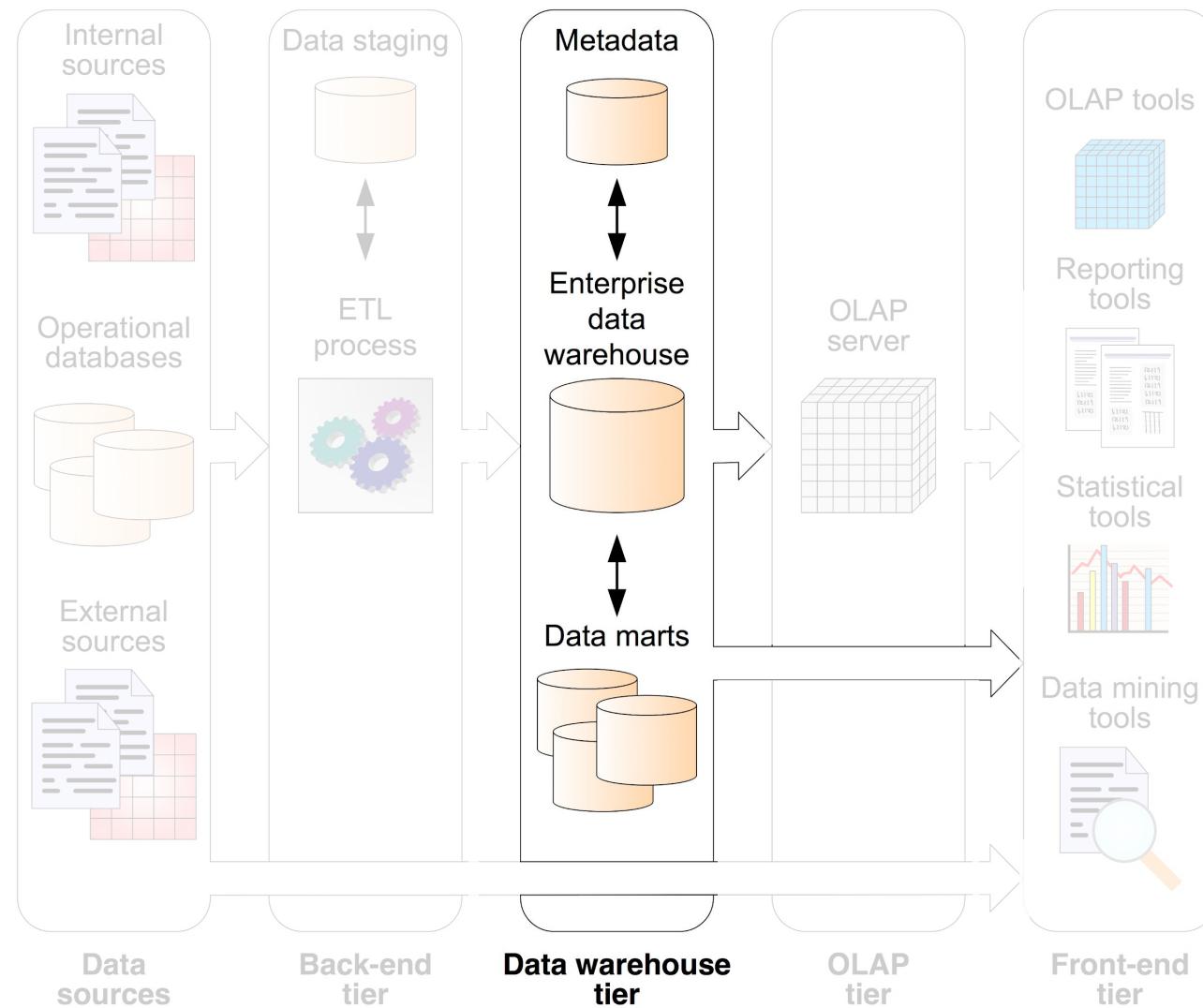
- Performs Extraction, Transformation, and Loading. It is a three-step process:
 - **Extraction** gathers data from multiple, heterogeneous data sources internal or external to the organization
 - **Transformation** modifies the data from the format of the data sources to the warehouse format; this includes:
 - **Cleaning**: removes errors and inconsistencies in the data and converts it into a standardized format
 - **Integration**: reconciles data from different data sources, both at the schema and at the data level
 - **Aggregation**: summarizes the data obtained from data sources according granularity of the data warehouse
 - **Loading** feeds the data warehouse with the transformed data, including refreshing the data warehouse, that is, propagating updates from the data sources to the data warehouse at a specified frequency



Back-end Tier 3/3

- Back-end tier has also a **data staging area** (usually called operational data store)
- The data staging area is a **database** where data extracted from the sources undergoes **successive modifications** before being loaded into the data warehouse

Data Warehouse Tier 1/3





Data Warehouse Tier 2/3

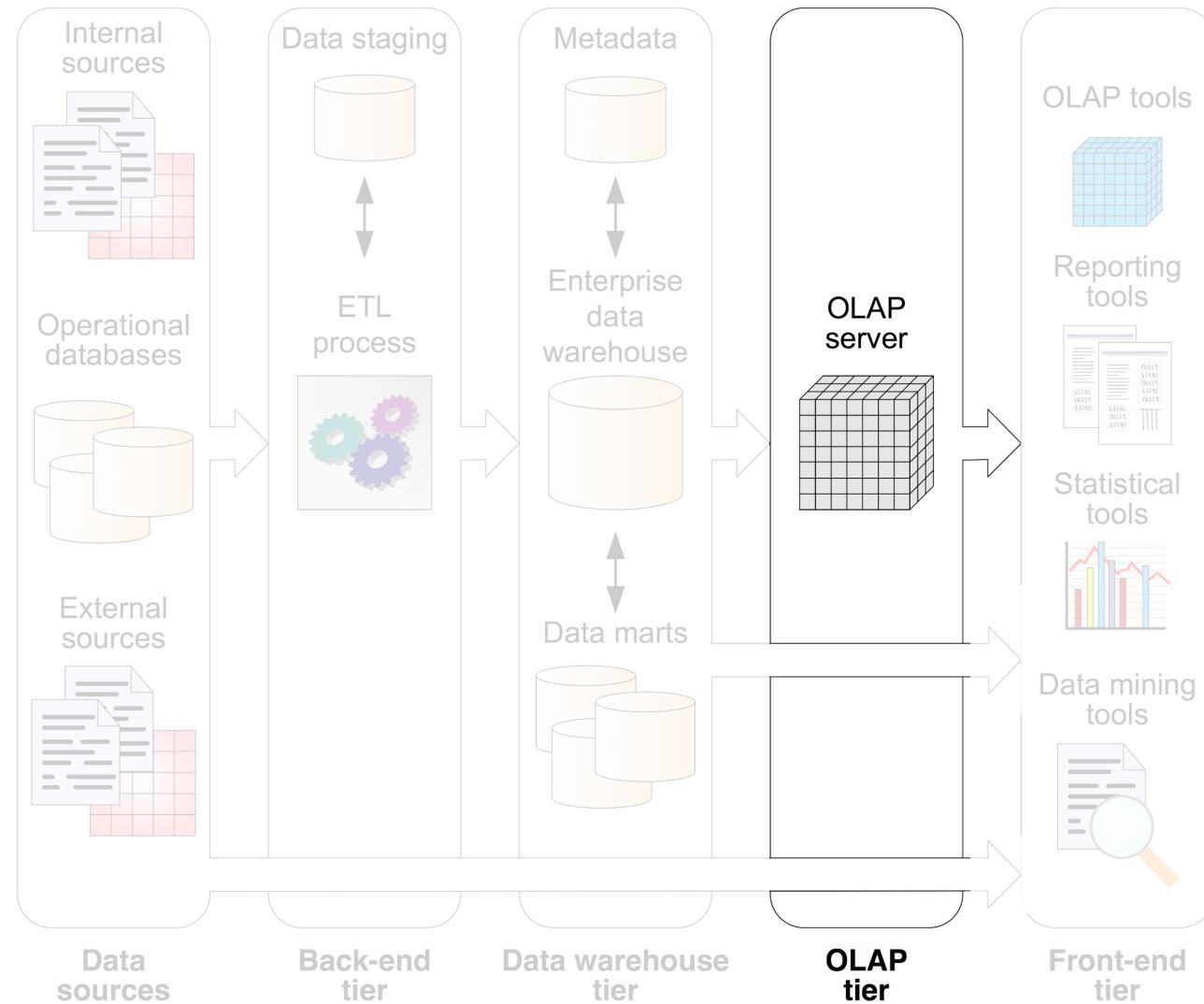
- **Components:**
 - An **enterprise data warehouse**, centralized and encompassing an entire organization
 - Several **data marts**: specialized departmental data warehouses
- **Metadata**
 - **Business metadata** describes the semantics of the data, and organizational rules, policies, and constraints related to the data
 - **Technical metadata** describes how data are structured and stored in a computer system, and the applications and processes that manipulate the data



Data Warehouse Tier 3/3

- The **metadata repository** of the data warehouse tier may contain information such as:
 - Metadata describing the **structure** of the data warehouse and the data marts, at the conceptual/logical level (facts, dimensions, hierarchies, ...) and at the physical level (indexes, partitions, ...)
 - **Security information** (user authorization and access control), and **monitoring information** (usage statistics, error reports, audit trails)
 - Metadata describing **data sources**: schemas, ownership, update frequencies, legal limitations, access methods
 - Metadata describing the **ETL**: data lineage, data extraction, cleaning, transformation rules, etc.

OLAP Tier 1/2



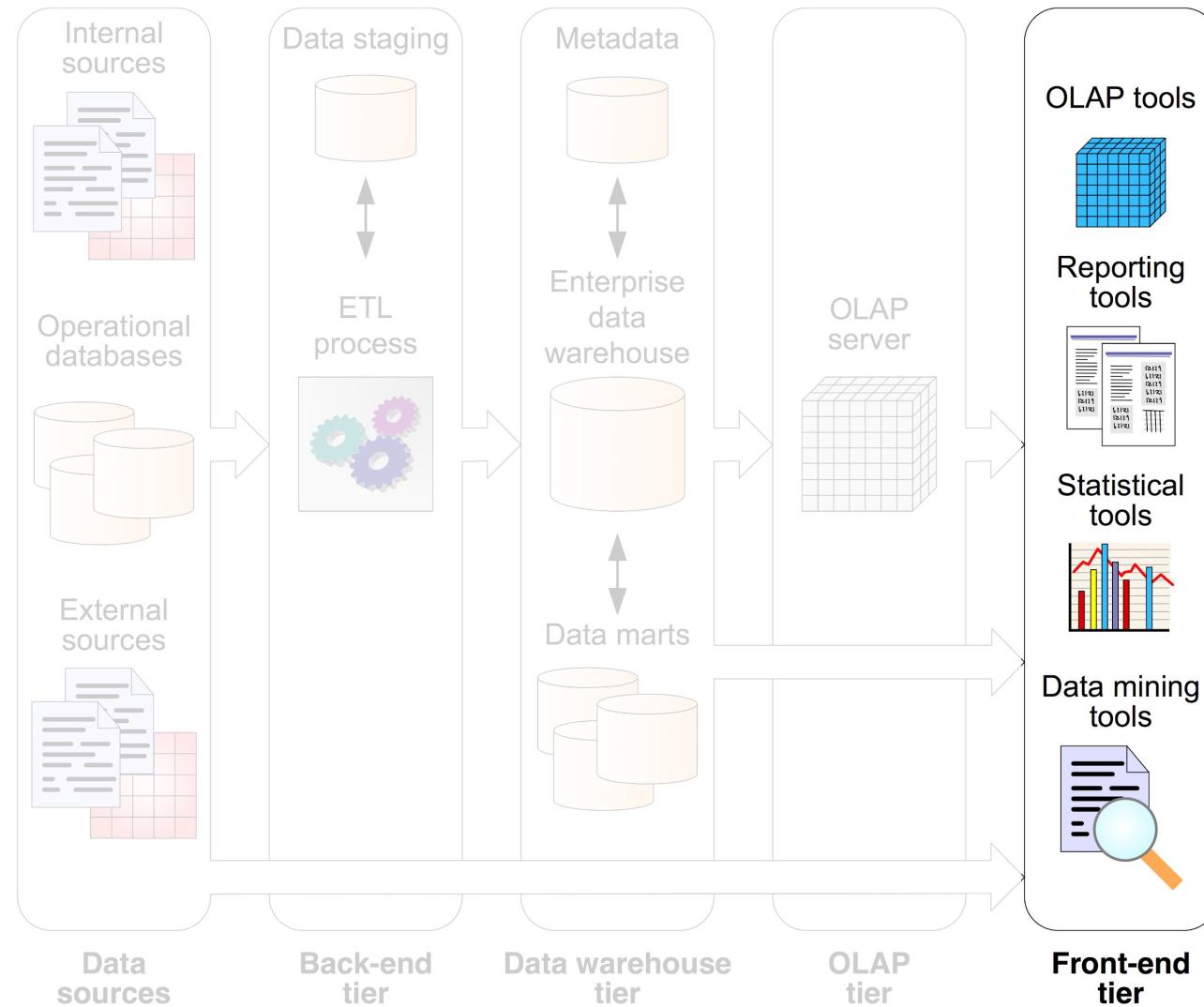


OLAP Tier 2/2

- A core component of data warehousing implementations for business intelligence (BI) and decision support applications
- OLAP cube allows to perform **multidimensional analysis at high speeds on large volumes** of data from a data warehouse, data mart, or some other unified, centralized data store
- The core of most OLAP systems, the OLAP cube is an array-based multidimensional database that makes it possible to process and analyze multiple data dimensions much more quickly and efficiently than a traditional relational database



Front-end Tier 1/2





Front-end Tier 2/2

Composed of client tools that allow users to exploit the content of the data warehouse:

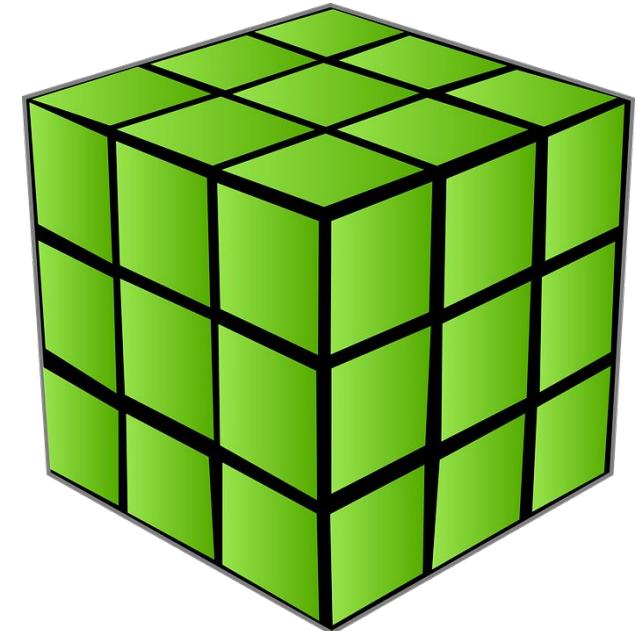
- **OLAP tools** allow interactive exploration and manipulation of the warehouse data and formulation of complex *ad hoc* queries
- **Reporting tools** enable the production, delivery, and management of reports, which can be paper-based, interactive, or web-based
 - Reports use **predefined queries** asking for specific information in a specific format, performed on a regular basis
- **Statistical tools** used to analyze and visualize the cube data using statistical methods
- **Data-mining tools** allow users to analyze data in order to discover valuable knowledge such as patterns and trends, and also allow to make predictions based on current data



Design: Multidimensional model

Design: Multidimensional model

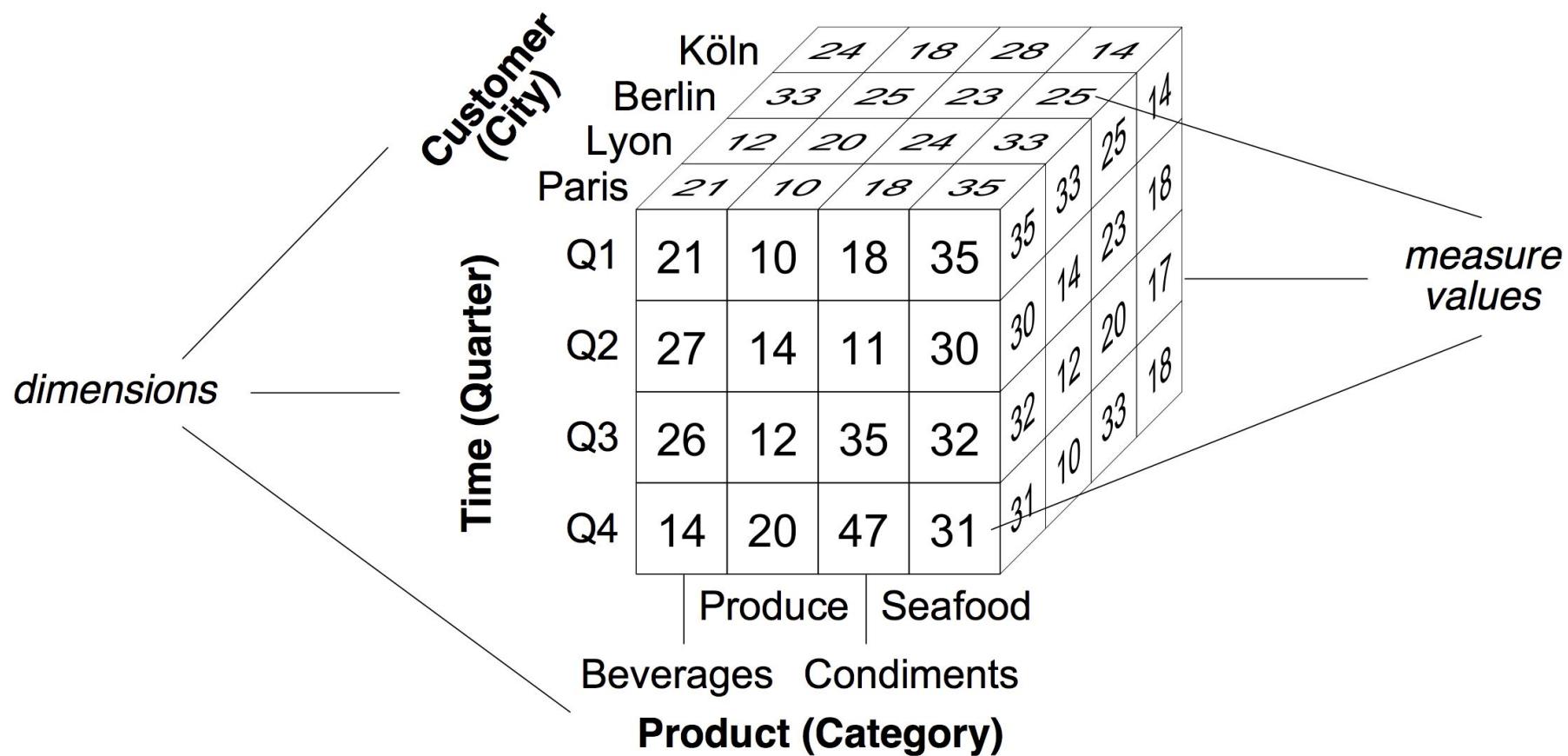
- The multidimensional model (MultiDim) views data in an **n-dimensional** space: a data cube
- A data cube is composed of **dimensions** and **facts**
- Remember that dimensions are **perspectives** used to analyze the data
 - Example: A three-dimensional cube for **sales** data with dimensions **Product**, **Time**, and **Customer**, and a measure **Quantity**



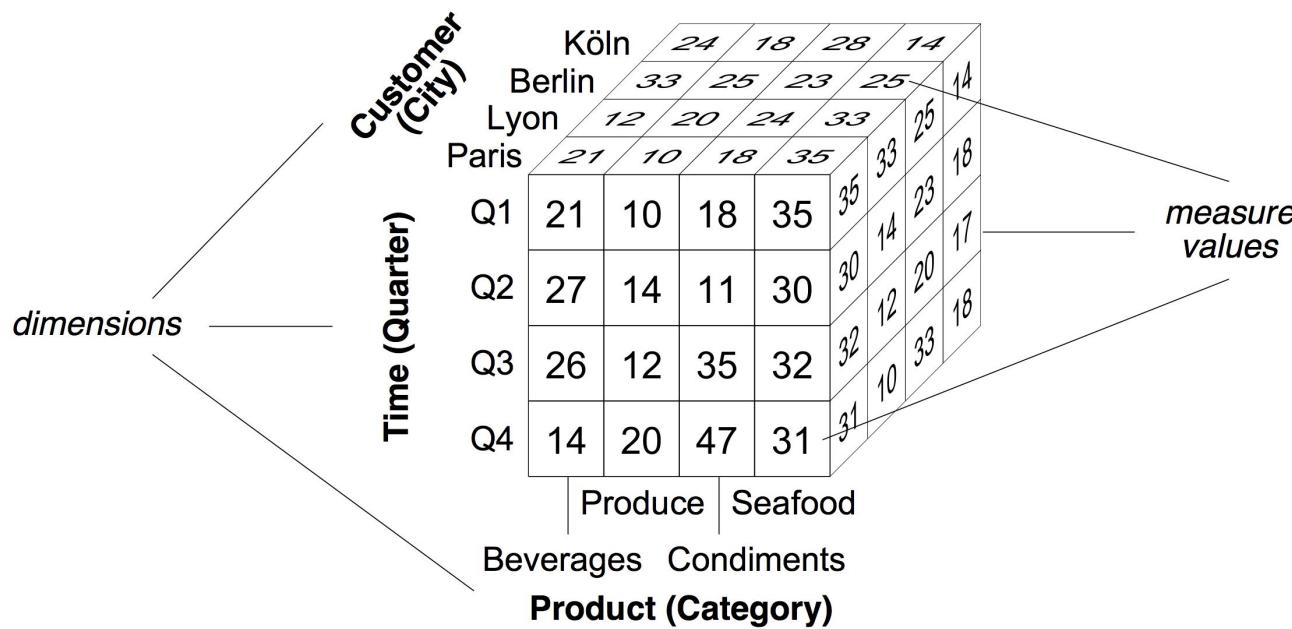


MultiDim: Example 1/2

- **Dimensions:** Customer, Time, Product
 - **Facts:** Sales (each “little cube”)
 - **Measures:** Quantity



MultiDim: Example 2/2

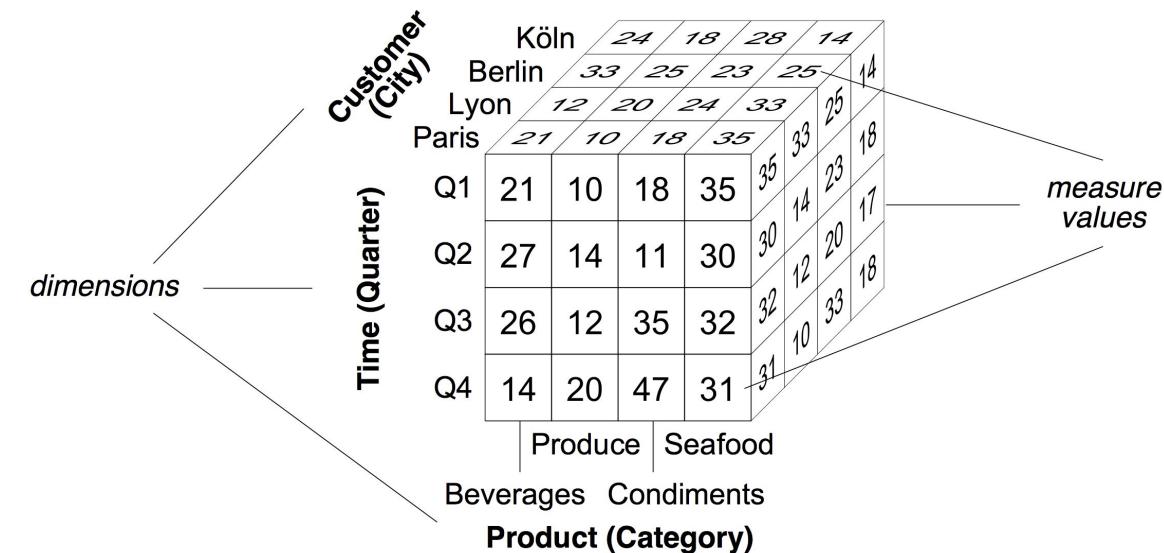


- **Attributes describe dimensions**: Product dimension may have attributes ProductNumber and UnitPrice (not shown in the figure)
- The cells or facts of a data cube have **associated numeric values** called **measures**
 - **Each cell (little cube) of the data cube represents Quantity of units sold by category, quarter, and customer's city**

Data granularity

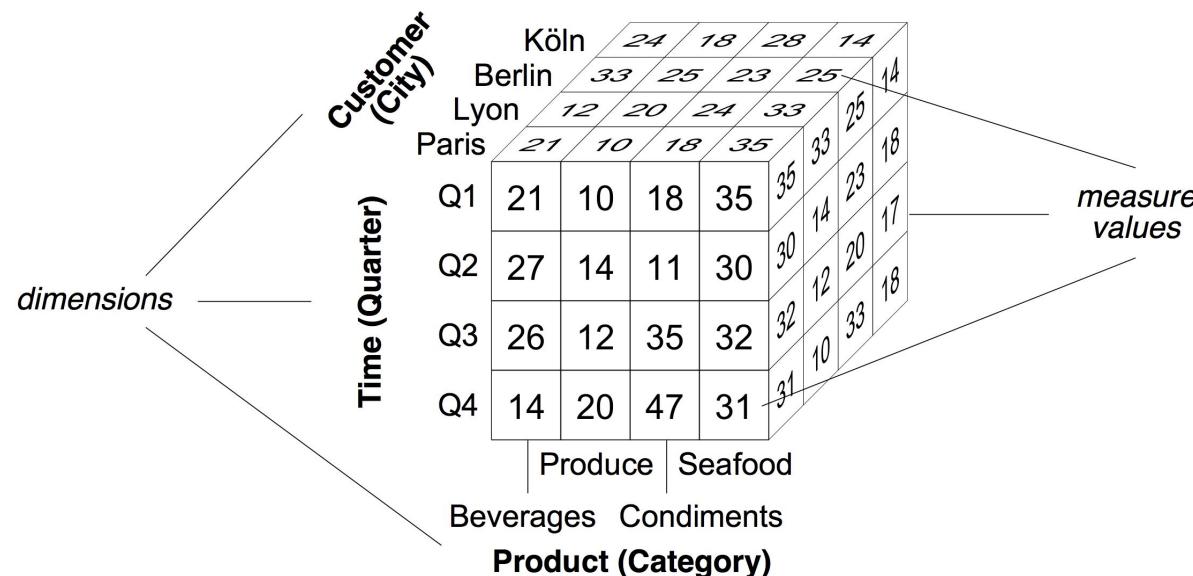
- **Data granularity:** level of detail at which measures are represented for each dimension of the cube
 - sales figures aggregated to granularities *Category*, *Quarter*, and *City*
 - **Instances** of a dimension are called **members**
 - *Seafood* and *Beverages* are members of the *Product* at the granularity *Category*
 - A data cube contains **several measures**
 - amount, indicating the total sales amount (not shown)
 - A data cube may be **sparse** (typical case) or dense
 - not all customers may have ordered products of all categories during all quarters

	Köln	Berlin	Lyon	Paris	
Q1	21	10	18	35	35
Q2	27	14	11	30	30
Q3	26	12	35	32	32
Q4	14	20	47	31	31
	24	18	28	14	14
	33	25	23	25	25
	12	20	24	33	33
	21	10	18	35	35



Data granularity: Hierarchies

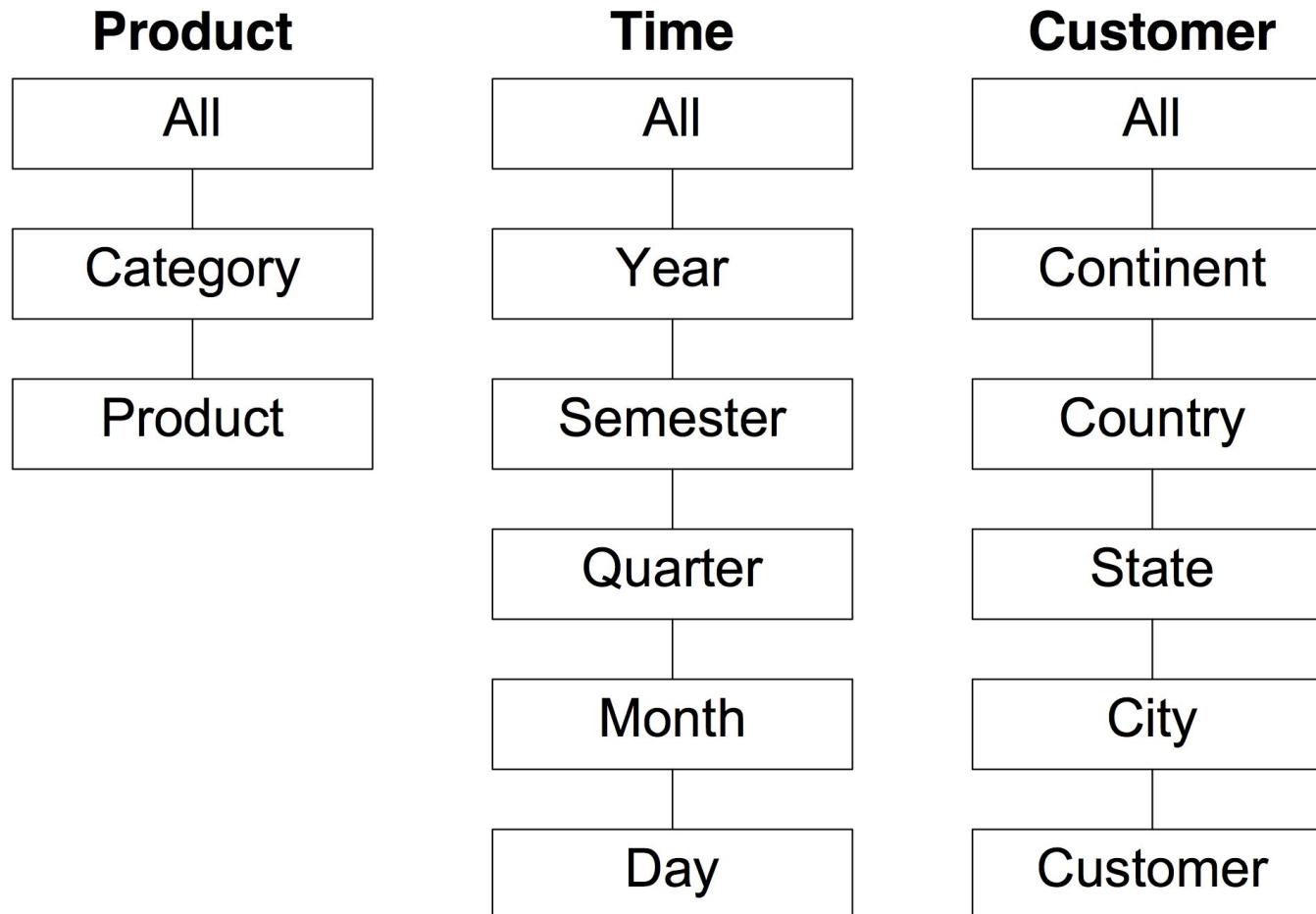
- **Hierarchies**: allow viewing data at **several granularities**
 - Define mappings relating **lower-level** detailed concepts to **higher-level** ones
 - The lower level is called the **child** and the higher level is called the **parent**
 - The hierarchical structure of a dimension is called the **dimension schema**
 - A **dimension instance** comprises **all members at all levels** in a dimension
 - In the example, granularity of each dimension indicated between parentheses: *Category* for the *Product* dimension, *Quarter* for *Time*, and *City* for *Customer*
 - We may want sales figures at a **finer granularity** (*Month*), or at a **coarser granularity** (*Country*)





