ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

inf | Informatik
Computer Science

The Systems Group
at ETH Zurich

XML and Databases
Exercise Session 11

courtesy of Ghislain Fourny

Where we are: XQuery Implementation

Behind the scenes

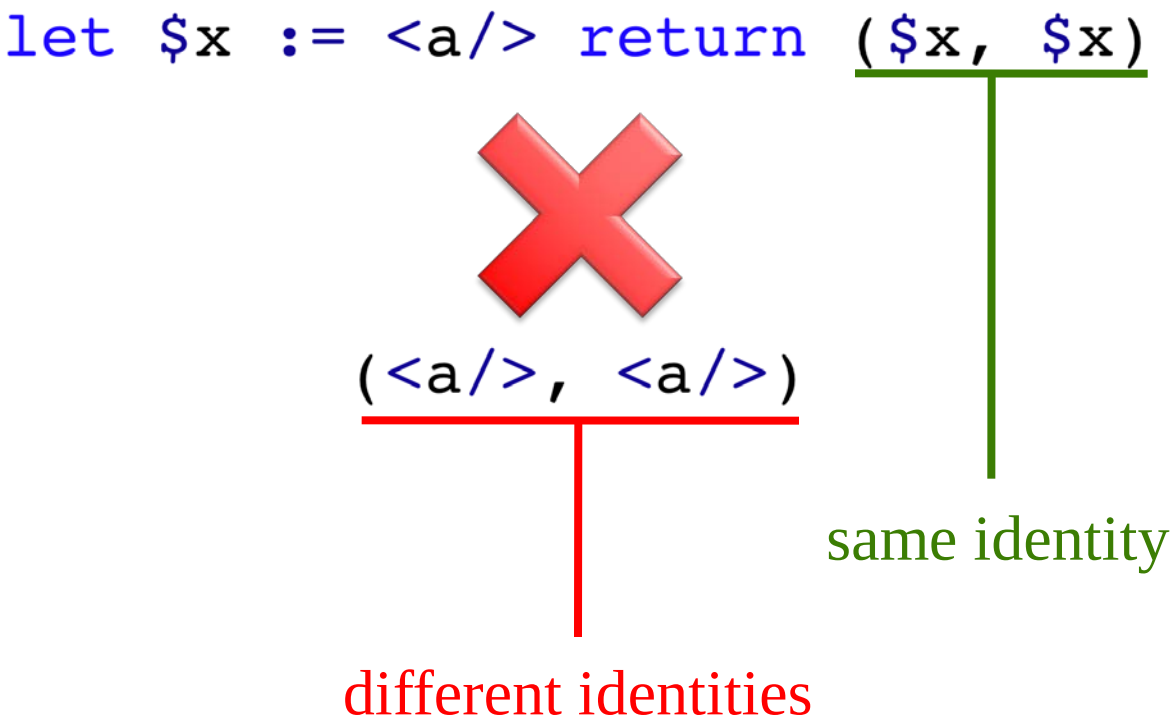
- The code has to be executed somewhere! New interesting problems arise:
  - Implementation
  - Optimization

Legal rewrite?

```
let $x := <a/> return ($x, $x)
```

```
(<a/>, <a/>)
```

Legal rewrite?



See it to believe it?

```
let $a:=<a><b/></a>
let $c:= ($a, $a)
return $c/b
```

```
let $a := <a><b/></a>
let $c := (<a><b/></a>, <a><b/></a>)
return $c/b
```

See it to believe it?

```
let $a:=<a><b/></a>
let $c:= ($a, $a)
return $c/b
```

<b/>

```
let $c := (<a><b/></a>, <a><b/></a>)
return $c/b
```

<b/><b/>

See it to believe it?

```
let $a:=<a><b/></a>
let $c:= ($a, $a)
return $c/b
```

<b/>

In a Path expression, at each step, document reordering and **duplicate elimination** are performed.

```
let $c := (<a><b/></a>, <a><b/></a>)
return $c/b
```

<b/><b/>

Document order

//a/b

Oh, and speaking of that...

reordering
+
duplicate elimination

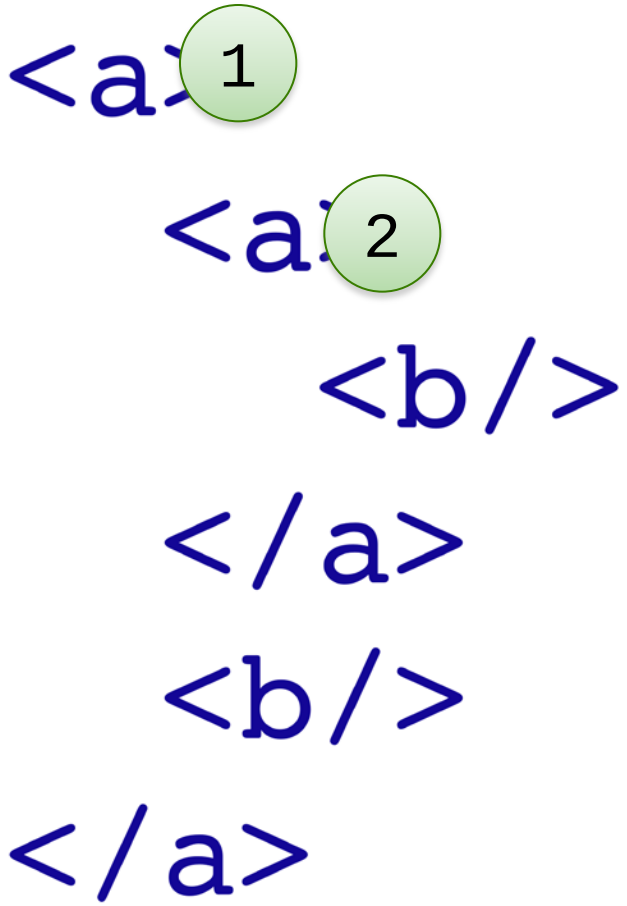
Document order

Why do we need this?

Document order

```
<a>
    <a>
        <b/>
    </a>
    <b/>
</a>
```

Document order



//a

Document order

```
<a>
    <a>
        <b/>
    </a>
    <b/>
</a>
```

1

2

```
//a/b
```

Document order

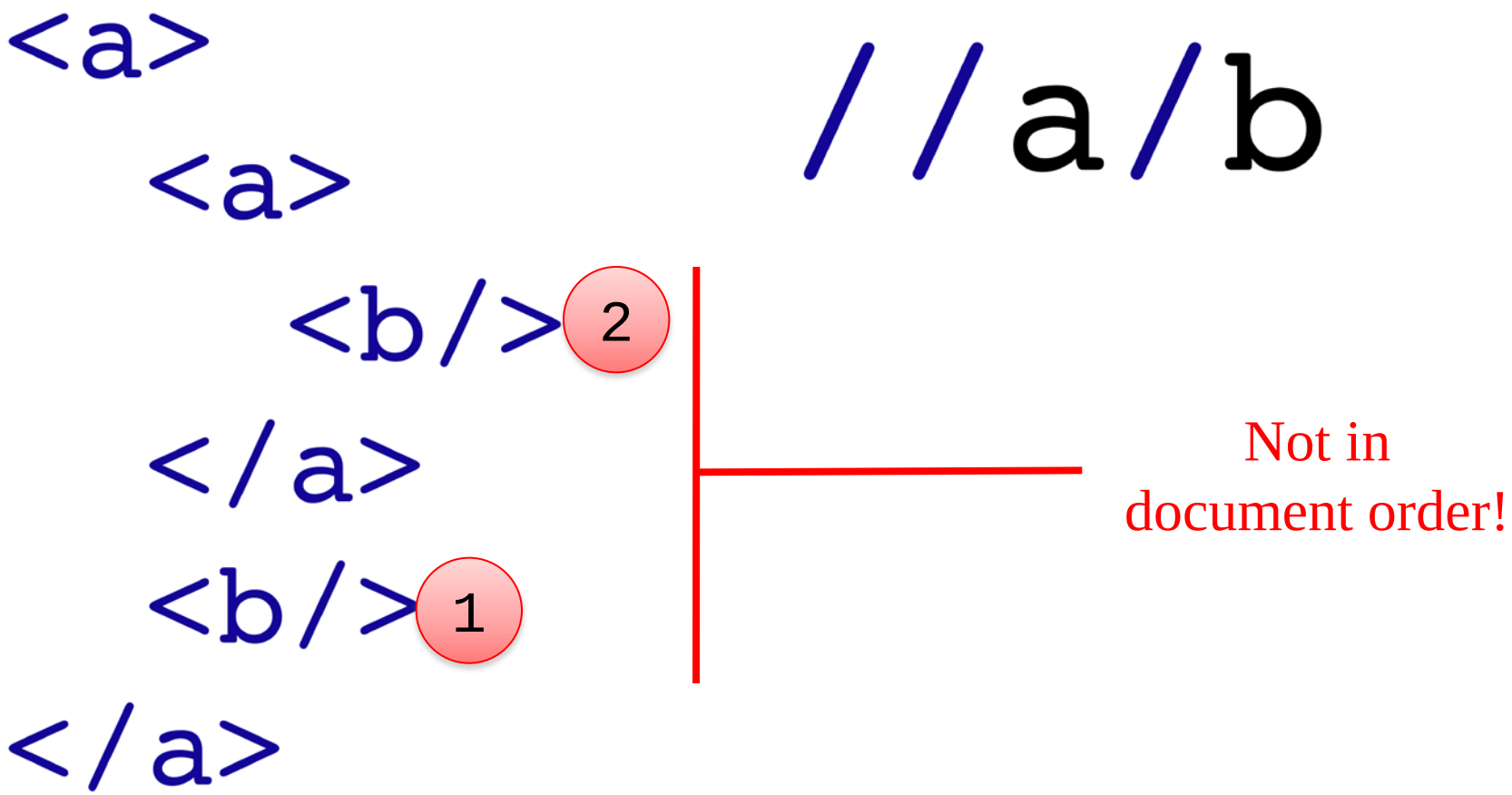
```
<a>
    <a>
        <b/>
    </a>
    <b/>
</a>
```

2

1

```
//a/b
```

Document order



On an actual engine

```
<a>
   <a>
      <b id="1"/>
   </a>
   <b id="2"/>
</a>
```

```
//a/b
```

```
<b id="1"/>
<b id="2"/>
```

In document order!

