Enterprise Programmering 1
Lesson 09: Selenium and JaCoCo

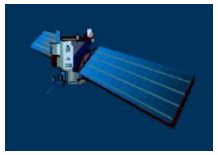
Dr. Andrea Arcuri
Westerdals Oslo ACT
University of Luxembourg

About these slides

- These slides are just high level overviews of the topics covered in class
- The details are directly in the code comments on the Git repository

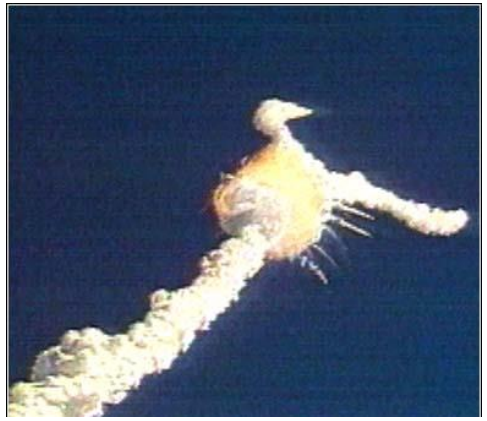
Software Testing

Software is everywhere!!! (not just enterprise systems…)



Are software applications doing what are they supposed to do?

Ariane 5 – ESA



On June 4, 1996, the flight of the Ariane 5 launcher ended in a failure.

$500 millions in cost

Software bug

F-18 crash

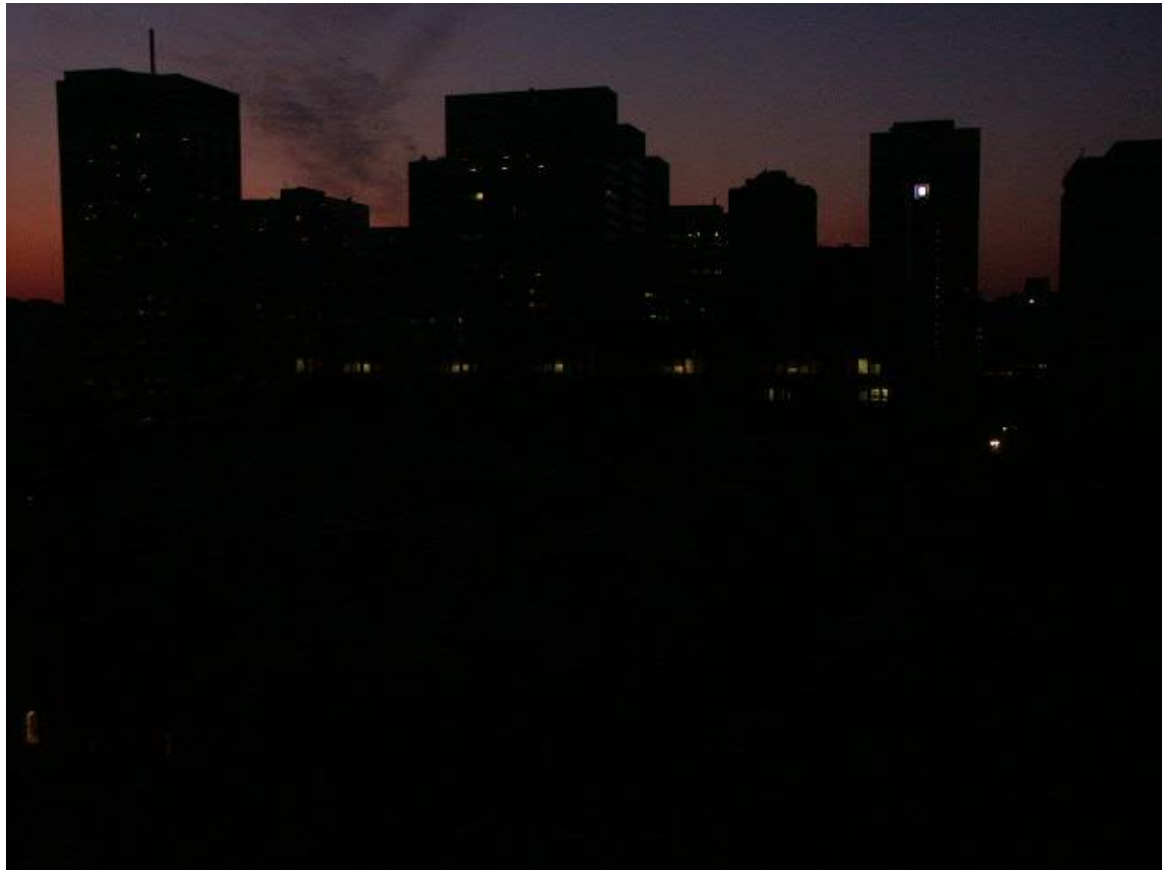
An F-18 crashed because of a missing exception condition, thought not possible…

Fatal Therac-25 Radiation

1986, Texas, person died

Power Shutdown in 2003

Nearly 50 million people affected in Canada/US



2010, Toyota, bug in braking system, recalled 436,000 vehicles…

Knight Capital Group 2012

$460 millions lost in 45 minutes of trading due to bug

And I could go on the whole day…

- As of 2013, estimated that software testing costing **$312 billions** worldwide
- In 2016, 548 recorded and documented software failures impacted **4.4 billion** people and **$1.1 trillion** in assets worldwide

But what about every-day life in Oslo???

Why??? A common problem… no tests!

Software Testing

- Software has *bugs,* ie implementation mistakes
- All developers write bugs, not just students, even senior ones
  - eg, in the writing of the code of this course there were (and might still be) several bugs, which you can check by looking at the commit logs in the Git history
- Testing helps you to find bugs before it is delivered to customers
- However, testing does NOT guarantee the absence of bugs…

Manual Testing

- Start the application you are developing
- Use it, and manually check if it is working correctly
  - eg, should not crash and should get expected behavior
- Try to follow common usage scenarios for the application
- Also try uncommon scenarios, to check if still working correctly
- *Software Tester*: an employee whose work is to test the application before each new release/feature

Problems of Manual Testing

- Many people finds it *boring…*
- It is not *systematic:* a manual procedure can miss/forget to test important scenarios and edge cases
- It is *expensive*: software testers still need a salary to feed their families…
- Every time there is a code change, new bugs might be introduced, and previous testing is invalidated
  - ie, have to do it again, and again, and again, and …
- Still, you need to have a manual testing phase (eg before a major release to customers), but you also need something more besides manual testing

Beta Testing

- A type of manual testing you might be familiar with, especially in regards to video-games
- Release the software in a "*beta*" state to a group of potential users, which will test it for *free*!
- Very common in online games, especially to test large loads on servers before a main release

Testing Depends on Context

- If there is a bug in a video-game, players will just have a to wait for a *patch*…
  - although if too many bugs at launch, it will hurt sales…
- If there is a bug in an *enterprise* system dealing with critical assets (e.g., a bank), you might be screwed…

Automated Testing

- Write code that can automatically run tests against an application, or parts of it (e.g., single functions or classes)
- *Benefits*:
  - you can tests single parts of the application
  - *can re-execute all tests every time there is a code change to see if these new changes break current functionalities*

Kinds of Testing

- *Unit Testing*
  - test units (functions/classes) in isolation
- *Integration Testing*
  - scenarios in which different components are involved
- *System Testing*
  - test the *whole* application, going through the same interfaces (e.g., a GUI) that a user would use to interact with it
- But there are many others
  - User Testing, Security Testing, Performance Testing, Robustness Testing, etc.