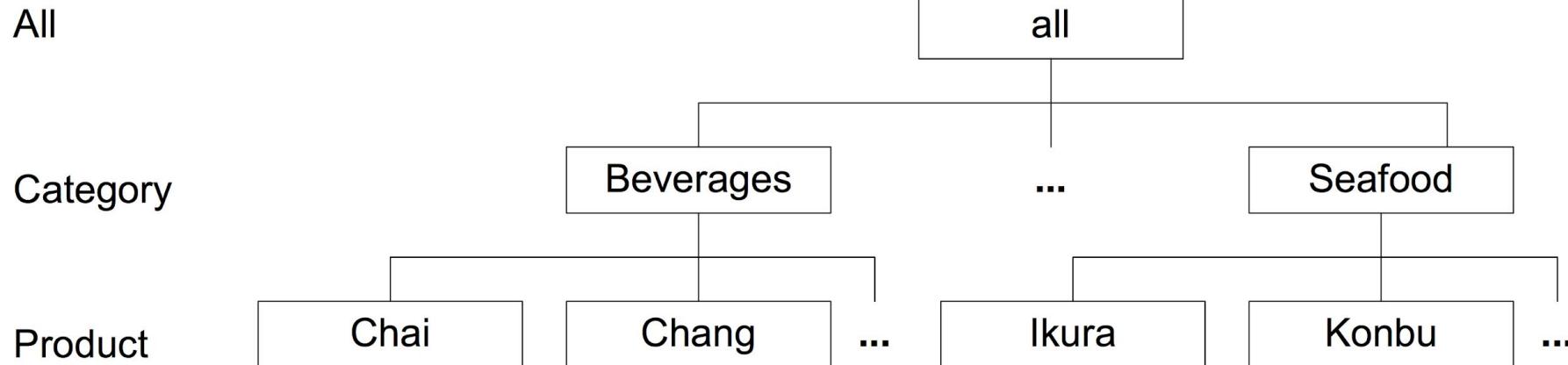
Hierarchies: an example

- **Hierarchies** of the *Product*, *Time*, and *Customer* dimensions



Hierarchies members

- In the same example, we can look at the **members** (instances) of the Product hierarchy (Product->Category->All)





Measures and Dimensions

- Aggregation of measures changes the abstraction level at which data in a cube is visualized
- Measures can be:
 - **Additive:** can be meaningfully summarized along all the dimensions, using addition
 - The most common type of measures
 - **Semiadditive:** can be meaningfully summarized using addition along some dimensions
 - Example: inventory quantities, which cannot be added along the Time dimension
 - **Nonadditive:** cannot be meaningfully summarized using addition across any dimension
 - Example: item price, cost per unit, and exchange rate

Other measures classifications

- Measures can also be:
 - **Distributive** measures are defined by an aggregation function that can be computed in a distributed way
 - Functions **count**, **sum**, **minimum**, and **maximum** are distributive, **distinct (values) count** is not
 - Example: $S = \{3,3,4,5,8,4,7,3,8\}$ partitioned in subsets $\{3,3,4\}$, $\{5,8,4\}$, $\{7,3,8\}$ gives a result of 8, while the answer over the original set is 5
 - **Algebraic** measures can be computed by an algebraic function with m arguments, each of which is obtained by applying a distributive aggregate function
 - Example: **average**, computed by dividing the sum by the count where both are distributive
 - **Holistic** measures cannot be computed from other subaggregates (e.g., median, rank, mostFrequent)



OLAP Querying

OLAP operations

- Taking the same example, we will now see a set of OLAP **operations** we can apply to **explore the cube** of sales (in thousands) by product category and customer cities for 2003:

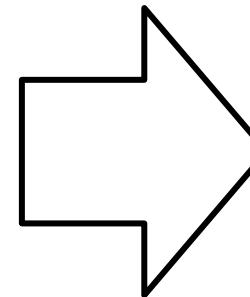
- **Roll-up**
- **Drill-down**
- **Sort**
- **Pivot**
- **Slice**
- **Dice**

		Customer (City)		Time (Quarter)					
		Köln	Berlin	Lyon	Paris	Q1	Q2	Q3	Q4
		24	18	28	14	21	10	18	35
		33	25	23	25	27	14	11	30
		12	20	24	33	26	12	35	32
		21	10	18	35	14	30	10	31
		35	23	17	18	35	12	20	18
		30	20	18	18	32	10	33	31
		32	20	18	18	31	10	33	31
		31	18	17	17	31	10	33	31
		Produce	Seafood	Beverages	Condiments	Product (Category)			

OLAP operation: Roll-up 1/2

- We compute the sales quantities by country: a **roll-up operation** to the *Country* level along the *Customer* dimension

Customer (City)	Köln				14
	Berlin	12	20	24	
Lyon	21	10	18	35	33
Paris	21	10	18	35	33
Q1	21	10	18	35	35
Q2	27	14	11	30	30
Q3	26	12	35	32	32
Q4	14	20	47	31	31
	Produce		Seafood		
Beverages		Condiments			
Product (Category)					



**Roll-up to the
Country level**

Customer (Country)	Germany				39
	France	57	43	51	
Q1	33	30	42	68	68
Q2	39	26	41	44	44
Q3	30	22	46	44	44
Q4	25	29	49	41	41
	Produce		Seafood		
Beverages		Condiments			
Product (Category)					

OLAP operation: Roll-up 2/2

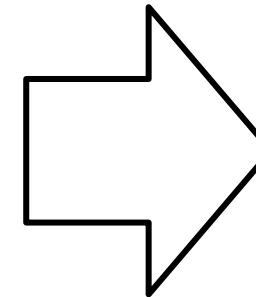
- After rolling-up we can look at measures at *Country* level
- We notice that France seafood sales for the first quarter are significantly higher

		Customer (Country)					
		Germany	France	57	43	51	39
		33	30	42	68	68	39
Q1		33	30	42	68	68	41
Q2		39	26	41	44	44	37
Q3		30	22	46	44	44	51
Q4		25	29	49	41	41	
				Produce	Seafood		
				Beverages	Condiments		
Product (Category)							

- To find out if this occurred during a particular month of quarter Q1, we take cube back to *City* aggregation level, and **drill-down** along *Time* to the *Month* level ...

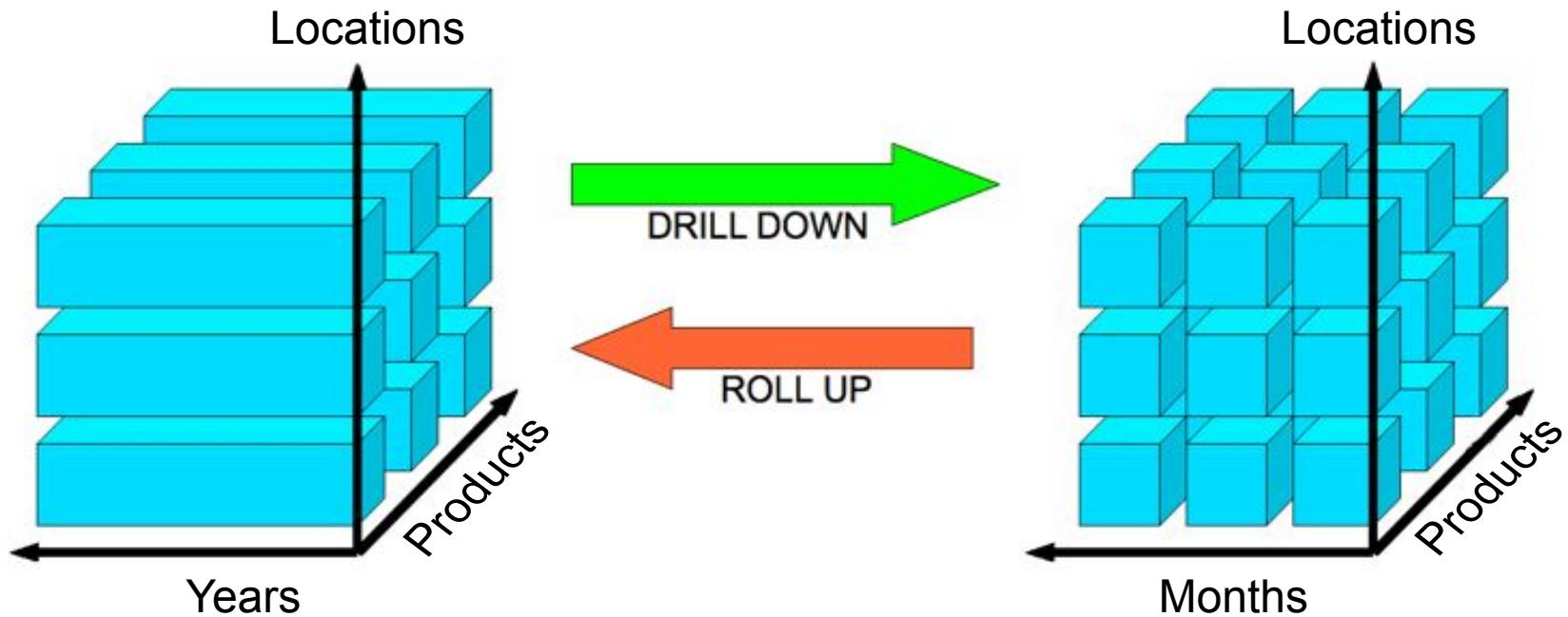
OLAP operation: Drill-down

- **Drilling-down**, we go from coarser granular data to finer granular data: from *Quarter* to *Month* granularity



Drill-down to the *Month* level

Roll-up/Drill-down in SQL: Example

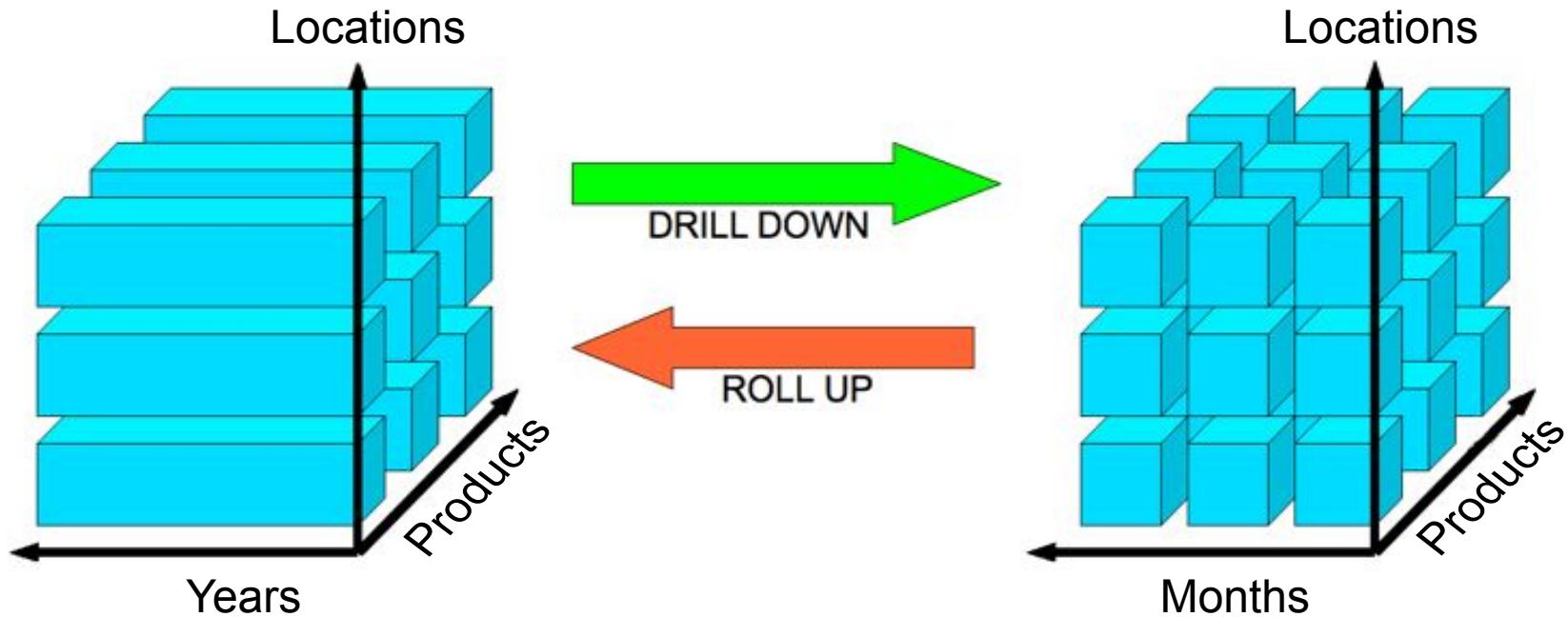


■ Roll-up:

```
SELECT SUM(F.sales)
FROM Time T JOIN Fact F JOIN Product P JOIN Location L
WHERE T.year = 2003
GROUP BY T.year, L.region, P.brand
```



Roll-up/Drill-down in SQL: Example



- **Drill-down:**

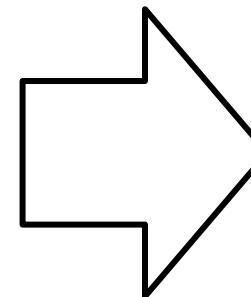
```
SELECT SUM(F.sales)
FROM Time T JOIN Fact F JOIN Product P JOIN Location L
WHERE T.year = 2003
GROUP BY T.month, L.region, P.brand
```

- Products are now in random order, we now want to visualize the original cube with sorted product ...

OLAP operation: Sort

- **Sorting** the *Product* dimension → the products, at *Category* level, are sorted alphabetically

Customer (City)	Köln	Q1				Q2				Q3				Q4									
		24	18	28	14	24	23	25	14	25	23	18	17	35	33	30	32	10	12	35	30	31	
Berlin	33	25	23	25	33	24	20	23	33	25	20	18	17	21	18	10	35	35	33	30	32	14	
Lyon	12	20	24	33	12	24	20	33	12	20	18	17	18	21	18	10	35	35	33	30	32	14	
Paris	21	10	18	35	21	10	18	35	21	18	10	35	18	21	18	10	35	35	33	30	32	14	
Time (Quarter)	Köln	24	18	28	14	24	23	25	14	25	23	18	17	35	33	30	32	10	12	35	30	31	14
Q1	21	10	18	35	21	10	18	35	21	18	10	35	18	21	18	10	35	35	33	30	32	14	
Q2	27	14	11	30	27	14	11	30	27	12	10	30	18	27	11	14	30	30	27	20	18	14	
Q3	26	12	35	32	26	12	35	32	26	32	10	32	18	26	35	12	32	32	26	10	33	18	
Q4	14	20	47	31	14	20	47	31	14	47	20	31	17	14	47	20	31	31	14	20	31	17	
Product (Category)	Produce	Seafood	Condiments	Beverages	Produce	Seafood	Condiments	Beverages	Produce	Seafood	Condiments	Beverages	Produce	Produce	Seafood	Condiments	Beverages	Produce	Seafood	Condiments	Beverages		

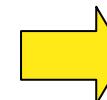


**Sort of the
Product dimension**

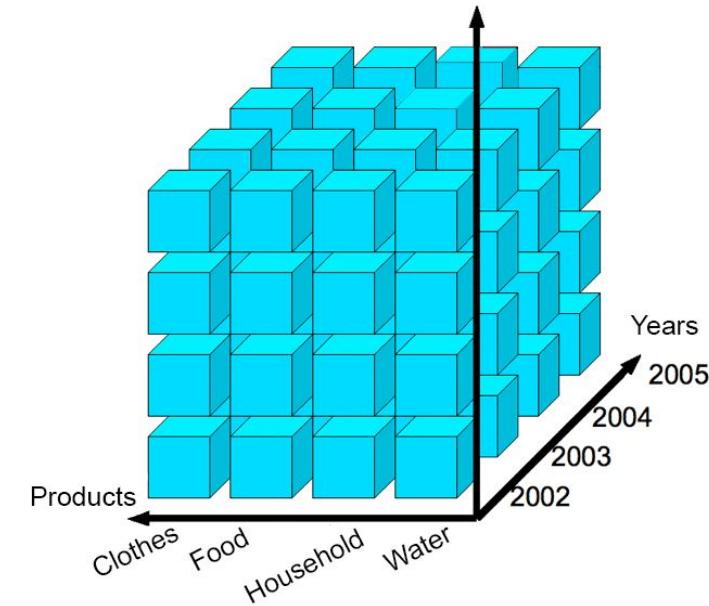
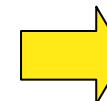
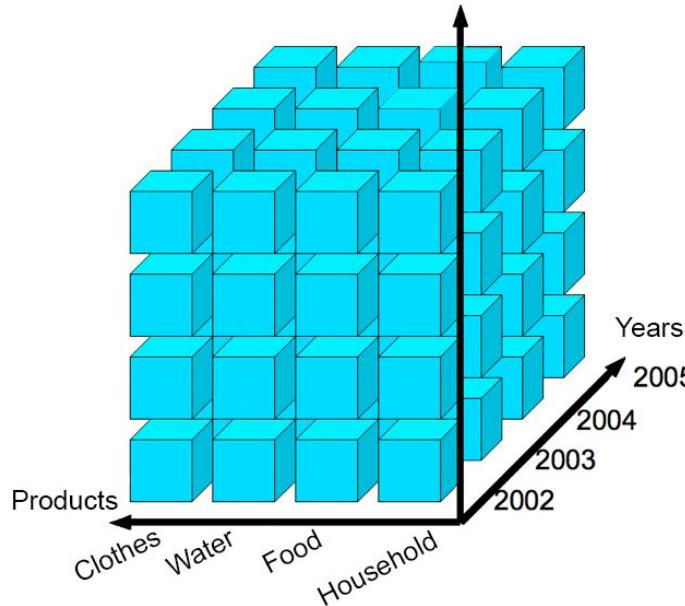
Customer (City)	Köln	Q1				Q2				Q3				Q4									
		24	28	18	14	24	23	25	25	25	23	20	18	35	33	30	32	10	12	35	30	31	14
Berlin	33	25	23	25	33	24	20	33	33	25	20	18	17	21	18	10	35	35	33	30	32	14	
Lyon	12	20	24	33	12	24	20	33	12	20	18	17	18	21	18	10	35	35	33	30	32	14	
Paris	21	10	18	35	21	10	18	35	21	18	10	35	18	21	18	10	35	35	33	30	32	14	
Time (Quarter)	Köln	24	28	18	14	24	23	25	25	25	23	20	18	35	33	30	32	10	12	35	30	31	14
Q1	21	18	10	35	21	18	10	35	21	18	10	35	18	21	18	10	35	35	33	30	32	14	
Q2	27	11	14	30	27	11	14	30	27	12	10	30	18	27	11	14	30	30	27	20	18	14	
Q3	26	35	12	32	26	35	12	32	26	32	10	32	18	26	35	12	32	32	26	10	33	18	
Q4	14	47	20	31	14	47	20	31	14	47	20	31	17	14	47	20	31	31	14	20	31	17	
Product (Category)	Condiments	Seafood	Condiments	Beverages	Produce	Seafood	Condiments	Beverages	Produce	Seafood	Condiments	Beverages	Produce	Produce	Seafood	Condiments	Beverages	Produce	Seafood	Condiments	Beverages		

Sort in SQL: Example

```
SELECT SUM(F.sales)
FROM Time T JOIN Fact F
JOIN Products P
JOIN Locations L
```



```
SELECT SUM(F.sales)
FROM Time T JOIN Fact F
JOIN Products P
JOIN Locations L
ORDER BY P.category
```

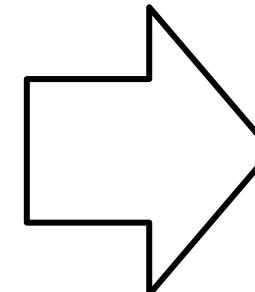


- To introduce the next operation, suppose that we want to rotate the cube to see the *Time* dimension on the X axis ...

OLAP operation: Pivot

- By using the **pivot** operation, the *Time* dimension is now on the **X axis**, the *Customer* dimension is now on the **Y axis**, while the *Product* dimension is on the **Z axis**

Customer (City)	Time (Quarter)	Köln				Customer (City)	
		Berlin	Lyon	Paris	Köln		
	Q1	21	10	18	35		
	Q2	27	14	11	30		
	Q3	26	12	35	32		
	Q4	14	20	47	31		
		Produce	Seafood	Beverages	Condiments		
		Product (Category)					



Pivot

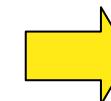
Customer (City)	Time (Quarter)	Seafood				Customer (City)	
		Condiments	Produce	Beverages	Seafood		
	Q1	21	27	26	14		
	Q2	12	14	11	13	13	28
	Q3	33	28	35	32	32	19
	Q4	24	23	25	18	18	47
		Time (Quarter)					

- Let's see an example in SQL, how rotate the data axes to provide a substitute presentation of data

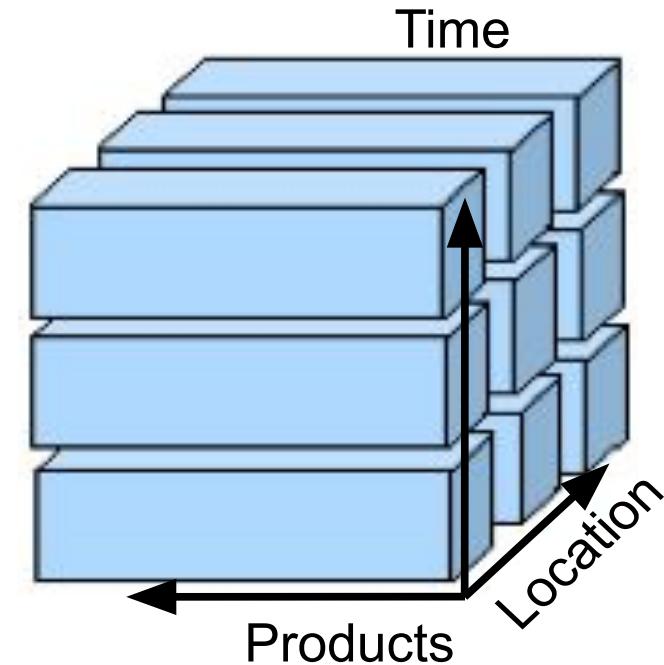
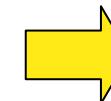
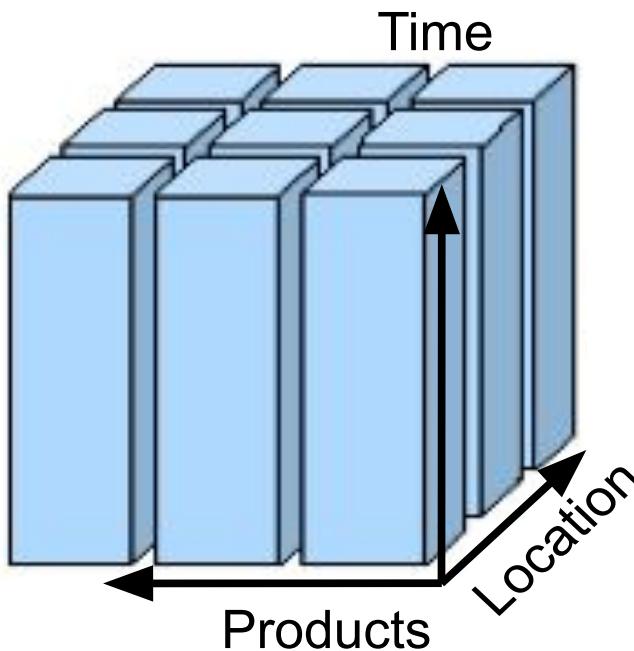


Pivot in SQL: Example

```
SELECT SUM(F.sales)
FROM Time T JOIN Fact F
JOIN Products P
JOIN Locations L
GROUP BY P.category
```



```
SELECT SUM(F.sales)
FROM Time T JOIN Fact F
JOIN Products P
JOIN Locations L
GROUP BY T.year
```



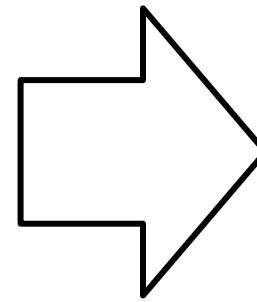
- What about filtering our data? Let's suppose we want to look at a single *City*

...

OLAP operation: Slice

- The **slice** operation visualizes the data for Paris only, resulting in a 2-dimensional “subcube”

Customer (City)	Köln				Berlin				Lyon				Paris			
	Beverages	Condiments	Produce	Seafood	Beverages	Condiments	Produce	Seafood	Beverages	Condiments	Produce	Seafood	Beverages	Condiments	Produce	Seafood
Time (Quarter)	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
Köln	24	18	28	14	33	25	23	25	12	20	24	33	21	10	18	35
Berlin																
Lyon																
Paris																
Q1	21	10	18	35	35	33	23	17	27	14	11	30	26	12	35	32
Q2	27	14	11	30	30	12	20	18	26	19	15	34	28	16	39	31
Q3	26	12	35	32	32	10	33	19	29	17	14	41	27	13	37	33
Q4	14	20	47	31	31	30	38	25	35	22	19	49	33	15	45	36
	Produce	Seafood														
	Beverages	Condiments														
	Product (Category)															



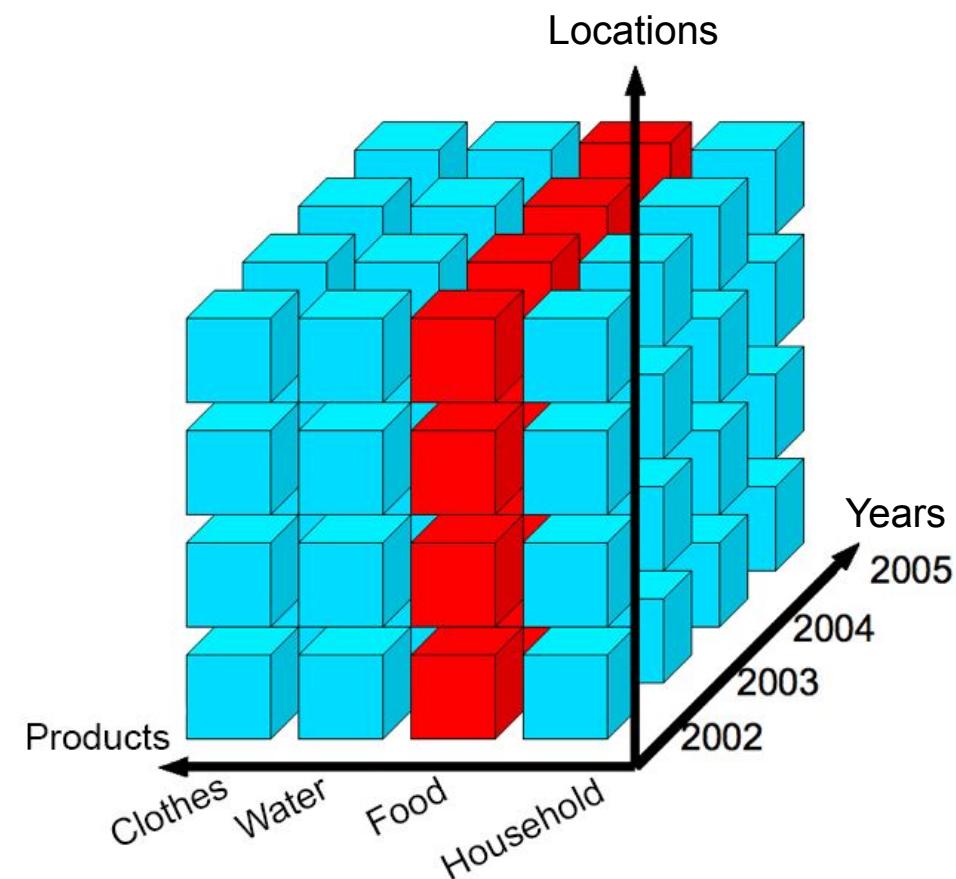
**Slice on
City = “Paris”**

Time (Quarter)	Paris			
	Q1	Q2	Q3	Q4
Produce	21	10	18	35
Seafood	35	33	23	17
Beverages	27	14	11	30
Condiments	26	12	35	32
Product (Category)	14	20	47	31

- Let's see an example in SQL

Slice in SQL: Example 1

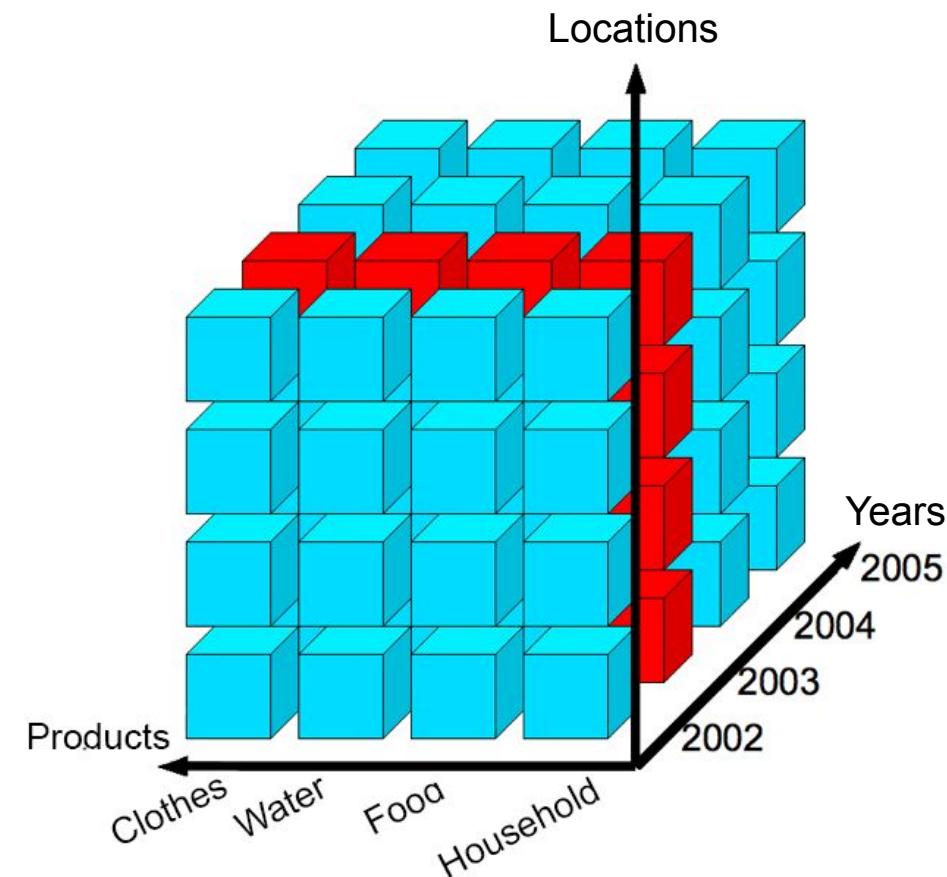
```
SELECT SUM(F.sales)
FROM Time T JOIN Fact F
JOIN Products P
JOIN Locations L
WHERE P.category="Food"
```





Slice in SQL: Example 2

```
SELECT SUM(F.sales)
FROM Time T JOIN Fact F
JOIN Products P
JOIN Locations L
WHERE T.years = 2003
```

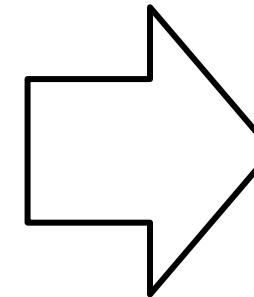


- Slice allows to show only a single value for a single dimension. What if we want to select multiple values from different dimensions?