Value Comparison

```
substring("Hello World", 7) = "World"
```

```
substring("Hello World", 7) eq "World"
```

Value Comparison

```
substring("Hello World", 7) = "World"
```

```
substring("Hello World", 7) eq "World"
```

- We are comparing
  - two one-item sequences
  - of type string (no untyped, no promotion)

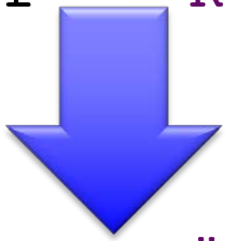
Value Comparison

```
doc('books.xml')/bib/book[author = "Kossmann"]
```

```
doc('books.xml')/bib/book[author eq "Kossmann"]
```

Value Comparison

```
doc('books.xml')/bib/book[author = "Kossmann"]
```

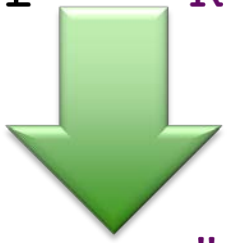


```
doc('books.xml')/bib/book[author eq "Kossmann"]
```

- Necessary conditions
  - Document validated against a schema
  - With author occuring exactly once
- Also sufficient?

Value Comparison

```
doc('books.xml')/bib/book[author = "Kossmann"]
```

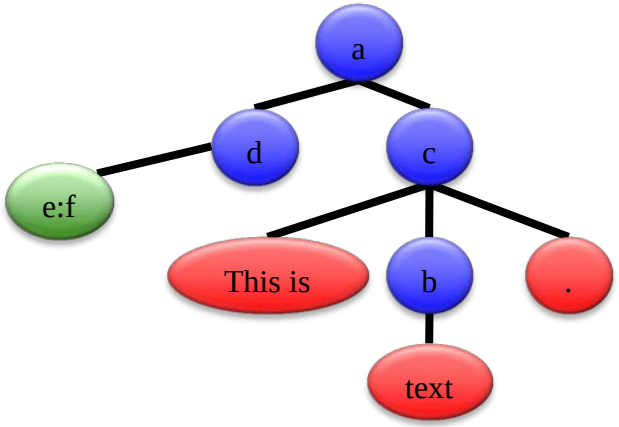


```
doc('books.xml')/bib/book[author eq "Kossmann"]
```

- Necessary conditions
  - Document validated against a schema
  - With author occuring exactly once
- Also sufficient? Yes.

Previously: XML and Data Models

Logical view
(data model)

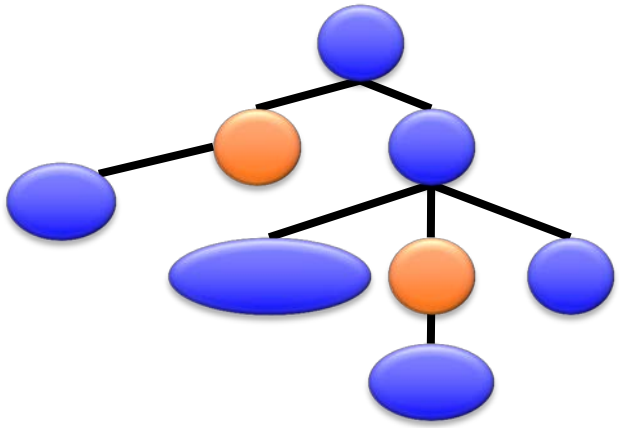


Physical view
(syntax)

```
<a>
  <d e="f"/>
  <c>This is <b>text</b>.</c>
</a>
```

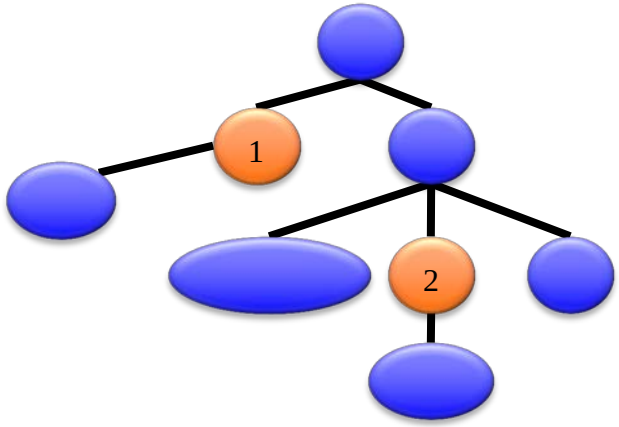
Important concepts

- Given two nodes:



Important concepts

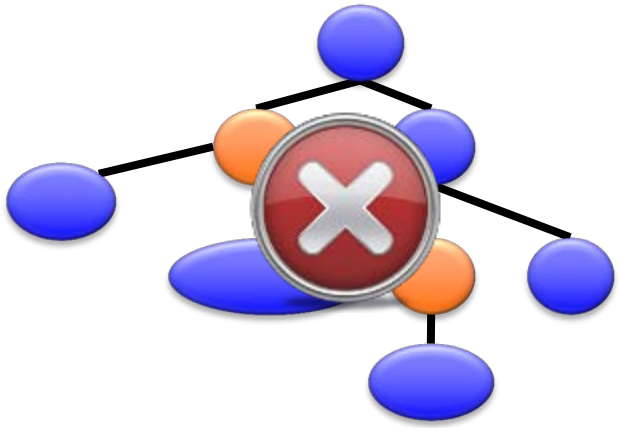
- Given two nodes:



- Document order: which one comes first?

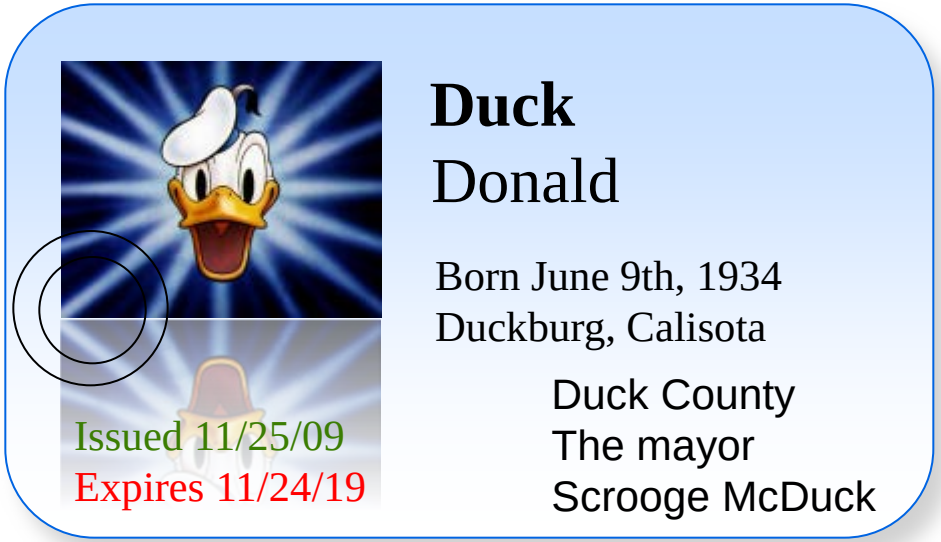
Important concepts

- Given two nodes:



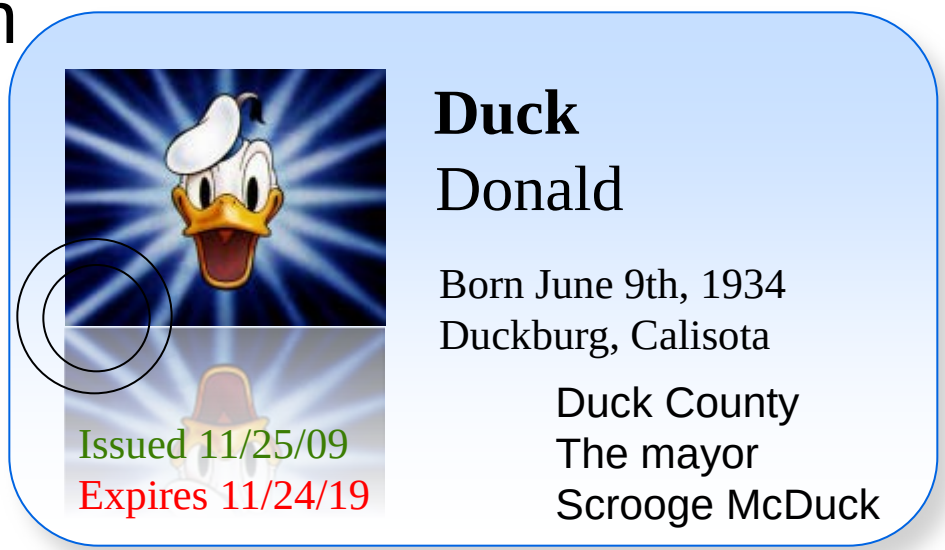
- Document order: which one comes first?
- Determine Ancestor/Descendant, Parent/Child, Sibling relationship