Running Tests in Java

- The main framework is *JUnit*
  - others are like for example *TestNG* and *Spock*
- Not just for unit testing, but any kind of testing
  - eg, the Arquilliain and Spring service tests can be considered as *integration tests*

Maven and Tests

- Maven can run tests as part of the build
- It makes distinctions between *unit* and *integration* tests
- But these are JUST names, and NOT necessarily related to the concepts of unit and integration testing
  - ie, could run unit tests in Maven like they were integration ones
- *unit*: run by Surefire plugin, name pattern “*Test.java”, executed BEFORE the “mvn package” phase (jar/war file not built yet)
- *integration*: run by Failsafe plugin, name pattern “*IT.java”, executed AFTER “mvn package” (so can use the built JAR/WAR file)

When to use “*IT.java”

- If your tests need to use the generated JAR/WAR file
  - eg, you want to start the actual built application
- If your tests are *long* to execute (e.g., system tests), and you do not want to run them each time you just need to do a “*mvn package*”
  - eg, you just want to quickly build the application for some manual testing

Selenium

Selenium

- Tool that enables you to interact and control a browser from code
- Written in Java, but used as a tool in many different languages, eg C#/.Net
- Main use: ability to write *system tests* for web applications, in which you first start the server, and then control the browser (eg click buttons) like it was a real user

Selenium Requirements

- In Java, imported as a JAR library
- Can be called directly from JUnit
- **MUST** have a browser (e.g., Chrome) AND the Selenium *drivers* for such browser (which you need to install separately)
  - Note: can also use Docker to run an image with a browser and drivers already configured… but then it is more difficult to debug, as no simple direct access to the browser GUI running inside the Docker image
- Note: can run same tests against different browsers

HTML Interactions

- For Selenium to interact with the HTML components, it needs to *locate* them first
- *Easiest* way is to add “id” attributes to the HTML tags we want to click on or type inputs in
- At times though, we need more sophisticate *queries*
  - eg, check how many rows there are in a table, or how many tags have a specific attribute
  - Needed for validation: e.g., after you click a button, you want to check that indeed a new row was added on a displayed table

XPath

- Query language on tags in HTML/XML documents
  - https://www.w3schools.com/xml/xpath_syntax.asp
- A HTML/XML document can be represented as a tree, on which tags/nodes can be identified with a *path*
- A path expression can match several tags/nodes
- An alternative to XPath is “CSS Selectors”

Just a simple example

- Can have links to the other files in webapp, like for example foo/foo.html
- But the files under WEB-INF are not accessible, e.g. WEB-INF/web.xml should return an error

To run this web application, you need to use Docker. Recall to use "-p 8080:8080" to expose the port on which Wildfly is listening.

Then, you can open this page by pointing your browser to **http://localhost:8080/base**.

Elements Console Sources Network Performance Memory »

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
▶#shadow-root (open)
▶<head>…</head>
...▼<body> == $0
    <h2>
        Just a simple example
    </h2>
  ▼<ul>
    ▼<li>
        "Can have links to the other files in webapp, like for example "
        <a href="foo/foo.html">foo/foo.html</a>
      </li>
    ▼<li>
        "But the files under WEB-INF are not accessible, e.g. "
        <a href="WEB-INF/web.xml">WEB-INF/web.xml</a>
        " should return
                an error
            "
      </li>
    </ul>
  ▶<p>…</p>
  ▶<p>…</p>
```

html body

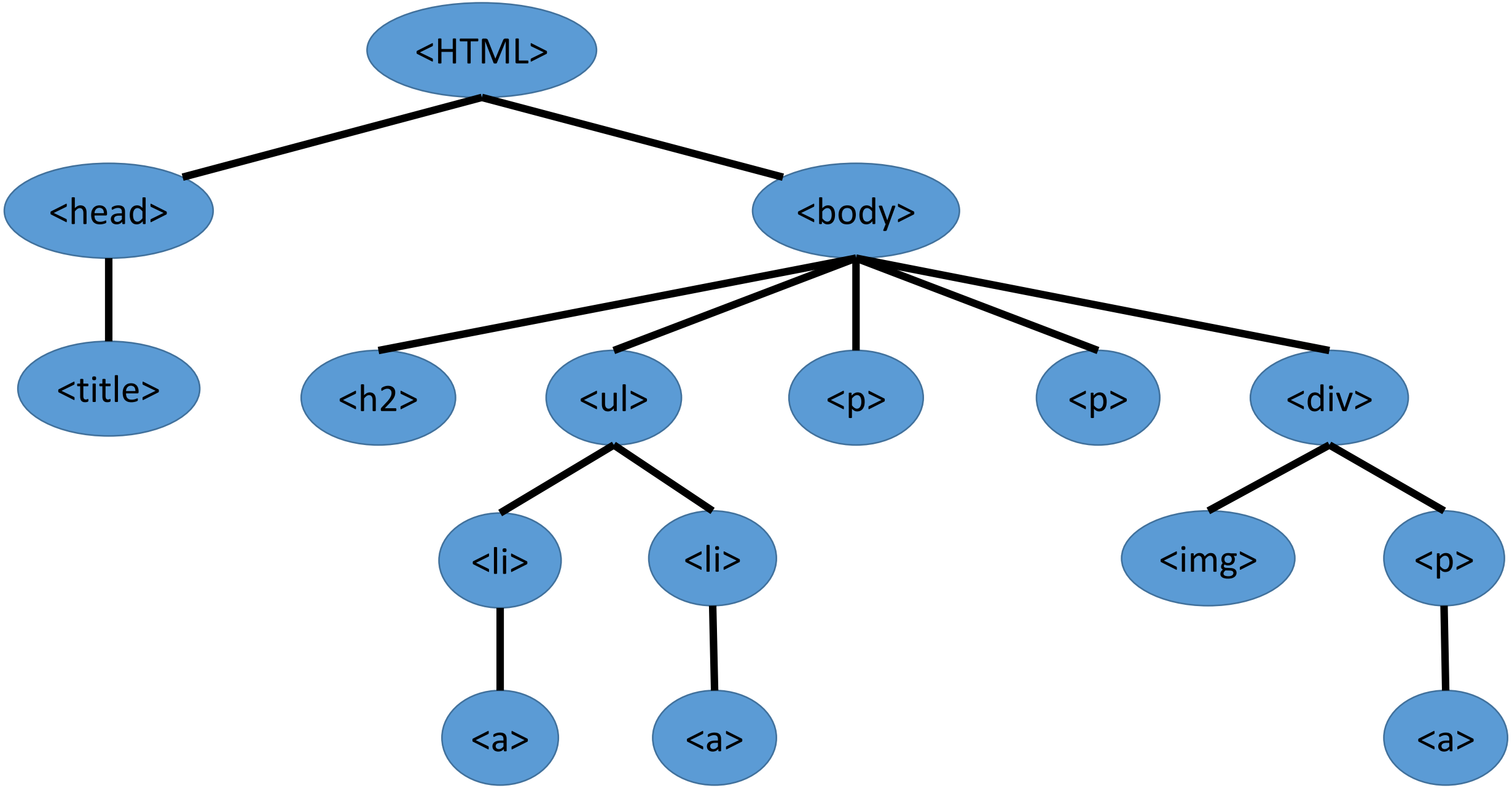
Styles Event Listeners DOM Breakpoints Properties

Filter :hov .cls +