OLAP operation: Dice

- With **dice** we can select over multiple conditions and multiple dimensions, producing a 3-dimensional subcube

Customer (City)	Time (Quarter)	Köln				14
		24	18	28	14	
Berlin		33	25	23	25	14
Lyon		12	20	24	33	25
Paris		21	10	18	35	18
Q1	21	10	18	35	35	17
Q2	27	14	11	30	30	20
Q3	26	12	35	32	32	18
Q4	14	20	47	31	31	33
		Produce	Seafood			
		Beverages	Condiments			
		Product (Category)				



Dice on *City*=‘Paris’ or
‘Lyon’ and
Quarter=‘Q1’ or ‘Q2’

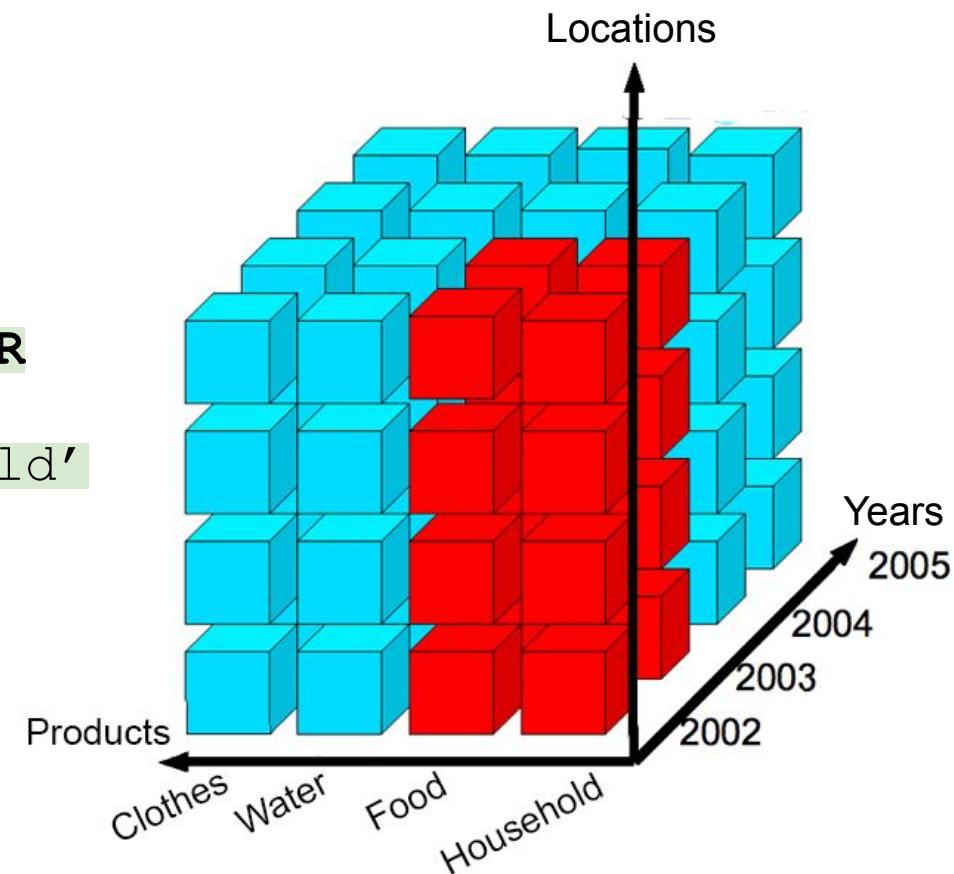
Customer (City)	Time (Quarter)	Lyon				33
		12	20	24	33	
Paris		21	10	18	35	35
Q1	21	10	18	35	35	14
Q2	27	14	11	30	30	33
		Produce	Seafood			
		Beverages	Condiments			
		Product (Category)				

- Let’s see an example in SQL



Dice in SQL: Example

```
SELECT SUM(F.sales)
FROM Time T JOIN Fact F
JOIN Products P
JOIN Locations L
WHERE T.years < 2003 AND
      P.category LIKE
          'Food' OR
      P.category LIKE
          'Household'
```





Data Warehouses examples

- [Snowflake](#)
- [Google BigQuery](#)
- [Amazon Redshift](#)
- [Azure Synapse Analytics](#)
- [IBM Db2 Warehouse](#)
- [Firebolt](#)



Do data streams turn to data floods?

- In complex business environment, **data is essential**. The **greatest challenge** organizations face is **how to organize** their data, rethinking data infrastructure
- Moving to an **as-a-service model** to store and manage data can offload data-management tedium and free up resources
 1. Assess your current needs and forecast your future ones before choosing a vendor
 2. Understand your potential vendors' roadmaps for achieving the capabilities you expect to need
 3. Beware of lock-in
 4. Promote data competency among your employees, for both business decision-making and innovation
 5. Assess and strengthen your cyber-resilience

A new era for data: What's possible with as-a-service

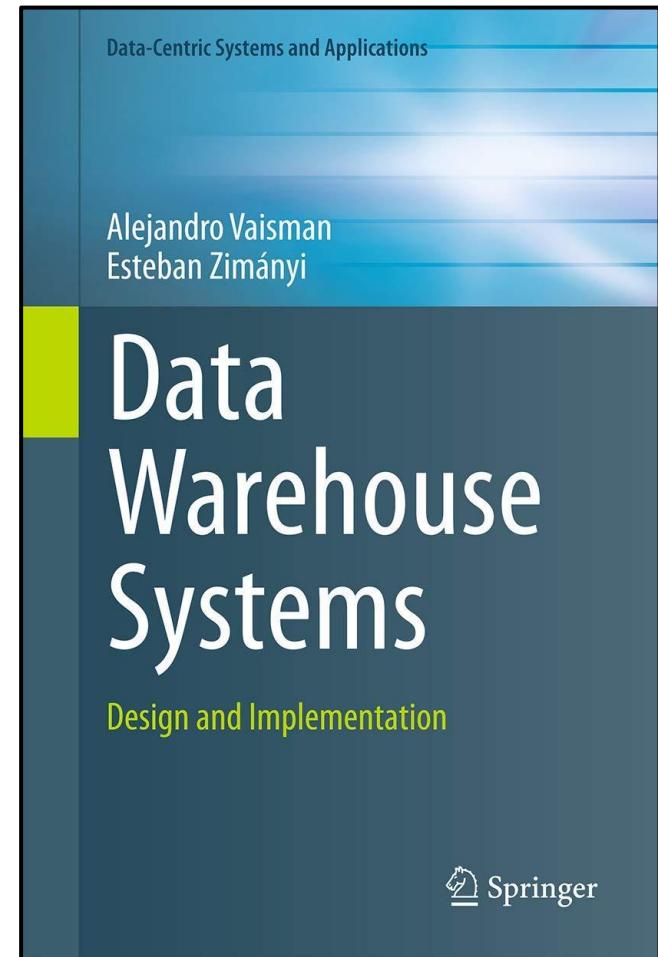


References

Data Warehouse Systems Design and Implementation

Authors: Vaisman, Alejandro,
Zimányi, Esteban

Extensive coverage of all data warehouse issues, ranging from basic technologies to the most recent findings and systems





Big Data and Data Mining

Data Mining Introduction and Preprocessing

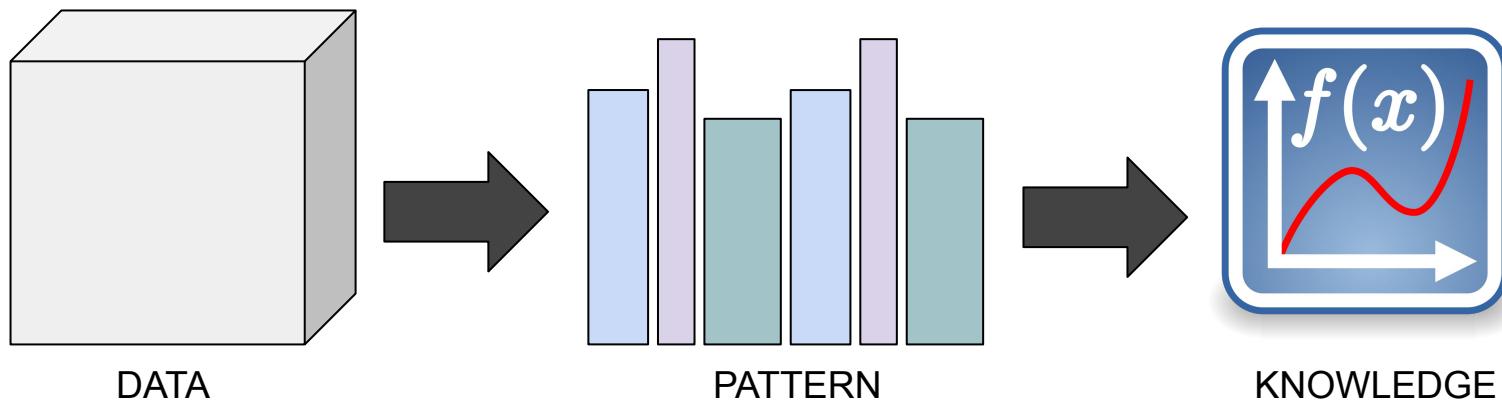
Flavio Bertini

flavio.bertini@unipr.it

Data Mining

*“Data mining is the computing process of discovering patterns in large data sets involving methods at the intersection of **machine learning, statistics, and database systems**”*

- The goal is the extraction of **patterns** and **knowledge** from large amount of data
- We will see now some example of patterns, knowledge and large amount of data involved in data mining





Patterns

- **Patterns** are regularities in the data, that - approximately - repeat for all of the observations in a predictable manner
- Examples of patterns:
 - In most cases, houses price in € is approximately 2500 times their square meters
 - Frequently, milk and cereals are bought together
 - It has been observed that people buy digital equipment in this order
 - 1) Personal Computer
 - 2) Digital Camera
 - 3) Memory Card



Knowledge

- Once a pattern is discovered, we can understand the logic behind the pattern, so that we can **describe** the pattern and **predict** similar phenomena
- **Knowledge** is a novel understanding on a subject
- Examples discovered knowledge:
 - How house prices are being set depending on square meters
 - Which products should be put closer in the supermarket
 - In which order digital equipment should be advertised in targeted marketing



Large amount of data

- In order to find patterns and deduce knowledge, we need to start from several **observations**
- In Data Mining observations are provided in the form of homogeneous **data** examples:
 - Many examples of house prices and their related size
 - Many examples of groceries transaction with the set of bought items
 - Many examples of sequences of purchases in the digital department of a superstore
- **Large amount** of data is needed in order to extract pattern or knowledge which are statistically significant
 - If we know only the price of three houses it may be due to chance, not a regularity!



Data Mining: Steps

1. **Define the problem:** what is the goal we are trying to achieve? What knowledge we'd like to extract?
2. **Identify required data:** what data do we need in order to pursue the goal? In this step we collect and understand the data
3. **Prepare and pre-process:** select and cleanse the required data. Format the data to be the input of a *machine learning* algorithm (we will see what machine learning is in a moment)
4. **Model the hypothesis:** select machine learning algorithms and tune parameters to build an accurate “predictive” model
5. **Train and test:** train algorithms using a sample of the data, test it on unseen data
6. **Verify and deploy:** verify final model with stakeholders (final users), prepare visualization and deploy



In this course

- In this course we will assume that:
 1. Goal is already given
 2. Required data is already given
 6. Visualization and deployment vary strongly depending on the stakeholders needs and may go outside the scope of data mining techniques
- This leaves us with 3 steps:
 3. **Prepare and pre-process:** select and cleanse the required data. Format the data to be the input of a machine learning algorithm
 4. **Model the hypothesis:** select machine learning algorithms and tune parameters to build an accurate “descriptive” and then “predictive” model
 5. **Train and test:** train algorithms using a sample of the data, test it on unseen data



Machine Learning

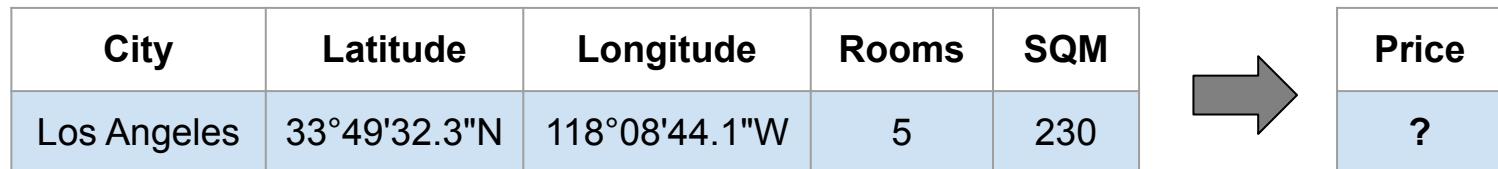
- Machine learning (ML) is at the core of data mining, because it is where the abstraction from data to patterns happens
- **Basic idea:** given many observations, a function with a specified goal (e.g., guess a house's price given its features) is “learned” automatically from the data
- Why humans can't learn patterns directly from the data?
 - Data can be **very large** (many thousands of records) and **high dimensional** (many variables per record)
 - **Non-linear** relations between variables can be very hard to catch
 - Manually coding some algorithms can be incredibly **hard**: how would you code a program to detect cats in a picture, starting from raw pixels?

Example

- We have the data from 10 thousand houses in the U.S.

City	Latitude	Longitude	Rooms	SQM	Price
Los Angeles	33°51'37.0"N	118°08'22.2"W	3	130	420000
Los Angeles	33°50'17.7"N	118°09'12.6"W	2	60	380000
...
Albuquerque	35°14'22.0"N	106°26'26.2"W	2	140	220000
Albuquerque	35°32'23.0"N	106°38'21.2"W	3	150	250000

- Goal:** learn a function (model) that can infer the *Price* given all the other variables, for houses not in the 10 thousand dataset:





Probably, approximately correct!

- Being based on statistics, the patterns and knowledge extracted in data mining is almost **never 100% accurate**
 - many variables could be involved which are not in the data (e.g., a house seller may want to quick sell the house at a lower price)
 - data coming from the real world is affected by noise and randomness (e.g. mistyped price value)
- Machine learning foundation is the P.A.C. theory: **Probably Approximately Correct**
 - **Probably**: the model of the real world abstracted using machine learning **should be correct** with high probability
 - **Approximately**: the model should have low **generalization** error, meaning that it should **approximate the general case** and be accurate with unseen data



ML: common notions

- **Example:** an observation, such as the data of a single house
- **Input variables:** the variables that are given in input for each example (e.g., the square meters)
- **Output variable:** the variable we want to infer on the basis of the input variables (e.g., the house price)
- **Hypothesis** (a.k.a. model): the function from the input variables to the output variable learned by a machine learning algorithm
- **Label:** in supervised tasks (we'll see in a moment), it's the value of the output variable for a certain input instance. It is given from the data, it's considered the true value
- **Prediction:** it's the value of the output variable "guessed" by the hypothesis function. It should be correct with high probability
 - Note that here *prediction* does not stand for *forecast*, time may be not involved at all
 - Any time you predict into the future it is a forecast. All forecasts are predictions, but not all predictions are forecasts, as when you would use regression to explain the relationship between two variables

Notation

- A common notation is to denote with the:
 - capital letter \mathbf{X} the input data matrix
 - lowercase \mathbf{y} the output variable vector
 - lowercase m the number of examples
 - lowercase n the number of variables of the examples

\mathbf{X}

City	Latitude	Longitude	Rooms	SQM
Los Angeles	33°51'37.0"N	118°08'22.2"W	3	130
Los Angeles	33°50'17.7"N	118°09'12.6"W	2	60
Los Angeles	33°49'32.3"N	118°08'44.1"W	5	230
...
Albuquerque	35°14'22.0"N	106°26'26.2"W	2	140
Albuquerque	35°32'23.0"N	106°38'21.2"W	3	150

m

\mathbf{y}

Price
420000
380000
2500000
...
220000
250000

n



Supervised techniques

- **Supervised** techniques: a “training set” of data is given to the machine learning algorithm, with a label for each example, representing the *ground truth* for the output value
- It’s **supervised** because we tell the algorithm **what should be the output for a set of examples**
 - The output variable is part of the dataset
 - **Example:** the actual price is given, along with the input variables
- Supervised learning has two tasks depending on the type of output variable:
 - **Regression:** when the output variable is a numerical value. For example in the problem of predicting the price of the house
 - **Classification:** when the output variable is a class. The easiest scenario is the binary classification problem where we want to predict if an example belong to a class or not (e.g., is the house located in Los Angeles? Yes or No)

Supervised ML: an example (1)

- The houses' prices example is a **supervised** learning task:
 - We have a large training set (data from 10K houses)
 - Among the variables there is the output variable (Price)
 - The output variable is the *ground truth*, because it's the actual price of the house
- The kind of task is a **regression** problem:
 - The **output variable we want to predict is a numerical value**

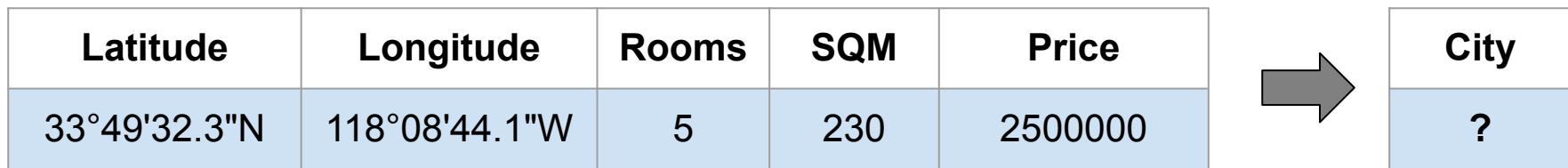
City	Latitude	Longitude	Rooms	SQM	Price
Los Angeles	33°49'32.3"N	118°08'44.1"W	5	230	?

- Note: given a testing dataset, which is labeled with the actual price but unseen for the learning algorithm, it's easy to measure the **accuracy** or, conversely, the **error**



Supervised ML: an example (2)

- Suppose we want instead to predict if the house is in L.A. or Albuquerque, given all the other variables (including the price)
- The kind of task is a **classification** problem:
 - The **output variable we want to predict is a class value**



- When the classes are 2, it's a **binary** classification problem: output can be either 0 or 1. Example: is the house from L.A.?
- When the classes are more than two, it's a **multi-class** classification problem: **the output is an element of a finite set.** Example: from which city is the house, among these 20 cities?

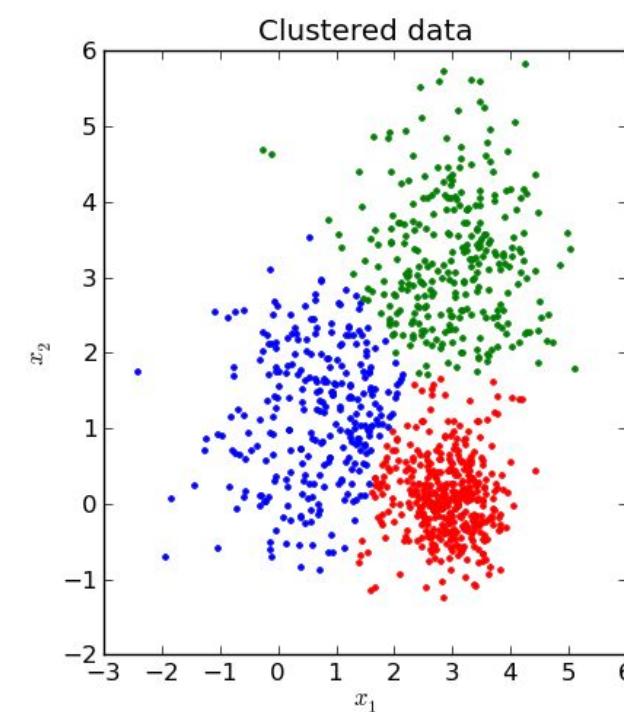
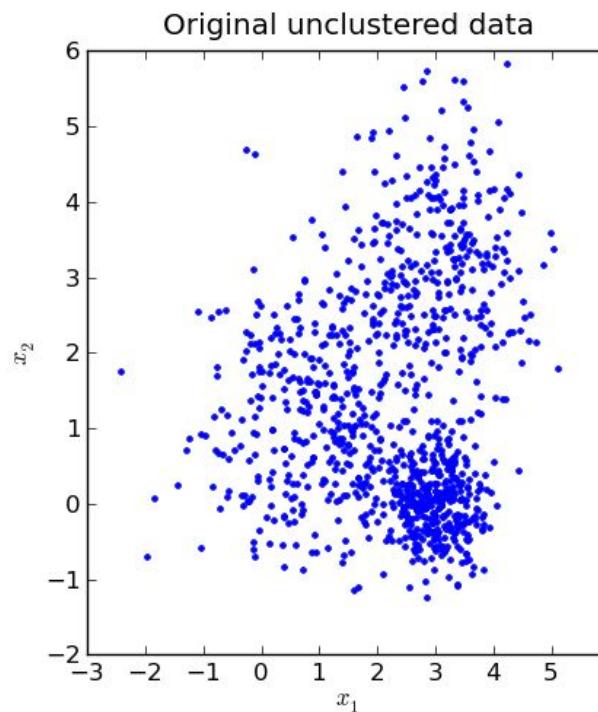


Unsupervised techniques

- **Unsupervised** techniques: there is no given value for the output variable, or there is **no output variable** at all
- Sometimes **there is no goal set** a-priori and we just want to find out regularities and relations inside the data
- Most common tasks (we will see the first two):
 - **Clustering**: split the examples into different groups (clusters). The examples in the same group should be similar, while examples in different groups should be dissimilar
 - **Association rules**: discover interesting relations between variables. E.g.,
 $\{onions, potatoes\} \Rightarrow \{burger\}$ in supermarket sales
 - **Anomaly detection**: find the “outliers” in a set of examples. E.g., from a log of bank transactions find the suspect ones
 - **Generative models**: given many examples generate a new, “synthetic” instance with similar features. E.g., melodies generation, poetry/book automatic writing
 - **Feature extraction**: reduces the number of variables by generating new, characterizing features

Unsupervised ML: an example

- **Clustering** is the most common task among the unsupervised tasks
- In this example our observations have two variables: x_1 and x_2
- There is neither *ground truth* nor labels on the data. The output variable is “*the class the data point belongs to*” but it’s not given in the training



- Note: is this a good clustering? It's not easy to measure accuracy: without a *ground truth* the correctness is “in the eye of the beholder”



Interdisciplinarity

- One of the reasons of data mining's and machine learning's success are the virtually unlimited fields of **applications**:
 - Machine translation
 - Medical diagnosis
 - Recommendation systems (e.g. Amazon suggesting books)
 - Speech recognition
 - Information Retrieval (learning to rank)
 - Computer vision (e.g. self driving cars)
 - Cyber security (intrusion detection systems)
 - Trading
 - Weather forecast
 - ...



Data pre-processing



Data pre-processing

- Regardless of the specific goal, the pre-processing step is common to all data mining projects
- Why pre-processing? Real world data is generally:
 - **Incomplete**: lacking attribute values, lacking certain attributes of interest, or containing only aggregate data
 - E.g., occupation = “”
 - **Noisy**: containing errors or outliers
 - E.g., salary = -100
 - **Inconsistent**: containing discrepancies in code or names
 - E.g., Age = 30, Birthday = “03/07/2001”
 - E.g., Sex = Male, Pregnant = Yes
- The phrase “***garbage in, garbage out***” is particularly applicable in data mining



Pre-processing steps

- Main tasks of data pre-processing are:
 - **Data cleaning**
 - Fill in missing values, smooth noisy data, identify or remove outliers, resolve inconsistencies
 - **Data integration**
 - Integration of multiple databases, data cubes, or files
 - **Data transformation**
 - Machine learning algorithms work on the vector space: text and categories must be “encoded” into vectors of real numbers (e.g., bag of words, one-hot encoding)
 - Very large and very small values must be normalized, otherwise the optimization algorithms under machine learning models will not converge in reasonable time



Data cleaning

- We have seen its importance in **data warehousing**
 - It is said that data cleaning is the number one problem in data warehousing
- Similar issues are also faced in **data mining**
- Common data cleaning **tasks** in data mining are:
 - Fill in **missing** values
 - Identify **outliers** and smooth out noisy data
 - Correct **inconsistent** data
 - Resolve **redundancy** caused by data integration



Why is data dirty?

- Incomplete/**missing data** may come from:
 - “Not applicable” data value when collected
 - Different considerations between the time when the data was collected and when it is analyzed
 - Human/hardware/software problems
- **Noisy data** (incorrect values) may come from:
 - Faulty data collection instruments
 - Human or computer error at data entry
 - Errors in data transmission
- **Inconsistent data** may come from
 - Different data sources
(e.g., to establish the position of a geographic location on a map, common map projections in current use include the Universal Transverse Mercator, the Military Grid Reference System, the United States National Grid, the Global Area Reference System and the World Geographic Reference System)
 - Functional dependency violation (e.g., modify some linked data)
- **Duplicate records** also need data cleaning



Missing data

- Data is not always available
 - e.g., many tuples have no recorded value for several attributes, such as customer income in sales data
- Missing data may be due to
 - Equipment malfunction
 - Inconsistent with other recorded data and thus deleted
 - Data not entered due to misunderstanding
 - Certain data may not be considered important at the time of entry
 - Not register history changes of the data
- Missing data may need to be **inferred**



How to handle missing data?

- **Ignore** the tuple: usually done when class label is missing (assuming the tasks in classification) not effective when the percentage of missing values per attribute varies considerably
- Fill in the missing value **manually**: tedious, but could be even infeasible if the value is totally unknown
- Fill in it **automatically** with
 - A **global constant**: e.g., “unknown”, a new class?
 - The attribute **mean** (e.g., replace the missing house price with the average price of all houses)
 - The attribute **mean** for all samples belonging to the **same class** (e.g. replace the missing L.A.’s house price with the average price of the houses in L.A.)
 - The **most probable value**: inference-based such as regression (needs machine learning already)



Noisy data

- Noise: random error or variance in a measured variable
- Incorrect attribute values may due to
 - Faulty data collection instruments
 - Data entry problems
 - Data transmission problems
 - Technology limitation
 - Inconsistency in naming convention
- Other data problems which requires data cleaning
 - Duplicate records
 - Incomplete data
 - Inconsistent data

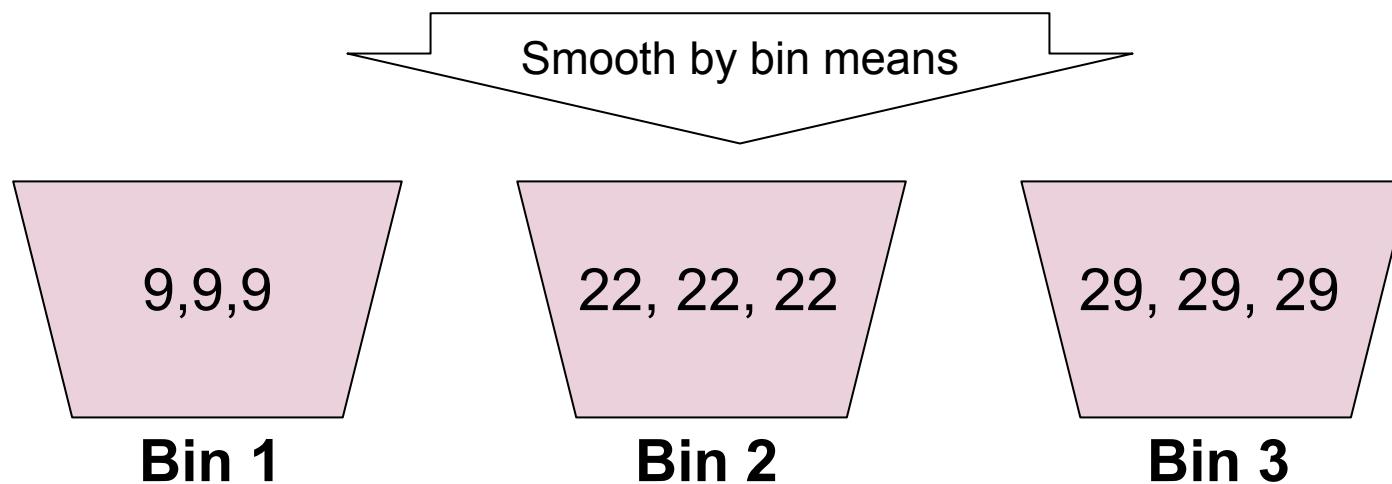
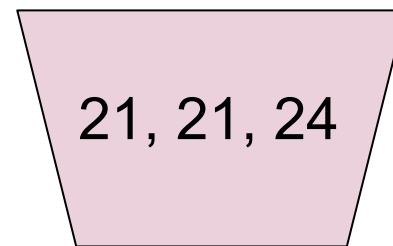


How to handle noisy data

- **Binning:** smooth a sorted data value by consulting its “neighborhood”, that is, the values around it
 - First sort data and partition into (equal-frequency) bins
 - Then we can smooth by bin means
- **Regression:**
 - Smooth by fitting the data into regression functions
- **Clustering:**
 - Detect and remove outliers
 - Combined computer and human inspection
 - Detect suspicious values and check by human (e.g., deal with possible outliers)

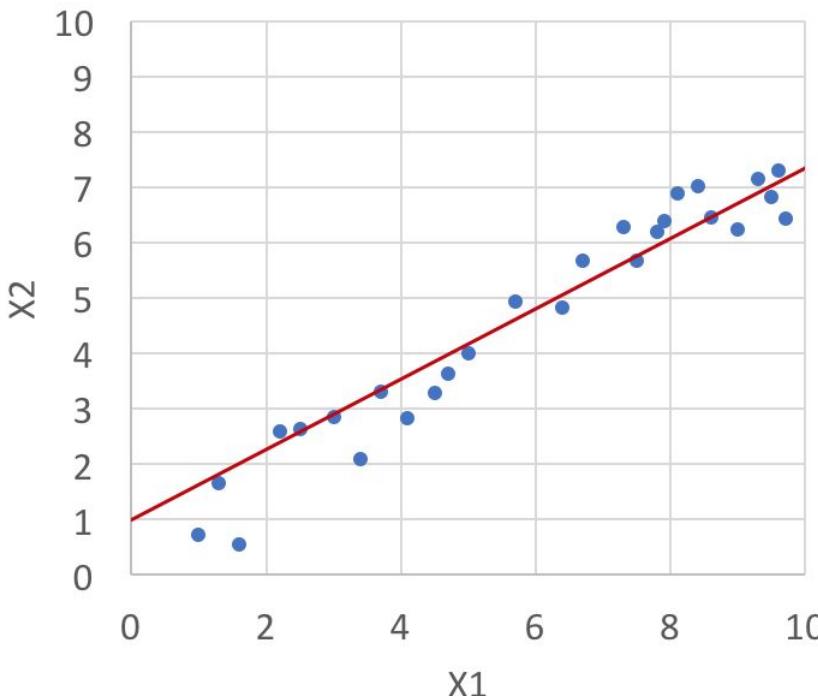
Denoising: Smoothing by bins

- Once the bins are populated, the values are substituted with aggregate values. A typical replacement is the means of the bin
- Because binning methods consider only the values in the same bin, they perform *local smoothing*