The solution: Node IDs



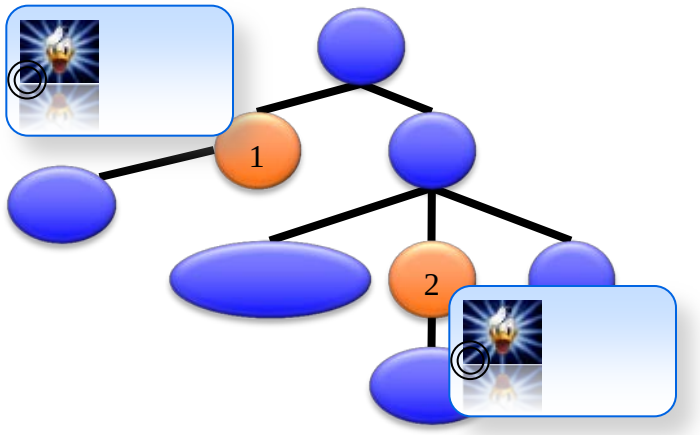
Node IDs

- Identify a node within a tree



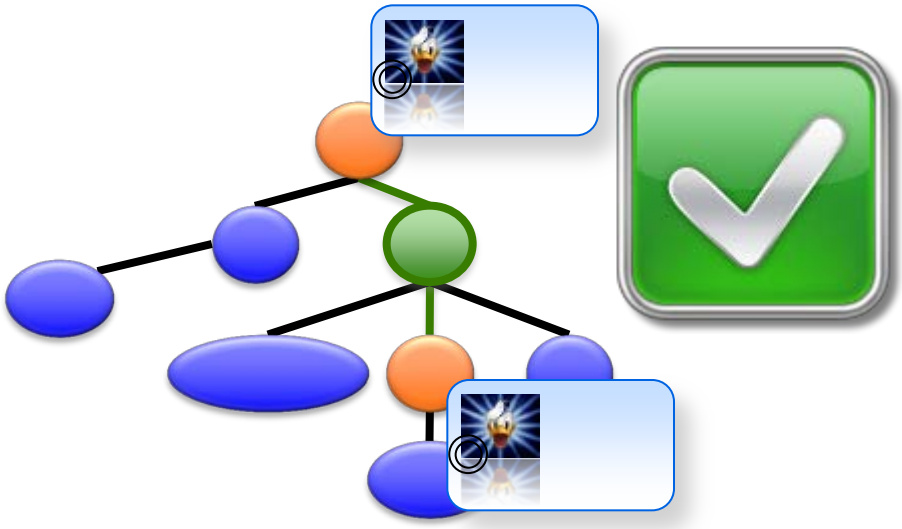
Node IDs

- Identify a node within a tree
- Allow efficient document ordering



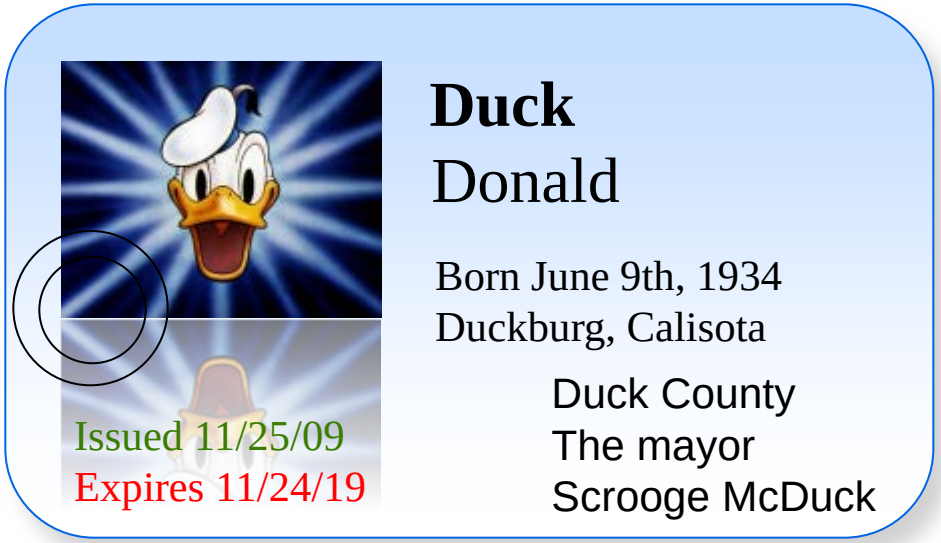
Node IDs

- Identify a node within a tree
- Allow efficient document ordering
- (Optionally) Allow efficient computation of "family" relationship



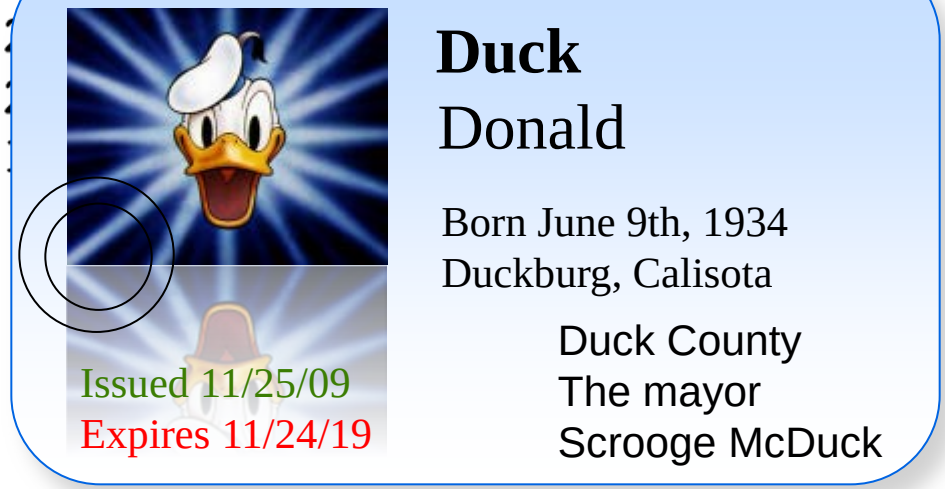
Node IDs

- Integer 7
- Double 7.07
- Dewey 1.6.2.3
- ORDPATH 1.7.1



Node IDs for this document?

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
    <Passenger>
        <name>Santa Claus</name>
        <passnumber>000112</passnumber>
        <address>Somewhere</address>
    </Passenger>
    <Reservation>
        <date>2006-12-
        <flightRef>LX1
        <passRef>00011
    </Reservation>
</doc>
```



Integer IDs

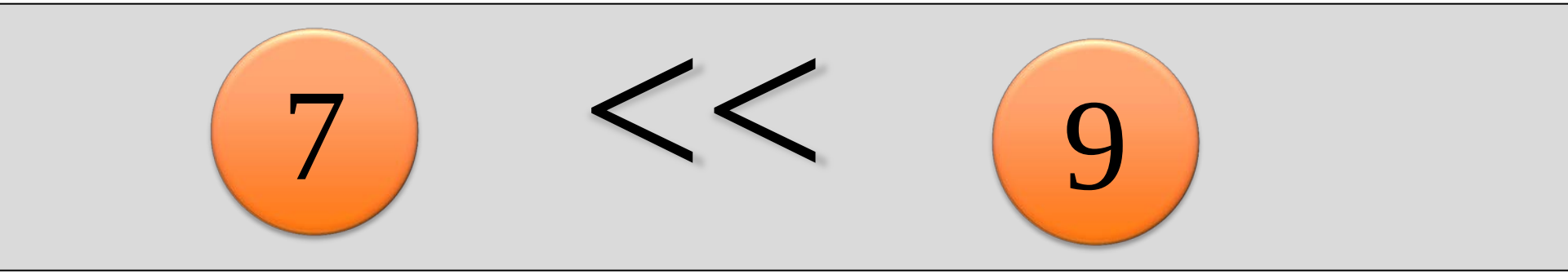
```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
    <Passenger>
        <name>Santa Claus</name>
        <passnumber>000112</passnumber>
        <address>Somewhere</address>
    </Passenger>
    <Reservation>
        <date>2006-12-24</date>
        <flightRef>LX124</flightRef>
        <passRef>000111</passRef>
    </Reservation>
</doc>
```

Integer IDs

```
<?xml version="1.0" encoding="UTF-8"?>
[1]<doc>
     [2]<Passenger>
          [3]<name>[4]Santa Claus</name>
          [5]<passnumber>[6]000112</passnumber>
          [7]<address>[8]Somewhere</address>
     </Passenger>
     [9]<Reservation>
          [10]<date>[11]2006-12-24</date>
          [12]<flightRef>[13]LX124</flightRef>
          [14]<passRef>[15]000111</passRef>
     </Reservation>
</doc>
```

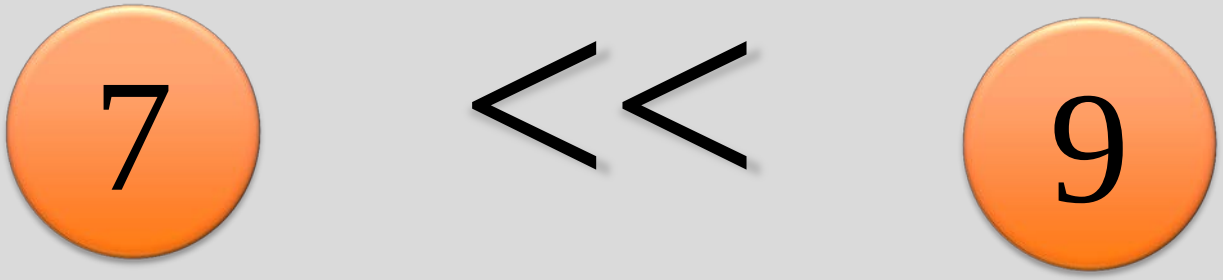
Integer IDs allow:

- Document ordering



Integer IDs allow:

- Document ordering



Comparing the integers gives you the document order!

Double IDs

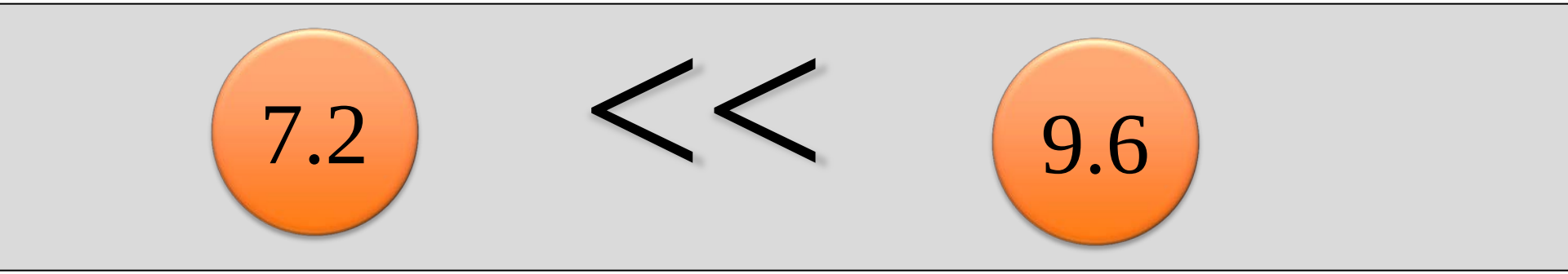
```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
    <Passenger>
        <name>Santa Claus</name>
        <passnumber>000112</passnumber>
        <address>Somewhere</address>
    </Passenger>
    <Reservation>
        <date>2006-12-24</date>
        <flightRef>LX124</flightRef>
        <passRef>000111</passRef>
    </Reservation>
</doc>
```

Double IDs