```
element.style {
}
body {                                  user agent stylesheet
  display: block;
  margin: ▶ 8px;
}
```

Console What's New ×

Recall first example of web page we had


<HTML>
<head>
<body>
<title>
<h2>

<p>
<p>

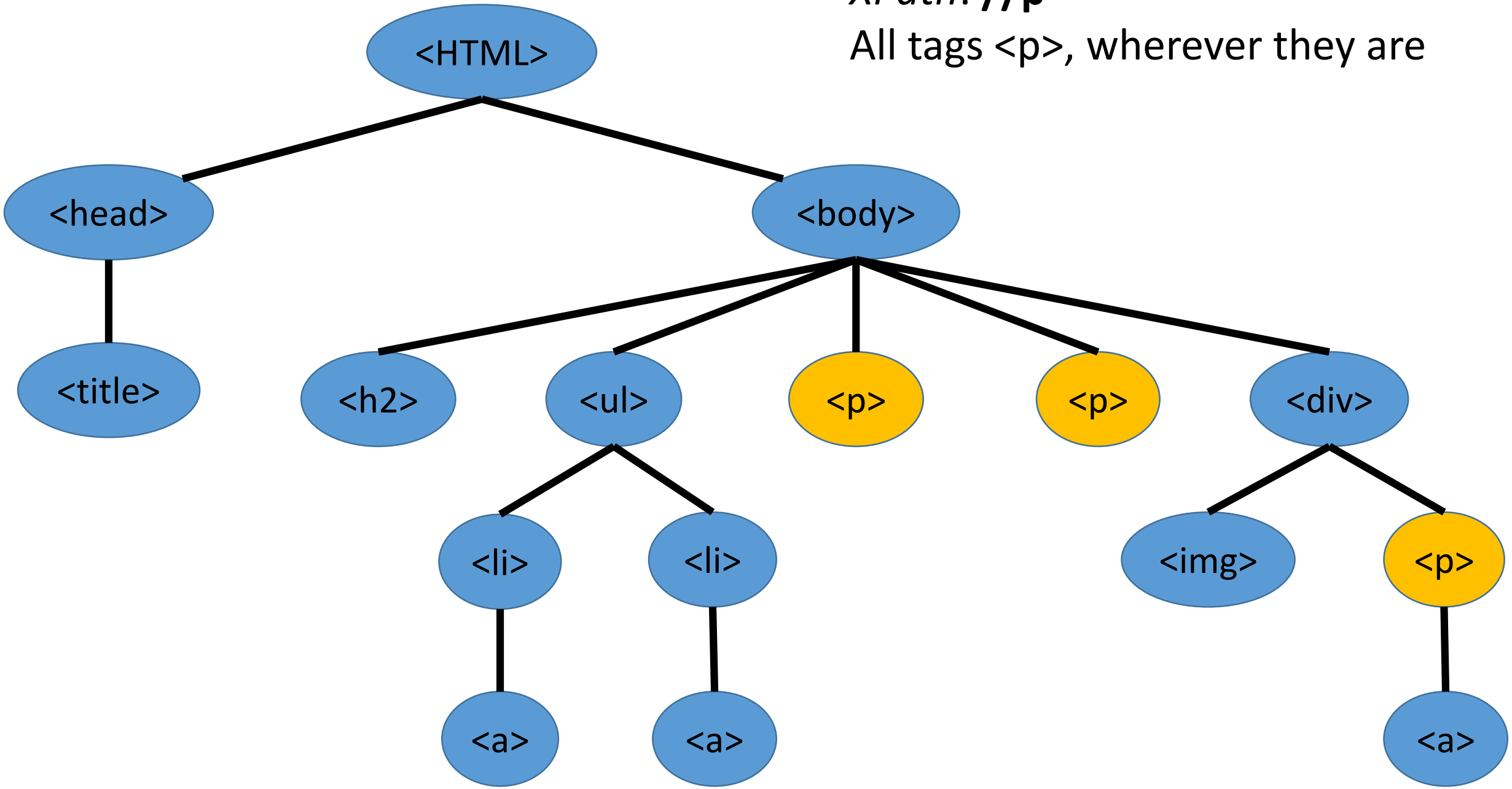
<img>
<p>
<a>
<a>
<a>

XPath: Path Expressions

- “nodename”: select all tags with given node name
- “/”: select from root node
- “//”: select anywhere inside current sub-tree
- “.”: select current node
- “..”: select parent of current node
