Integration and System Testing

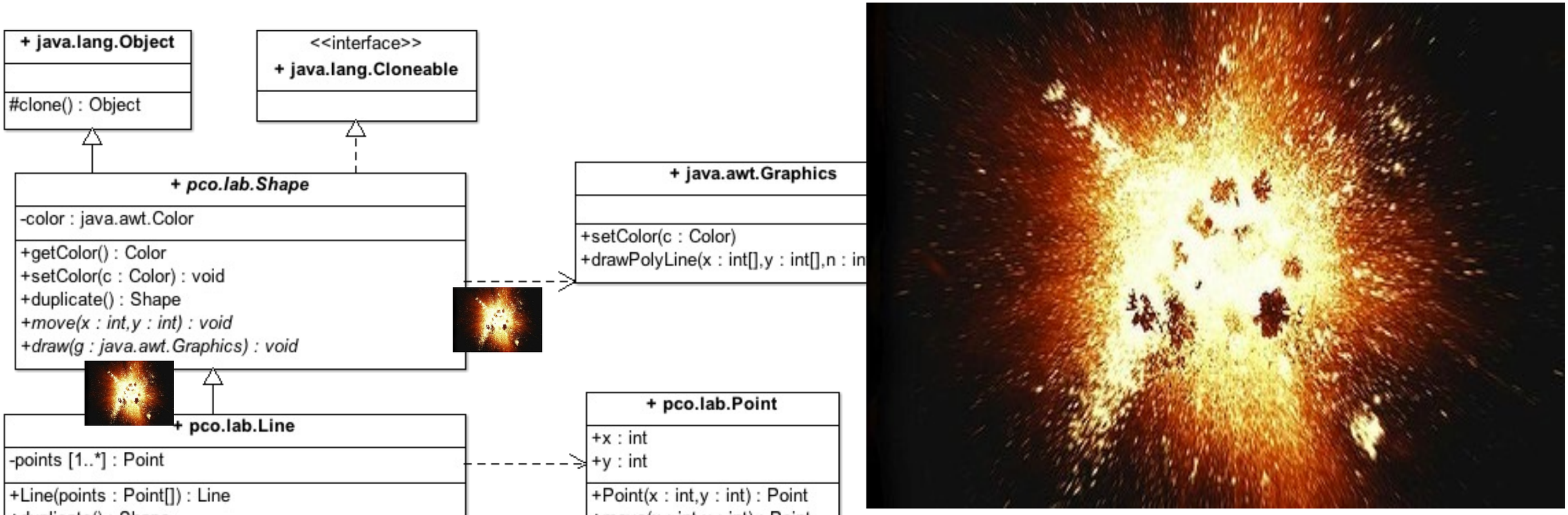
- From module to integration and system testing
- The case of web applications