```
<?xml version="1.0" encoding="UTF-8"?>
[1.0]<doc>
    [2.0]<Passenger>
        [3.0]<name>[4.0]Santa Claus</name>
        [5.0]<passnumber>[6.0]000112</passnumber>
        [7.0]<address>[8.0]Somewhere</address>
    </Passenger>
    [9.0]<Reservation>
        [10.0]<date>[11.0]2006-12-24</date>
        [12.0]<flightRef>[13.0]LX124</flightRef>
        [14.0]<passRef>[15.0]000111</passRef>
    </Reservation>
</doc>
```

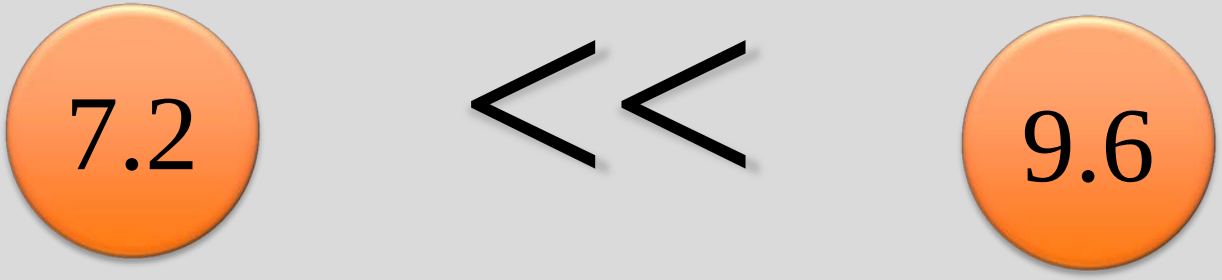
Double IDs allow:

- Document ordering



Double IDs allow:

- Document ordering



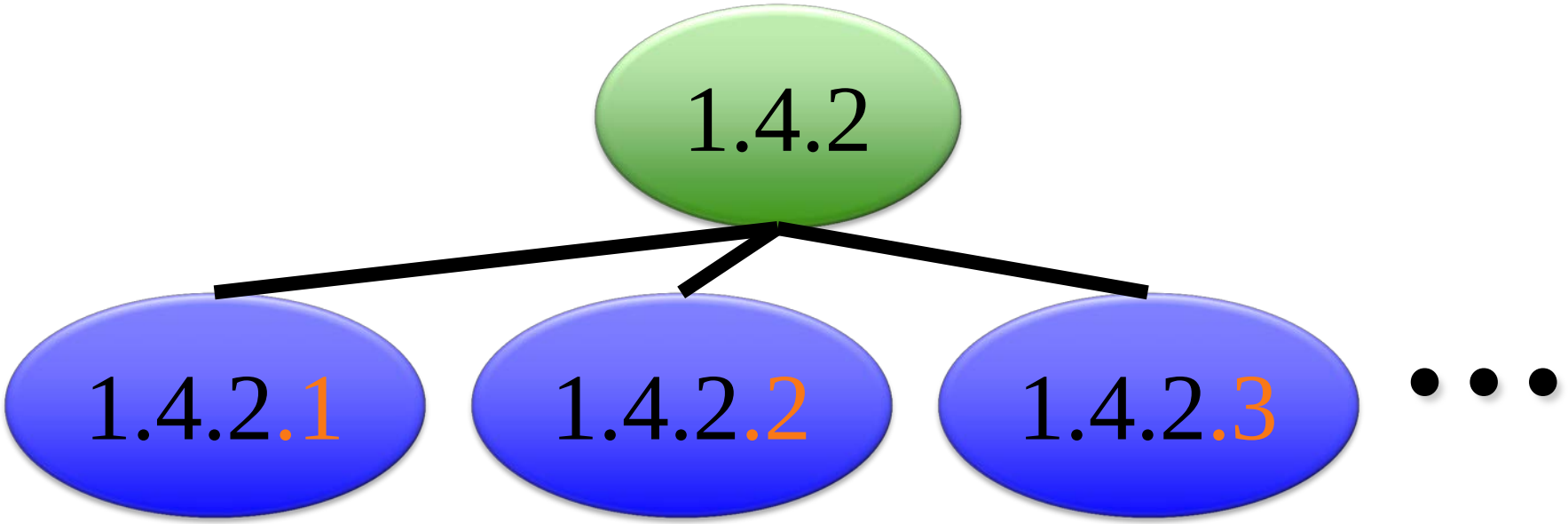
Just like integer IDs! Compare the numbers.

Dewey IDs

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
    <Passenger>
        <name>Santa Claus</name>
        <passnumber>000112</passnumber>
        <address>Somewhere</address>
    </Passenger>
    <Reservation>
        <date>2006-12-24</date>
        <flightRef>LX124</flightRef>
        <passRef>000111</passRef>
    </Reservation>
</doc>
```

Dewey IDs: Background

- Labeling the children of a node (recursively)



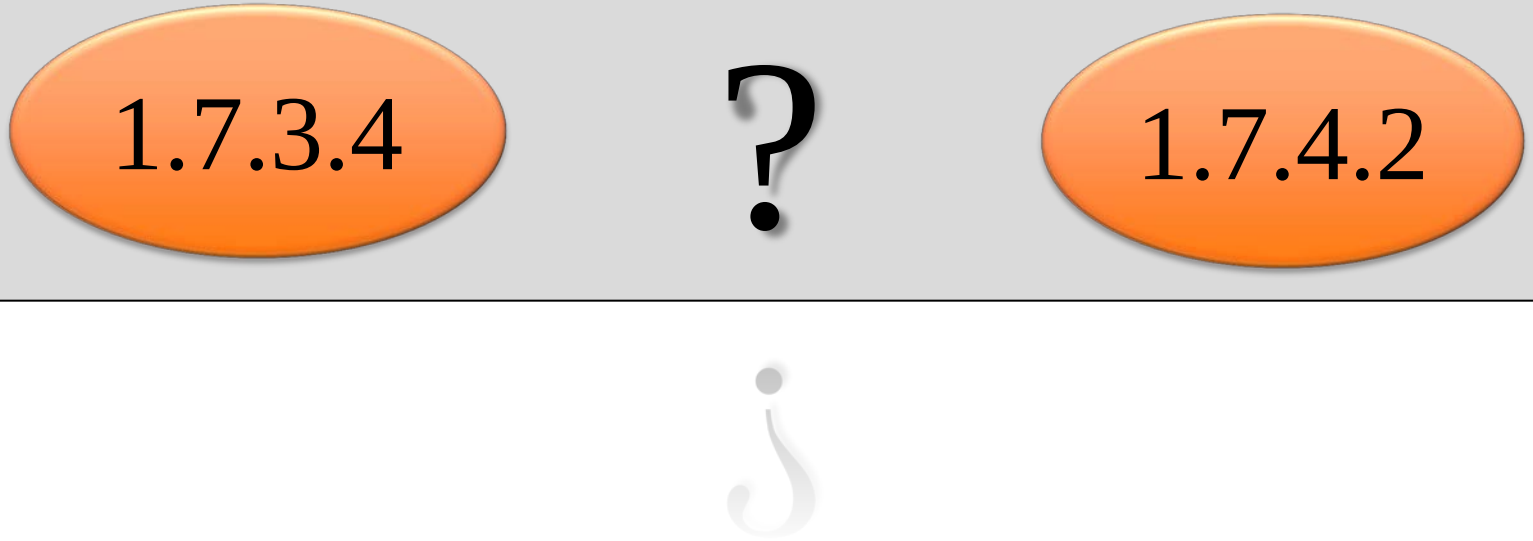
- Root is 1

Dewey IDs

```
<?xml version="1.0" encoding="UTF-8"?>
[1]<doc>
    [1.1]<Passenger>
        [1.1.1]<name>[1.1.1.1]Santa Claus</name>
        [1.1.2]<passnumber>[1.1.2.1]000112</passnumber>
        [1.1.3]<address>[1.1.3.1]Somewhere</address>
    </Passenger>
    [1.2]<Reservation>
        [1.2.1]<date>[1.2.1.1]2006-12-24</date>
        [1.2.2]<flightRef>[1.2.2.1]LX124</flightRef>
        [1.2.3]<passRef>[1.2.3.1]000111</passRef>
    </Reservation>
</doc>
```

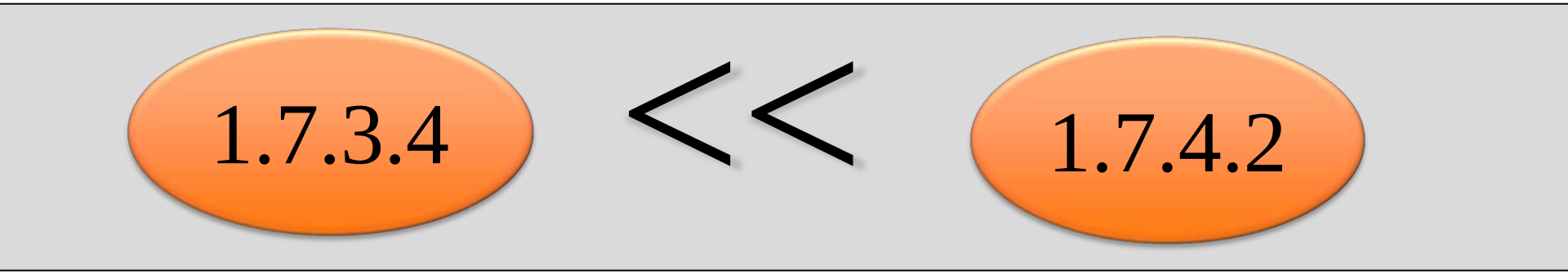
Dewey IDs allow:

- Document ordering



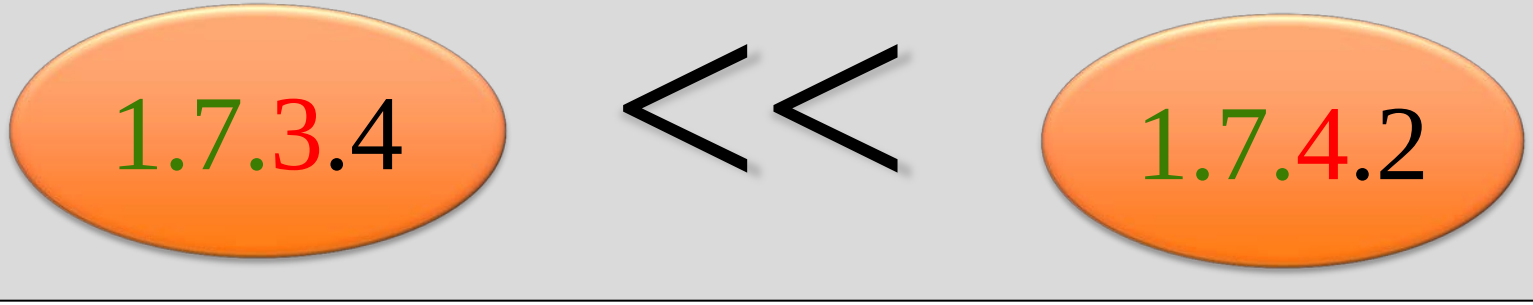
Dewey IDs allow:

- Document ordering



Dewey IDs allow:

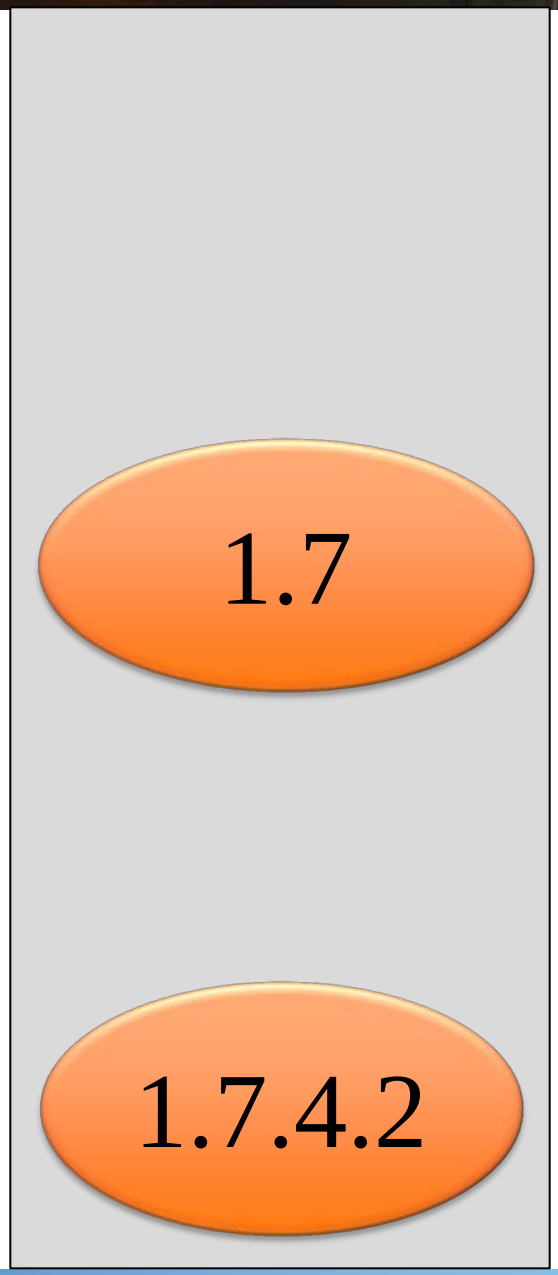
- Document ordering



Just apply lexicographical order w.r.t. the dots, then integer order.