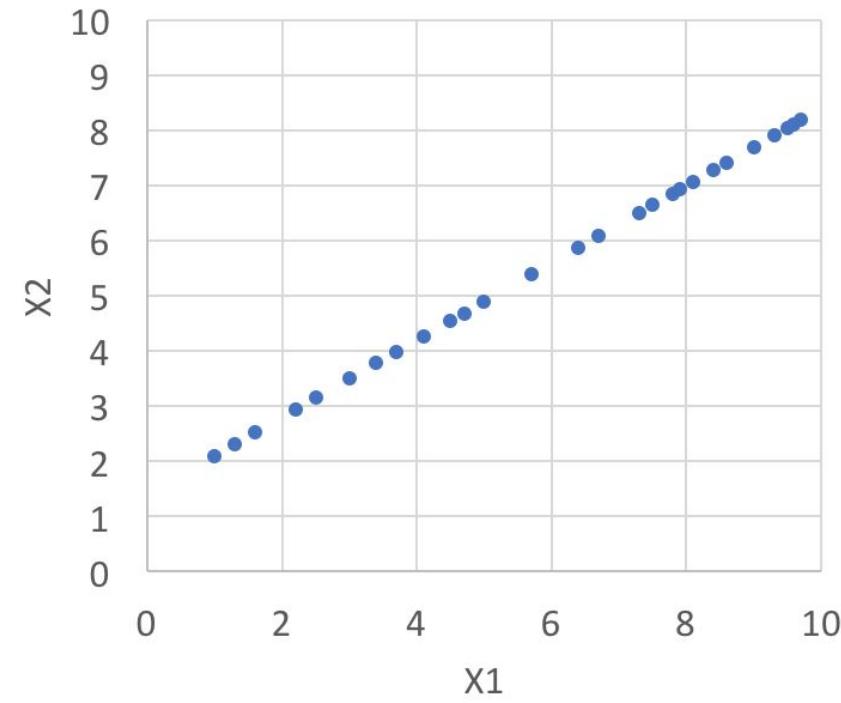


Denoising data: Regression

- Regression: linear regression involves finding the “best” **line** to fit two attributes (or variables) so that one attribute can be used to predict the other
- Regression could be the **final goal** of a data mining project, but it can also be used **a-priori** in the project in order to **smooth** the data



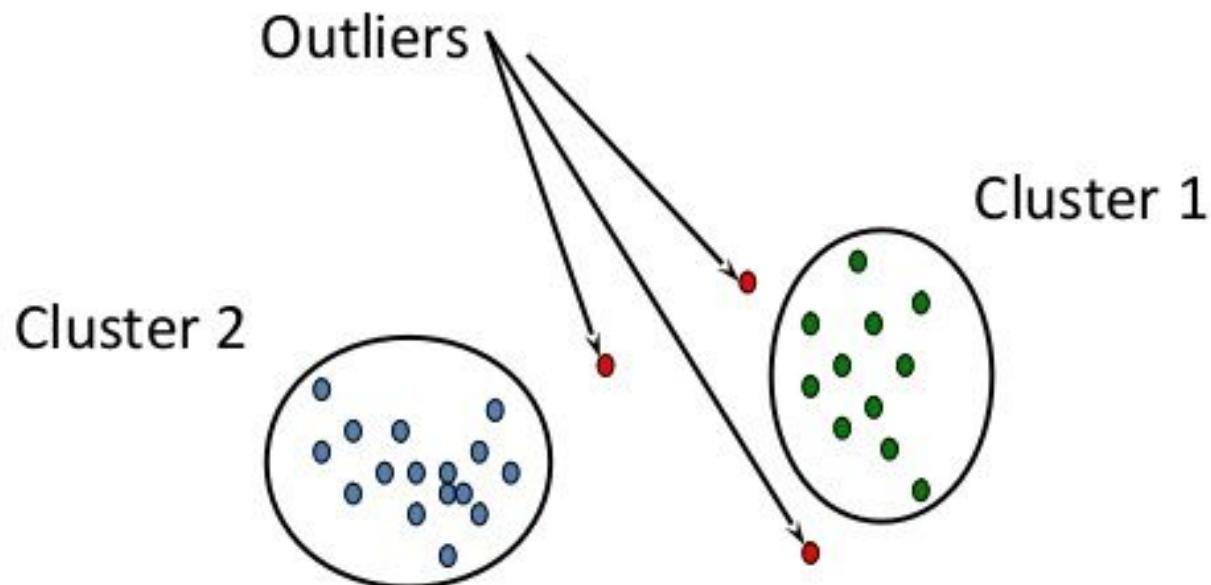
1) Regression



2) Smoothing, moving data points to the regression line

Denoising data: Clustering

- Outliers may be detected by clustering: similar values are organized into groups (clusters)
- Intuitively, **values that fall outside** of the clusters may be considered as outliers (anomalies)
- As for the linear regression method, clustering can be also the ultimate task of a data mining project, but could be applied in the preprocessing task to improve the quality of data





A note on noisy data

- When thinking of data noise we must keep in mind the ultimate tasks and their differences:
 - **Data warehousing:** next step is loading in the DW. Handling noisy data in the pre-processing phase is imperative because the decision maker will have no means of denoising data
 - **Data mining:** next step is machine learning. Machine learning models (even the most simple such as Logistic Regressions) have means of denoising data through “regularization” techniques
- **Practical suggestion:** build a first model without denoising, then reiterate training steps with pre-processing steps



Data integration

- Data mining often requires **data integration**: the merging of data from multiple data stores
- The semantic heterogeneity and structure of data pose great challenges in data integration
- How can we match schema and objects from different sources?
 - E.g., in the table A, the customer identification number is named A.cust_id, while in the table B, the customer identification number is called B.cust_numb
- This problem is called the ***Entity Identification Problem***



Entity Identification Problem

- *Schema integration* and *object matching* can be tricky. Both tasks are affected by the **Entity Identification Problem**
 - For example, how can a data analyst or a computer be sure that *customer_id* on one database and *cust_number* in another refer to the same attribute?

Schema A

customer_id	birth	city
109238	07/12/87	Rome
113125	23/08/89	London
159483	28/11/90	Paris
198828	22/12/92	Bristol

Schema B

order numb	cust numb	cost
4982812389	113125	34.50
4982812390	113125	110.02
4982812391	151514	98.49
4982812392	129827	32.40



Entity Identification solution 1/2

- We can use the attribute **metadata**, for example:
 - **Name of the attribute**: we can use distances on string such as the edit distance to measure how much two names are similar
 - **Data type**: are the attributes both strings? Both dates?
 - **Range of values**: if one attribute contains values in the thousands and the other values between -1 and 1, we are probably looking at two different things

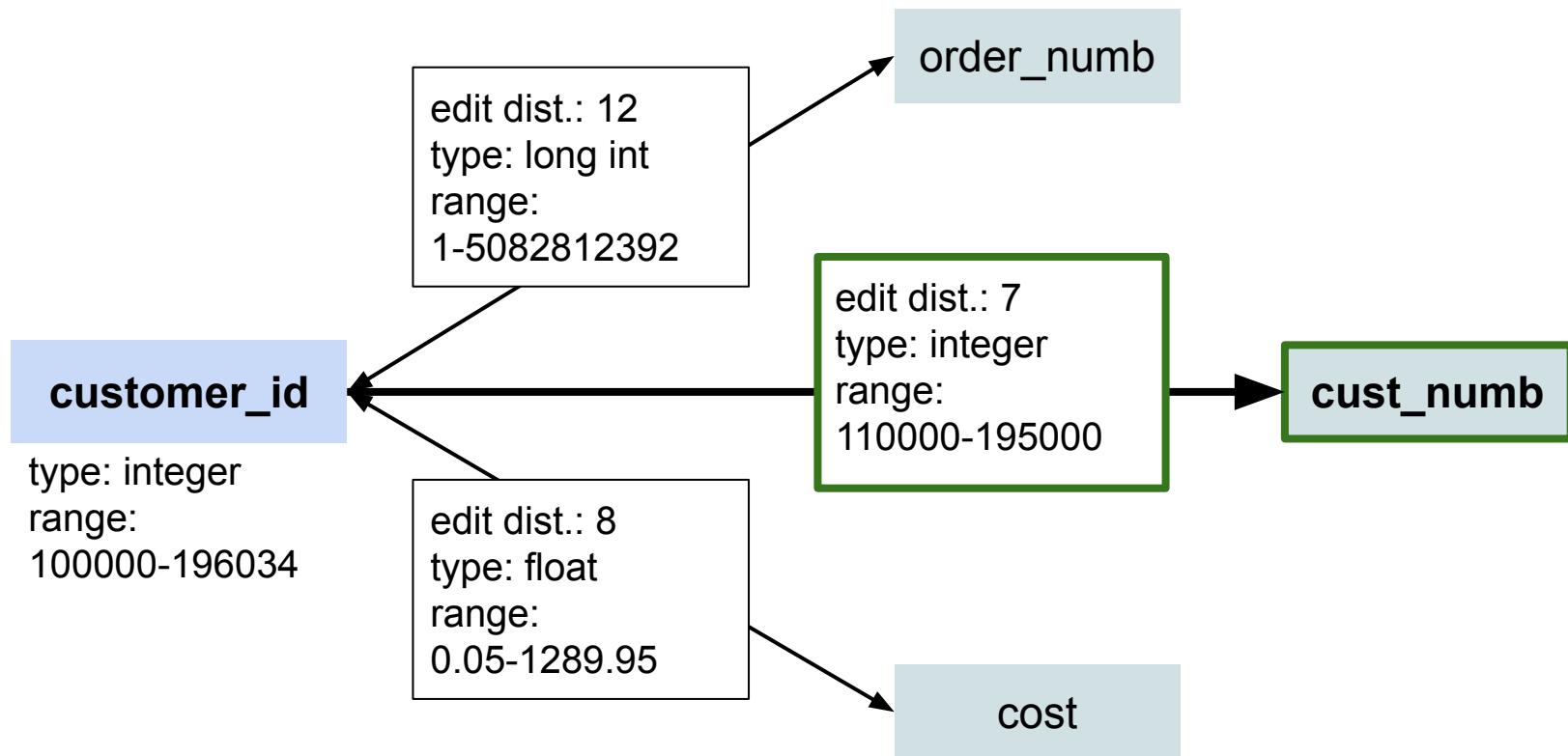
Schema A

customer_id	birth	city
109238	07/12/87	Rome
113125	23/08/89	London
159483	28/11/90	Paris
198828	22/12/92	Bristol

Schema B

order numb	cust numb	cost
4982812389	113125	34.50
4982812390	113125	110.02
4982812391	151514	98.49
4982812392	129827	32.40

Entity Identification solution 2/2

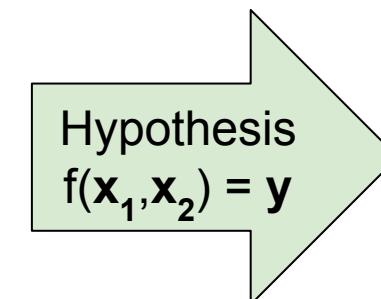


- We select **cust_numb** because it has the **least edit distance**, **same data type** and the **most similar values range**
- Keep in mind that it is **not possible** to determine fully automatically the correspondences (with 100% accuracy)

Data transformation

- Recall: the hypothesis is the function modeled by machine learning algorithms, from the input variables to the output variable (e.g., a regression line that fits 2D data points)
- This function is a mathematical function, which takes **numerical values in input**, and output one or more numerical values

SquareMeters x_1	Bedrooms x_2
120	2
200	3
80	2
160	3
45	1
140	1



Price y
380,000
550,000
230,000
500,000
135,000
410,000



Why we need transformation

- **Encoding** problem: what if some variables of our data are **not numerical**?
 - It could be a **categorical** values such as the city
 - It could be a **boolean** yes/no value such as “hasGarage”
 - It could be a **free text** value such as “*Very nice cozy flat near the Centrum shopping mall, friendly neighborhood*”
- **Scaling** problem: very big numerical values are not good in ML
 - Machine learning models are based on **numerical optimization** techniques which minimizes the error between the hypothesis and the training data
 - Optimization algorithms **converge faster** with normalized values (e.g., between 0 and 1 or between -1 and 1)

Encoding solution for text

- We have seen in Information Retrieval how **free text** can be represented as a **vector of numerical variables**:
 - Each variable of the vector is for a specific word
 - The value of a variable x_i is 1 if the *i-th word* of the set of all words is present in the text, 0 otherwise
 - This representation is called “**bag-of-words**” (BOW)
 - This is a “**sparse**” representation: the resulting vector for a text has most of the variables set to 0, while only the words present in the text are set to 1

“Very nice flat with wonderful view”

...	<i>apartment</i>	<i>close</i>	<i>cozy</i>	<i>flat</i>	<i>floors</i>	<i>friendly</i>	<i>great</i>	<i>mall</i>	<i>view</i>	...
...	0	0	0	1	0	0	0	0	1	...

- This encoding process that goes from free text to a real-valued vector is called **vectorization**



Encoding solution for categories

- Encoding **categorical** variables is similar to bag-of-word encoding, with some differences:
 - Only **one category** at a time is true (e.g., a house cannot be both in N.Y. and in Miami, or cannot have a garage while not having a garage!)
 - No need to tokenize (split strings into words) or removing stop-words: each unique value will have its own variable (e.g., one variable for each city)

City: New York

...	<i>Albuquerque</i>	<i>Boston</i>	<i>New York</i>	<i>Seattle</i>	<i>Miami</i>	<i>Las Vegas</i>	<i>Chicago</i>	...
...	0	0	1	0	0	0	0	...

- This encoding process is called **one hot encoding**, because **only one** of the generated variables is active (“hot”) while all the other are set to 0



One Hot Encoding: an example

Human-Readable

Pet
Cat
Dog
Turtle
Fish
Cat

Machine-Readable

Cat	Dog	Turtle	Fish
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
1	0	0	0

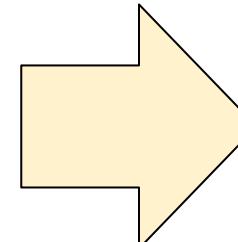


Scaling - normalization

- The reasons why scaling is important in a data mining project will become more clear when we will see how machine learning models tune the hypothesis function to “fit” the data
- For now, let’s just say it **improves performance and accuracy** of most machine learning models
- Most common scaling range is [0,1], it can be obtained with linear scaling:

$$scale(value) = \frac{value - min(values)}{max(values) - min(values)}$$

SQM	Price
130	420000
60	80000
180	500000
140	220000



SQM	Price
0.583	0.810
0.000	0.000
1.000	1.000
0.666	0.333

Summary

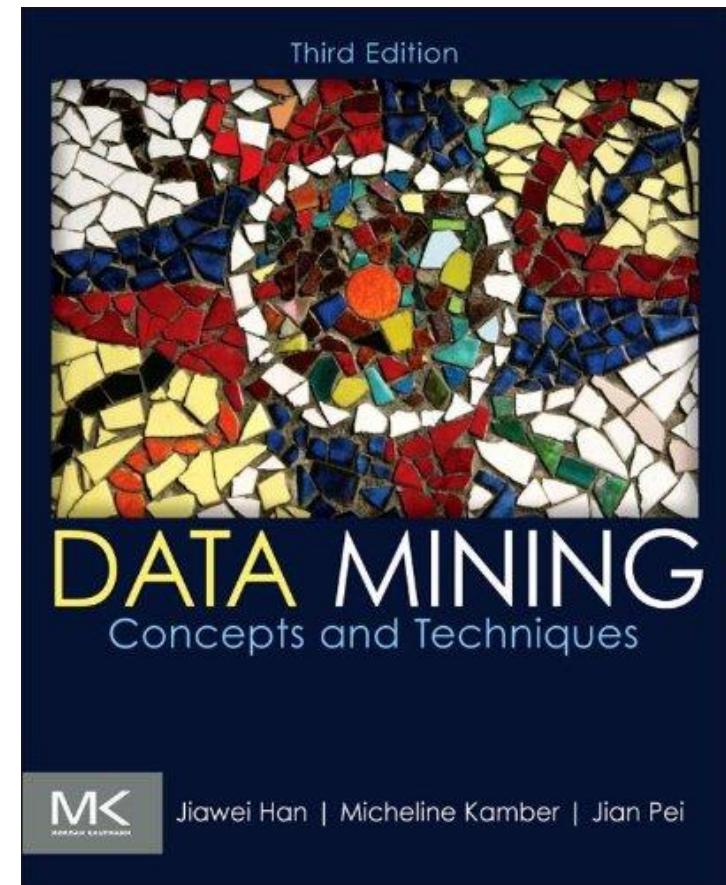
- We've introduced the Data Mining process: it allows to extract new knowledge and patterns from large amount of data
- It consists of several steps, we have covered a preliminary step, which is the pre-processing phase:
 - **Cleaning data:** by removing or replacing erroneous or inconsistent values
 - **Denoising data:** by smoothing values toward central measures and removing outliers
 - **Integrating data:** by matching entities and attributes across different data sources
 - **Transforming data:** by encoding categorical and text data into numerical vectors, and scaling values into fixed ranges
- Data pre-processed, in the form of a **normalized matrix “X”**, of **m** examples and **n** variables, will be the input of machine learning algorithms



References

Data Mining Concepts and Techniques

Authors: Jiawei Han, Micheline Kamber, Jian Pei





Big Data and Data Mining

Supervised Learning

Flavio Bertini

flavio.bertini@unipr.it

Regression

- The regression problem asks to predict a numerical variable's value given the values of other variables
- Easiest example is with two variables x and y :
 - x is the variable in input
 - y is the variable we want to predict

x	y
1	3
2	5
3	7
4	9
5	11

Training data: x is the input data, y is the label data

Learning
Learning task: **what's the mapping from x to y ?**

Testing
Try to “learn” from the training data an hypothesis function h
What is $h(6)$? That is, what's y when x is 6?

x	y
6	?

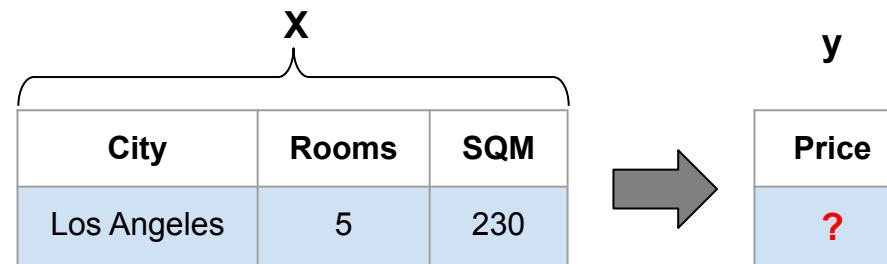
Regression example 1/2

- We have the data from 10 thousand houses in the U.S.

$m=10000$

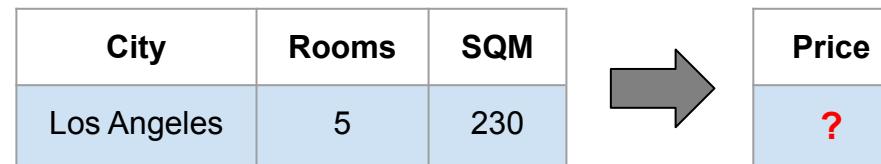
City	Rooms	SQM	Price
Los Angeles	3	130	420000
Los Angeles	2	60	380000
...
Albuquerque	2	140	220000
Albuquerque	3	150	250000

- Goal:** learn a function (model) that can infer the *Price* given all the other variables, for houses not in the 10 thousand dataset:

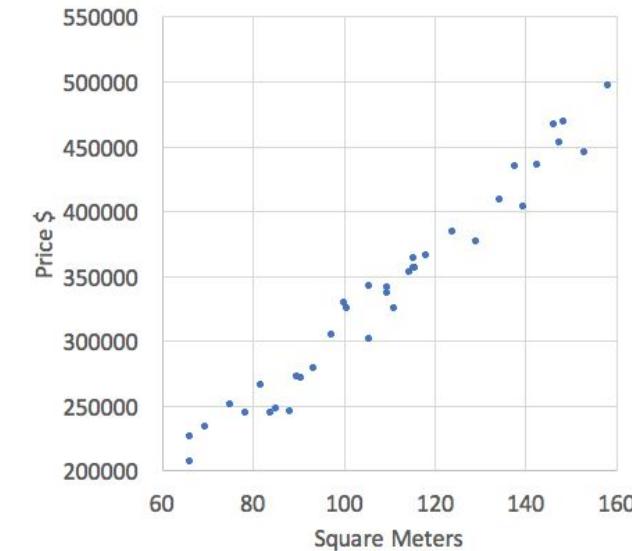


Regression example 2/2

- The houses' prices example is an example of **regression** task:
 - Among the variables there is the output variable (Price)
 - The output variable is the *ground truth*, because it's the actual price of the house
 - The output variable we want to predict is a numerical value



- To visualize things better we will have examples with only two variables (one used as input one as output)
- All the process can be generalized for n variables





Linear regression

- We defined regression as the general task of predicting the real value of a variable using the other variables as input
- The model we want to learn is a function f from n real values x_i to one real value y :

$$y = f(x_1, \dots, x_n)$$

- The learned model is called *hypothesis* function h :

$$y = h(x_1, \dots, x_n)$$

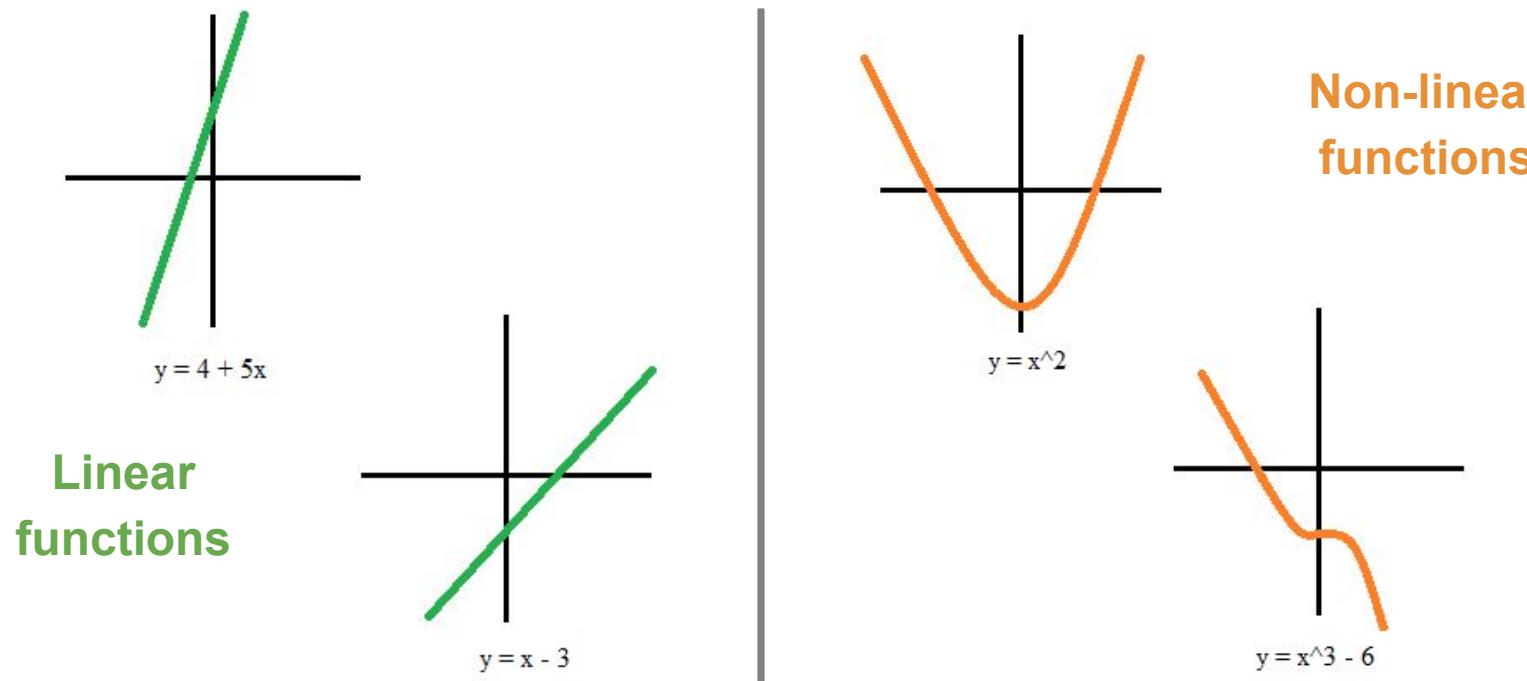
- When h is a **linear function** on the input variables x_i , the regression task is called **linear regression**

Linear function

- A linear function is a function of this form:

$$y = f(x_1, \dots, x_n) = w_1 x_1 + \dots + w_n x_n + b$$

- When $n = 1$ (only one input variable), the plotted function has the **shape of a straight line**. When $n = 2$ it takes the shape of a straight plane etc.
- Straight lines and straight planes are examples of linear functions





Linear regression: parameters

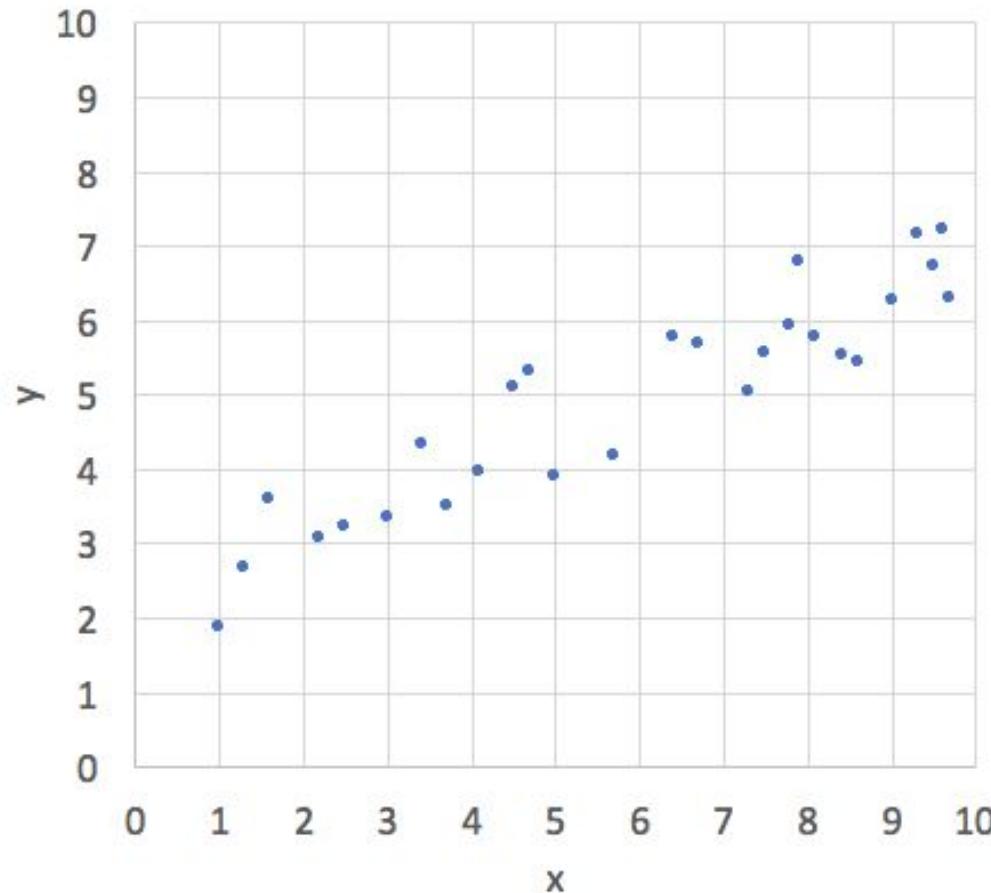
- Recall, a linear function is a function of this form:

$$y = f(x_1, \dots, x_n) = w_1 x_1 + \dots + w_n x_n + b$$

- This means that the function that models our data is already defined!
- What's missing? **What are the parameters we need to “learn”?**
 - The x_i variables are the ones **given in input**, so we already have them (e.g., the square meters of the house)
 - The w_i **coefficients are not known!**
 - In a straight line, the coefficient is the **slope** of the linear function!
 - In ML they are also called **weights**: assuming that you have scaled your variables (e.g. between 0 and 1) they tell you **how much a variable contributes** to the final result (the y)
 - The b parameter is **also not known!**
 - In a straight line, this is the **y-intercept** of the line: the point where the line intercept the y axis
 - In ML this is also called **bias term** or **bias weight**

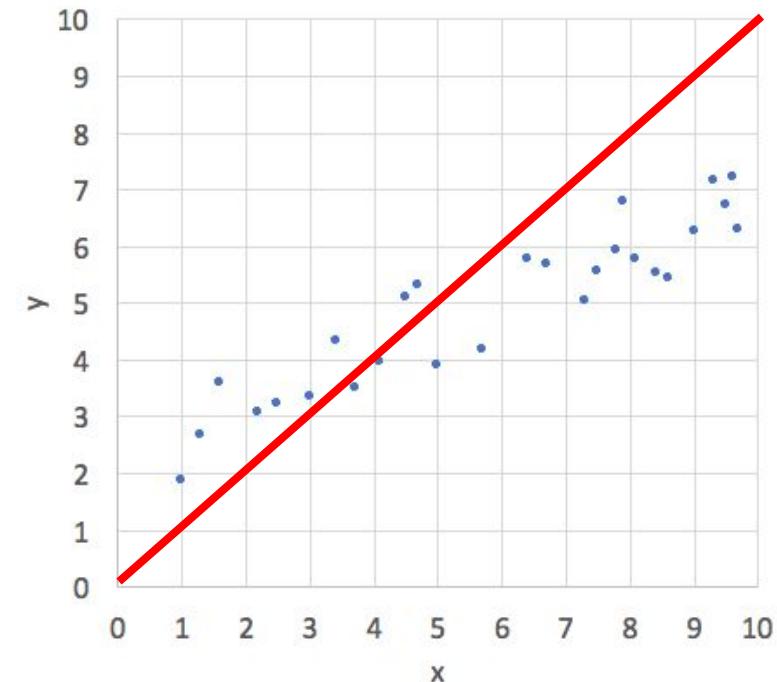
Example: data points

- Let's clarify these new notions with an example
- We have plotted some data points, each point is an observation with two variables: x and y



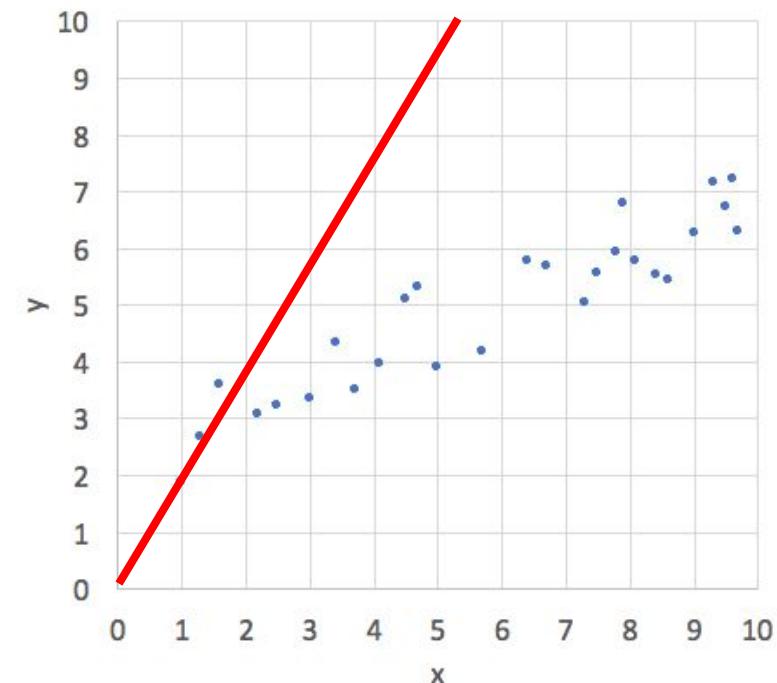
Example: slope 1/3

- Let's simulate a learning process: we need to learn the slope on the line
 - To keep things simple, we focus on the angle only, setting the y-intercept to 0
1. **Let's try with a slope $w = 1$:** this means the line will have a 45° slope
 2. Having a y-intercept (b parameters) set to 0, the hypothesis function is:
$$y = h(x) = wx = x$$
 3. By plotting the line on the figure we can visually evaluate how much this hypothesis match the actual data
 - Is this hypothesis good enough?
 - We will see later how to **measure precisely the distance between the actual data and the hypothesis**
 - But for sure can do better!