Ciências ULisboa

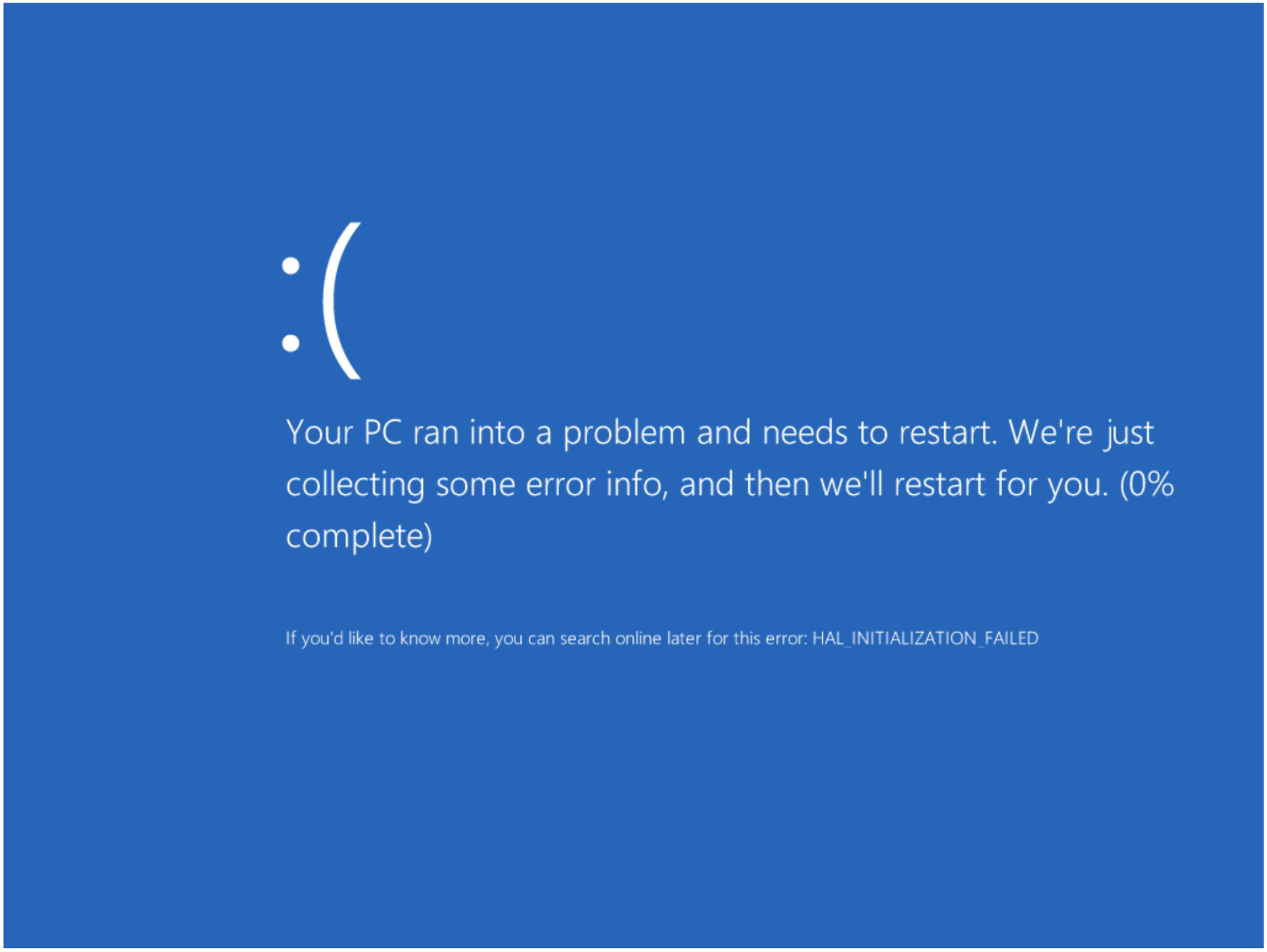
slides: Eduardo Marques, João Neto, José Campos

Integration/system testing

- **Problem**
  - Real-world software applications have multiple components. There are multiple, frequently quite intricate, dependencies among these components.
  - How do I test the integration of components in a sane manner, and the system as a whole when all components are integrated?
- **"Big bang" integration testing?**
  - Combine all components together, and test the system right away!
  - *What could go wrong?*



A big bang:
Windows 8 BSOD (“blue screen of death”)



Integration Testing

- Testing of groups of components integrated to create a sub-system. Components should be tested previously.

- Usually the responsibility of an independent testing team (except sometimes in small projects).

- Integration testing should be black-box testing with tests derived from the technical specification.

- A principal goal is to detect defects that occur on the interfaces of units.

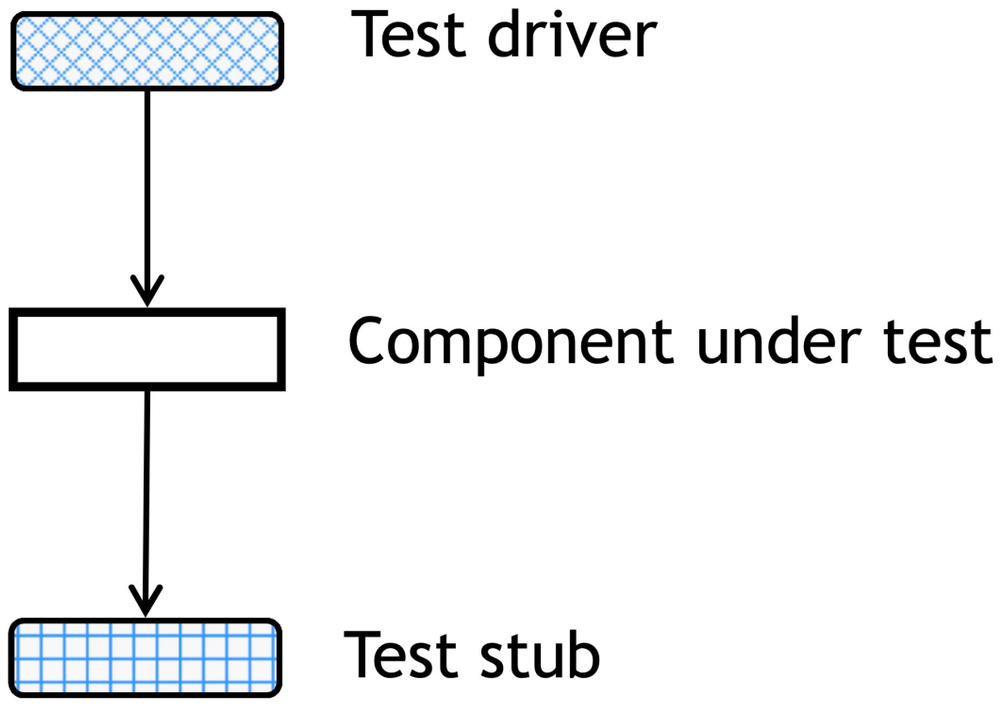
- Main difficulty is localizing errors.