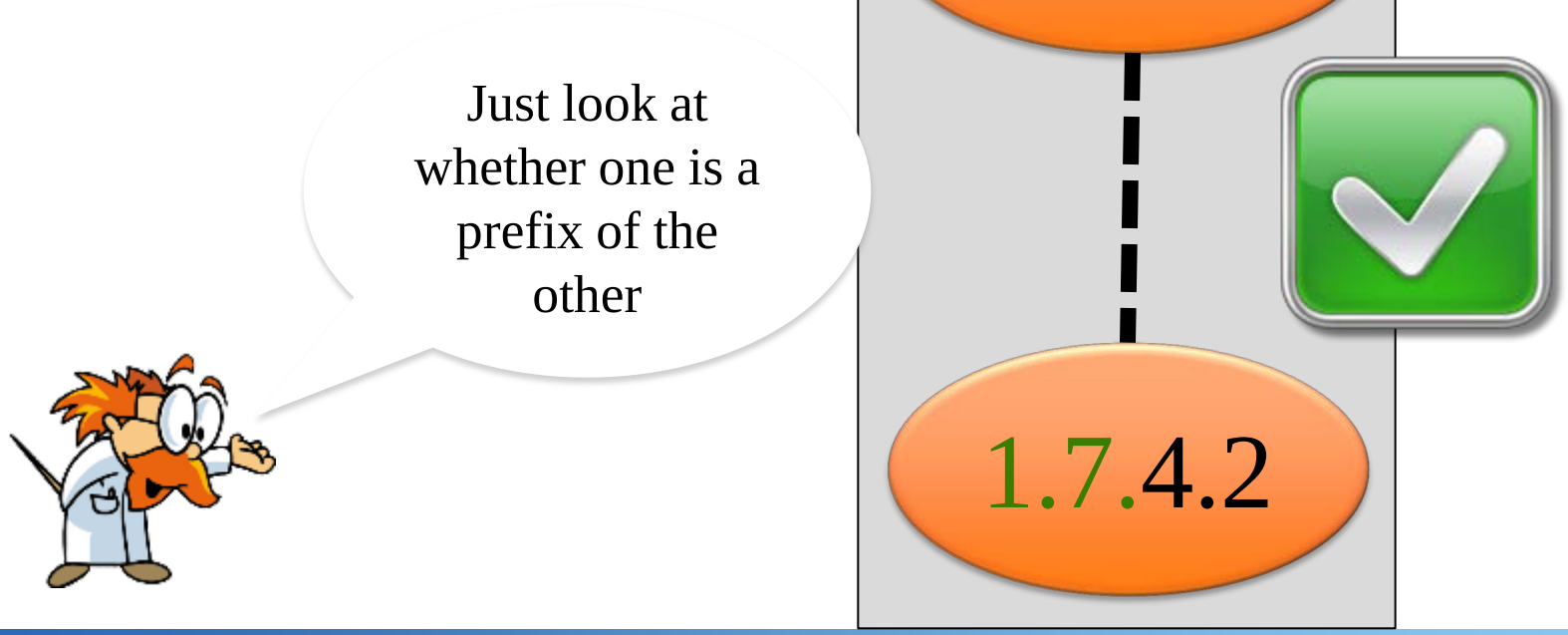
Dewey IDs allow:

- Document ordering
- Ancestor relationship



Dewey IDs allow:

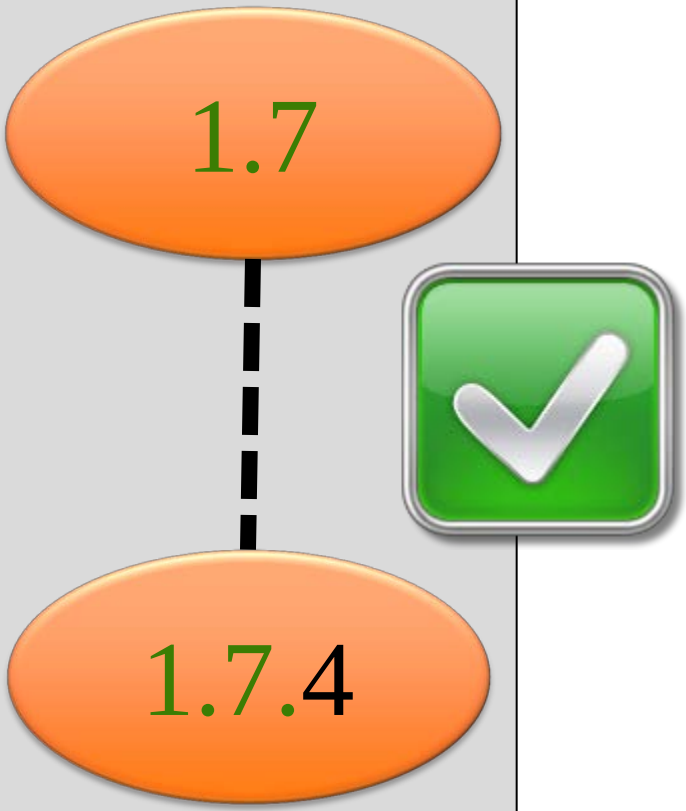
- Document ordering
- Ancestor relationship



Dewey IDs allow:

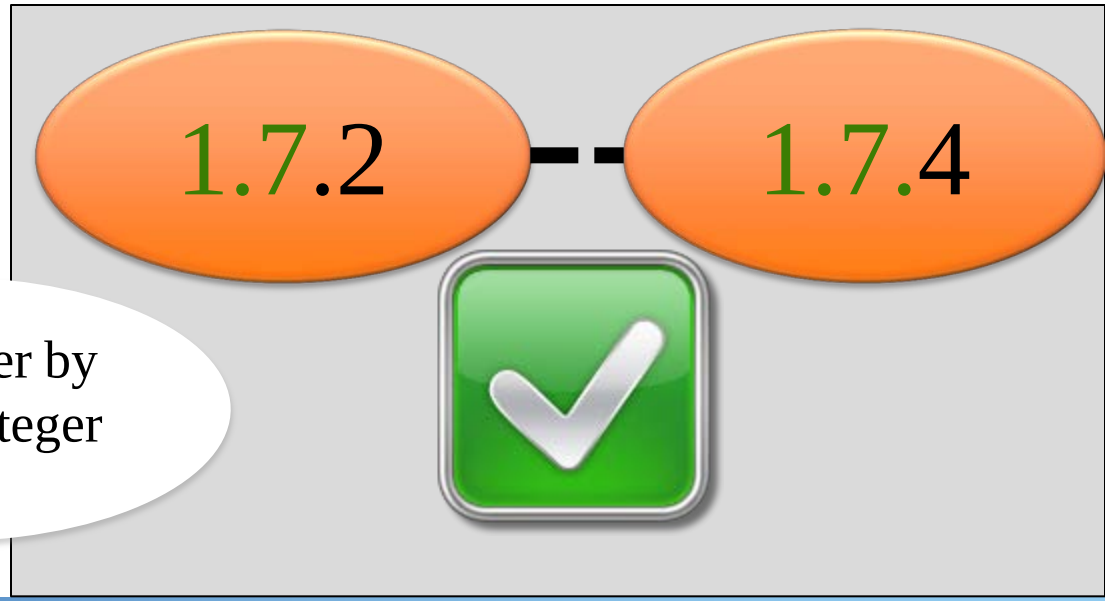
- Document ordering
- Ancestor relationship
- Parent relationship

The same, but with only one additional integer



Dewey IDs allow:

- Document ordering
- Ancestor relationship
- Parent relationship
- Sibling relationship