Example: slope 2/3

- Let's simulate a learning process: we need to learn the slope on the line
 - To keep things simple, we focus on the angle only, setting the y-intercept to 0
1. Now let's try with a slope $w = 2$!
 2. Having a y-intercept (b parameters) set to 0, the hypothesis function is:
$$y = h(x) = 2x$$
 3. By plotting the line on the figure we can visually evaluate how much this hypothesis match the actual data
 - Is this hypothesis better?
 - The hypothesis **seems worse than before**



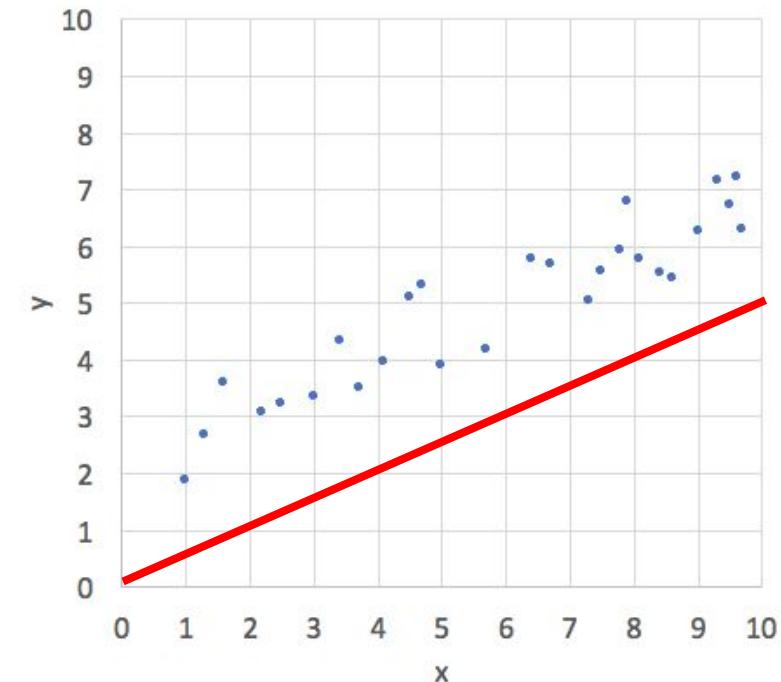
Example: slope 3/3

- Let's simulate a learning process: we need to learn the slope on the line
- To keep things simple, we focus on the angle only, setting the y-intercept to 0

1. Now let's try with a slope $w = 0.5$!
2. Having a y-intercept (b parameters) set to 0, the hypothesis function is:

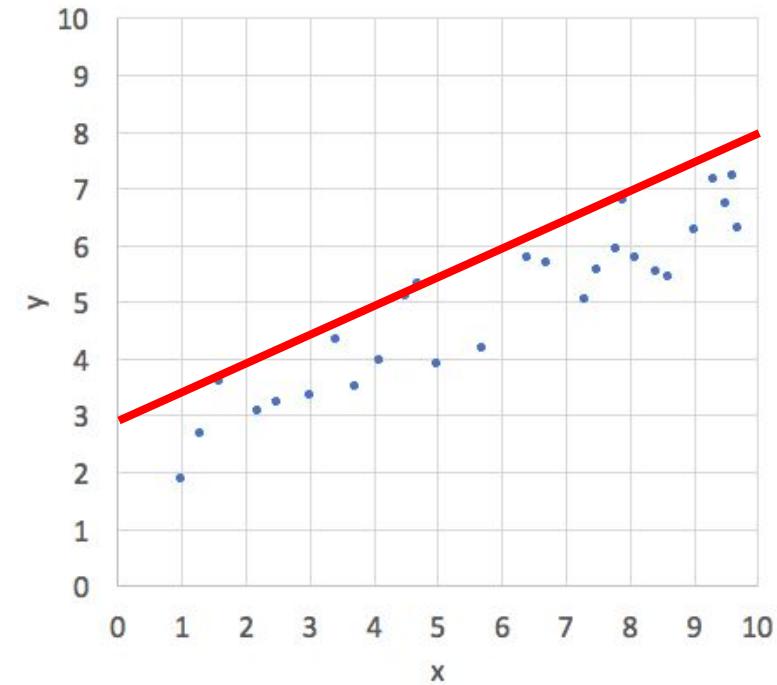
$$y = h(x) = \frac{1}{2}x$$

3. By plotting the line on the figure we can visually evaluate how much this hypothesis match the actual data
 - Is this hypothesis better?
 - The hypothesis seems **quite accurate, at least for the slope**
 - We should now learn **what's the best value for the bias term b**



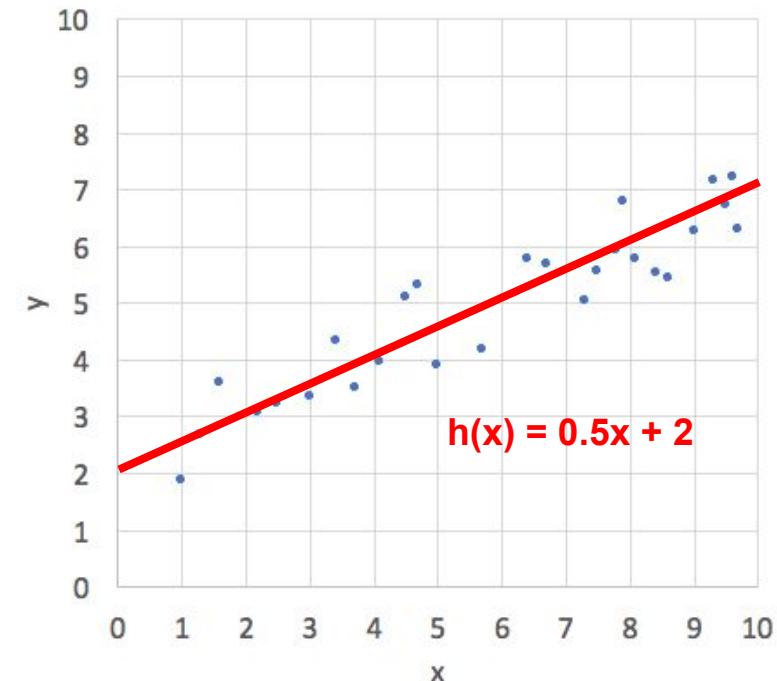
Example: bias term 1/2

- We have now a **good slope but a clearly wrong bias term** (the y intercept parameter)
 - Let's try again some reasonable value
1. **Now let's try with a bias $b = 3$!**
 2. Having already set the weight to 0.5, the hypothesis function is:
$$y = h(x) = \frac{1}{2}x + 3$$
 3. By plotting the line on the figure we can visually evaluate how much this hypothesis match the actual data
 - Are the points approximately around the line?
 - **It seems slightly above. We should try a lower value for b**



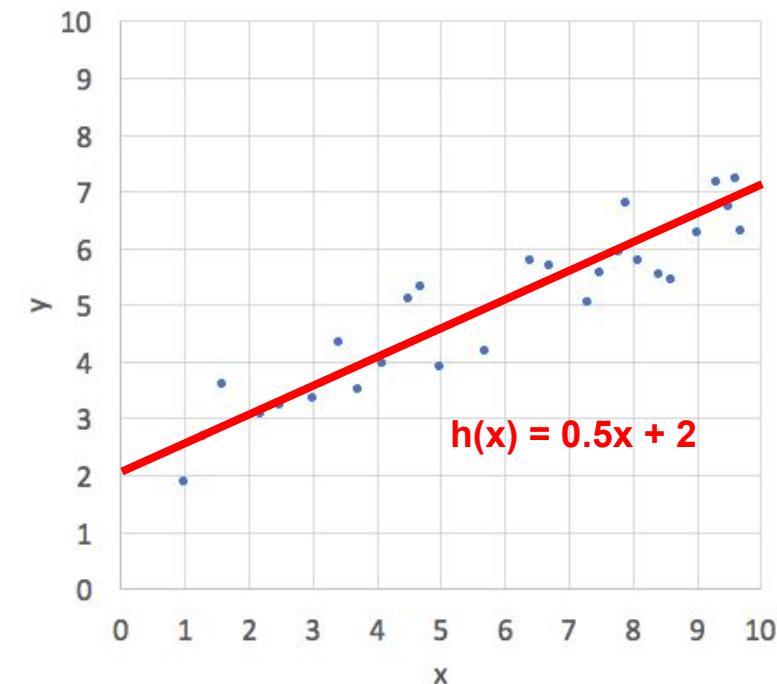
Example: bias term 2/2

- We have now a **good slope but a clearly wrong bias term** (the y intercept parameter)
 - Let's try again some reasonable value
1. **Now let's try with a bias $b = 2$!**
 2. Having already set the weight to 0.5, the hypothesis function is:
$$y = h(x) = \frac{1}{2}x + 3$$
 3. By plotting the line on the figure we can visually evaluate how much this hypothesis match the actual data
- **This hypothesis looks quite right!**



Example: observations

- The machine learning algorithm we have simulated is rather naive:
 - It has no precise way to measure how close we are to the optimal hypothesis
 - It makes almost random guesses to generate new hypothesis
- This means that a good machine learning model should have:
 - **A way to measure how good (or how bad) an hypothesis is**
 - **A smart algorithm that tunes the weights** until the measure of badness is very low (or conversely the measure of goodness is very high)



Loss function

- In ML, the measure used to evaluate the hypothesis is called a **loss function**
 - Intuitively, a **loss** function measures how **bad** the model is with respect to the actual data
- The most common it's the mean squared error, that is the **average squared distance** between the training data and the hypothesis
 - Mean because we want to consider all the m data points in the training set, so we average all the distances
 - Squared to enforce the distance to be non-negative

$$\frac{1}{m} \sum_{i=1}^m (h(x_i) - y_i)^2$$

Arithmetic mean

Difference between the y predicted using the hypothesis function and the actual y

We square the difference to enforce non-negativity



Optimization

- Now that we have a tool to measure the error of an hypothesis function, we want the error to be the smallest possible
- This means our **objective function** is to find the variables values (weights and bias) that **minimize** the error

$$\text{minimize} : \frac{1}{m} \sum_{i=1}^m (h(x_i) - y_i)^2$$

- Using the *argmin* notation and replacing $h(x)$ with the linear function:

$$\operatorname{argmin}_{w,b} \frac{1}{m} \sum_{i=1}^m ((wx_i + b) - y_i)^2$$

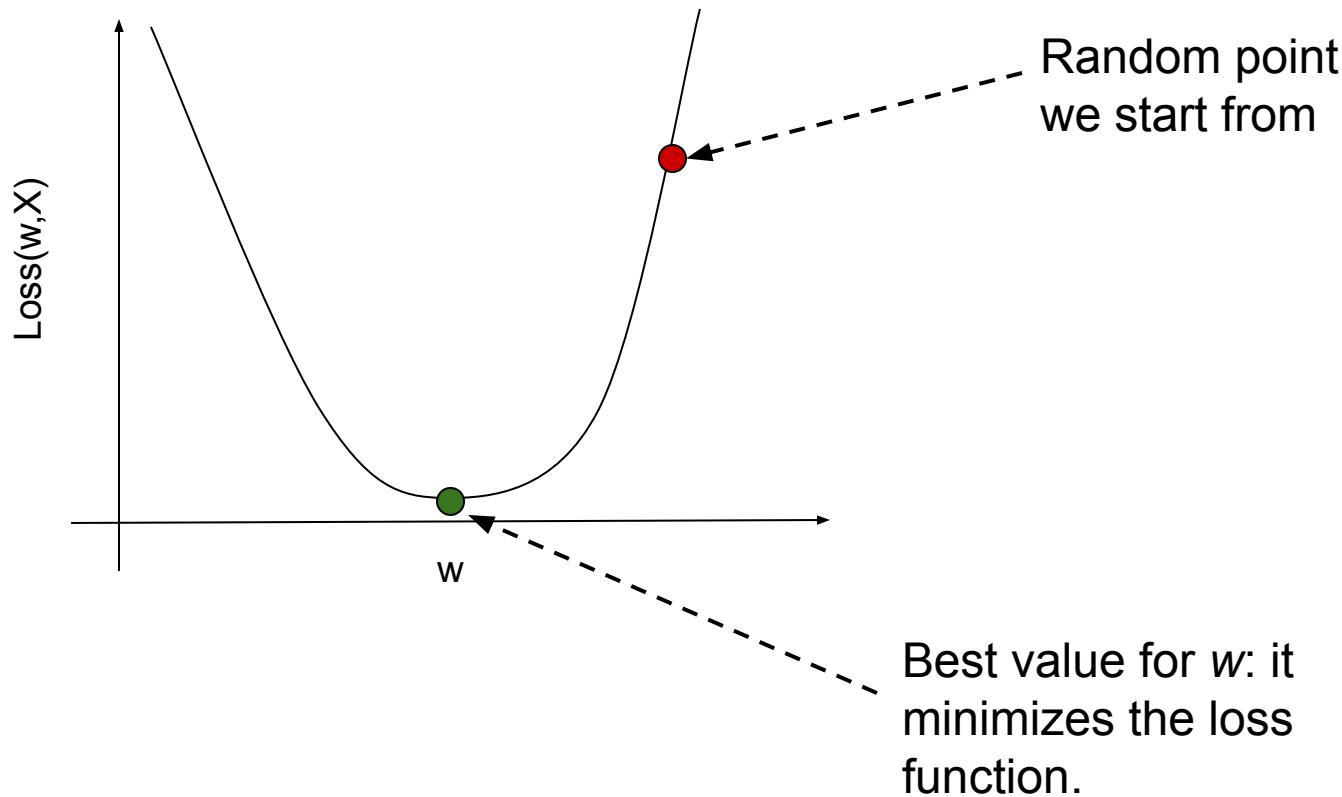


Optimization algorithms

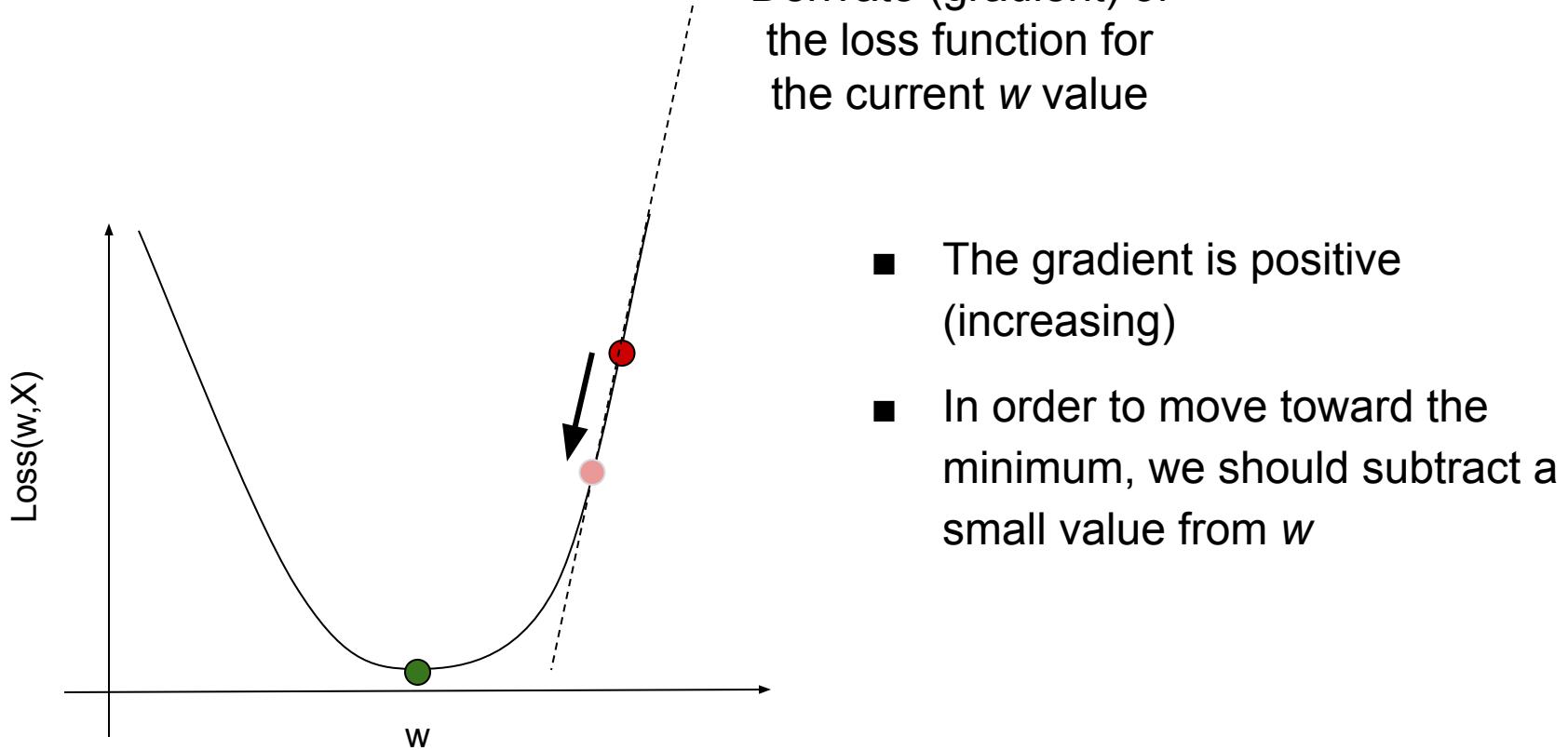
- Most popular and effective optimization algorithm in ML is **gradient descent**
- It **iterates** through 4 steps until the **loss is smaller than a tolerance value**:
 1. Compute the gradient (that is the derivative or slope) of the loss function
 2. The **sign of the gradient** tells us if the loss increase (positive) or decrease (negative) moving to the right. Recall that we want to reach the minimum:
 - a. If it increase, we should move the weight to the left (subtract a small value)
 - b. If it decrease, we should move to the weight to the right (add a small value)
 3. Update the hypothesis with the new weights
 4. Compute again the loss

Gradient descent example 1/6

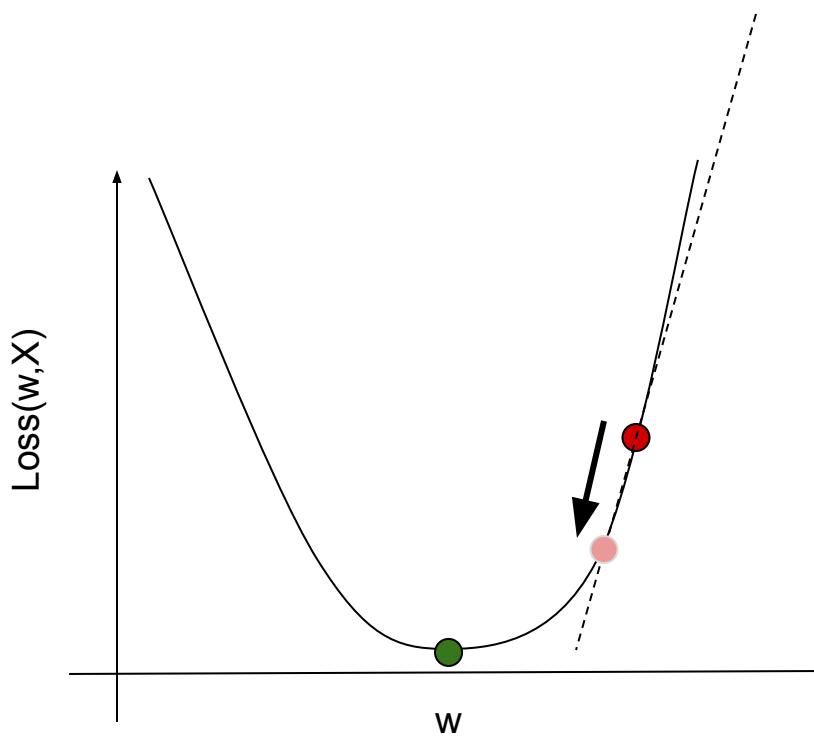
- For simplicity, we consider only one parameter w to be learned
- The loss function with respect to w it's a convex function, thus it has a global minimum



Gradient descent example 2/6



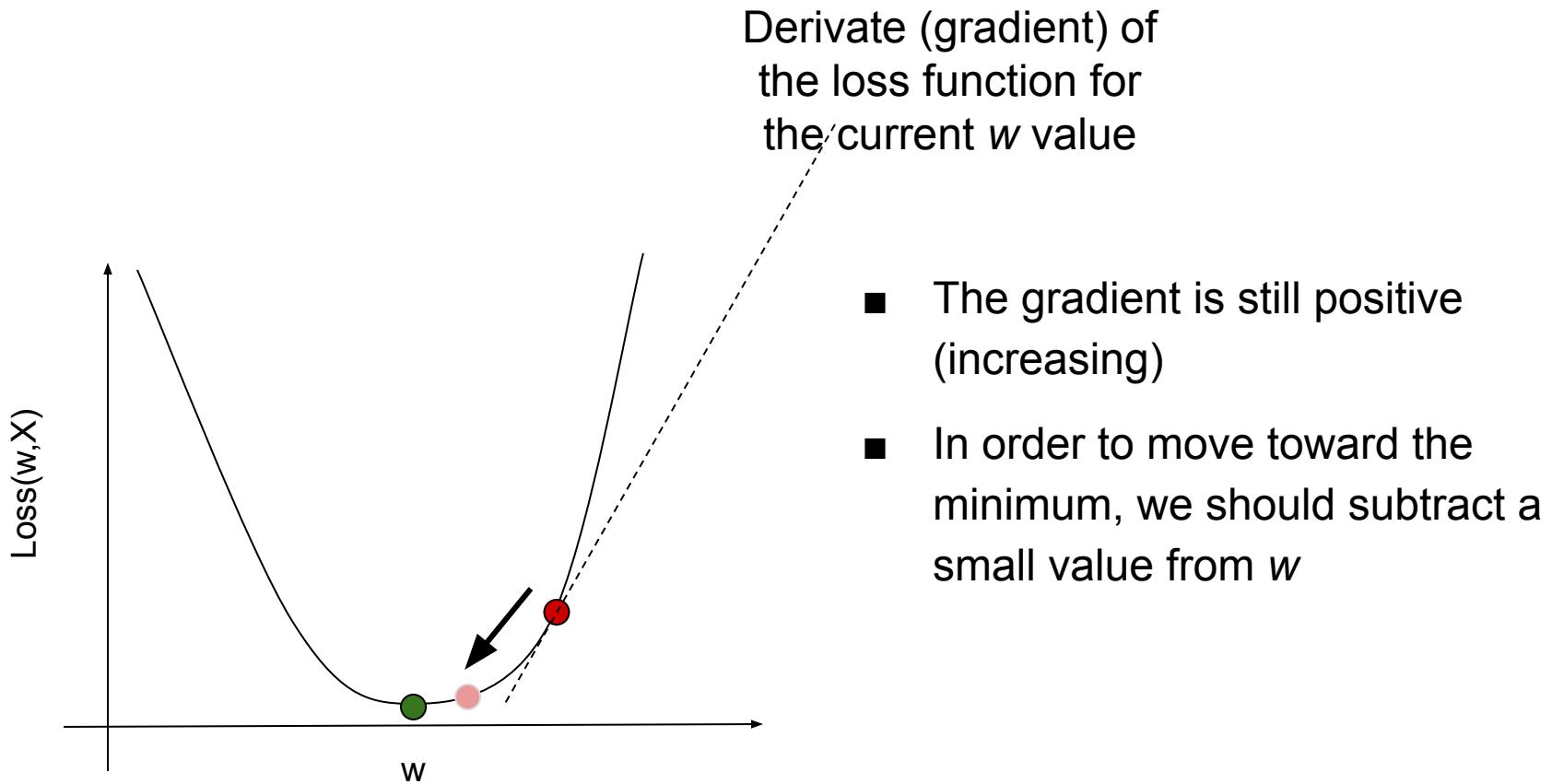
Gradient descent example 3/6



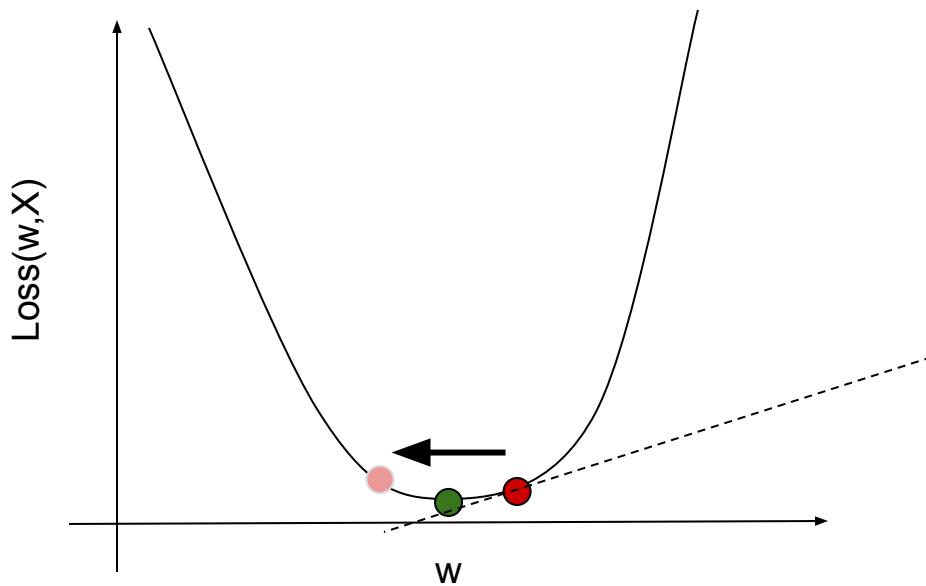
Derivate (gradient) of
the loss function for
the current w value

- The gradient is still positive (increasing)
- In order to move toward the minimum, we should subtract a small value from w

Gradient descent example 4/6

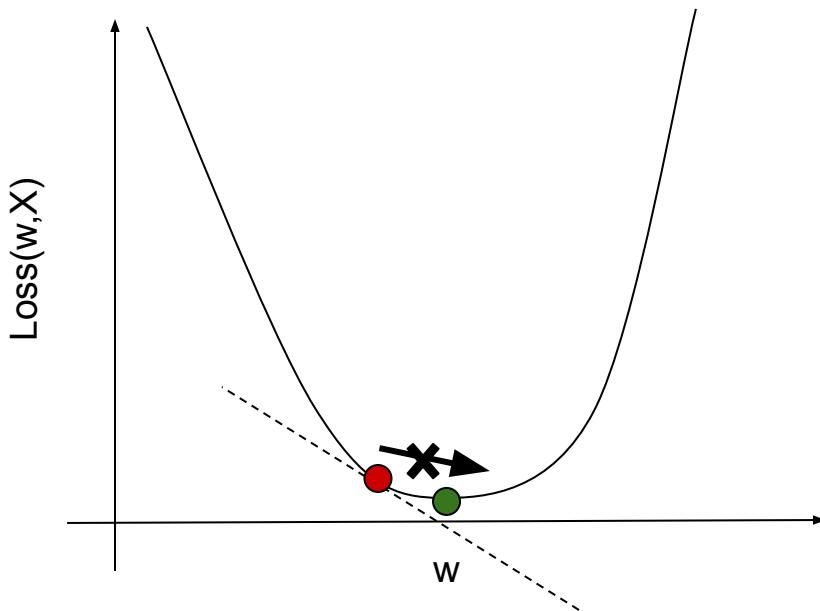


Gradient descent example 5/6



- The gradient is still positive (increasing)
- In order to move toward the minimum, we should subtract a small value from w

Gradient descent example 6/6



- The gradient **now is negative** (decreasing)
- In order to move toward the minimum, we should **add a small value to w**
- However, we could decide that the loss is **small enough** and **stop our descent iterations**



Testing set

- **Important:** the loss function is made to evaluate the hypothesis **on the training set!** It measures the distance between the training data and the hypothesis
- If the loss is 0, it means that the hypothesis is **perfectly fitting** the training data
- Yet, this hypothesis could be inaccurate **with unseen data:** data that is not in the training set
 - This phenomenon is called ***overfitting***: the model is extremely good (overfitted) on the training data but unable to generalize on new data
- In order to actually test our model, we must use a set of data that the learning algorithm has never seen, that is the ***testing set***



Train/test split

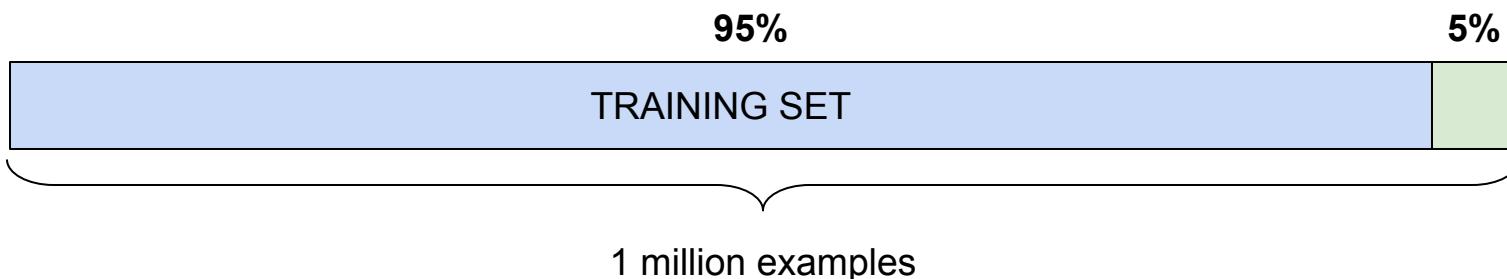
- Usually in a data mining process the data **is not already** splitted into training and testing set
- In splitting between training and testing set you should consider the following:
 - The more data you have in the training the better will perform the learned model
 - The more data you have in the testing the better will be the estimation of accuracy (e.g., if you guess just one example you will have 100% accuracy, but it's not a significant estimation)
- **IMPORTANT:** both training and testing set **must come from the same distribution.** For example:
 - You can't train on the houses' prices in L.A. and test on the houses in N.Y.
 - You can't train on pictures taken with the smartphone and test it on pictures taken from Google Images

Train/test split ratio

- The rule-of-thumb is to split **80/20** following the Pareto principle: 80 for training and 20 for testing



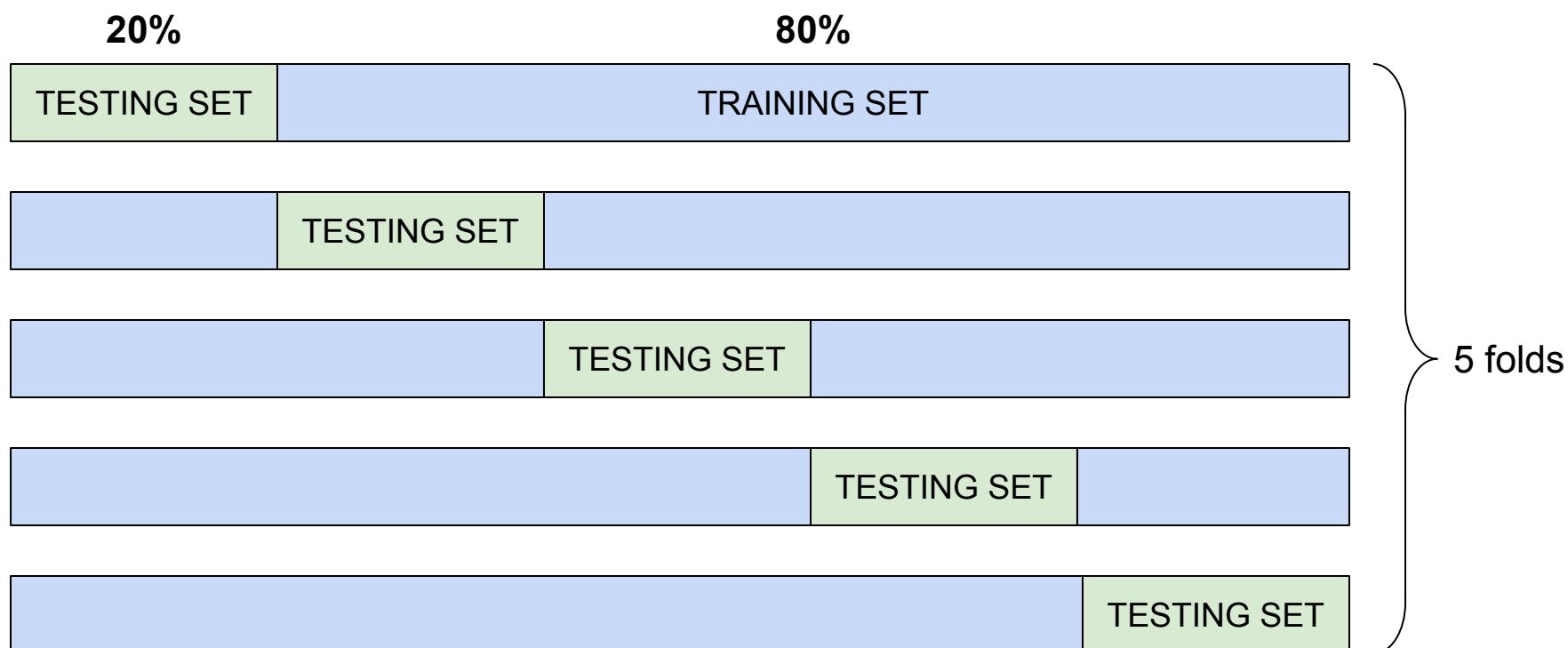
- However, if you have many examples, you may decrease the size of testing set to have a more accurate model with a still significant testing set



- What if we want to use all the data both for training and testing?