- Incremental integration testing (as opposed to big-bang integration testing) reduces this difficulty.

Integration Testing: Terminology

- **Test harness**: auxiliary code developed to support testing.
- **Test drivers**: call the target code, simulating calling units or a user.
- **Test stubs**: Simulate modules/units/systems called by the target code. Mock objects can be used for this purpose.

(read *Practical Unit Testing with JUnit and Mockito* by Tomek Kaczanowsk)

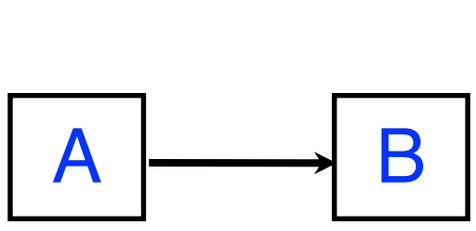


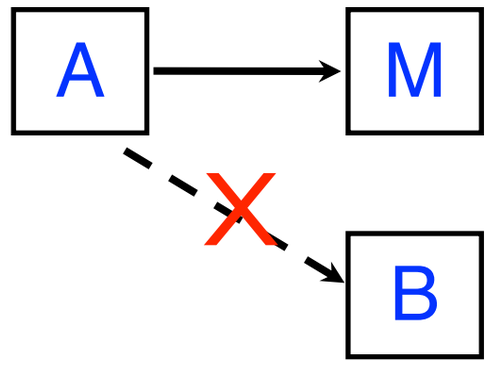
Alternative to the “big bang”

- It’s wiser to start “small” rather than with a “big bang” system test (or an infinitely dense “black hole” we cannot understand)
  - Progress with unit testing, module testing, and integration testing.

- Recall:
  - **Module testing** tests a software component in isolation from other components (i.e., program pieces that can be tested independently).
  - **Integration testing** tests the relations among a (small) subset of all software components, independently of the overall system functionality.
  - **System testing** considers an environment where all components are integrated together.

Module testing and dependency isolation





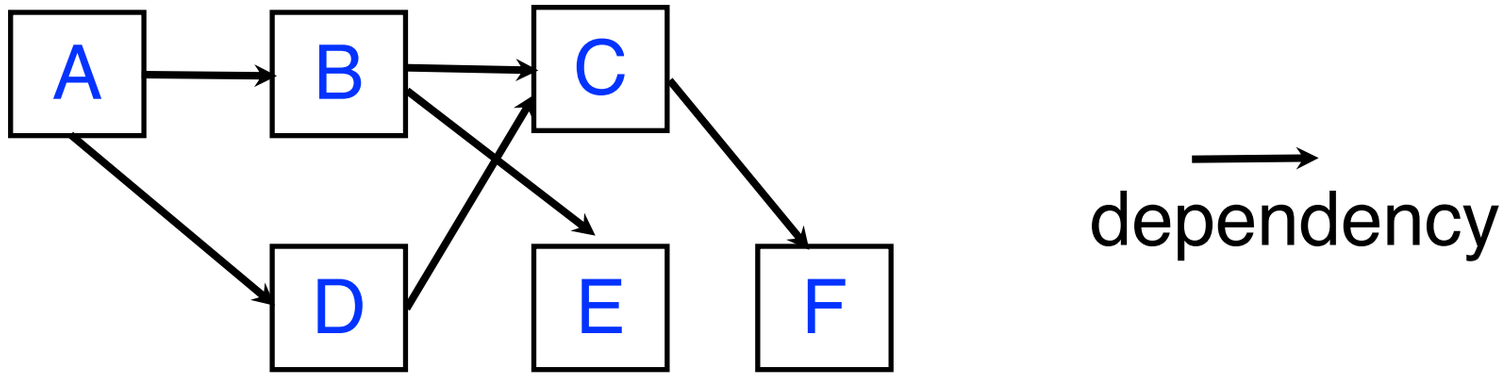
- **Dependency isolation and module testing**
  - **A** interacts with/depends on **B**.
  - How do we test **A** in isolation of **B**?
  - Use **mock object M** (see http://mockito.org)

Basic interaction testing

- **Basic interaction testing of two components**
  - **A** interacts with/depends on **B**.
  - **A** and **B** have been (module-)tested in isolation.
  - We now integrate **A** and **B**.