- “@”: select attribute
- “*”: select all nodes
- “@*”: select all attributes

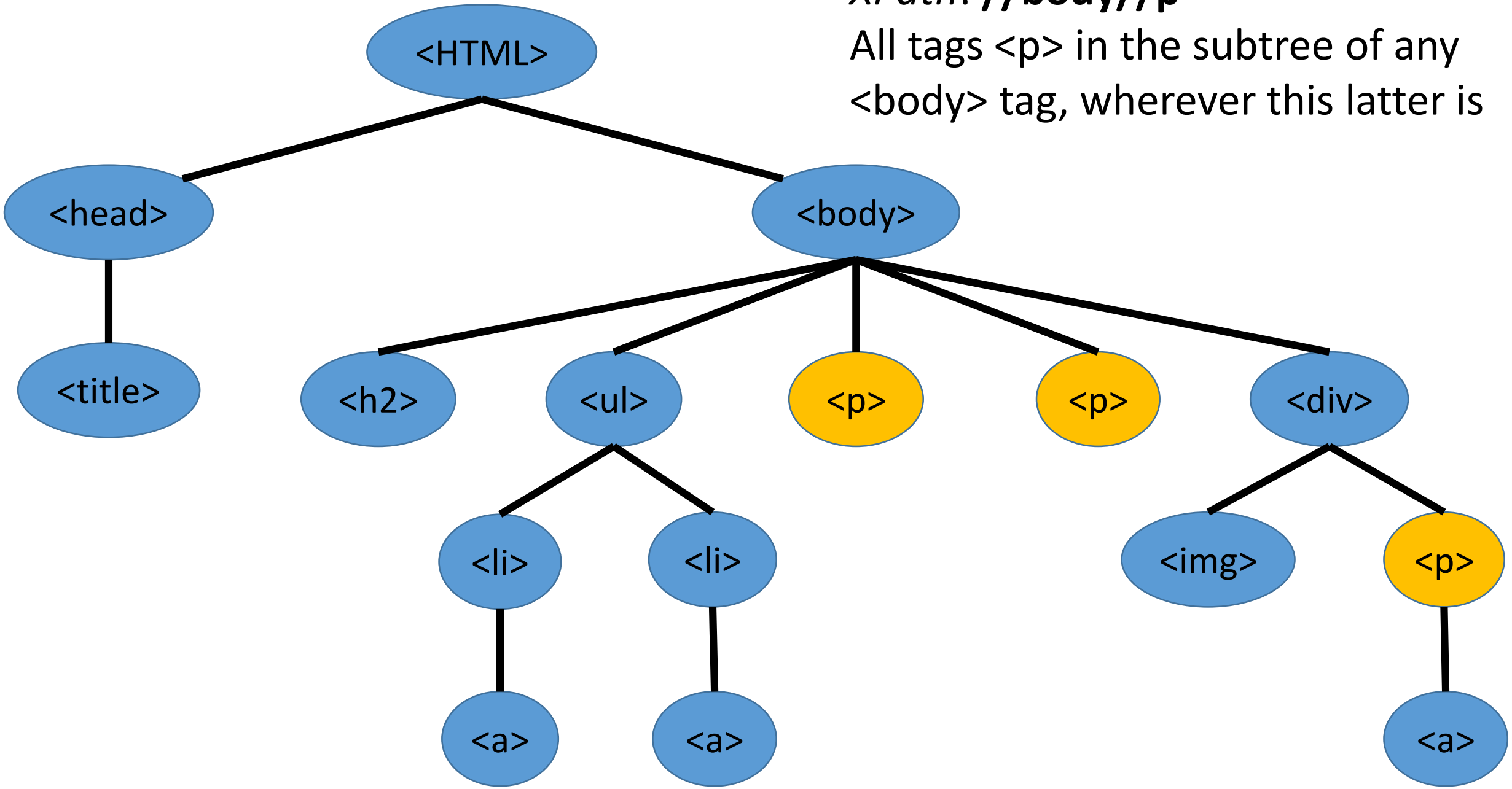
XPath: **//p**

All tags <p>, wherever they are



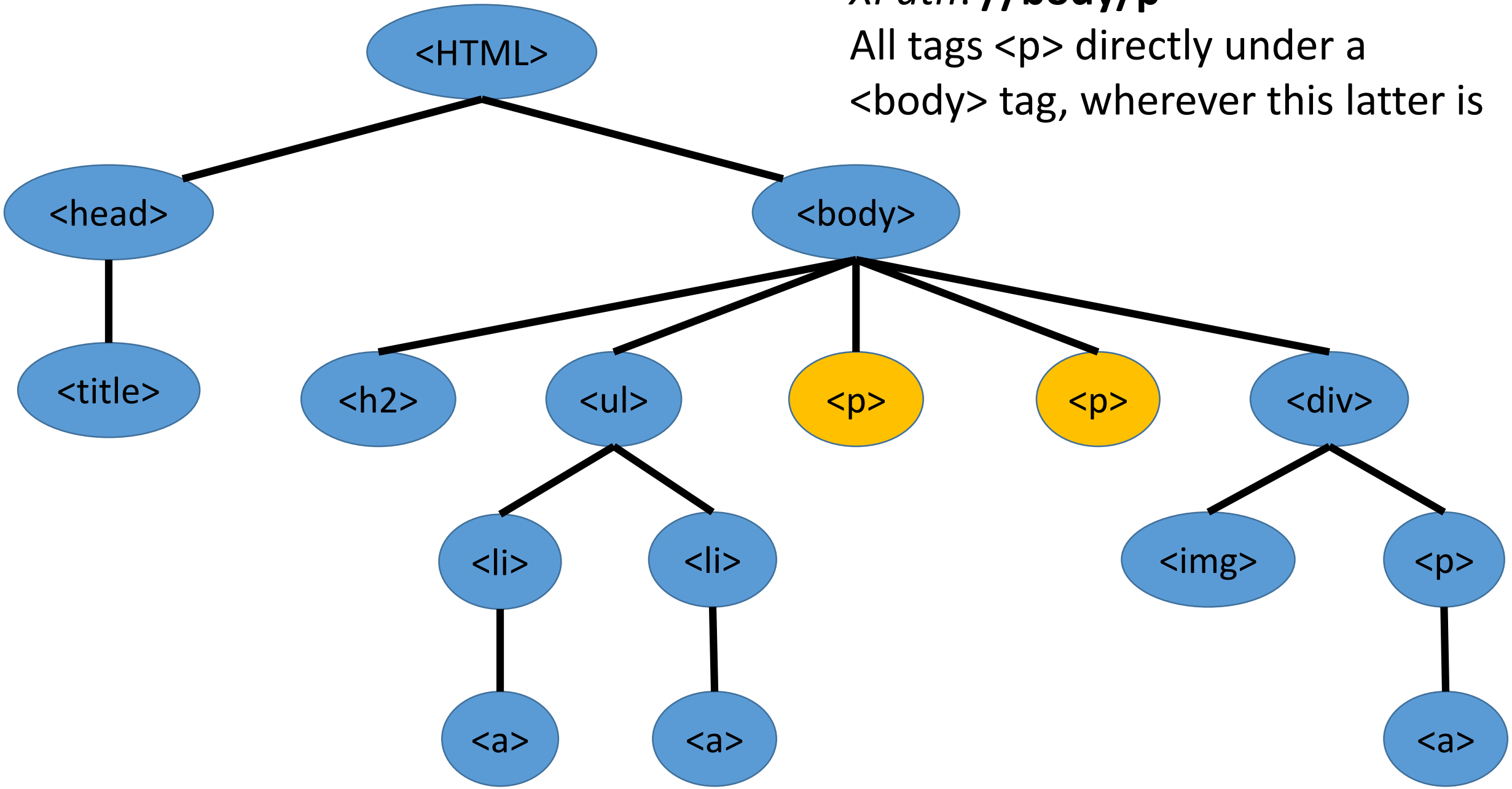
XPath: **//body//p**

All tags <p> in the subtree of any <body> tag, wherever this latter is



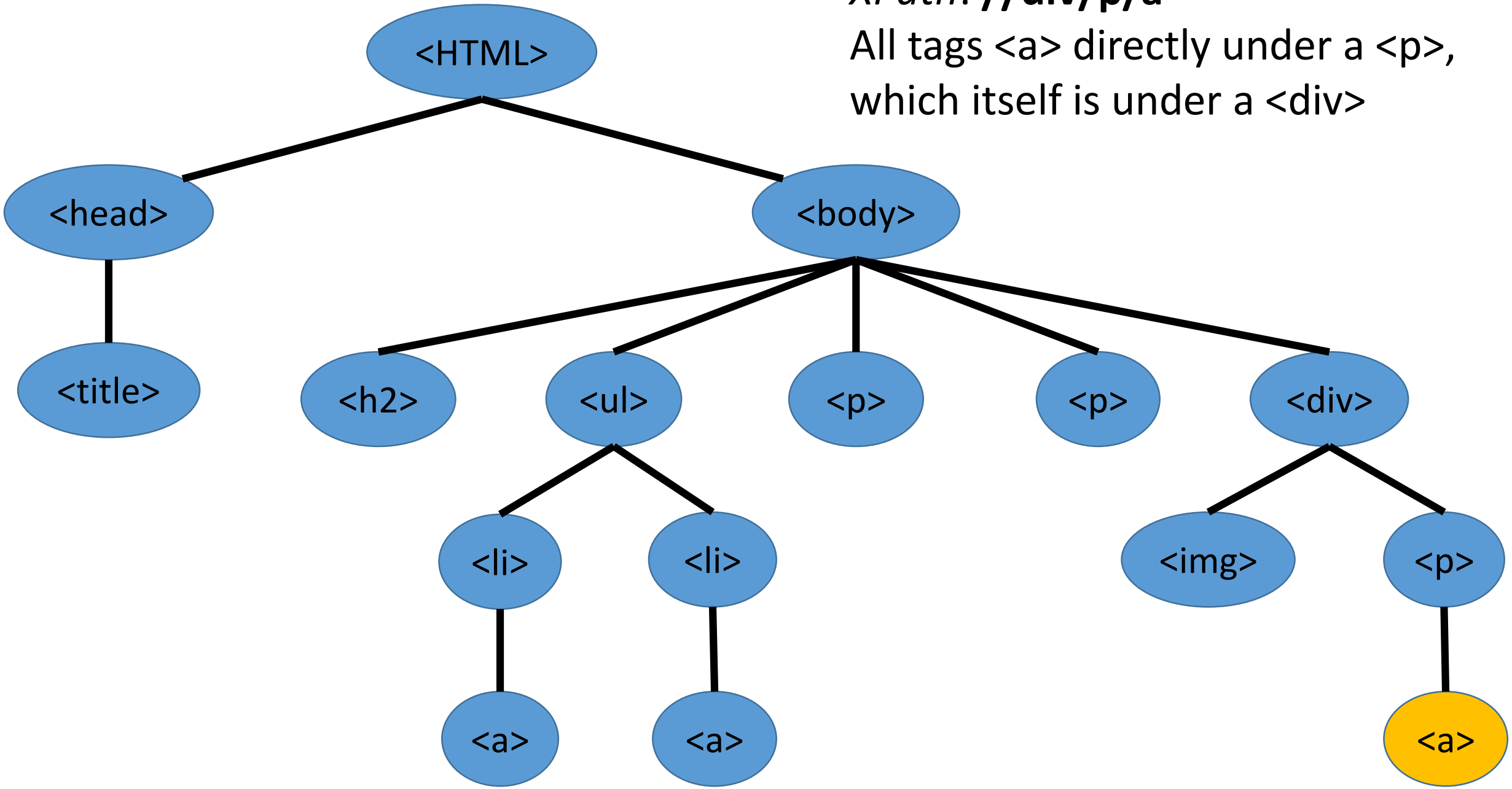
XPath: **//body/p**

All tags <p> directly under a <body> tag, wherever this latter is



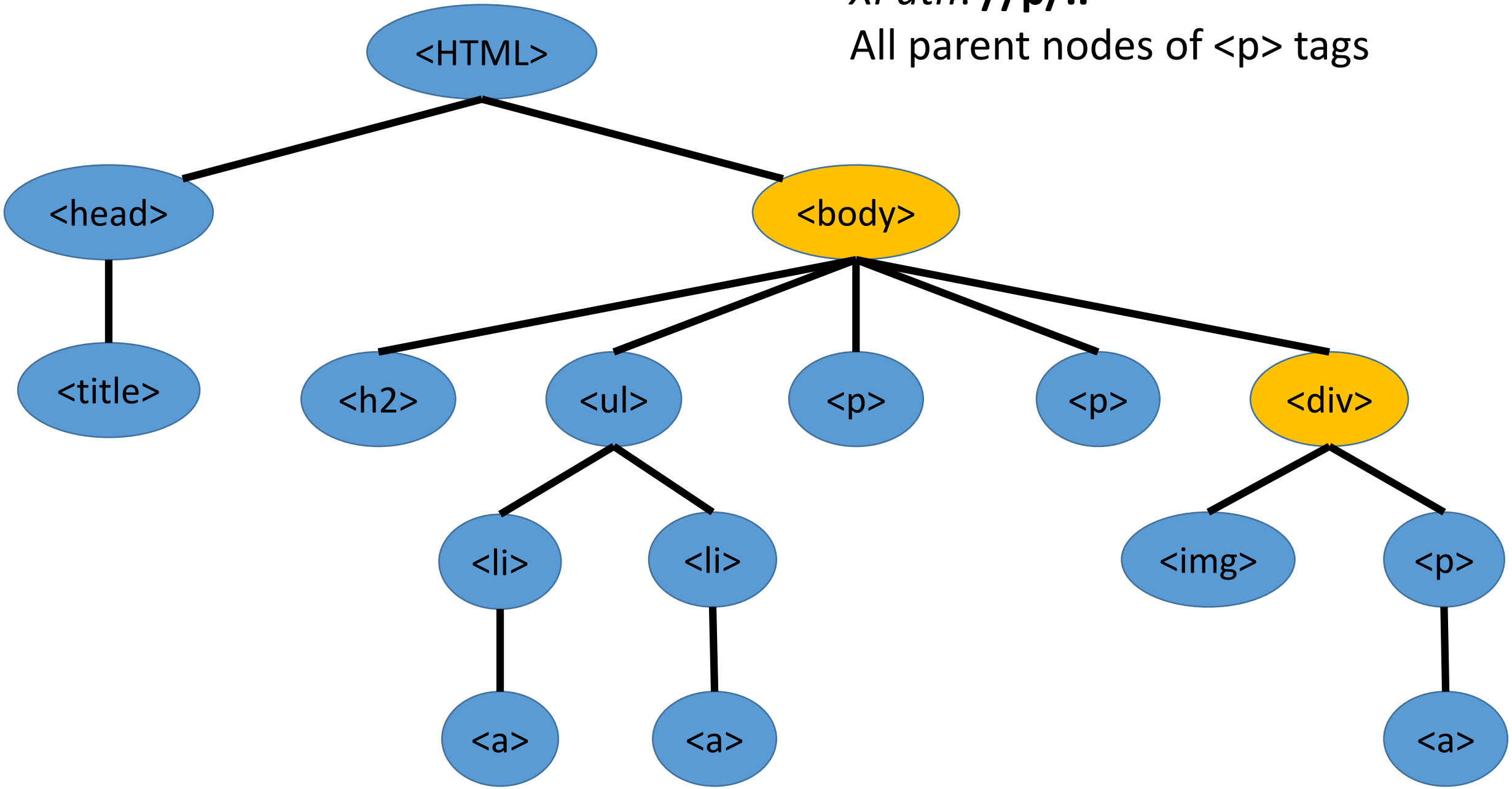