Cross-Validation (CV)

- The idea of cross-validation is to rotate over testing/training set split, so that each example will be used for training and for testing
- The fraction of the testing set will determine the number of rotations, also called “folds”. For example, using a testing set of 20%, that is $\frac{1}{5}$, we will have a 5-folds cross validation:





Cross-Validation: limit scenarios

- A **2-fold** cross-validation splits the data 50/50:
 - 1st fold: the first half is used for testing, the other for training
 - 2nd fold: the first half is used for training, the other for testing
- A **m-fold** cross-validation (m is the number of examples in the whole dataset) splits the data $1/m$:
 - 1st fold: first example is used for testing, all the rest ($m-1$ examples) is used for training
 - 2nd fold: second example is used for testing, the first example and all the other examples from the third are used for training
 - ...
 - m -th fold: first $m-1$ examples are used for training, last example is used for testing

m -fold a.k.a. **Leave-One-Out Cross-Validation**

- Note on CV: **the error of the model will be the average of all the errors**: in the m -fold CV it will be the average of m error values

What about non-linear data?

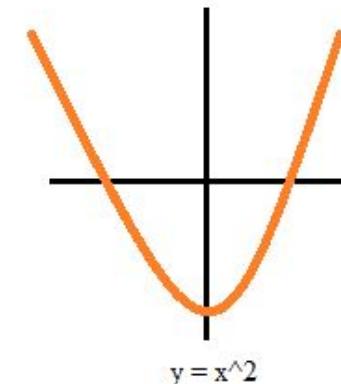
- So far we have seen a regression method that only works if the function from the input variables to the output variable is linear. E.g., with only one input variable x :

$$y = f(x) = wx + b$$

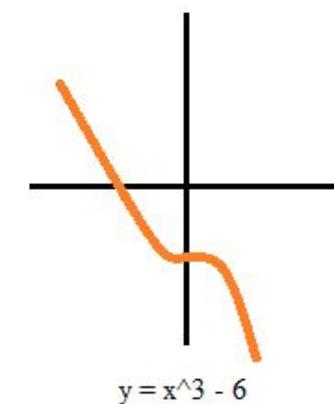
- What if the function is non-linear? The associated expression would be like:

$$y = f(x) = w_1x + w_2x^2 + w_3x^3 \dots$$

with a different weight w for each exponential



Non-linear functions



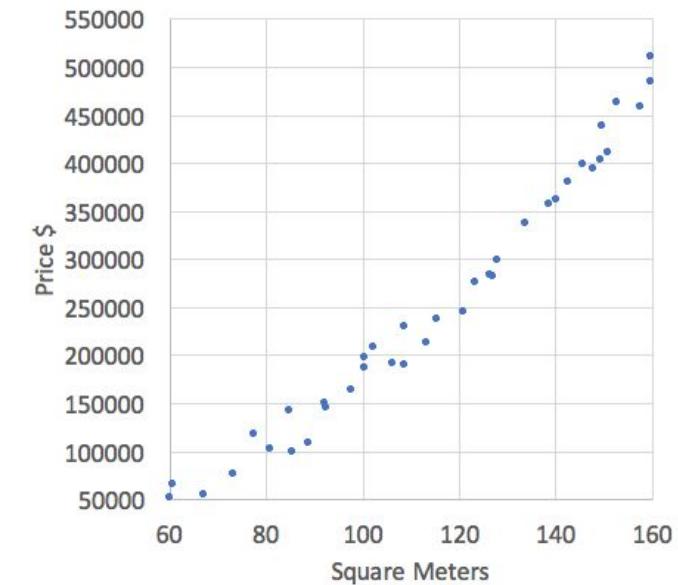
- A simple trick to obtain non-linearity is to artificially generate new features:
 - We start from a feature x_1
 - We can compute the feature x_1^2 and add this new feature to our data
 - We then compute the feature x_1^3 and add it to the data
 - And so on...

Example: house prices

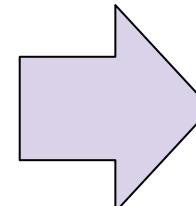
- Let's suppose that the price is a non-linear function of the size in square meters
- In this example, the function that relates the price to the size is:

$$\text{Price} = 0 * \text{Size} + 20 * \text{Size}^2 + 0$$

- We kept the zeros to show all the coefficients and variables
- We can model the above function by augmenting our dataset as follows:



Size (sqm)	Price
60	63193.69799
65.04312529	70911.59045
68.8814367	80082.32417
69.35554098	80602.04965
69.73350476	114729.8332
74.53091304	131513.3546



Size (sqm)	Size ²	Price
60	3600	63193.69799
65.04312529	3887.220733	70911.59045
68.8814367	4436.82399	80082.32417
69.35554098	4764.013352	80602.04965
69.73350476	5225.380597	114729.8332
74.53091304	5321.441446	131513.3546



Limits and advanced methods

- There are limits to the previous “trick”:
 - We have to **manually** add features before training
 - We don’t know where to **stop**
 - to which degree of the polynomial? x^3 ? x^4 ? ... x^{20} ?
 - How about products of features like x_1x_2 or $x_1^2x_2^3x_2^2$?
 - There can be so many features that this would make dimensionality (number of features) to explode
- There are **advanced methods** for regression that do not need to manually add features but have different ways of coping with non-linearity



Classification

- The classification problem ask to predict a categorical variable's value from the values of other variables
- Easiest example is with two variables x and y :
 - x is the variable in input
 - y is the variable we want to predict

x	y
1	A
2	A
3	B
4	B
5	A

Training data: x is the input data, y is the label data.

Learning
Learning task: **what's the mapping from x to y ?**

Testing
Try to “learn” from the training data an hypothesis function h
What is $h(6)$? That is, what's y when x is 6?

x	y
6	?

Classification example 1/2

- We have again the data from 10 thousand houses in the U.S.

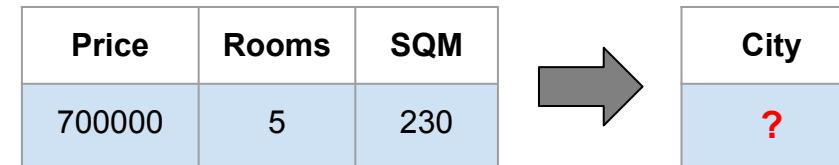
City	Rooms	SQM	Price
Los Angeles	3	130	420000
Los Angeles	2	60	380000
...
Albuquerque	2	140	220000
Albuquerque	3	150	250000

- Goal: learn a function (model) that can infer the City given all the other variables, for houses not in the 10 thousand dataset:

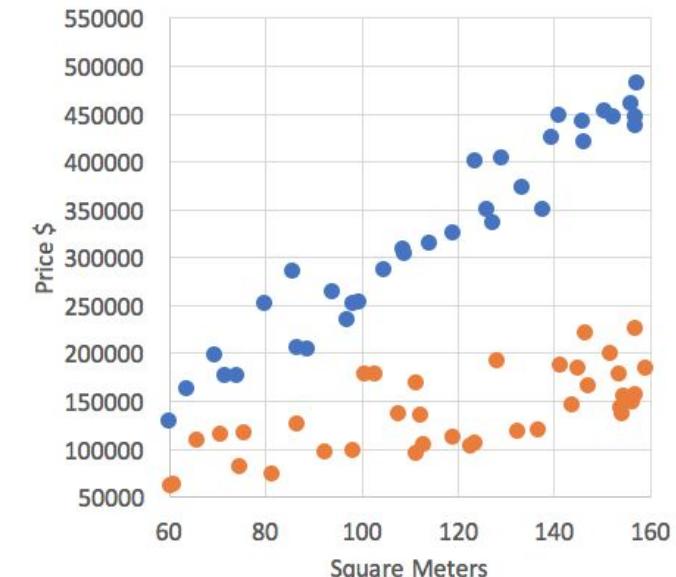


Classification example 2/2

- The houses' city can be an example of **classification** task:
 - Among the variables we choose the output variable (City)
 - The output variable is the *ground truth*, because it's the actual city of the house
 - The output variable we want to predict is a categorical value

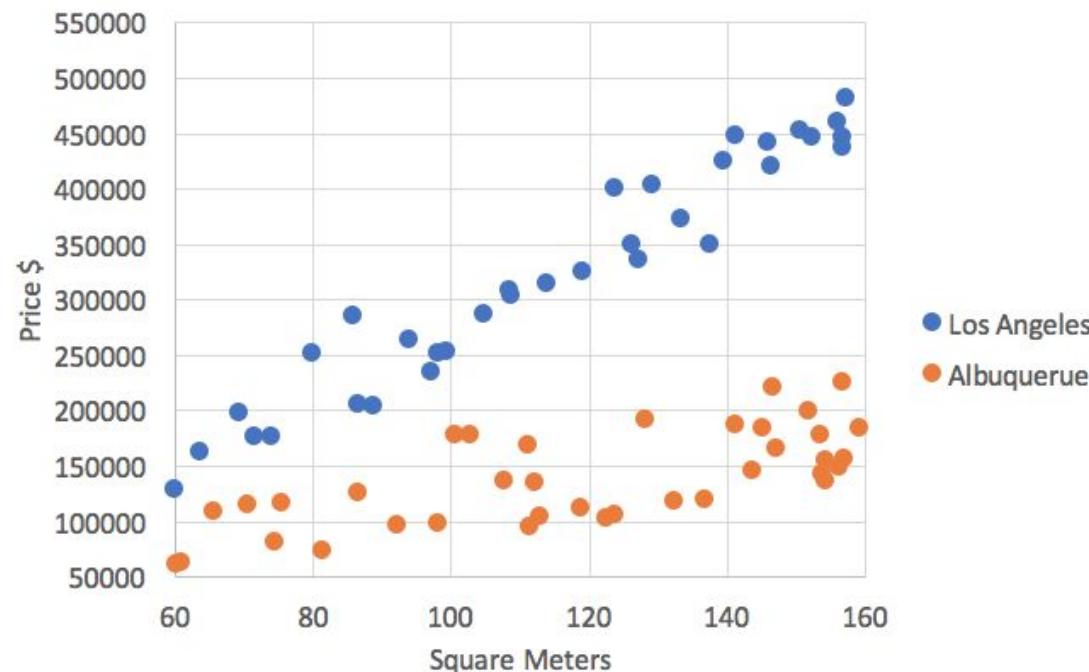


- To visualize things better we will have examples with two variables (no rooms) as input and one as output
- The categorical output will be visualized using a **different color for each category**



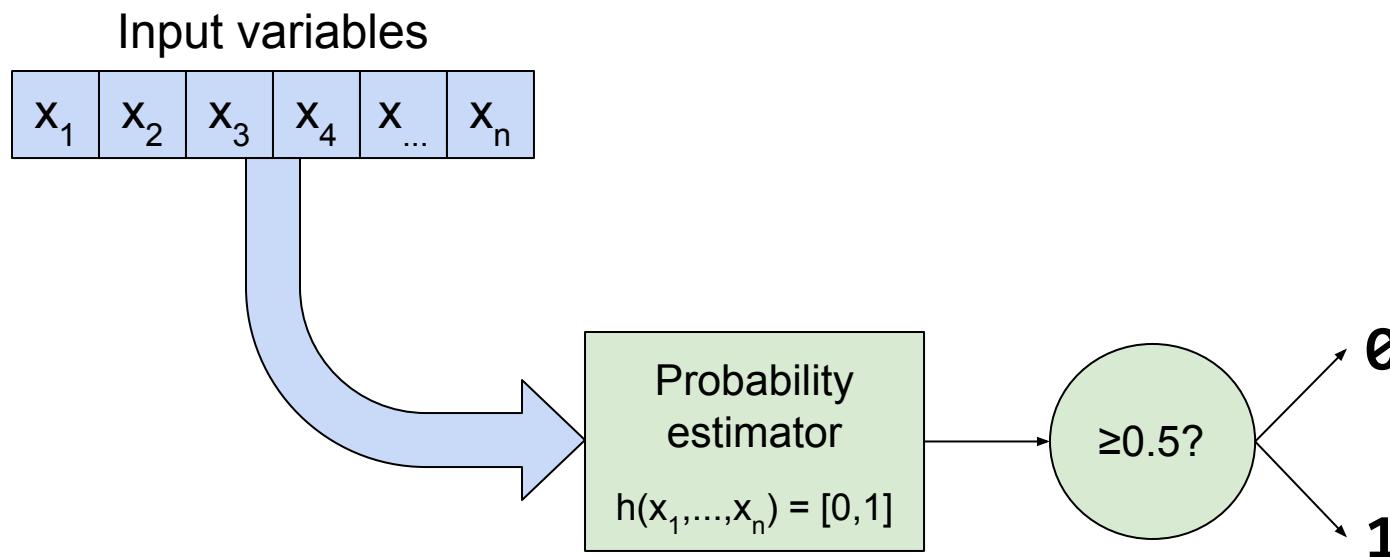
Binary classification

- The binary classification is the task of classifying between two classes.
For example:
 - Detecting if an email is spam or not
 - Deciding if an image has a cat in it
 - Predicting if a currency will go up or down
 - Classifying if a house is in L.A. or Albuquerque



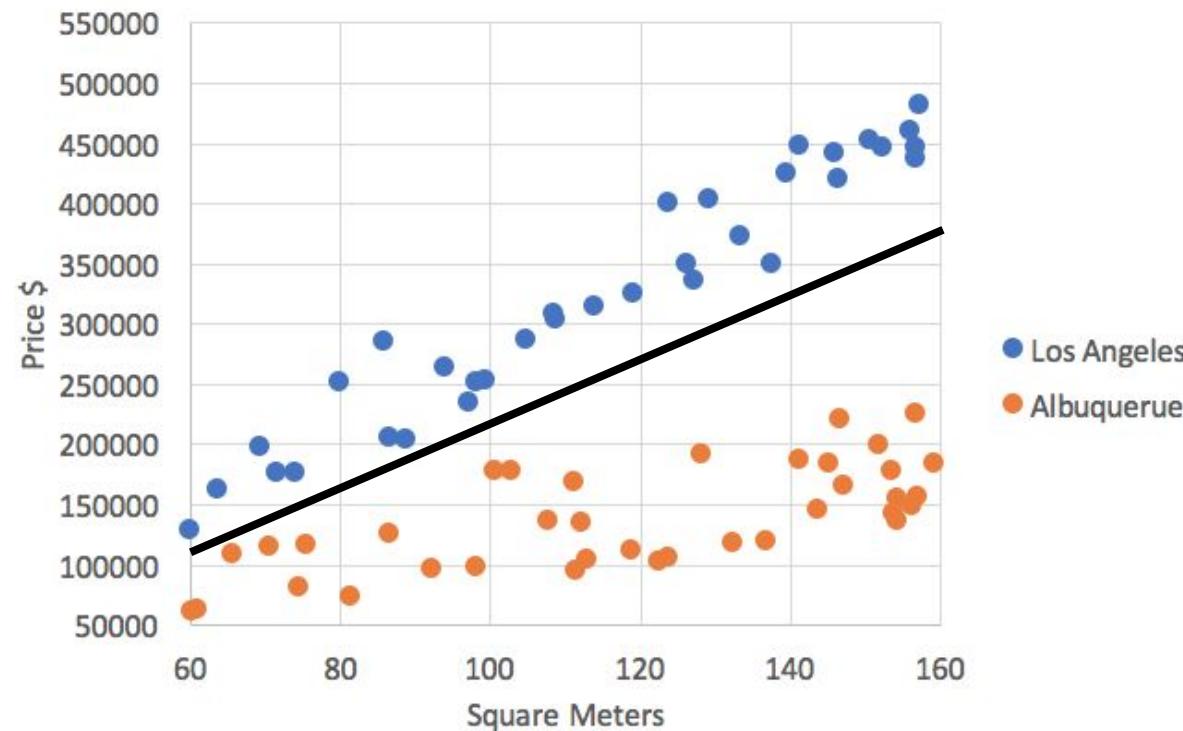
Binary classifier prediction

- A binary classifier prediction works as following:
 - Estimate a score of probability for the first class
 - Output a binary value:
 - 1 if the probability score is greater than or equal to 0.5
 - 0 if the probability score is less than 0.5



Separating the space

- Visually speaking, it is very easy to linearly separate (i.e. with a straight line) the space between Los Angeles houses and Albuquerque houses:



- But how could we **learn** a function that separate the space automatically?

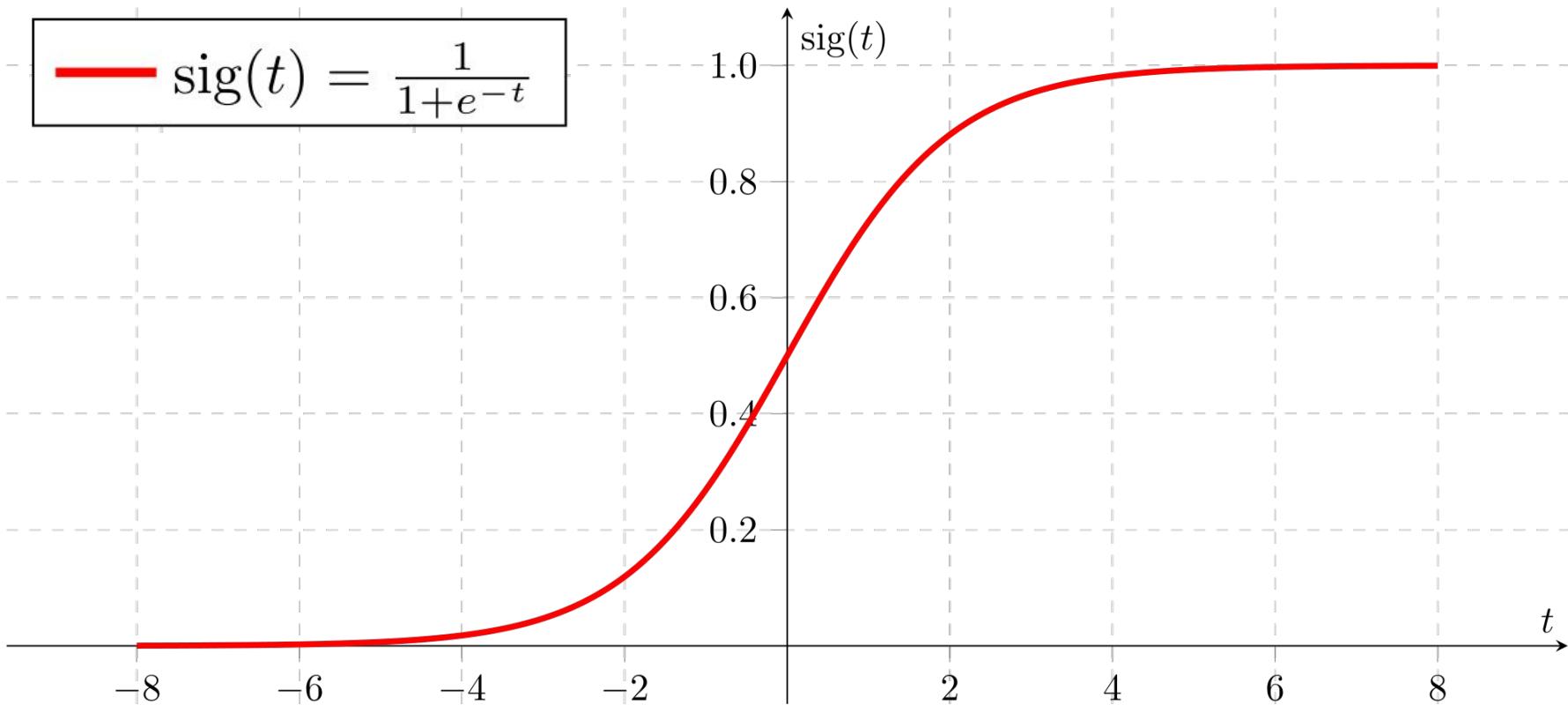


Linear regression for classification?

- In the case of classification, our hypothesis is a function that serves as a probability estimator
 - It takes the variables in input (e.g., house size and price)
 - It must outputs a **value between 0 and 1**
- Can we use **linear regression**? After all ...
 - ... the output value is actually numerical: 0 or 1
 - ... the line that separates our examples is a straight line
- **Problem:** we don't have control over unseen data, so we could have values bigger than 1, or lower than 0
- **Idea:**
 - We use a linear function to combines all the input variables into a value
 - We apply a function to constrain every possible value into a range of [0,1]

Logistic (sigmoid) function

- The ***sigmoid*** function is a special case of the **logistic function**
- Sigmoid functions have domain of **all real numbers**, with return value monotonically increasing **from 0 to 1**





Logistic Regression

- Despite the name, the logistic regression is actually a **classification** method
- Recall the linear regression hypothesis was a linear combination of the input variables, with one weight for each variable:
$$h_{\text{linear}}(x_1, \dots, x_n) = w_1 x_1 + \dots + w_n x_n + b$$
- Its hypothesis function is simply the sigmoid function applied to the linear combination:

$$h_{\text{logistic}}(x_1, \dots, x_n) = \text{sig}(w_1 x_1 + \dots + w_n x_n + b)$$



Logistic Regression: loss function

- Recall that we need a loss function to **tune the weights towards a minimum loss**
- We have a training input x . We consider two cases:
 - **When the true y is 0:** the hypothesis should output 0, anything more than 0 is a loss! So we can use simply the hypothesized score

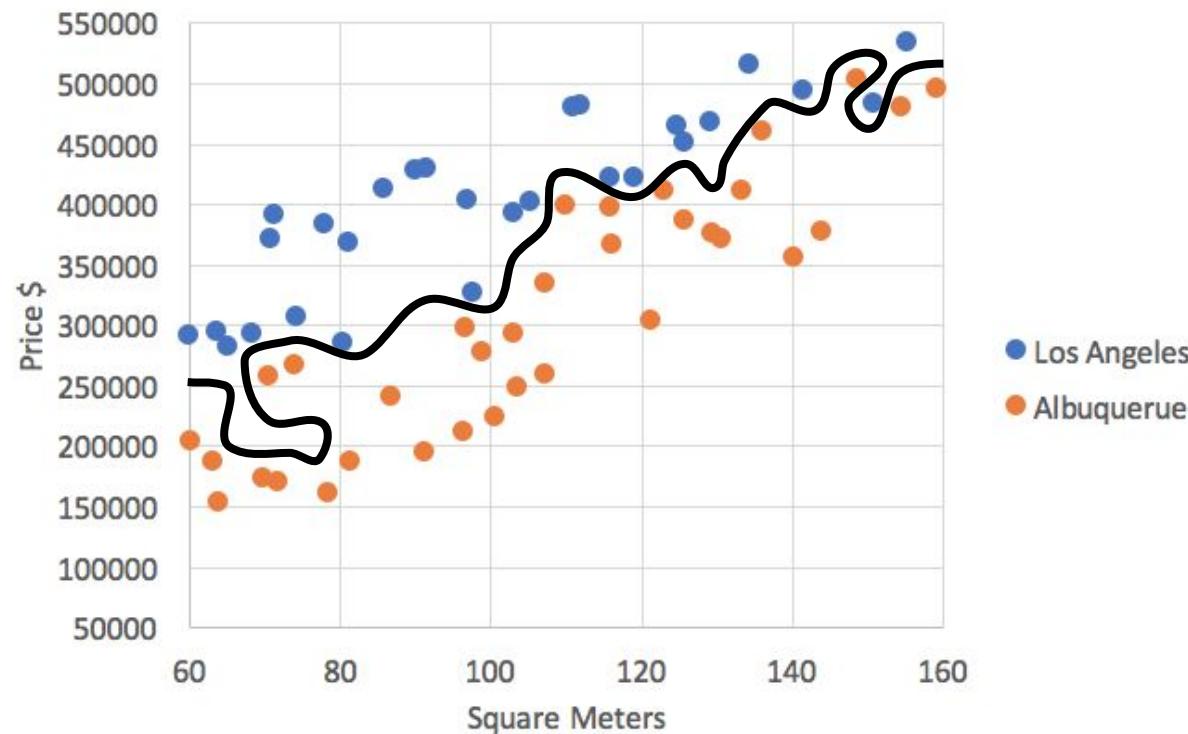
$$\text{loss}(x) = h_{\text{logistic}}(x)$$

- **When the true y is 1:** the hypothesis should output 1, anything less than 1 is a loss! So we can use the difference between 1 and the hypothesized score

$$\text{loss}(x) = 1 - h_{\text{logistic}}(x)$$

Problem: Overfitting 1/2

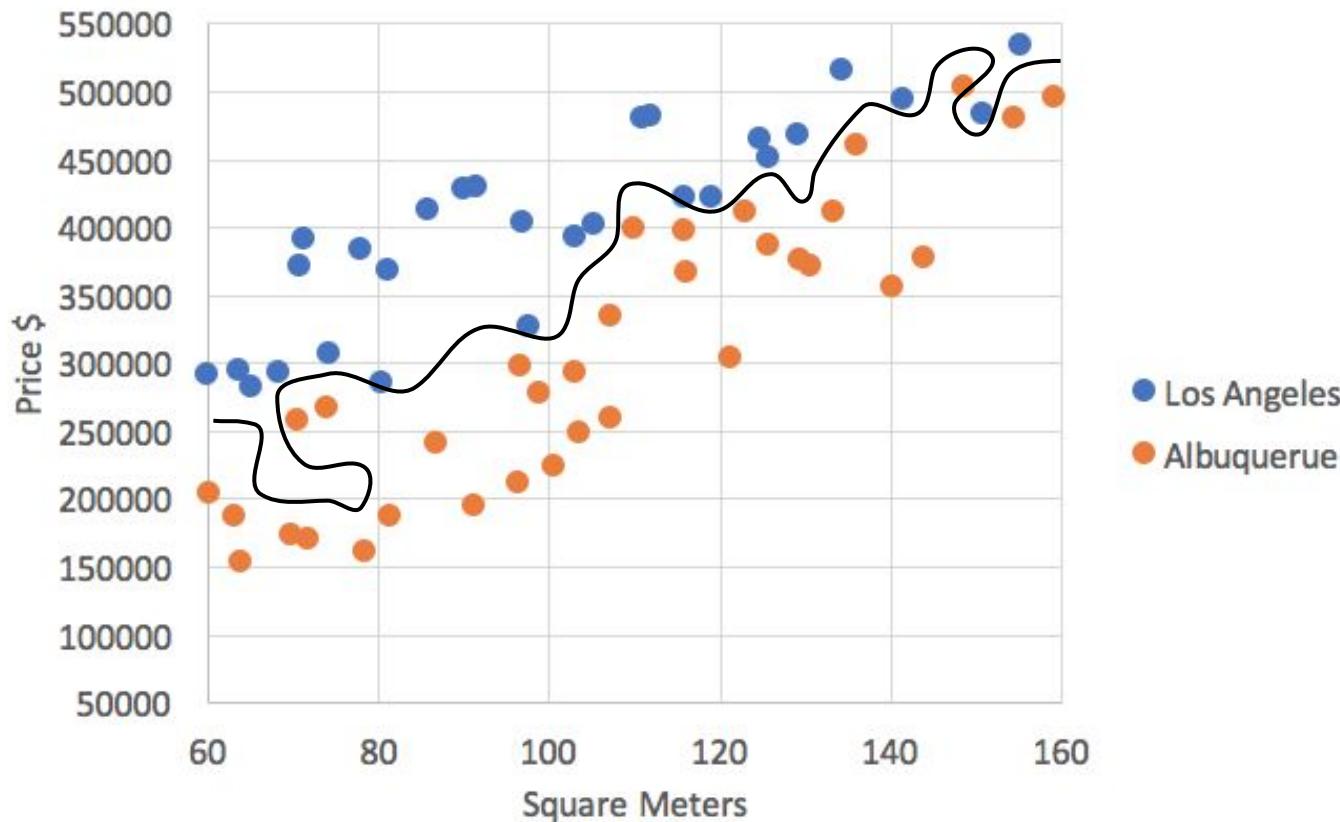
- Using the **same trick** of introducing artificial features using polynomials of different degrees (e.g. $x_1^2x_2^3x_2^2\dots$), we could train a very complex non-linear function:



- Is this a good thing?** For sure it's perfect for the training set ... what about **unseen data**?