- The idea is to put the system together component by component

Class integration testing order (CITO) problem



- Whenever possible, it is desirable to integrate and test classes in a sane order.
- **But what order** ?
- Consider the **graph of dependencies** among components. Then:
  - Start by testing components with no dependencies.
  - Integrate already tested components with others that depend only on the former. Repeat this until all components are tested.
  - For the graph above: **E** and **F** first; then **C**; then **B** and **D**; finally **A**.
- **Cyclic dependencies cannot be handled using this scheme**. They are common, e.g., in OO programs. Testers must "break" the cycle by choosing which component to test first, possibly with the aid of mock objects.
- Dependencies can be quite intricate … we'll see a "simple example" next …

Integration Testing

Integration testing is of four types:

- **Top-down:** Start with high-level system and integrate from the top-down replacing individual components by stubs (aka mocks) where appropriate.

- **Bottom-up:** Integrate individual components in levels until the complete system is created.

- **Sandwich:** is the combination of bottom-up approach and top-down approach, so it uses the advantage of both bottom up approach and top down approach. Initially it uses the stubs and drivers where stubs simulate the behaviour of missing components.

- **Big-Bang**

Integration Testing: Top-down

Top-down integration testing technique used in order to simulate the behavior of the lower-level modules that are not yet integrated. In this integration testing, testing takes place from top to bottom. First high-level modules are tested and then low-level modules and finally integrating the low-level modules to a high level to ensure the system is working as intended.

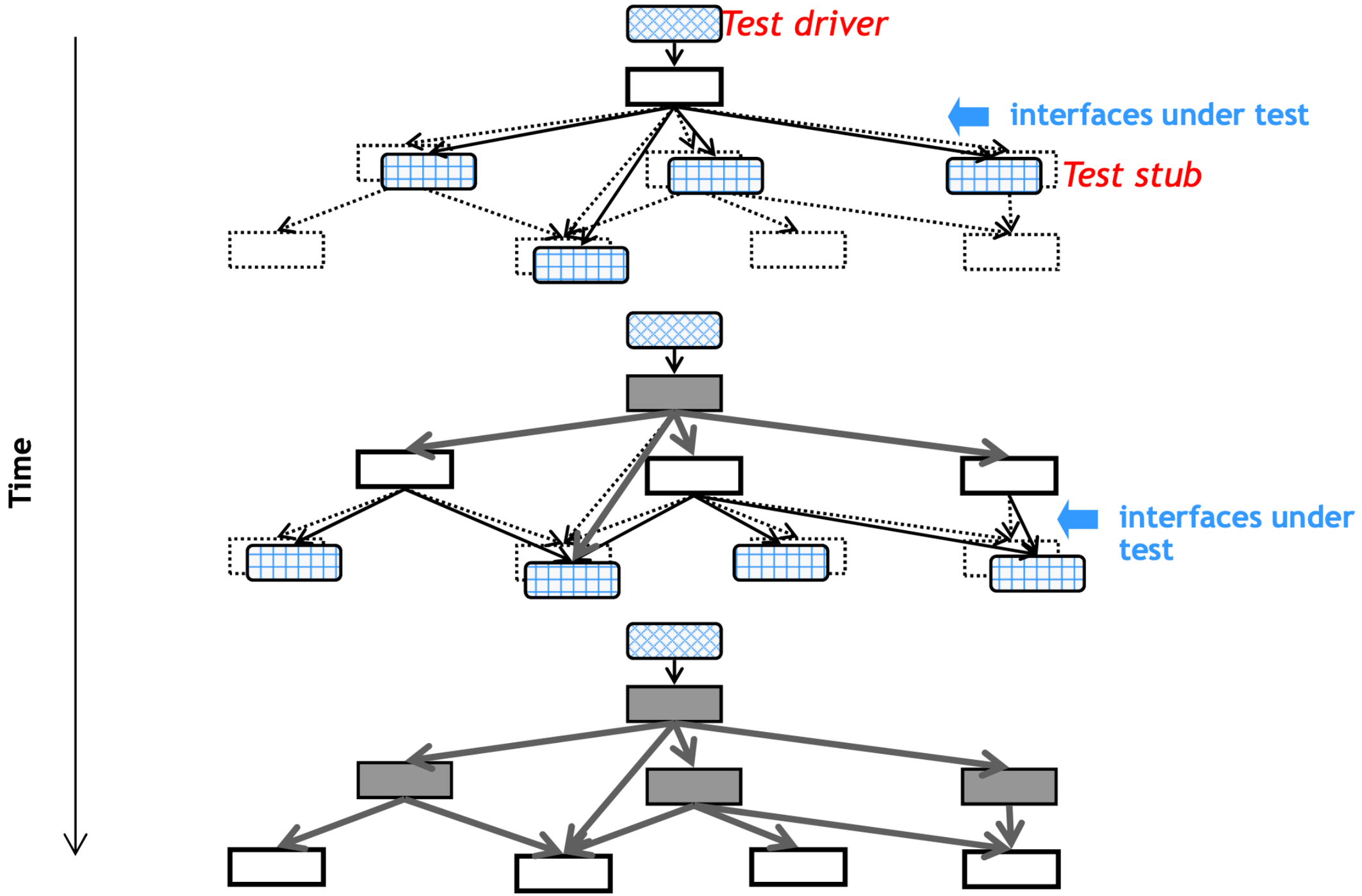
Advantages:

- Separately debugged module.
- Few or no drivers needed.
- It is more stable and accurate at the aggregate level.

Disadvantages:

- Needs many Stubs.
- Modules at lower level are tested inadequately.

Integration Testing: Top-down



Integration Testing: Bottom-up

In bottom-up testing, each module at lower levels is tested with higher modules until all modules are tested. The primary purpose of this integration testing is, each subsystem is to test the interfaces among various modules making up the subsystem. This integration testing uses test drivers to drive and pass appropriate data to the lower level modules.

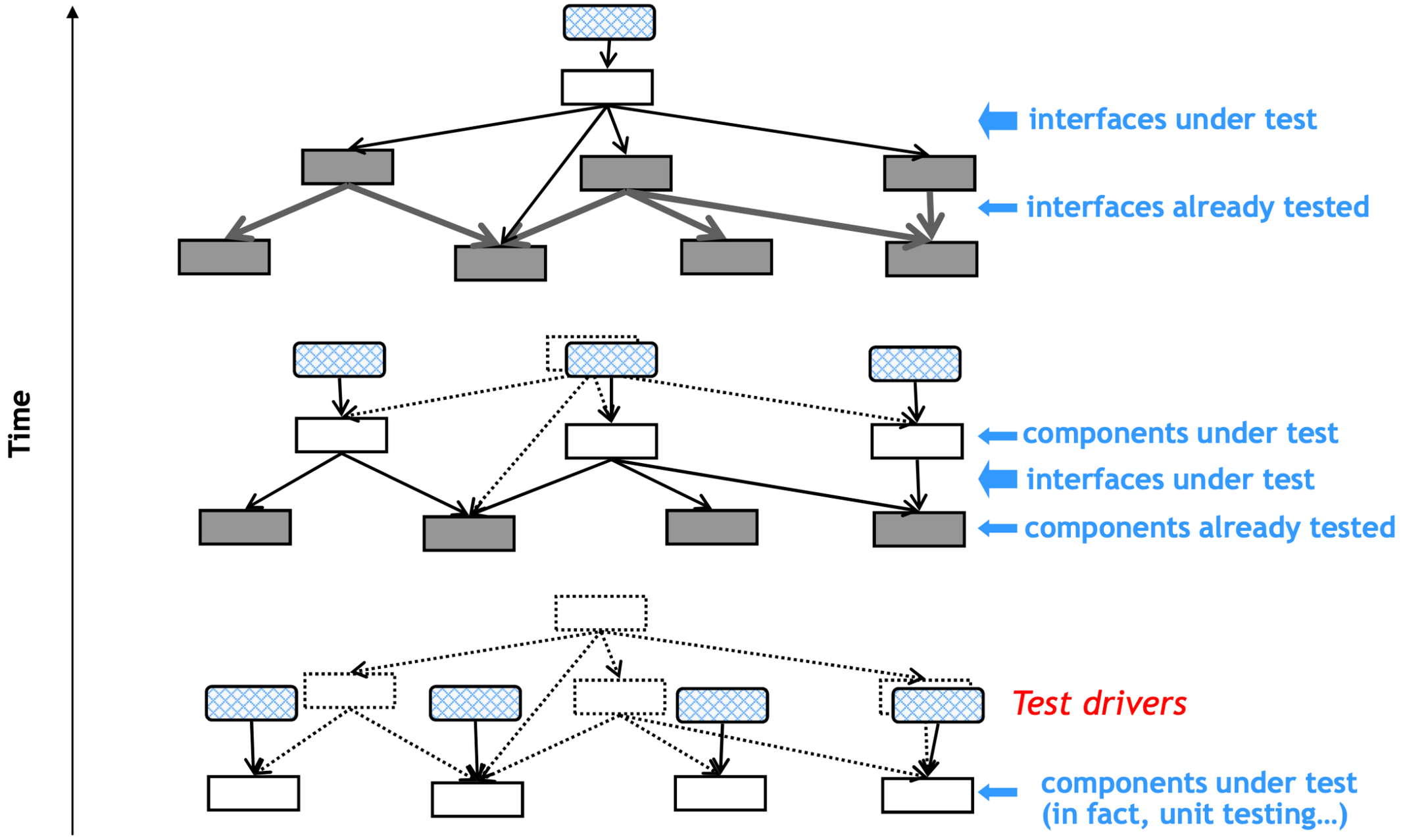
Advantages:

- In bottom-up testing, no stubs are required.
- A principle advantage of this integration testing is that several disjoint subsystems can be tested simultaneously.

Disadvantages:

- Driver modules must be produced.
- In this testing, the complexity that occurs when the system is made up of a large number of small subsystem.

Integration Testing: Bottom-up



System Testing

- Testing the system as a whole by an independent testing team.

- Often requires many resources: laboratory equipment, long test times, etc.

- Usually based on a requirements document, specifying both functional and non-functional (quality) requirements.

- Preparation should begin at the requirements phase with the development of a master test plan and requirements-based tests (black-box tests).

- Ensure the system performs according to its requirements, by evaluating both functional behavior and quality requirements such as reliability, usability, performance and security.

- Useful for detecting external hardware and software interface defects, for example, those causing race conditions, deadlocks, problems with interrupts and exception handling, and ineffective memory usage.

- Tests implemented on the parts and subsystems may be reused/repeated, and additional tests for the system as a whole may

System Testing: Performance Testing

Performance Testing ensures software applications to perform properly under their expected workload. It is a testing technique carried out to determine system performance in terms of sensitivity, reactivity and stability under a particular workload.

- Check whether the software meets the performance requirements.
- Check any hardware or software factors that impact on the system's performance.
- Provide valuable information to tune the system.
- Predict the system's future performance levels.

Results of performance tests should be quantified, and the corresponding environmental conditions should be recorded. Resources usually needed:

- A source of transactions to drive the experiments, typically a load generator.
- Instrumentation of probes that help to collect the performance data (event logging, counting, sampling, memory allocation counters, etc.).
- A set of tools to collect, store, process and interpret data from probes.

System Testing: Load Testing

Load Testing determines the performance of a system under real life based load conditions. Basically load testing determines the behavior of the application when multiple users use it at the same time. It is the response of the system measured under varying load conditions. The load testing is carried out for normal and extreme load conditions.

Objectives of Load Testing is to identify performance congestion before the software product is launched in market.

- To maximize the operating capacity of a software application.
- To determine whether the latest infrastructure is capable to run the software application or not.
- To determine the sustainability of application with respect to extreme user load.
- To find out the total count of users that can access the application at the same time.
- To determine scalability of the application.
- To allow more users to access the application.

System Testing: Stress Testing

Stress Testing determines the robustness of software by testing beyond the limits of normal operation. Stress testing is particularly important for critical software but is used for all types of software.

Stress testing emphasizes on robustness, availability and error handling under a heavy load rather than on what is correct behavior under normal situations. It even tests beyond the normal operating point and analyses how the system works under the extreme conditions. It ensures the system would not crash under crunch situations.

Characteristics of Stress Testing:
- Stress testing analyzes the behavior of the system after a failure.
- Stress testing makes sure that the system recovers after failure.
- It checks whether the system works under the abnormal conditions.
- It ensures to display appropriate error message when the system is under stress.
- It verifies that unexpected failures do not cause security issues.
- It verifies whether the system has saved the data before crashing or not.