XPath: **//div/p/a**

All tags <a> directly under a <p>, which itself is under a



XPath: **//p/..**

All parent nodes of <p> tags

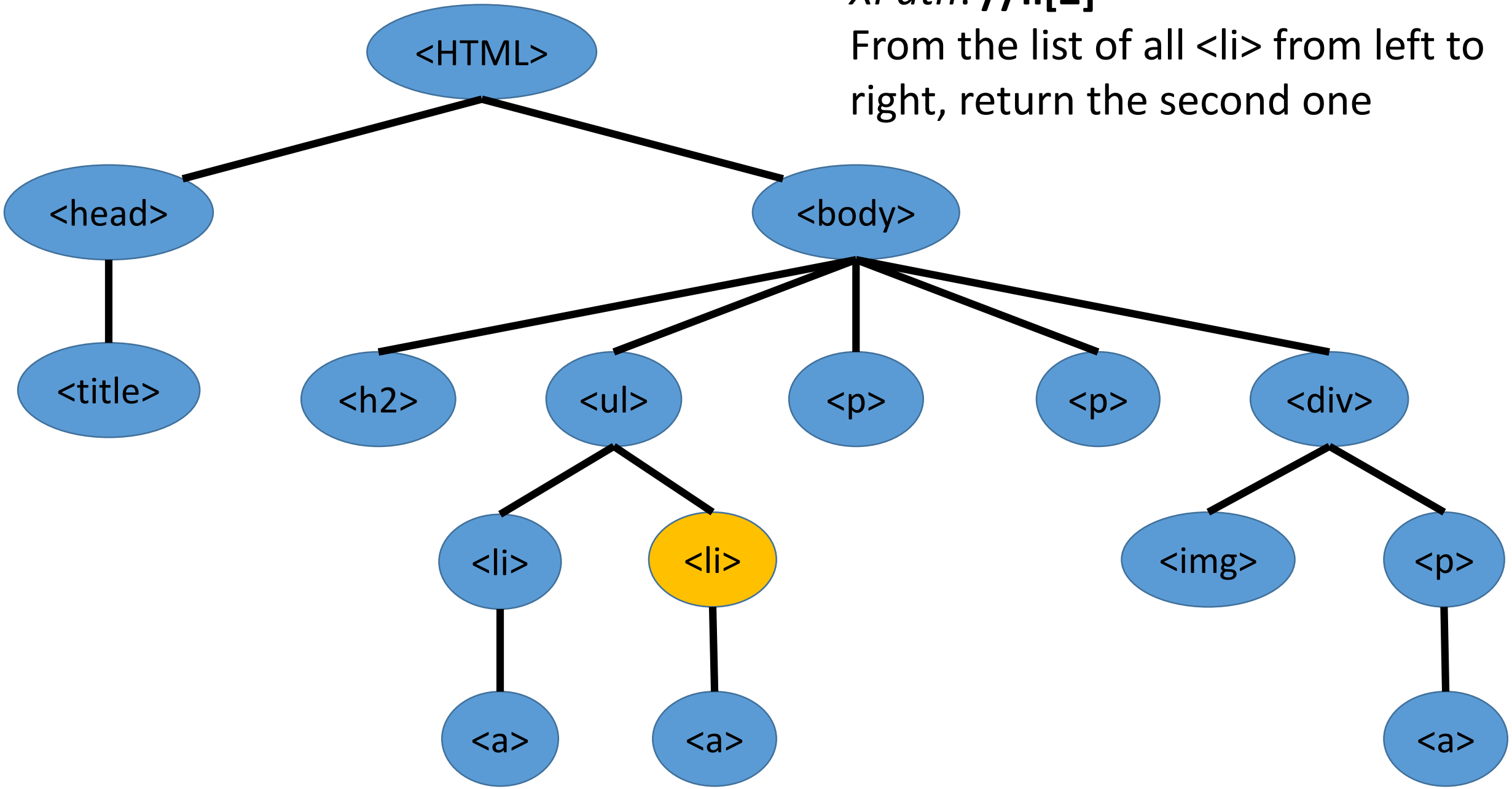


XPath: Predicates

- Predicate inside “[]” to select a subset of all nodes matching the path
- Can also be the index to specify one node among the list of the returned ones from the XPath
  - WARNING: it starts from 1, not 0…

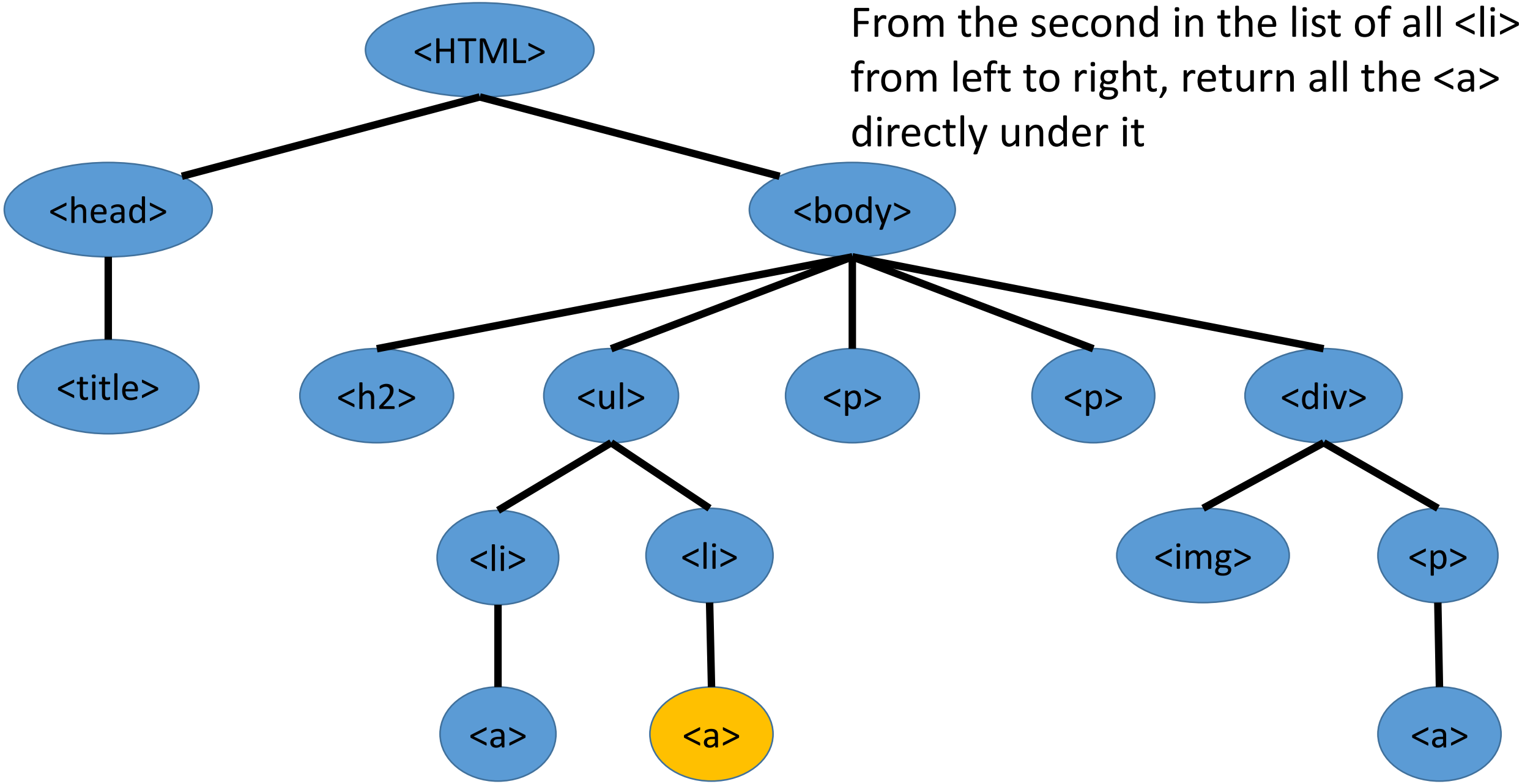
XPath: **//li[2]**

From the list of all <li> from left to right, return the second one