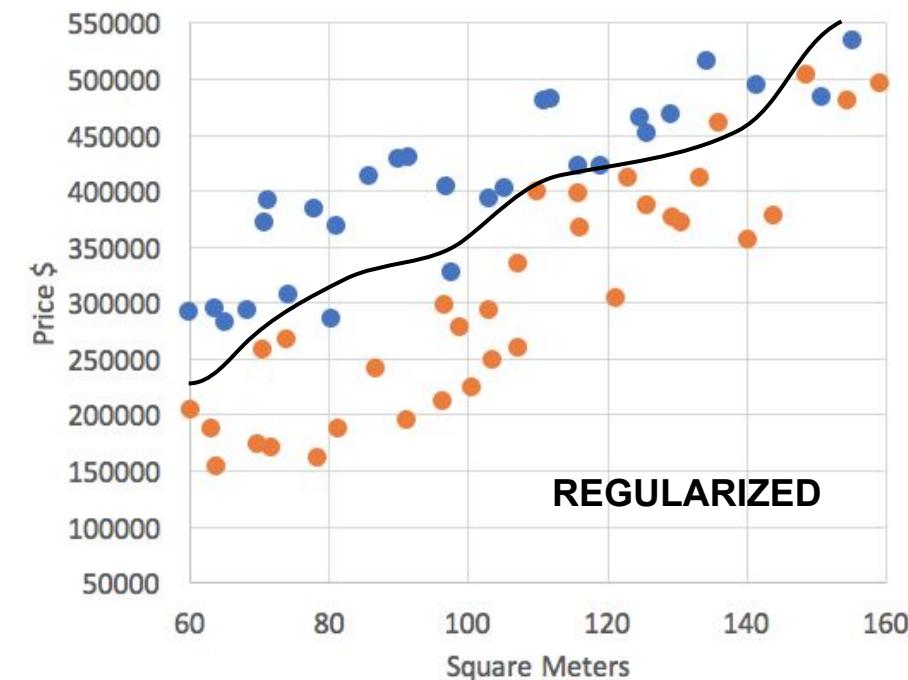
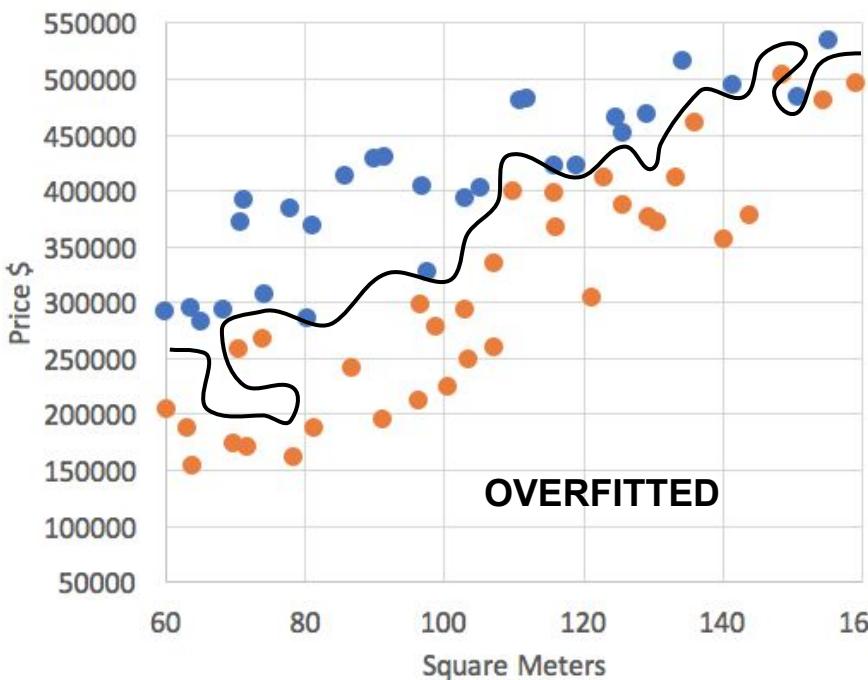
Problem: Overfitting 2/2

- It turns out it is actually not good: the model is **overly fitted** on the training set the unseen data could fall randomly on one side or the other
- The model should be more robust and **generalized** to work also on unseen data (such as the testing set)



Solution: Regularization

- Regularization is a very effective technique to prevent overfitting
 - It's a modification of the loss function
 - We add an additional value to the loss function that increases as the value of features weights (w) increase
- What is the effect of regularization?
 - It prevents the features weights (w) to be very large
 - Visually speaking, it will “smooth” the decision boundary line:





Multi-class classification

- Binary classification **can be generalized** to multi-class classification
- There are **two ways** to adapt binary classifiers to a **K-classes** classification problem:
 - **OVO - One vs. One:** Each possible pair of classes is taken in consideration to train a different classifier. This approach will **need to train $K*(K-1)/2$ classifiers!**
 - **OVA - One vs. All** (more common): for each class, a different classifiers is trained to distinguish between that class and all the other classes merged together. This approach will **need to train only K classifiers**



References

[Stanford Lectures on Machine Learning](#)

Andrew Ng



Big Data and Data Mining

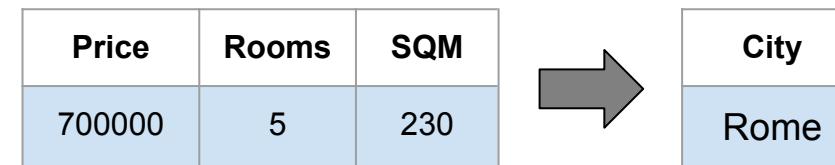
Unsupervised Learning

Flavio Bertini

flavio.bertini@unipr.it

We have seen so far...

- Until now we have seen supervised methods, in which a human **training** (supervision) was needed
 - Training was given as a **large set of examples**
- Each example was an association from a set of **input variables** (X) to an **output variable** (y)



- Now we will see **unsupervised methods**, in which no training is involved. They are used to:
 - Mine frequent patterns to make **associations** between examples
 - Group together similar objects creating **clusters**



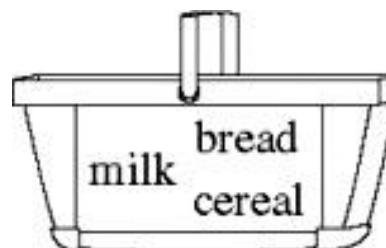


Mining frequent patterns

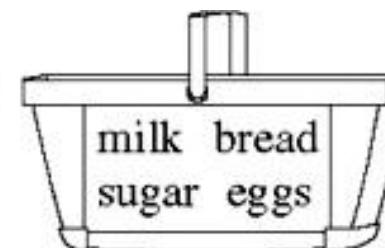
- Mine frequent patterns to make **associations** between examples allows to discover new relations from data
 - Frequent patterns are patterns that appear frequently in a data set
- This kind of technique can be applied in different scenarios, such as:
 - Marketing basket analysis (frequent patterns in products buying)
 - Bioinformatics (associating genes and diseases)
 - Intrusion detection systems (finding malicious activities)

Market basket analysis

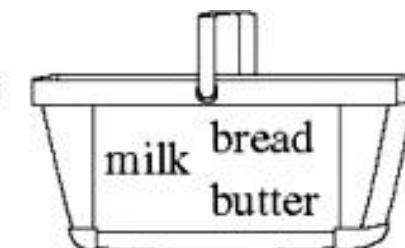
- A typical example of frequent pattern mining is **market basket analysis**
- This process analyzes customer **buying habits** by finding associations between the different items that customers place in their “shopping baskets”
 - For instance, if customers are buying milk, how likely are they **to buy also** bread (and what kind of bread) on the same trip to the supermarket?



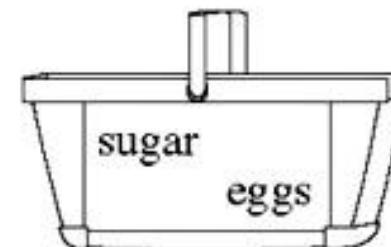
Customer 1



Customer 2



Customer 3



Customer n

Example: AllElectronics

- As a manager of an *AllElectronics* branch you wonder: “Which groups or sets of items are customers likely to purchase on a given trip to the store?”
 - To answer your question, **market basket analysis** may be performed on the data of customer transactions at your store
- You can then use the results to design a smart store layout
 - Items that are frequently purchased together can be placed in proximity to encourage the combined sale of such items
 - Example: if customers who purchase computers also tend to buy antivirus software at the same time, then placing the hardware display close to the software display may help increase the sales of one or both items





Association Rules

- Transactions data can be analyzed for buying patterns that reflect items that are frequently associated (purchased together)
- These patterns can be represented in the form of **association rules**
 - For example, the information that *customers who purchase computers also tend to buy antivirus software at the same time* is represented in the following **association rule**:

computer => antivirus_software [support = 2%, confidence = 60%]

- [*support = 2%*] means that in the 2% of all the transactions, computer and antivirus software are bought together
- [*confidence = 60%*] means that 60% of the people who bought a computer also bought an antivirus software at the same time



Support & Confidence

- We can now have a more formally look at the two measures
 - Let A be a set of items (can be a singlet $\{\text{computer}\}$)
 - Let B be a set of items (can be a singlet $\{\text{antivirus_software}\}$)
- The rule $A \Rightarrow B$ holds in the transactions set with **support** s , where s is the percentage of all transactions, that **contains** $A \cup B$

$$\text{support}(A \Rightarrow B) = P(A \cup B)$$

- The rule $A \Rightarrow B$ holds in the transactions set with **confidence** c , where c is the percentage of transactions containing A , that also contain B

$$\text{confidence}(A \Rightarrow B) = P(B | A)$$



Mining strong association rules

- Association rules are considered **strong** if they satisfy **both**:
 1. A **minimum support threshold**
 2. A **minimum confidence threshold**
- These thresholds can be set by users or domain experts
- In general, **association rule mining** can be viewed as a two-step process:
 1. **Find all frequent itemsets** A : by definition, each of these itemsets will occur at least as frequently as a predetermined minimum support (if A doesn't satisfy the threshold, neither will $A \cup B$: if milk is not among the transactions, neither is milk and coffee)
 2. **Generate strong association rules** from the frequent itemsets A to B : by definition, these rules must satisfy minimum support and minimum confidence



Example: AllElectronics data

- Consider the following transactions data for *AllElectronics*:

Transaction ID	Item set
T100	{Item1, Item2, Item5}
T200	{Item2, Item4}
T300	{Item2, Item3}
T400	{Item1, Item2, Item4}
T500	{Item1, Item3}
T600	{Item2, Item3}
T700	{Item1, Item3}
T800	{Item1, Item3}
T900	{Item1, Item2, Item3, Item5}
T1000	{Item1, Item2, Item3}

- We set minimum support 30% and confidence 70%

Example: iteration 1

- We first consider the *support* of itemsets with cardinality 1:

1-Itemsets	Support
{Item1}	70%
{Item2}	70%
{Item3}	70%
{Item4}	20%
{Item5}	20%

- With support 30% (and confidence 70%)
 - The itemsets that **does not** satisfy minimum support are **pruned**
 - The itemsets that **does** satisfy minimum support are **taken** in the next iteration



Example: iteration 2

- We now consider the *support* of itemsets with cardinality 2:

2-Itemsets	Support
{Item1, Item2}	40%
{Item1, Item3}	50%
{Item2, Item3}	40%

- All itemsets satisfy minimum support threshold and are taken in the next iteration

Example: iteration 3

- We now consider the *support* of itemsets with cardinality 3:

3-Itemsets	Support
{Item1, Item2, Item3}	20%

- The itemset **does not** satisfy minimum support are **pruned**
- We stop the iterations as there can be no larger itemsets
- We have therefore found the **largest frequent itemsets** as:

2-Itemsets	Support
{Item1, Item2}	40%
{Item1, Item3}	50%
{Item2, Item3}	40%



Example: association rules

- We found the frequent itemsets $\{\text{Item1}, \text{Item2}\}$, $\{\text{Item1}, \text{Item3}\}$, $\{\text{Item2}, \text{Item3}\}$
- What are the **association rules** that can be generated from the above frequent itemsets?
 - $\{\text{Item1}\} \Rightarrow \{\text{Item2}\}$ [support=40%]
 - $\{\text{Item2}\} \Rightarrow \{\text{Item1}\}$ [support=40%]
 - $\{\text{Item1}\} \Rightarrow \{\text{Item3}\}$ [support=50%]
 - $\{\text{Item3}\} \Rightarrow \{\text{Item1}\}$ [support=50%]
 - $\{\text{Item2}\} \Rightarrow \{\text{Item3}\}$ [support=40%]
 - $\{\text{Item3}\} \Rightarrow \{\text{Item2}\}$ [support=40%]
- We already computed their support in the search for frequent itemsets, therefore we know that minimum support is satisfied
- We now go back to the data to compute their confidence and check if they are **strong** association rules (confidence $\geq 70\%$)

Example: confidence

Trans. ID	Item set
T100	{Item1, Item2, Item5}
T200	{Item2, Item4}
T300	{Item2, Item3}
T400	{Item1, Item2, Item4}
T500	{Item1, Item3}
T600	{Item2, Item3}
T700	{Item1, Item3}
T800	{Item1, Item3}
T900	{Item1, Item2, Item3, Item5}
T1000	{Item1, Item2, Item3}

- We can compute confidence with the support values we already computed

$$\text{confidence}(A \Rightarrow B) = P(B | A) = \frac{\text{support}(A \cup B)}{\text{support}(A)}$$

- Goal: support 30% and confidence 70%

- {Item1} \Rightarrow {Item2} [support=40%, confidence=4/7= 57%]
- {Item2} \Rightarrow {Item1} [support=40%, confidence=4/7= 57%]
- {Item1} \Rightarrow {Item3} [support=50%, confidence=5/7= 71%]
- {Item3} \Rightarrow {Item1} [support=50%, confidence=5/7= 71%]
- {Item2} \Rightarrow {Item3} [support=40%, confidence=4/7= 57%]
- {Item3} \Rightarrow {Item2} [support=40%, confidence=4/7= 57%]



Mining association rules: recap

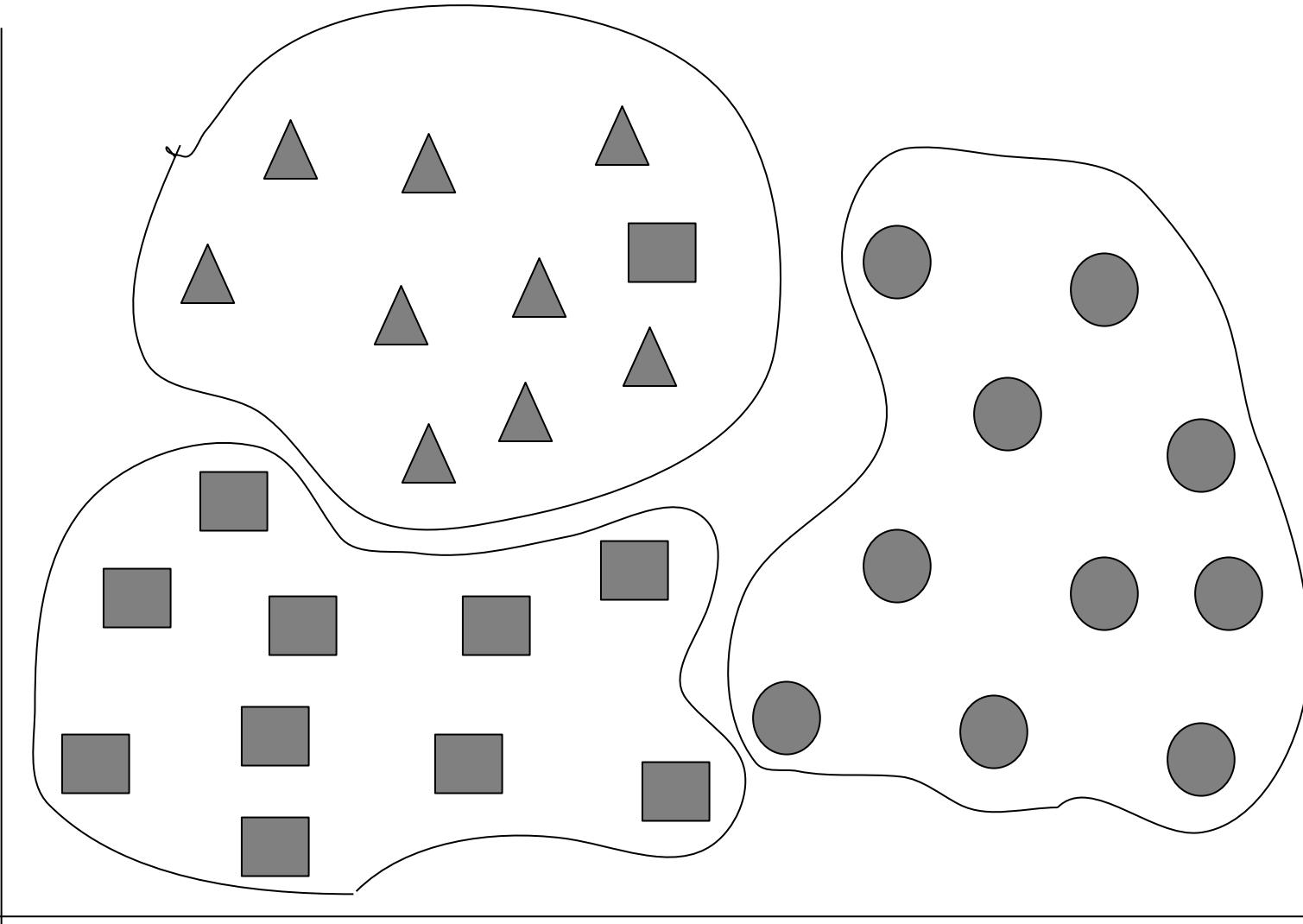
- We have seen how to iteratively find frequent itemsets and filter them by minimum support and minimum confidence rules
- The algorithm we used to find strong association rules is called ***a priori* algorithm**
 - The *a priori* property says that ***if a set cannot pass a test, all of its supersets will fail the same test as well.*** This allowed us to prune and reduce the search space
- **Association rules mining** of itemsets is an example of ***unsupervised learning*** task
- We will now see how items (or, more generally, objects) can be *clustered* together based on their common features
 - This method is called **clustering**



Clustering

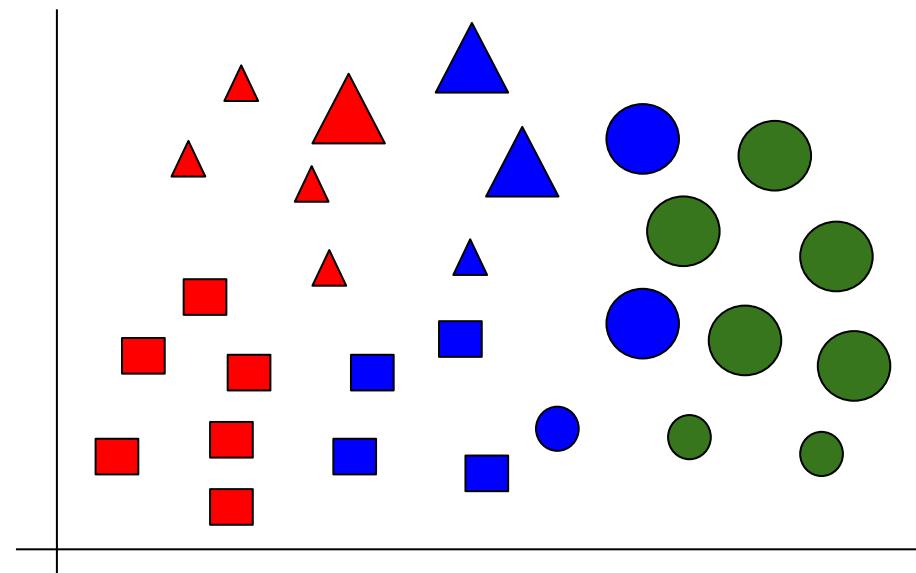
- Goal of clustering is to group together similar objects (documents, patients, houses) into “clusters”, depending on their features:
 - usually, objects are **grouped together if they belong to the same “category”** but...
 - **categories are not known!**
- Clustering has no “*ground truth*” since:
 - it’s an unsupervised technique: there is no labeled data
 - evaluation depends on perspective, may be different for different people
 - being without supervision restrictions, it may find new, unknown common features shared by objects: *pattern discovery*

Clustering example



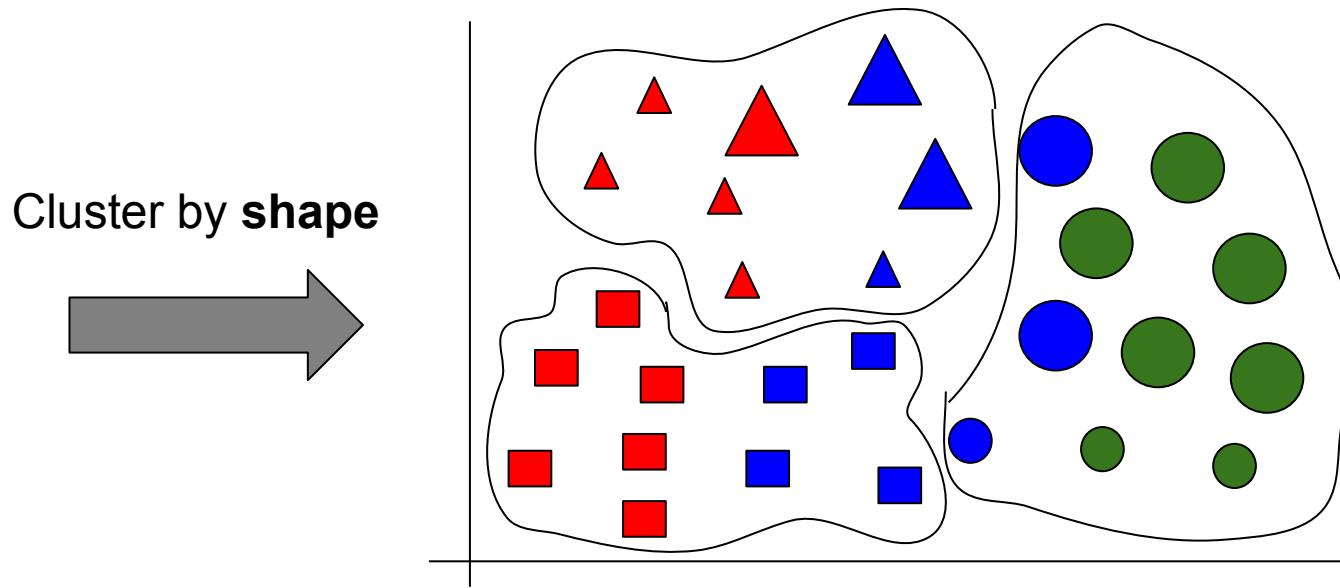
Clustering Bias (1)

- It's common to say that clusters are “in the eye of the beholder”:
 - are the terms “horse” and “car” similar?
 - in the previous example the clustering bias was the shape of each object, it was obvious
 - with more features it may be less obvious



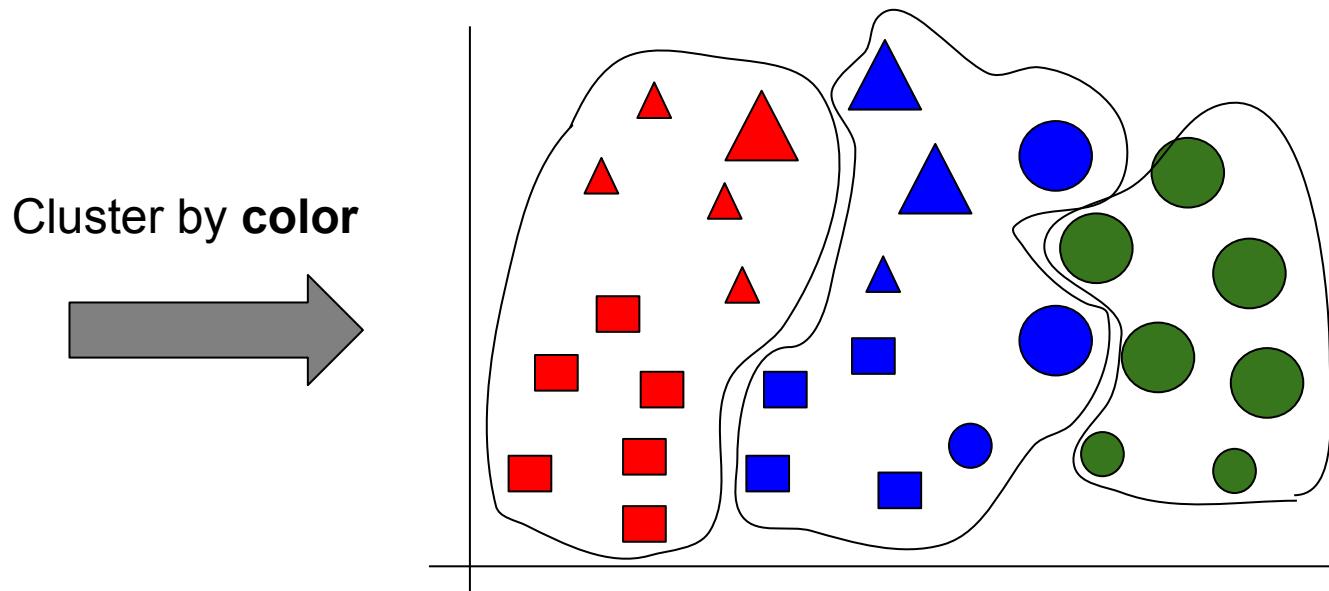
Clustering Bias (2)

- It's common to say that clusters are “in the eye of the beholder”:
 - are the terms “horse” and “car” similar?
 - in the previous example the clustering bias was the shape of each object, it was obvious
 - with more features it may be less obvious



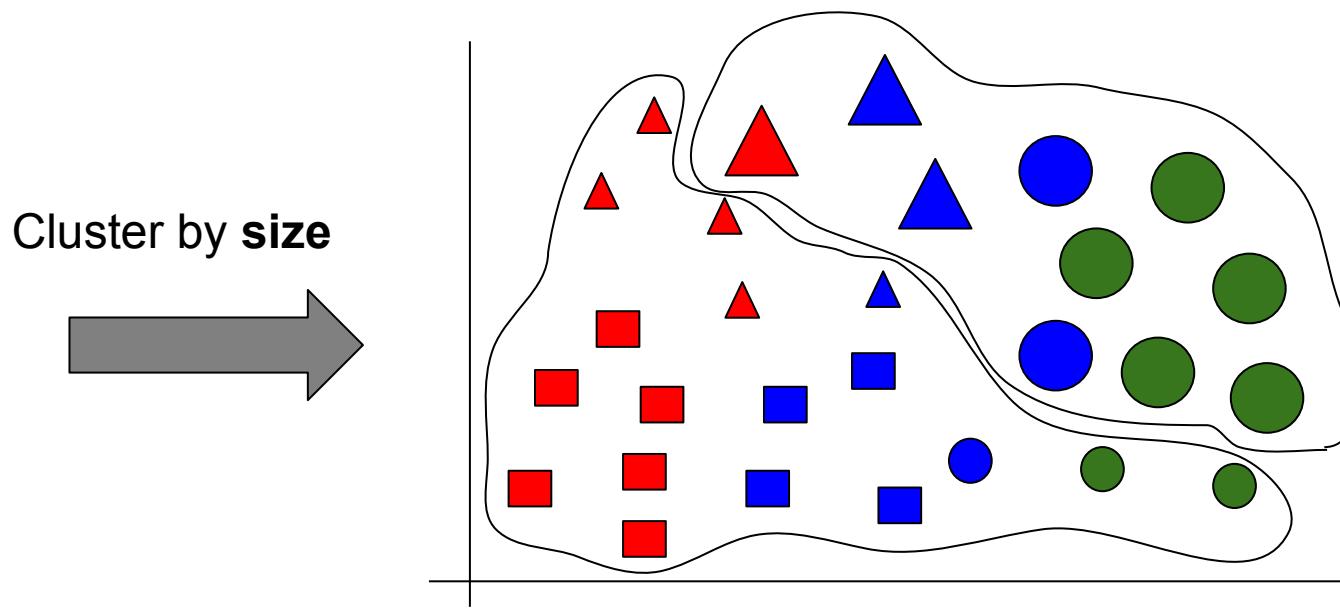
Clustering Bias (3)

- It's common to say that clusters are “in the eye of the beholder”:
 - are the terms “horse” and “car” similar?
 - in the previous example the clustering bias was the shape of each object, it was obvious
 - with more features it may be less obvious



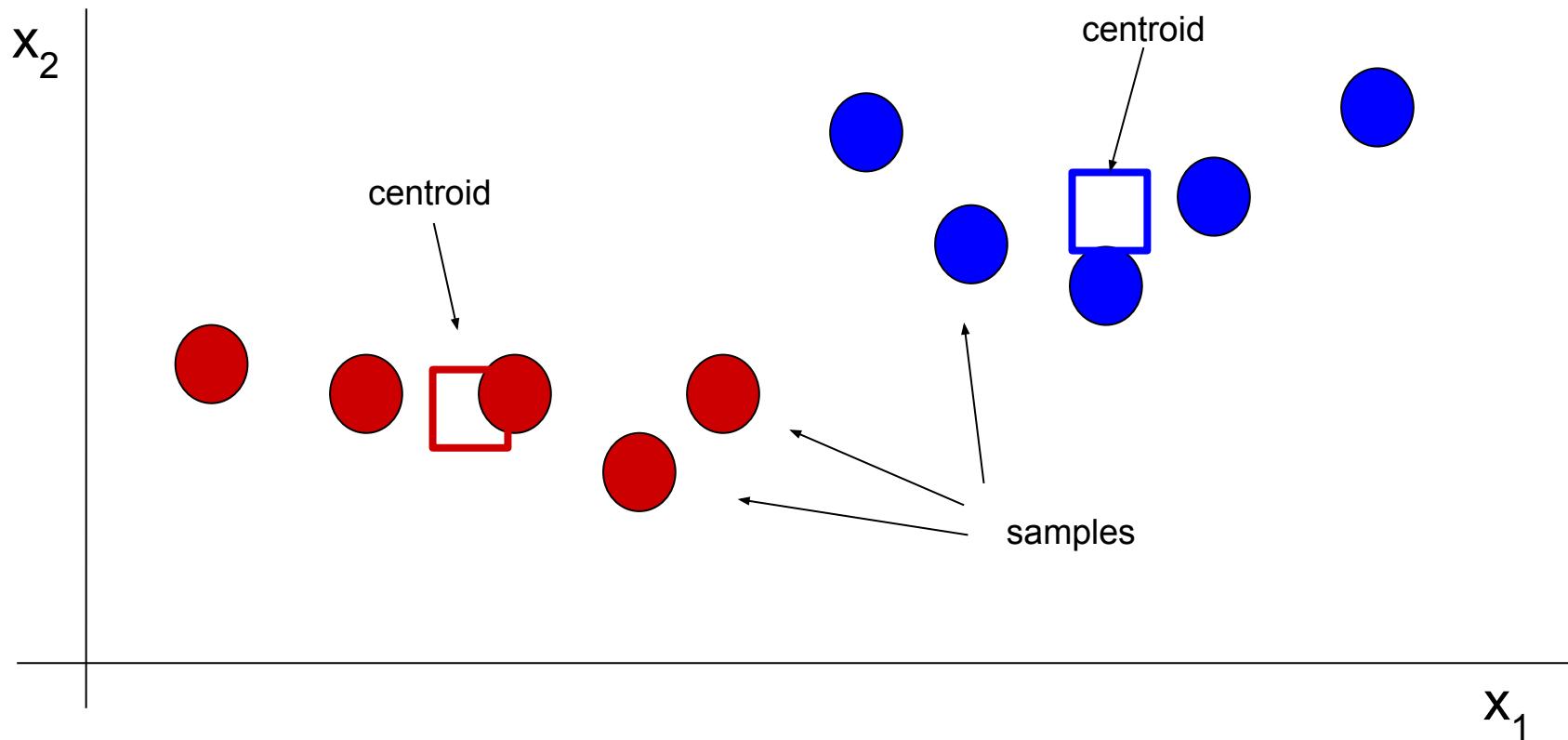
Clustering Bias (4)