System Testing: Load Testing vs Stress Testing

Load Testing	Stress Testing
Load Testing is performed to test the performance of the system or software application under extreme load.	Stress Testing is performed to test the robustness of the system or software application under extreme load.
In load testing load limit is the threshold of a break.	In stress testing load limit is above the threshold of a break.
In load testing, the performance of the software is tested under multiple number of users.	In stress testing, the performance is tested under varying data amounts.
Huge number of users.	Too much users and too much data.
Load testing is performed to find out the upper limit of the system or application.	Stress testing is performed to find the behavior of the system under pressure.
The factor tested during load testing is *performance*.	The factor tested during stress testing is *robustness* and *stability*.
Load testing determines the operating capacity of a system or application.	Stress testing ensures the system security.

System Testing: Configuration Testing

Configuration Testing verifies the performance of the system under development against various combinations of software and hardware to find out the best configuration under which the system can work without any flaws or issues while matching its functional requirements.

Goals:

- To determine whether the software application fulfills the configurability requirements.
- To identify the defects that were not efficiently found during different testing processes.
- To determine an optimal configuration of the application under test.
- To do analyze of the performance of software application by changing the hardware and software resources.
- To do analyze of the system efficiency based on the prioritization.
- To verify the degree of ease to how the *bugs* are reproducible irrespective of the configuration changes.

The case of web applications

- Web applications can be particularly challenging … we have a heterogeneous combination of technologies, subsystems and requirements:
  - Client-side functionality - browser/user interaction, embedded scripting, ...
  - Server-side functionality - web services, databases, ...
  - Data formats: HTML, CSS, XML, JSON, images, video ...
  - Protocols: HTTP, REST, ...
  - Languages: Javascript, PHP, JSP, SQL, Java ...
  - Frameworks: Ajax, J2EE, Google Web toolkit ...
- The so-called “web 2.0 applications” employ massive client / server / service / content interaction: think of Facebook, Youtube, Tumblr, ...

A really "simple" example

begin HTML

A Javascript method invoked by the HTML form below

Use of the AJAX Javascript API
HTTP communication
and XHTML/XML formats are implicit

```
<html> <head>
<script>
function showUser(str) {
if (str=="") {
  document.getElementById("txtHint").innerHTML="";
  return;
}
if (window.XMLHttpRequest) {// code for IE7+, Firefo
   xmlhttp=new XMLHttpRequest();
} else {// code for IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
xmlhttp.onreadystatechange=function()  {
  if (xmlhttp.readyState==4 && xmlhttp.status==200) {
     document.getElementById("txtHint").innerHTML=xmlhttp.responseText;
  }
}
xmlhttp.open("GET","getuser.php?q="+str,true);
xmlhttp.send();
}
</script>
</head>
<body>
<form>
<select name="users" onchange="showUser(this.value)">
<option value="">Select a person:</option>
<option value="1">Peter Griffin</option>
<option value="2">Lois Griffin</option>
<option value="3">Glenn Quagmire</option>
<option value="4">Joseph Swanson</option>
</select>
</form>
<br>
<div id="txtHint"><b>Person info will be listed here.</b></div>

</body>
</html>
```

invocation of server side PHP script

static HTML form

dynamic HTML section

end HTML

A "simple" example
"PHP - AJAX and MySQL"

http://www.w3schools.com/php/php_ajax_database.asp

```php
<?php
$q=$_GET["q"];
$con = mysql_connect('localhost', 'peter', 'abc123');
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
mysql_select_db("ajax_demo", $con);

$sql="SELECT * FROM user WHERE id = '".$q."'";
$result = mysql_query($sql);

echo "<table border='1'>
<tr>
<th>Firstname</th>
<th>Lastname</th>
<th>Age</th>
<th>Hometown</th>
<th>Job</th>
</tr>";
while($row = mysql_fetch_array($result))
  {
  echo "<tr>";
  echo "<td>" . $row['FirstName'] . "</td>";
  echo "<td>" . $row['LastName'] . "</td>";
  ...
  }
echo "</table>";
mysql_close($con);
?>
```