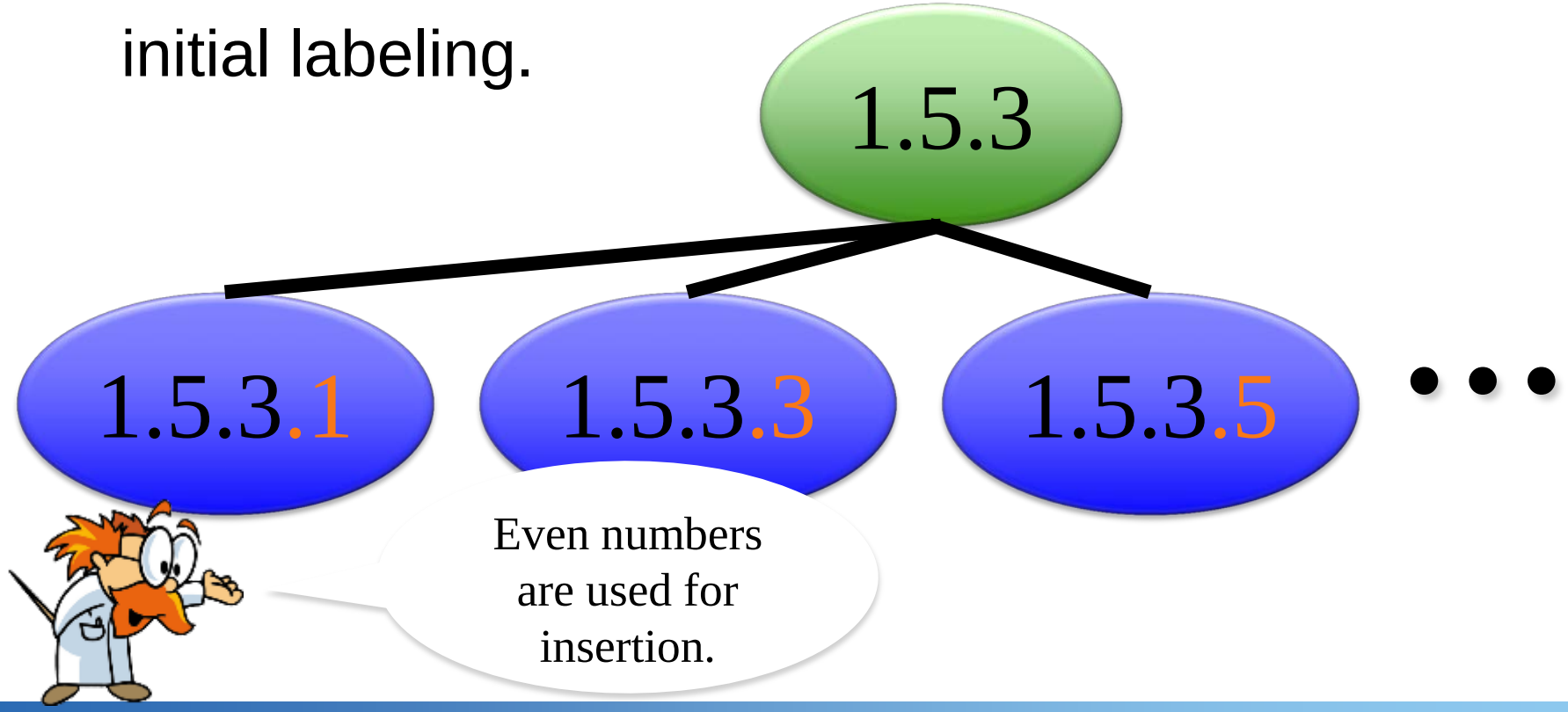
Only differ by
the last integer

ORDPATH IDs

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
    <Passenger>
        <name>Santa Claus</name>
        <passnumber>000112</passnumber>
        <address>Somewhere</address>
    </Passenger>
    <Reservation>
        <date>2006-12-24</date>
        <flightRef>LX124</flightRef>
        <passRef>000111</passRef>
    </Reservation>
</doc>
```

ORDPATH IDs: Background

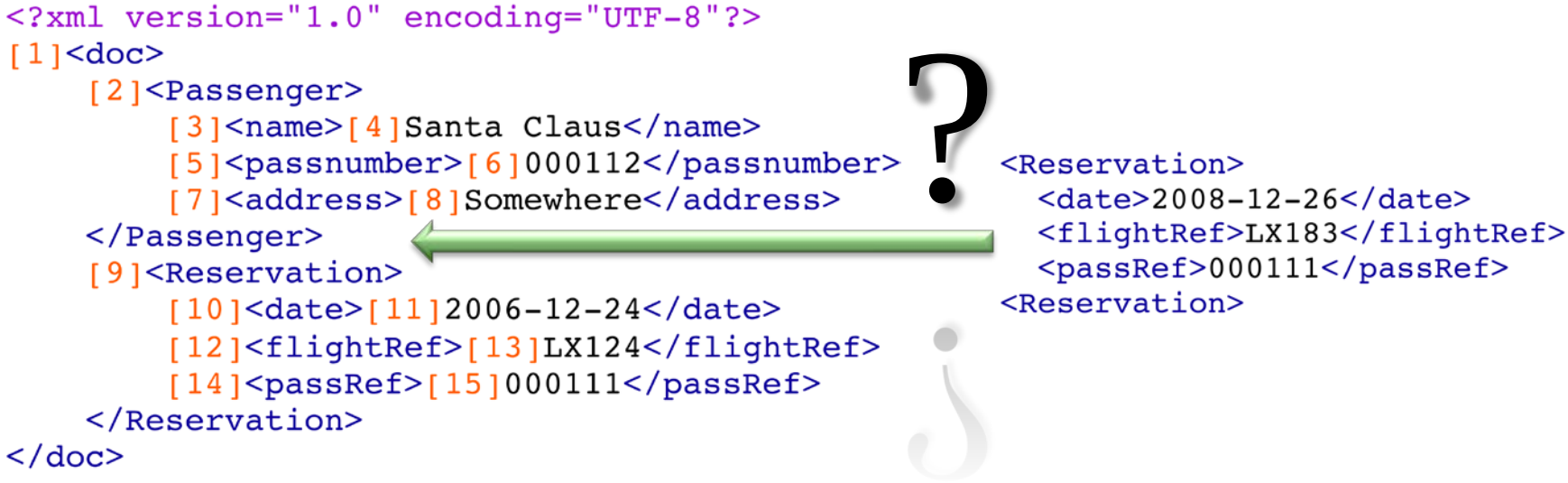
- Same as Dewey, but only with odd numbers for initial labeling.



ORDPATH IDs

```
<?xml version="1.0" encoding="UTF-8"?>
[1]<doc>
    [1.1]<Passenger>
        [1.1.1]<name>[1.1.1.1]Santa Claus</name>
        [1.1.3]<passnumber>[1.1.3.1]000112</passnumber>
        [1.1.5]<address>[1.1.5.1]Somewhere</address>
    </Passenger>
    [1.3]<Reservation>
        [1.3.1]<date>[1.3.1.1]2006-12-24</date>
        [1.3.3]<flightRef>[1.3.3.1]LX124</flightRef>
        [1.3.5]<passRef>[1.3.5.1]000111</passRef>
    </Reservation>
</doc>
```

Update: Integer IDs

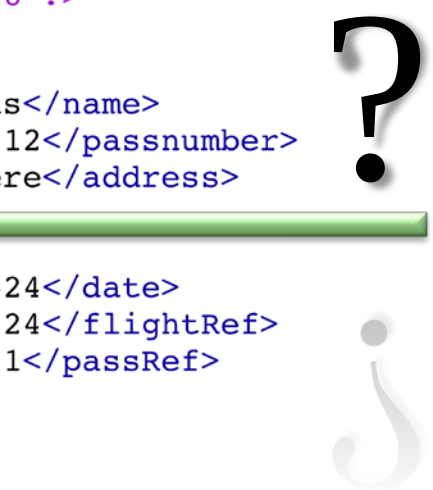


Update: Integer IDs

- Redistribute IDs
- Leave space between IDs (1001, 2001...)
- Add additional structure to maintain document order

Update: Double IDs

```
<?xml version="1.0" encoding="UTF-8"?>
[1.0]<doc>
    [2.0]<Passenger>
        [3.0]<name>[4.0]Santa Claus</name>
        [5.0]<passnumber>[6.0]000112</passnumber>
        [7.0]<address>[8.0]Somewhere</address>
    </Passenger>
    [9.0]<Reservation>
        [10.0]<date>[11.0]2006-12-24</date>
        [12.0]<flightRef>[13.0]LX124</flightRef>
        [14.0]<passRef>[15.0]000111</passRef>
    </Reservation>
</doc>
```



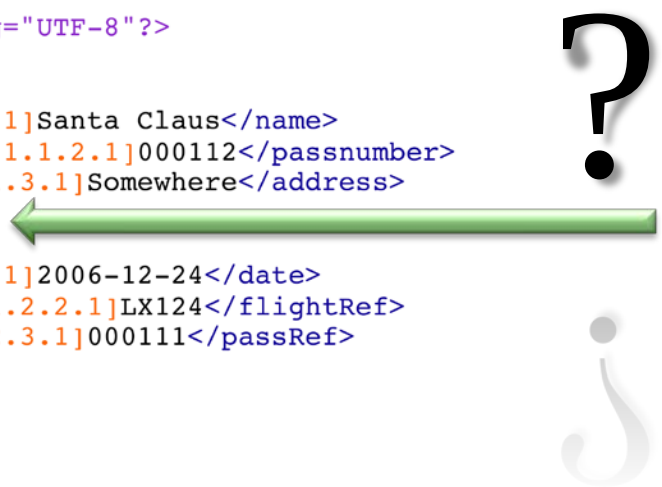
```
<Reservation>
  <date>2008-12-26</date>
  <flightRef>LX183</flightRef>
  <passRef>000111</passRef>
<Reservation>
```

Update: Double IDs

```
<?xml version="1.0" encoding="UTF-8"?>
[1.0]<doc>
    [2.0]<Passenger>
        [3.0]<name>[4.0]Santa Claus</name>
        [5.0]<passnumber>[6.0]000112</passnumber>
        [7.0]<address>[8.0]Somewhere</address>
    </Passenger>
    [8.125]<Reservation>
        [8.25]<date>[8.375]2008-12-26</date>
        [8.5]<flightRef>[8.625]LX183</flightRef>
        [8.75]<passRef>[8.875]000111</passRef>
    <Reservation>
    [9.0]<Reservation>
        [10.0]<date>[11.0]2006-12-24</date>
        [12.0]<flightRef>[13.0]LX124</flightRef>
        [14.0]<passRef>[15.0]000111</passRef>
    </Reservation>
</doc>
```

Update: Dewey IDs

```
<?xml version="1.0" encoding="UTF-8"?>
[1]<doc>
    [1.1]<Passenger>
        [1.1.1]<name>[1.1.1.1]Santa Claus</name>
        [1.1.2]<passnumber>[1.1.2.1]000112</passnumber>
        [1.1.3]<address>[1.1.3.1]Somewhere</address>
    </Passenger>
    [1.2]<Reservation>
        [1.2.1]<date>[1.2.1.1]2006-12-24</date>
        [1.2.2]<flightRef>[1.2.2.1]LX124</flightRef>
        [1.2.3]<passRef>[1.2.3.1]000111</passRef>
    </Reservation>
</doc>
```



```
<Reservation>
  <date>2008-12-26</date>
  <flightRef>LX183</flightRef>
  <passRef>000111</passRef>
<Reservation>
```

Update: Dewey IDs

```
<?xml version="1.0" encoding="UTF-8"?>
[1]<doc>
    [1.1]<Passenger>
        [1.1.1]<name>[1.1.1.1]Santa Claus</name>
        [1.1.2]<passnumber>[1.1.2.1]000112</passnumber>
        [1.1.3]<address>[1.1.3.1]Somewhere</address>
    </Passenger>
    [1.2]<Reservation>
        [1.2.1]<date>[1.2.1.1]2006-12-24</date>
        [1.2.2]<flightRef>[1.2.2.1]LX124</flightRef>
        [1.2.3]<passRef>[1.2.3.1]000111</passRef>
    </Reservation>
</doc>
```

?

```
<Reservation>
  <date>2008-12-26</date>
  <flightRef>LX183</flightRef>
  <passRef>000111</passRef>
<Reservation>
```

Here, we have the same problem as for integer IDs...

Update: ORDPATH IDs

```
<?xml version="1.0" encoding="UTF-8"?>
[1]<doc>
    [1.1]<Passenger>
        [1.1.1]<name>[1.1.1.1]Santa Claus</name>
        [1.1.3]<passnumber>[1.1.3.1]000112</passnumber>
        [1.1.5]<address>[1.1.5.1]Somewhere</address>
    </Passenger>
    [1.3]<Reservation>
        [1.3.1]<date>[1.3.1.1]2006-12-24</date>
        [1.3.3]<flightRef>[1.3.3.1]LX124</flightRef>
        [1.3.5]<passRef>[1.3.5.1]000111</passRef>
    </Reservation>
</doc>
```

?