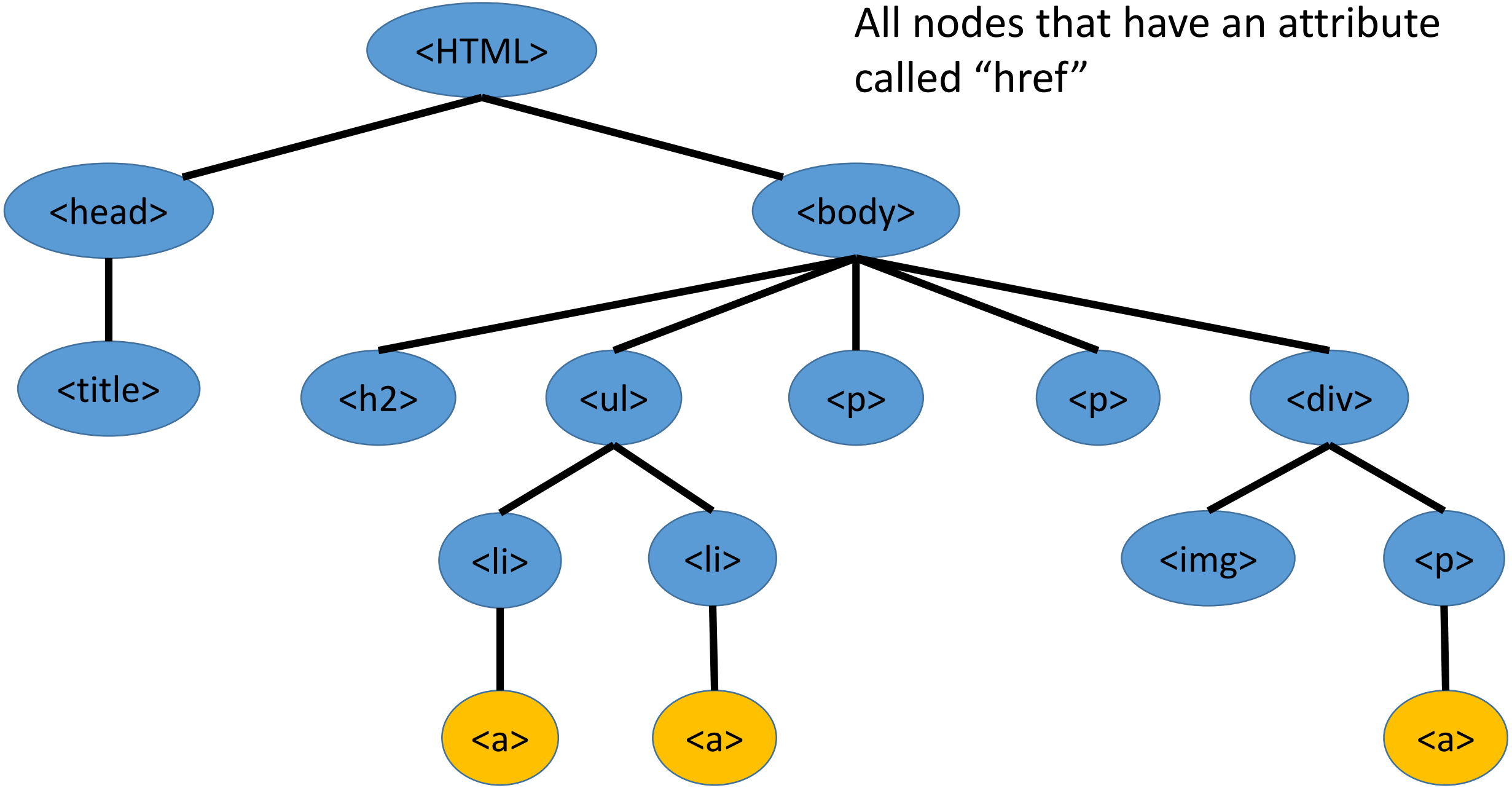
XPath: **//li[2]/a**

From the second in the list of all <li> from left to right, return all the <a> directly under it

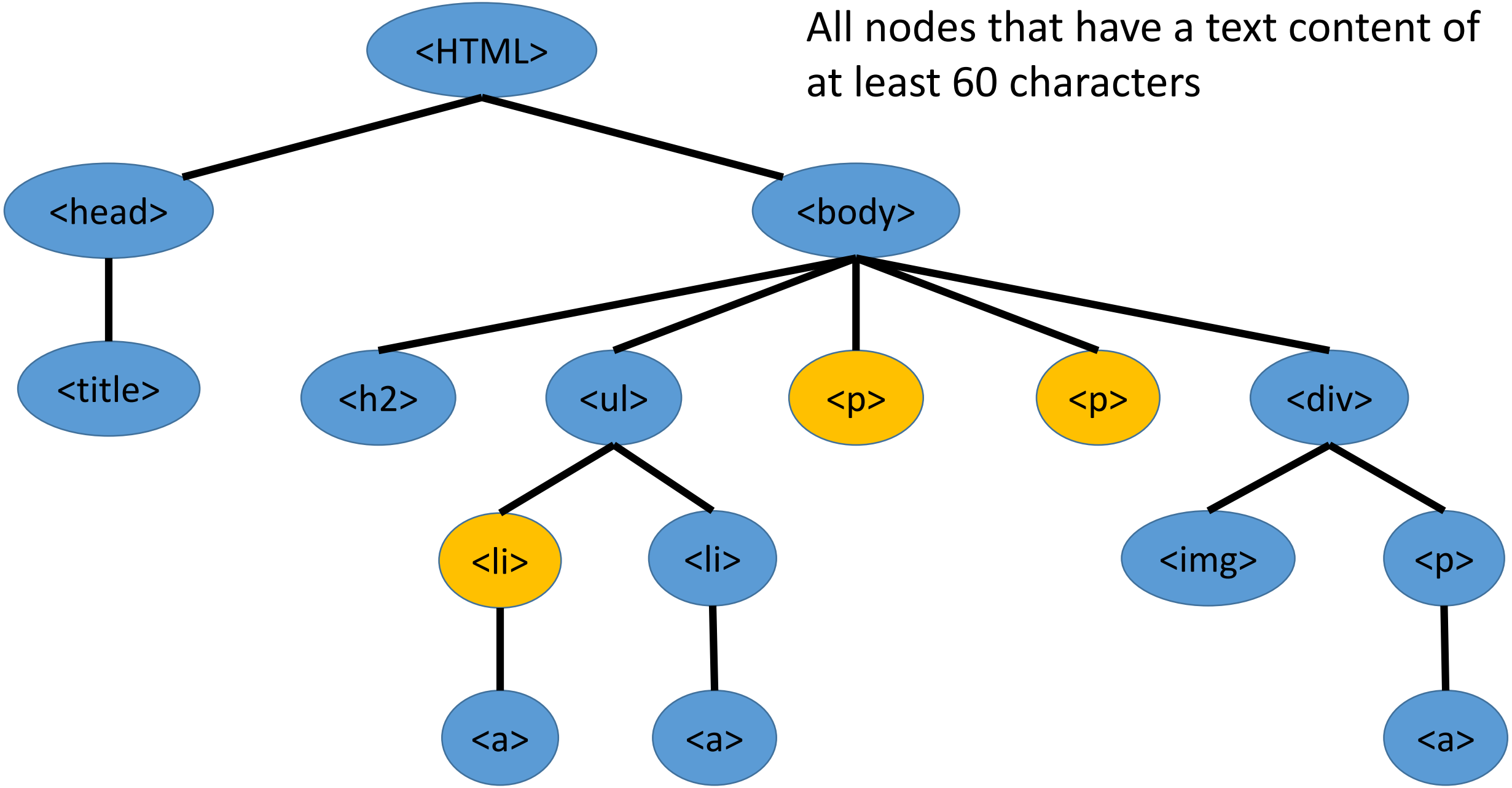


XPath: *//**[@href]**

All nodes that have an attribute called “href”



XPath: *//**[**string-length(text()) > 60**]**
All nodes that have a text content of at least 60 characters



- The “Search” in Chrome Developer Tools can use XPath
- Good to check validity of queries before running your Selenium tests

Just a simple example

- Can have links to the other files in webapp, like for example foo/foo.html
- But the files under WEB-INF are not accessible, e.g. WEB-INF/web.xml should return an error

To run this web application, you need to use Docker. Recall to use "-p 8080:8080" to expose the port on which Wildfly is listening.

Then, you can open this page by pointing your browser to **http://localhost:8080/base**.

Elements Console Sources Network »