database access

SQL query

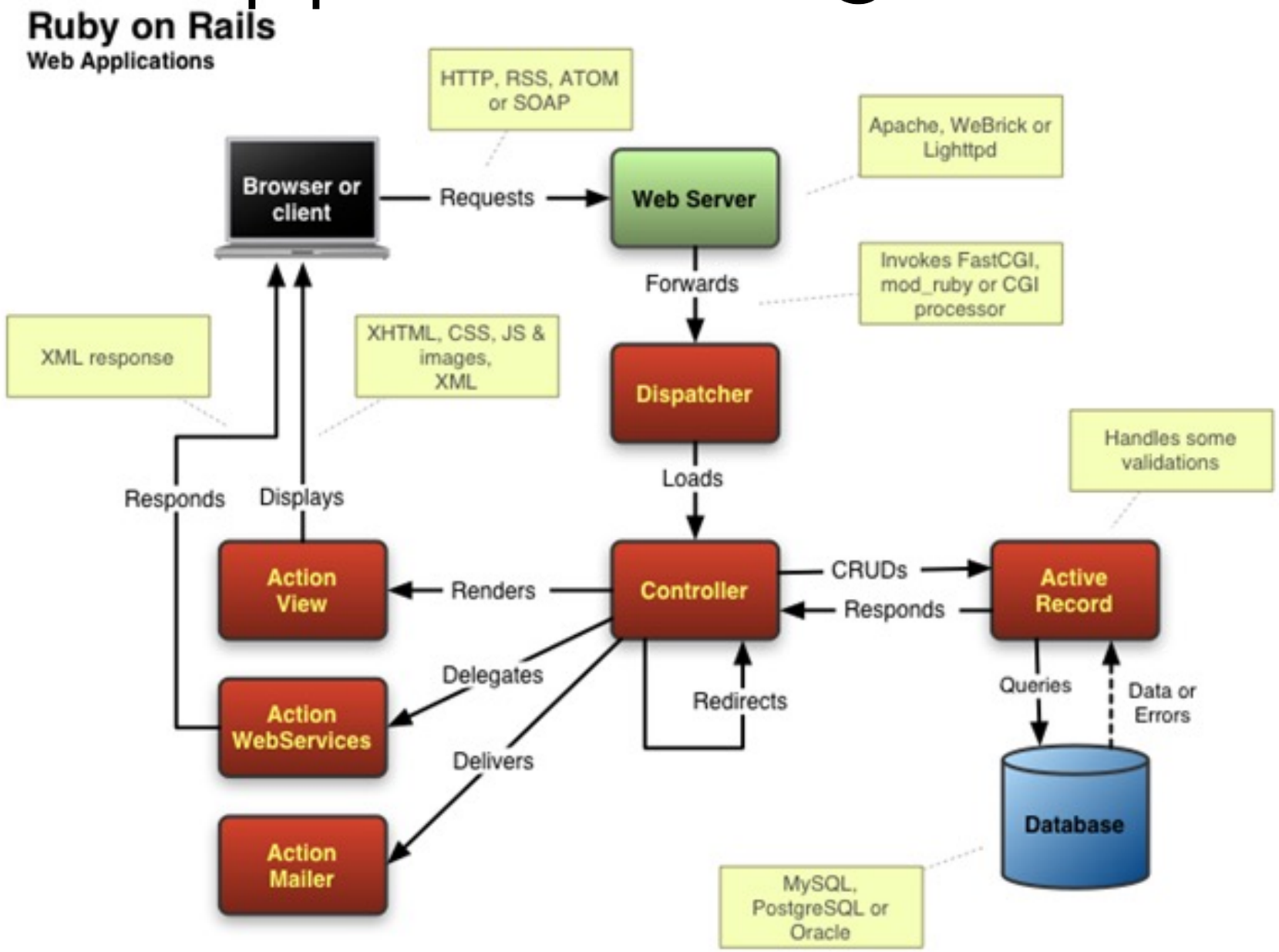
generation of dynamic HTML

use of database data

A “simple” example
“PHP - AJAX and MySQL”

http://www.w3schools.com/php/php_ajax_database.asp

Web application organisation



From: http://thepaisano.wordpress.com/2008/04/24/stop-flipping-the-bird/

- Client-side functionality: HTML, Javascript, CSS, JSON
- Server-side: application server comprising web services, part of the web page rendering logic, business logic, database access

Some testing frameworks

Katalon Studio	Browser interaction IDE
JWebUnit HtmlUnit	Browser interaction testing (HTML + Javascript)
DBunit DBsetup	Database testing
RESTAssured	REST API testing
HttpComponents	HTTP testing
JsUnit	Javascript unit tests
JSonUnit XMLunit	formatted data testing

Exercise: Webapp project

- Download and install Wildfly 10.x (needs Eclipse JEE)
- Download and import maven project `vvs_webapp`
- At Eclipse, create a new server on `Windows > Show View > Servers` (Eclipse might need to download some plugins)
- R-click on project and select `Run As > Run on Server`
- In your browser interact with the project at

  `http://localhost:8080/VVS_webappdemo/index.html`
- Student's project (might have bugs ☺)
  - We'll be using it in the next lectures
- Interact with this webapp
  - Eg, add five customers, and then list all customers in the database