```
<Reservation>
  <date>2008-12-26</date>
  <flightRef>LX183</flightRef>
  <passRef>000111</passRef>
<Reservation>
```

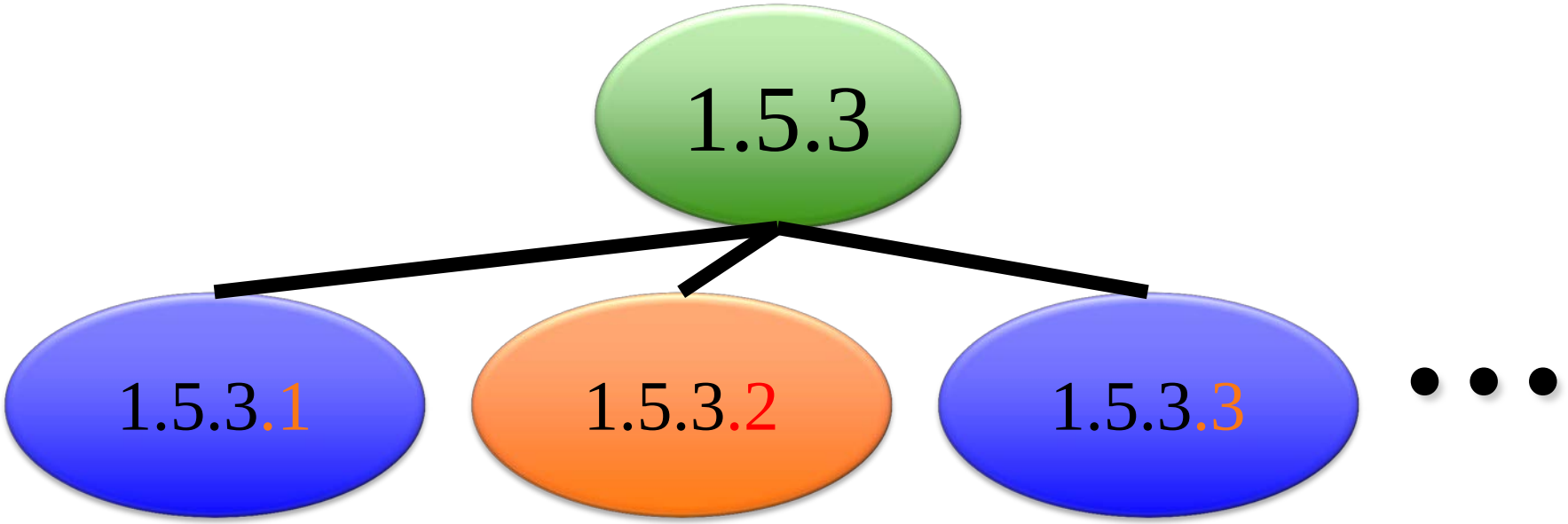
And ORDPATH solves this problem. How?

ORDPATH IDs: Background

- Use even numbers to insert!

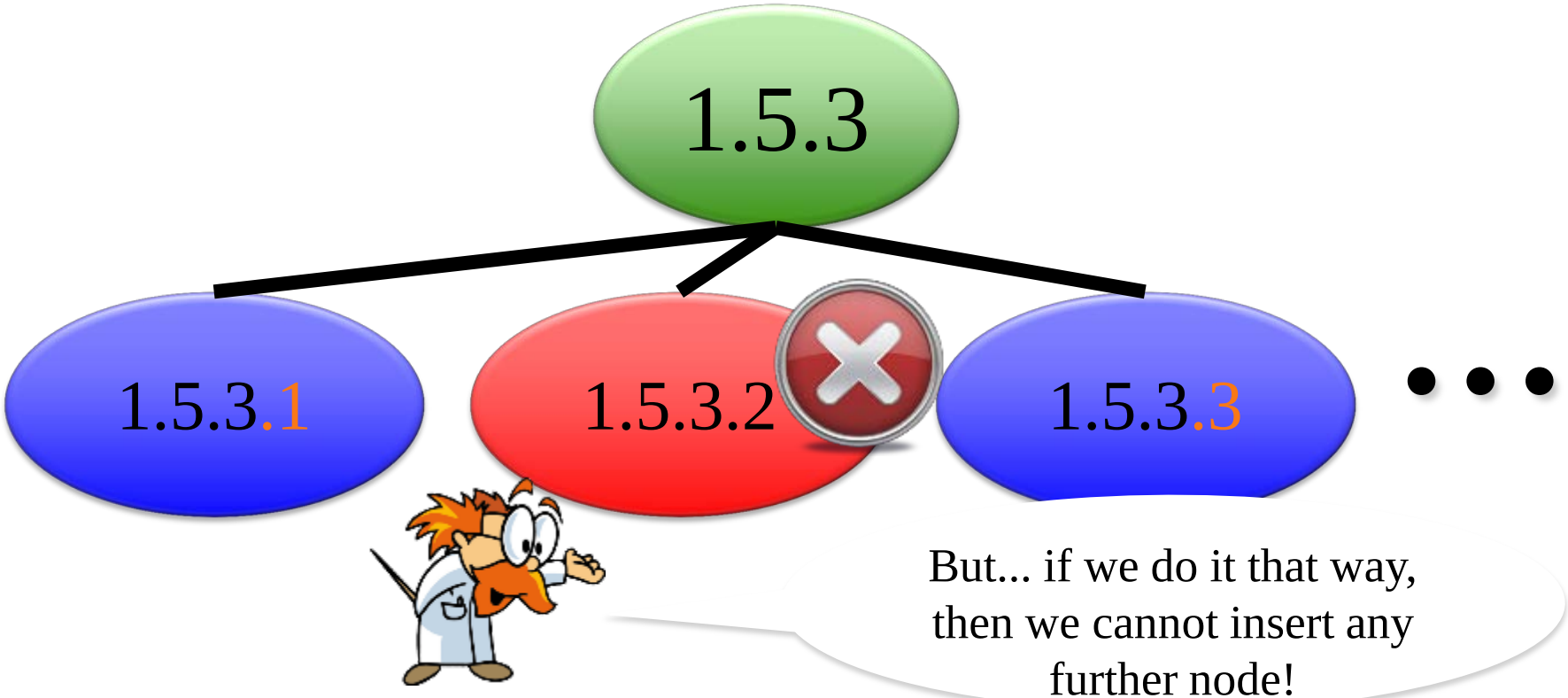
ORDPATH IDs: The wrong solution

- Use even numbers to insert!



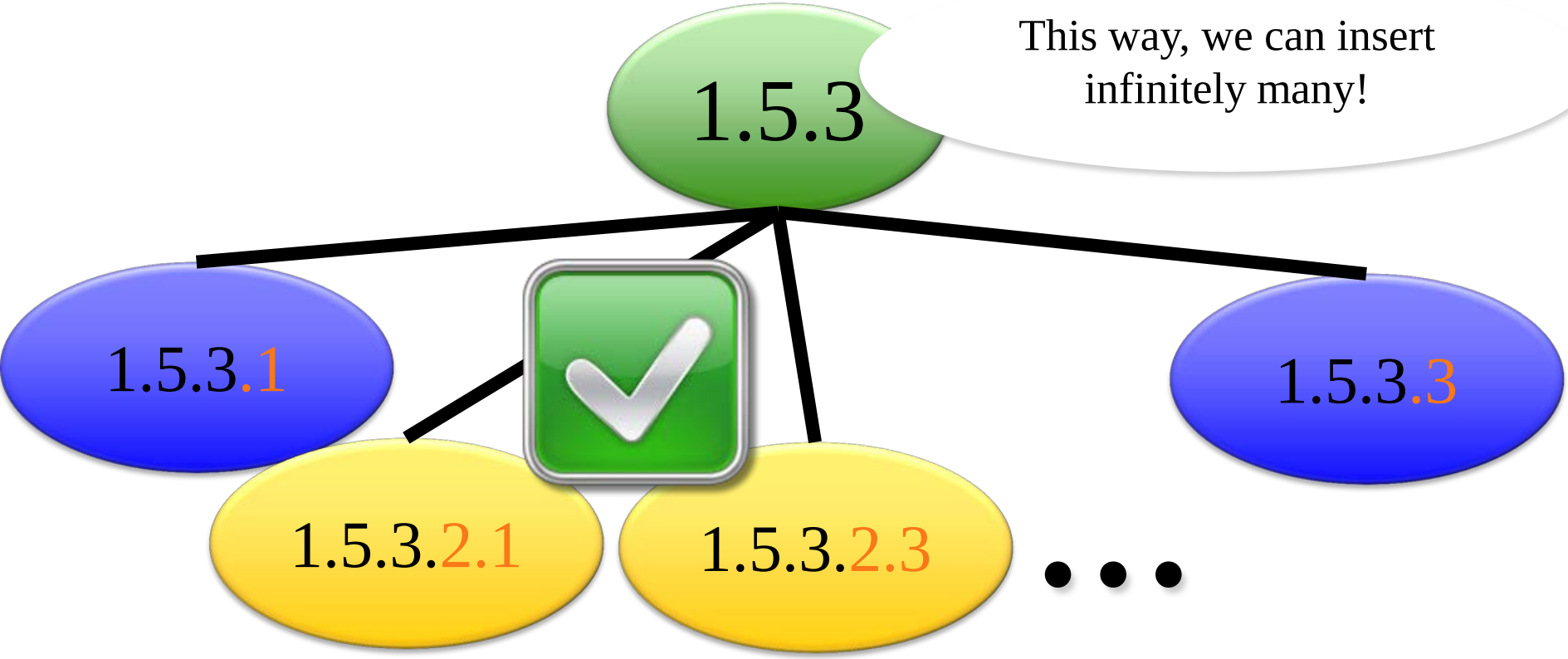
ORDPATH IDs: The wrong solution

- Use even numbers to insert!



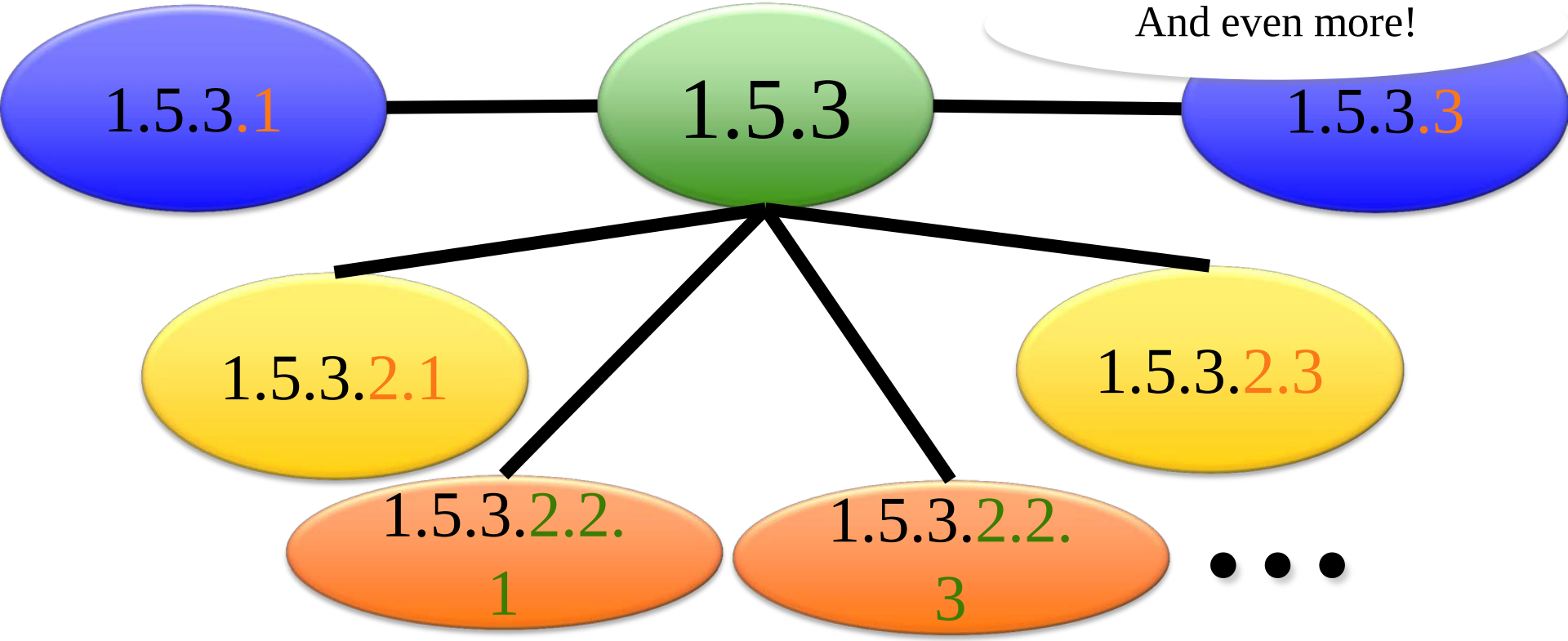
ORDPATH IDs: Background

- Use even numbers to insert!



ORDPATH IDs: Background

- Use even numbers to insert!



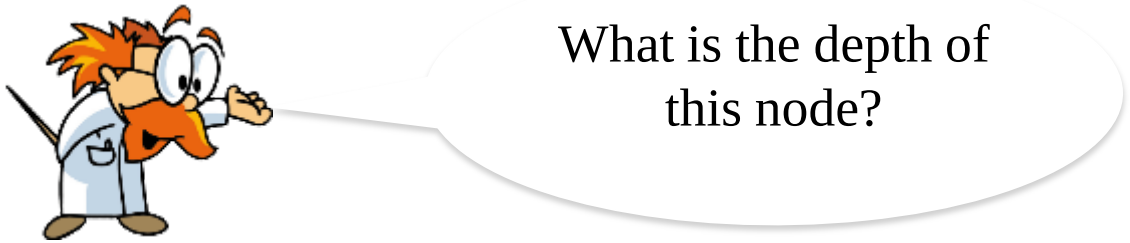
Update: ORDPATH IDs