```
        To run this web application, you need to use Docker. Recall
    to use "-p 8080:8080" to expose the port
        on which Wildfly is listening.
    "
  </p>
▼ <p>
    "
        Then, you can open this page by pointing your browser to "
    <b>http://localhost:8080/base</b>
    ".
    "
  </p>
▼ <div>
    <img src="img/kitty-cat-kitten-pet-45201.jpeg" alt>
  ▼ <p>
      "Note: the image above was taken from "
      <a href="https://www.pexels.com">https://www.pexels.com</a>
      ",
          and is released with free license CC0"
    </p>
  </div>
</body>
```

html body

//*[string-length(text()) > 60] 3 of 3 Cancel

Maintenance

- In the Selenium tests you need to use XPath to locate HTML elements, eg to click buttons
- XPaths can be arbitrarily complex
- You might need to traverse the same page in many different Selenium tests
  - eg, use Sign Up page to first create a user
- The HTML in a page can change, and you do not want to have to update the XPath in *every single* Selenium test using such page

Page Object (PO)

- A PO is a Java class used in testing to encapsulate all the actions you can do on a web page
- You will have a PO for each web page
- A PO can have methods like “clickLoginButton()”
  - the details of actually interacting with the DOM will be inside such PO method
- Selenium tests should not access the HTML/DOM directly, but rather just use the POs
- If a method in a PO represents a page transition (eg, clicking on a link), then such method should return the PO of the new page
- If the HTML of page change, you need to ONLY update its PO, and not the hundreds of Selenium tests using it…
- It also makes the Selenium tests much easier to understand

Unit or System Tests???

Which tests to write?

- Should you just write unit tests?
- Or should you just write Selenium tests?
- A mix of both?
- Which tests are more useful?
- Which ones should you prioritize?
- When to decide that you have enough unit tests and should add more Selenium ones?

Issues with Unit Testing (UT)

- A lot of code can be trivial (eg getters and setters). UT for them would be a waste of time
- When a class has complex dependencies to other classes (eg, inputs in constructors, like 5 other class instances), writing UT might be cumbersome
- To test your whole system at UT level, you might need *thousands* and *thousands* of tests… as each one may test very little

Issues With System Testing (ST)

- Usually **much** slower than UT
  - eg, start a browser, then start a server, click buttons on browser that send HTTP commands to the server, etc…
- If a ST fails, much more difficult to find out why, as much more code is executed

So, UT or ST???

- Write UT for classes/functions **YOU** think are complex
  - unit tests are easier to debug
- Make sure to have ST for all the main functionalities of your system
  - make sure main functionalities are working, and those will cover a lot of the basic code
- When a ST fails due to bug, to help debugging start writing UT and integration tests for related classes (if they have bugs, maybe they were not as easy as you thought…)

Testing as an Art

- Unfortunately, testing is more like *art* than *science*
- Usually just rules of thumbs, based on anecdotal experience
- There is some general consensus (e.g., testing is important), but no scientifically sound guidelines
- Example: typically, in literature many suggest to put more emphasis on unit tests (eg 90-95%), as quicker to run… *but I disagree* (at least in regards to enterprise applications, where I usually have mostly system and integration tests)

Mocking Frameworks

- Writing unit tests have challenges, especially when dealing with external dependencies
- Example: how to *unit* test a class interacting with a database?
  - Note: the tests you have seen so far on databases are *integration* tests, as we do start a database (eg H2) and also a container (JEE/Spring)
- To overcome such issues, there are testing frameworks like *Mockito* that allows you to mock dependencies
- Such mocking frameworks are relatively popular, *but I do not like them* (at least in the context of enterprise applications)
  - tests often become *brittle,* and harder to maintain
  - tests check your assumptions on the external dependencies, which could be wrong, ie you are not testing the “real” thing any more…
  - just better to write more integration/system tests…

Code Coverage

How Many Tests?

- How many tests should you write?
- As many as you can?
- When can you say you have enough tests?
- When can you say you really need to write more tests?

Code Coverage

- When running tests, automatically check which part of the code has been executed
- *A line that is not executed by any test might have a bug, and the tests would not find it*
- You can calculate *coverage* as percentage of statement executed
  - eg 160 covered lines out of 200 is a 80% line coverage
- Note: there are more sophisticated coverage criteria, but *statement coverage* is the most used/known

Limitations

- Code coverage does NOT tell you if you have good tests, but rather if you need more
- 100% code coverage? You might still have plenty of bugs
  - not all bugs lead the application to crash
- 15% code coverage? Your test suites really suck, go and write more tests
- Usually trying to have code coverage between 50% and 80%
- 100% is often impractical
  - Dead code, defensive programming, etc.

Economy of Testing

- Customers buy (software) products, they do not give a *damn* about the test cases
- The income in a company (can) come from the software, whereas *testing is a cost*
- Each time an employee writes a test, that is time taken away from writing new features that the customer wants
- But customers do not like *defective* products…
- The more time/resources you invest in testing, the lower the *risk* of having defects
- The right balance depends on the economical cost of having bugs
  - eg, recall difference between video-games and software for banks…

Example: Student Projects

- Given X amount of time, should you implement another feature to try to get a better grade?
- Or rather spend such time in testing your code, to avoid bugs that crash your application and possibly reduce your grade?
  - eg, you give a demo during an exam, and your application crash in front of the examiners…

Measuring Coverage

- The main tool to calculate code coverage in Java is *JaCoCo*
  - JaCoCo stands for “Java Code Coverage”
- Can be easily activated via a Maven plugin
- No need to do any change in your tests
- Will generate a report at the end of the build
  - eg, a web page
- Note: IDEs like IntelliJ also have their internal tools to measure coverage when your run tests

localhost:63342/testing_security_development_enterprise_systems/report/target/site/jacoco-aggregate/index.html?_ijt=ld3hk7aasheg2rbivgua3h4lgl

report Sessions

report

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
backend		89%		100%	3	14	3	27	3	12	0	4
frontend		89%		n/a	1	10	2	15	1	10	0	3
Total	14 of 136	89%	0 of 4	100%	4	24	5	42	4	22	0	7

Created with JaCoCo 0.7.9.201702052155

Instrumentation

- How can a tool like JaCoCo measure coverage?
- When a class is executed, its *bytecode* in the ".class" file is loaded into the JVM by the so called *Class Loaders*
- Code coverage tools *intercept* the loading of bytecode, and modify it on the fly
- The modifications do add *probes,* eg method calls after each statement to monitor if such statement is executed

Git Repository Modules

- *NOTE: most of the explanations will be directly in the code as comments, and not here in the slides*
- **intro/spring/testing/selenium/jsf-tests**
- **misc/test-utils**
- **intro/spring/testing/selenium/crawler**
- **intro/spring/testing/mocking**
- **intro/spring/testing/coverage/jacoco**
- **intro/spring/testing/coverage/instrumentation**
- Exercises for Lesson 09 (see documentation)