- It's common to say that clusters are “in the eye of the beholder”:
 - are the terms “horse” and “car” similar?
 - in the previous example the clustering bias was the shape of each object, it was obvious
 - with more features it may be less obvious



k-means

- k-means clustering aims to partition n samples into k clusters in which each sample **belongs to the cluster with the nearest mean**, serving as a prototype of the cluster





k-means

- Algorithm for k-means starts with a “weak” clustering that is **refined** in many rounds **until nothing changes**
- The parameter k is user-defined (number of clusters)

ALGORITHM

```
samples = [(x11,x12),..., (xm1,xm2)] # our dataset of m samples in 2-dimension space
centroids = pick_random(samples,k) # pick k random samples as centroids

for each sample in samples:           # first weak clustering
    cluster[sample] = nearest(sample,centroids) # assign to nearest centroids

anychange = True

while (anychange):
    anychange = False
    centroids = [mean(cluster1),...,mean(clusterk)] # recompute centroids
    for each sample in samples:
        if cluster[sample] != nearest(sample,centroids): # we should move
            cluster[sample] = nearest(sample,centroids)
            anychange = True
```

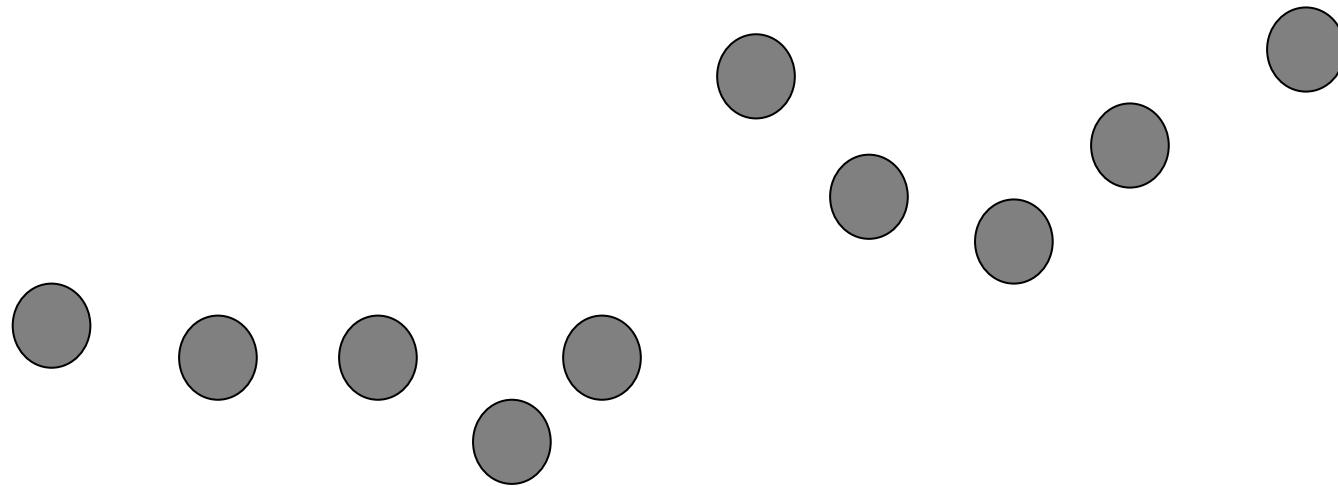


k-means convergence

- The **Stirling numbers of the Second kind** gives the number of subdivisions of a set of n objects into k cluster. It is a large but finite number
$$\left\{ \begin{matrix} n \\ k \end{matrix} \right\} = \frac{1}{k!} \sum_{j=1}^k (-1)^{k-j} \binom{k}{j} j^n$$
- For each iteration of the algorithm, we produce a new clustering based only on the old clustering
- At each round there can be one of two cases:
 1. The old clustering is the same as the new → the iteration stops
 2. The new clustering is different → the distance between the samples and the centroids is lower
- If the old clustering is the same as the new, then the next clustering will again be the same
- If the new clustering is different from the old then the newer one has a lower cost

k-means: example

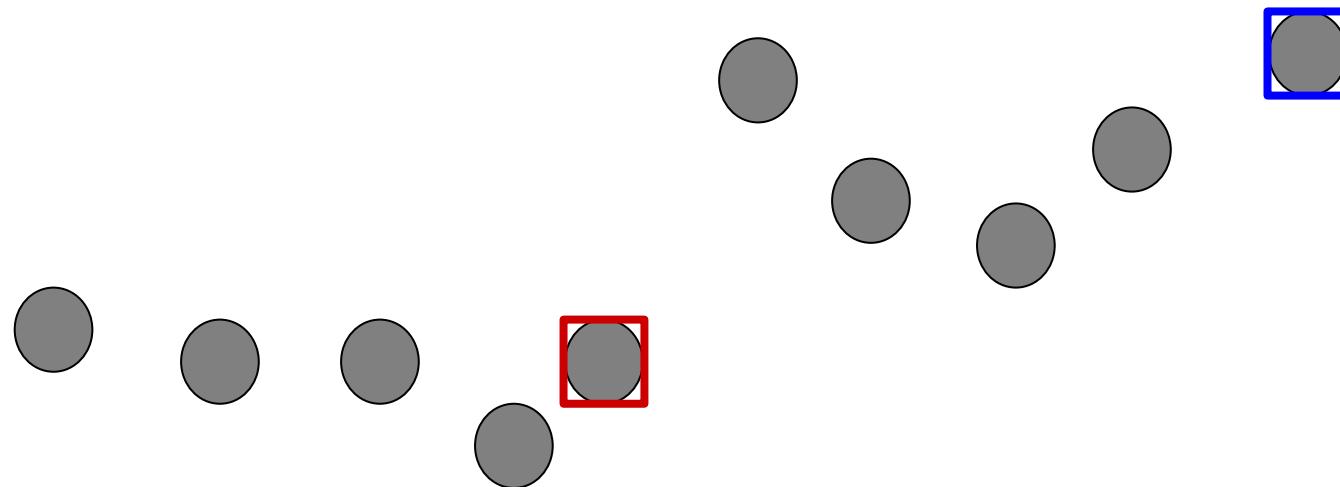
This is the collection of documents in the vector space



k-means: first round

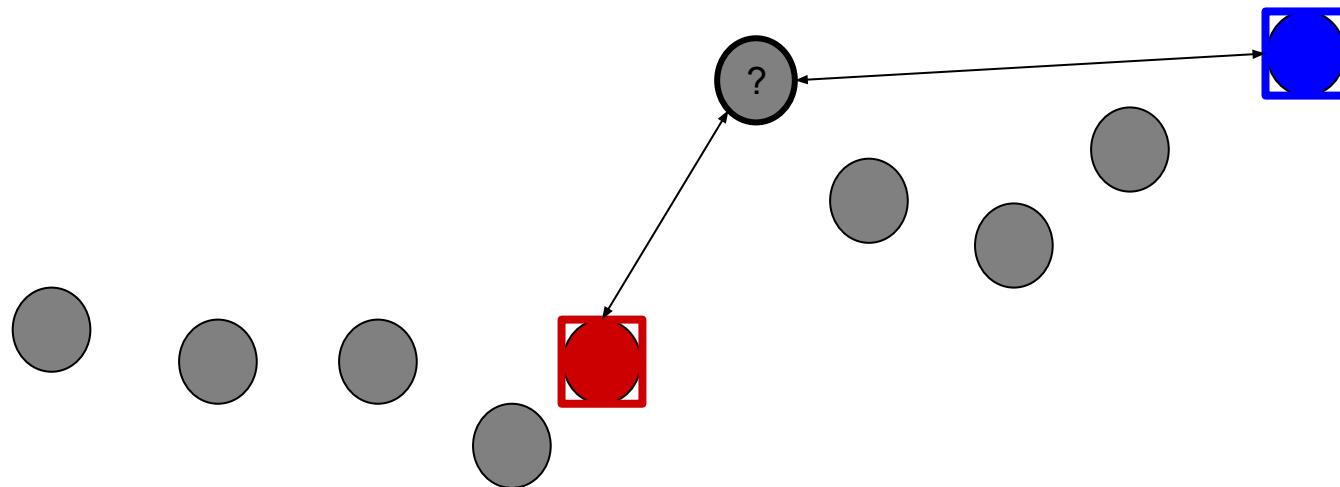
The hyperparameter k is set to 2 in this example

We pick 2 random documents as centroids



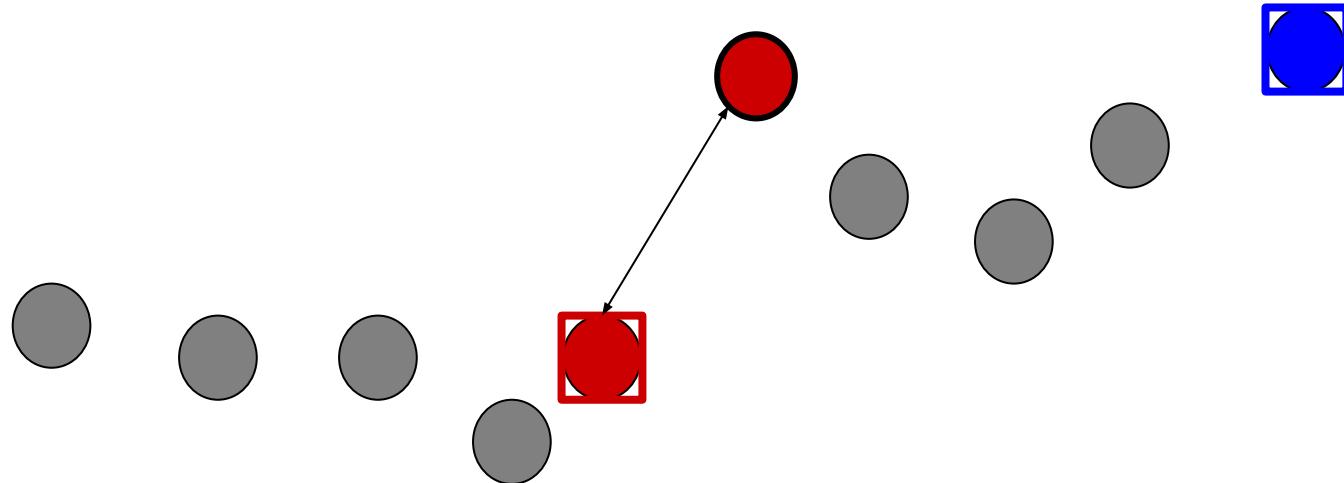
k-means: first round

Each document is assigned to the cluster of the nearest centroid



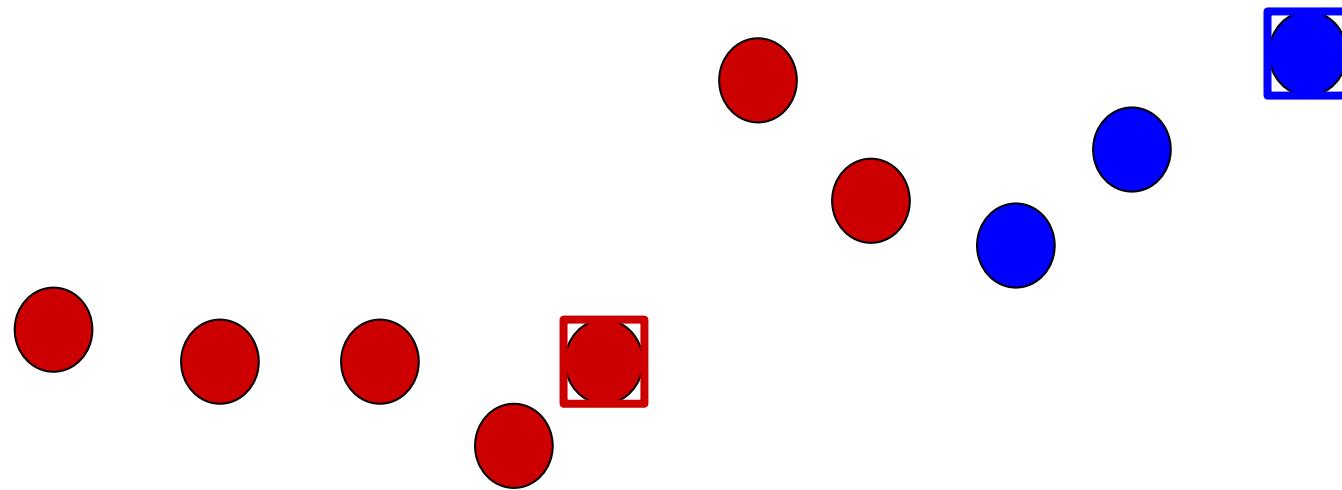
k-means: first round

Each document is assigned to the cluster of the nearest centroid



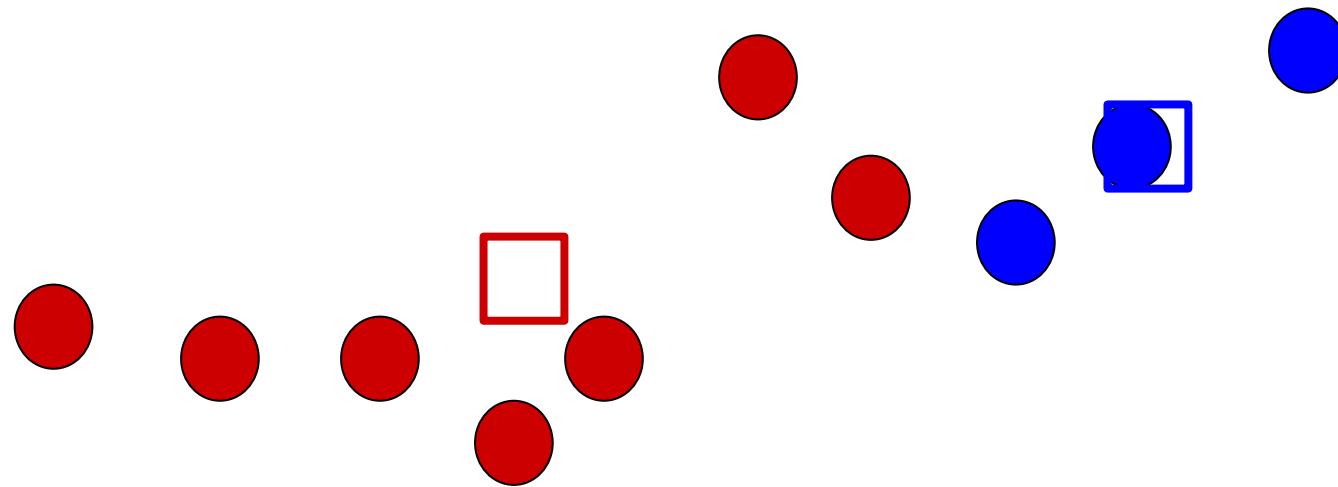
k-means: first round

Each document is assigned to the cluster of the nearest centroid



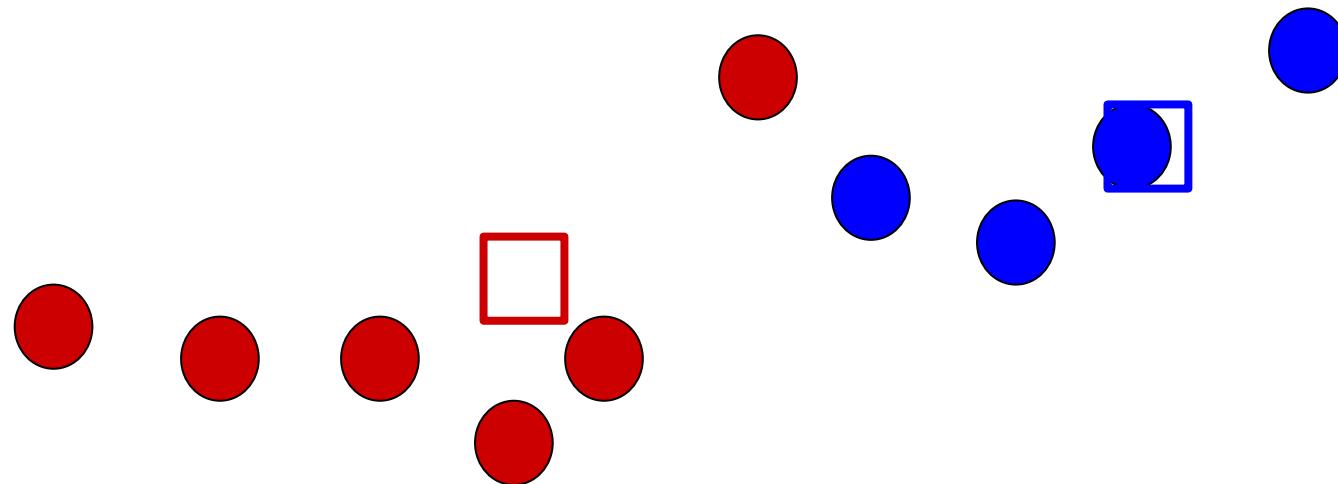
k-means: second round

Centroids are updated: real centroid of the assigned samples



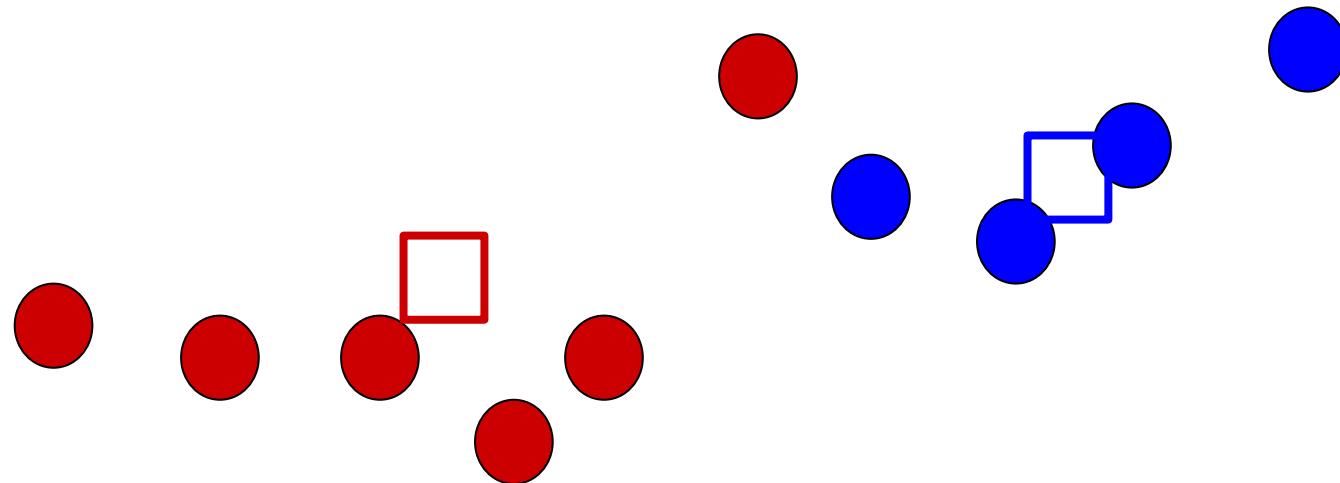
k-means: second round

Samples are re-assigned on the basis of the new centroids



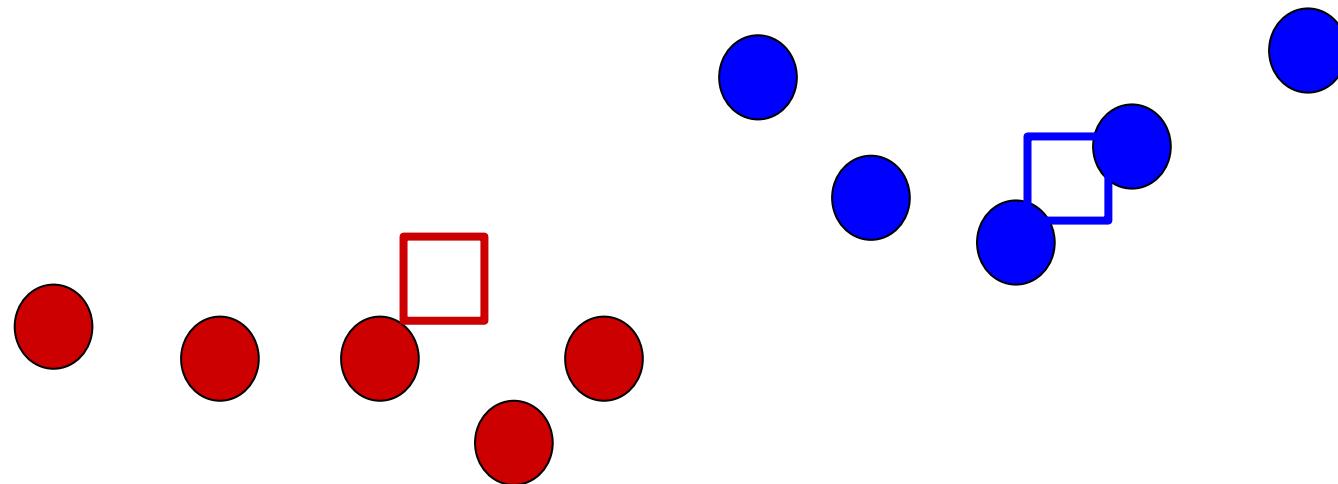
k-means: third round

Centroids are re-computed



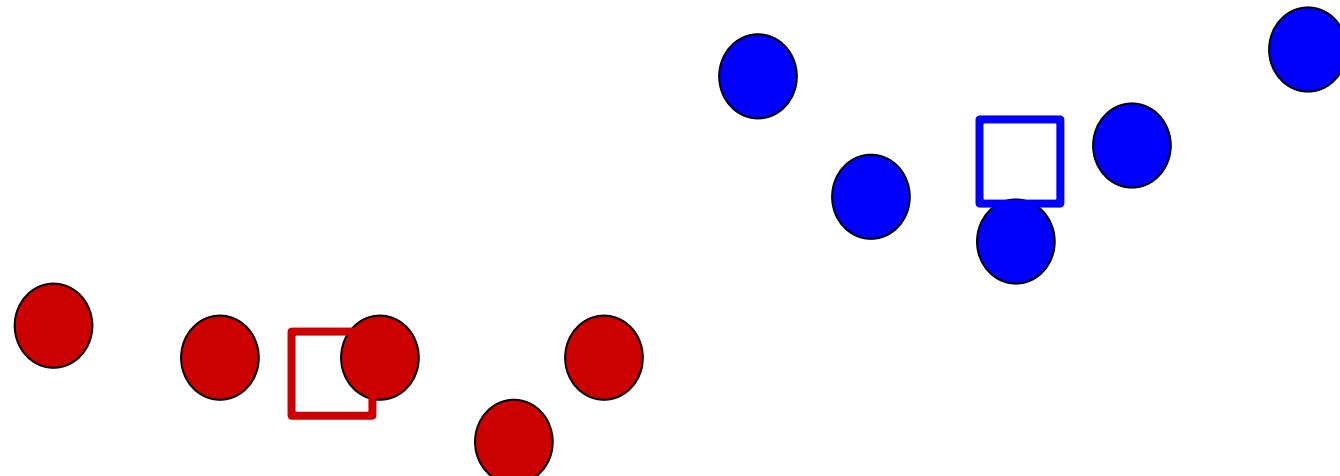
k-means: third round

Samples are re-assigned



k-means: fourth round

Centroids are re-computed

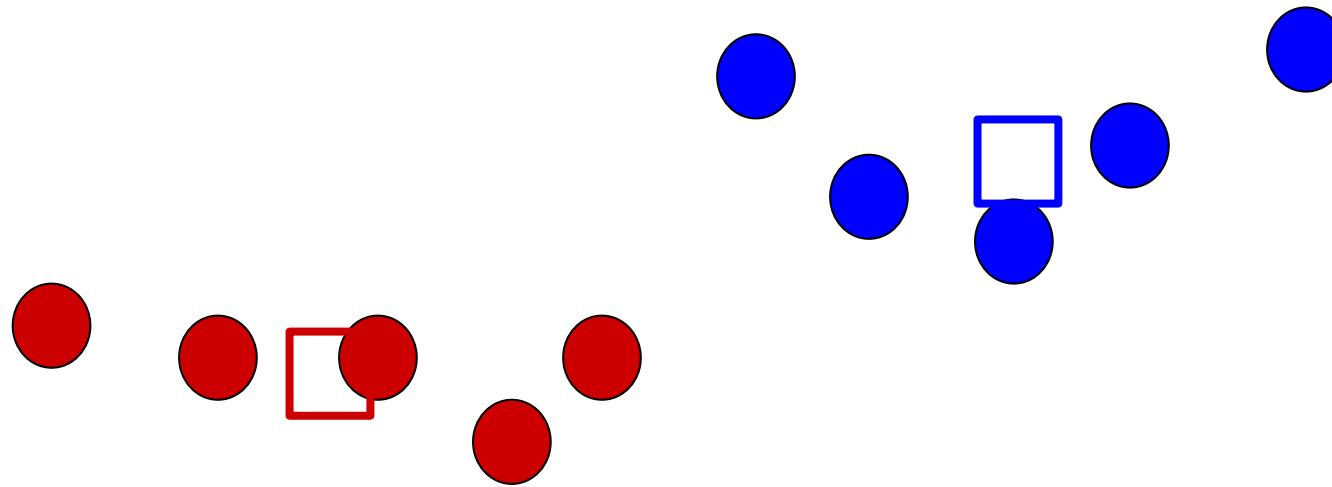




k-means: fourth and last round

No sample is re-assigned!

STOP THE ITERATION





Picking first k points

- The way how we choose the k points can make the k-means algorithm to **converge faster**
 - The assignment loop (internal loop) is linear on the number of points for each cluster... but number of rounds (external loop) is unknown!
 - Usually rounds << m but the worst case reaches 2^m rounds: **exponential time**
- Good way to choose first points is to pick **dispersed** set of points:
 - pick first point at random
 - pick as next point the one with the **largest minimum distance** from the already selected points
 - repeat until we have k points



Largest minimum distance?

- When we have selected only 1 centroid, picking the next is easy, because the above becomes:

$$\max(d(p_1, c_1), \dots, d(p_m, c_1))$$

- Let's say we already selected s centroids c and we want to pick up the next one among the m points p

$$\max(\min(d(p_1, c_1), \dots, d(p_1, c_s)), \dots, \min(d(p_m, c_1), \dots, d(p_m, c_s)))$$

- When we already have 2 or more centroids, for each point we
 - first search the nearest centroid (minimum)
 - then use that value to pick the point farther from any centroid (largest)

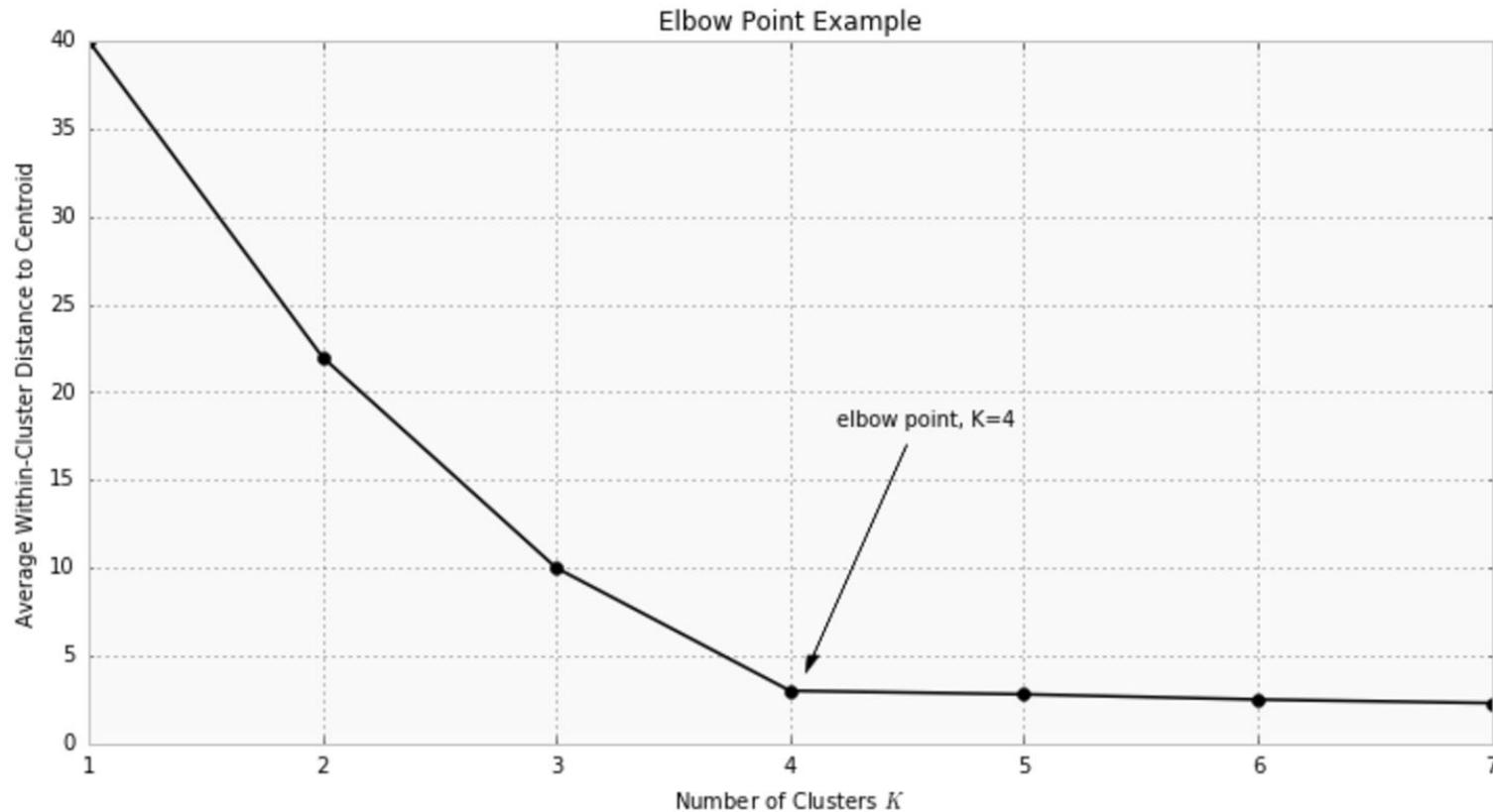


Choosing k

- Choosing into how many clusters we should group the documents depends on the task, there's no good automatic way
- However, some measures can help:
 - usually the average distance of all documents from the cluster centroid is used
 - k-means is then ran over different values of k
 - increasing k **will always** decrease the average distance
 - We should stop when the improvement begin to be small: **elbow point**

Elbow method

- The elbow method is a visual method to roughly find the best value for k
- When the line chart looks like an arm, then the "elbow" on the arm is the value of k that is the best





References

Data Mining Concepts and Techniques

Authors: Jiawei Han, Micheline Kamber, Jian Pei