```
<?xml version="1.0" encoding="UTF-8"?>
[1]<doc>
    [1.1]<Passenger>
        [1.1.1]<name>[1.1.1.1]Santa Claus</name>
        [1.1.3]<passnumber>[1.1.3.1]000112</passnumber>
        [1.1.5]<address>[1.1.5.1]Somewhere</address>
    </Passenger>
    [1.2.1]<Reservation>
        [1.2.1.1]<date>[1.2.1.1.1]2008-12-26</date>
        [1.2.1.3]<flightRef>[1.2.1.3.1]LX183</flightRef>
        [1.2.1.5]<passRef>[1.2.1.5.1]000111</passRef>
    </Reservation>
    [1.3]<Reservation>
        [1.3.1]<date>[1.3.1.1]2006-12-24</date>
        [1.3.3]<flightRef>[1.3.3.1]LX124</flightRef>
        [1.3.5]<passRef>[1.3.5.1]000111</passRef>
    </Reservation>
</doc>
```

ORDPATH IDs: Background

- Only odd numbers count as a level

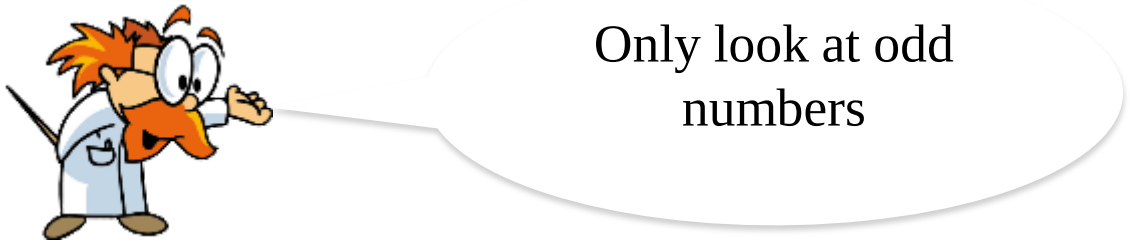
1.2.1.3.4.2.5.6.7



ORDPATH IDs: Background

- Only odd numbers count as a level

1.2.1.3.4.2.5.6.7



ORDPATH IDs: Background

- Only odd numbers count as a level

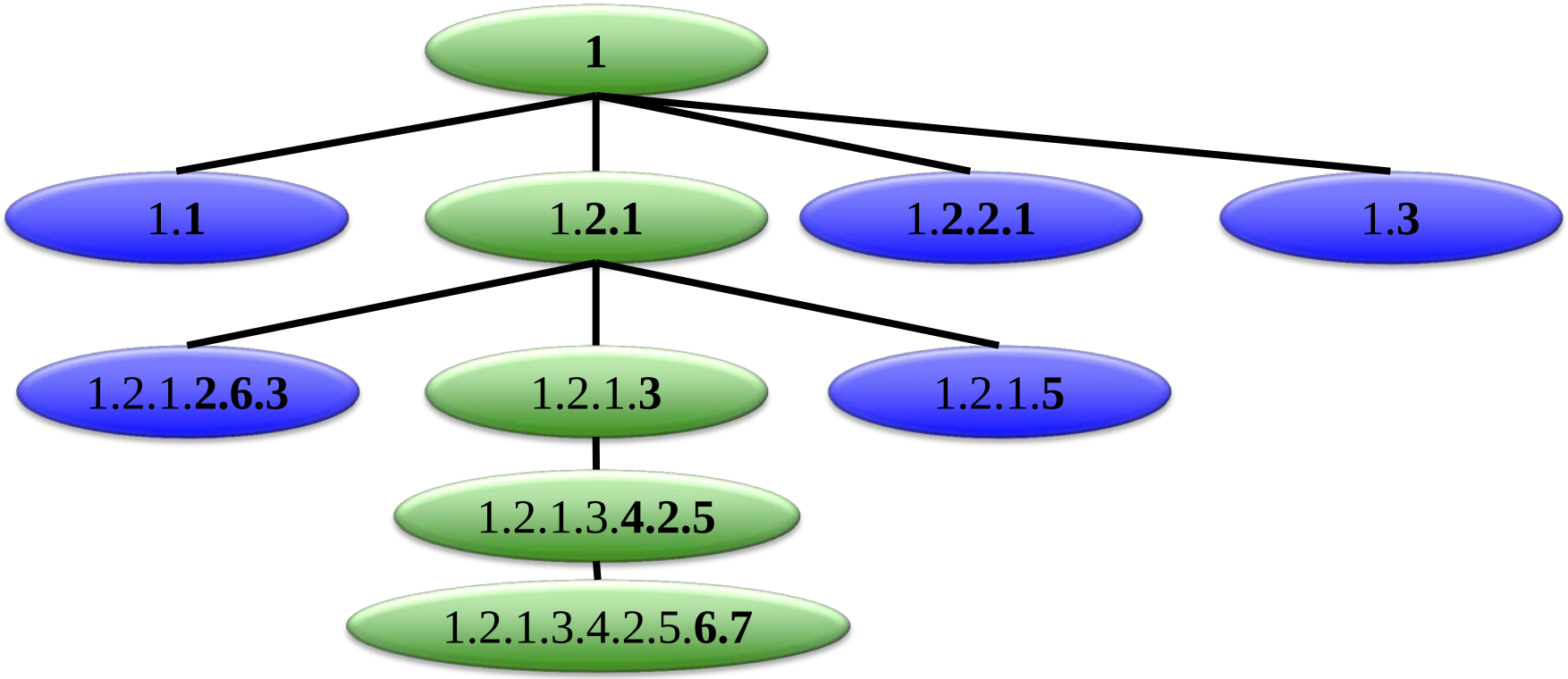
1.2.1.3.4.2.5.6.7

root depth 1 depth 2 depth 3 depth 4

However, the algorithms for computing document order and ancestor/parent relationship are the same as Dewey!

ORDPATH IDs: Background

- Only odd numbers count as a level



Why ORDPATH is very smart – and efficient

Why ORDPATH is very smart – and efficient

- IDs can be expressed as bitstrings (prefix-free encoding for integers).

```
0101100101101011011001101010101011110110101101101
```

Why ORDPATH is very smart – and efficient

- IDs can be expressed as bitstrings (prefix-free encoding for integers).
- Document ordering is just a bitstring comparison!

```
0101100101101011011001101010001001101010101010110110
0101100101101011011001101010101011110110101101011011
```

Why ORDPATH is very smart – and efficient

- IDs can be expressed as bitstrings (prefix-free encoding for integers).
- Document ordering is just a bitstring comparison!
- Ancestor relationship is determined by testing if one bitstring is a prefix of the other.

010110010110101101100110101010111101101011010110

IDs: Wrap-up

	Size	Doc order	Ancestor/Descendant	Sibling/Before/After node (navigation)	Insert	Computation intensive
Integer IDs						
double IDs						
Dewey IDs						
ORDPATH IDs						

IDs: Wrap-up

	Size	Doc order	Ancestor/Descendant	Sibling/Before/After node (navigation)	Insert	Computation intensive
Integer IDs	+	+				+
double IDs	+	+			+	+
Dewey IDs		+	+	+		
ORDPATH IDs		+	+	+	+	

Hope to see you next week!