Software Engineering Testing

Testing Web-based Software

The Technologies

- Earlier lectures emphasize criteria on four models of software
- Emphasis in each discussion was first on the criteria, then on how to construct the models from different software artifacts
- This lecture discusses how to apply the criteria to specific technologies
  - We shall be concentrating our discussion on Testing Web-based Software
  - Most of the ideas presented here were developed after the year 2000
  - Thus they are still evolving

Outline

Most of these ideas were developed after 2000
Few are widely used
Most adapt graph-based testing from earlier discussions

1. **Overview**
2. Static Hyper Text Web Sites
3. Dynamic Web Applications
    1. Client-side testing
    2. Server-side testing
4. Web Services

Issues in Testing Web Software

- A *web application* is a program that is deployed on the web
  - Usually uses HTML as the user interface
  - Web-deployment means they are available worldwide
  - They accept requests through HTTP and return responses
  - HTTP is stateless – each request/response pair is independent
- Web applications are usually very competitive
- A *web service* is a web-deployed program that accepts XML messages wrapped in SOAP
  - Usually no UI with humans
  - Service must be published so other services and applications can discover them

Web Software

- Composed of independent, loosely coupled software components
  - All communication is through messages
  - Web application messages always go through clients
  - The only shared memory is through the session object – which is very restricted
  - The definition of state is quite different
- Inherently concurrent and often distributed
- Most components are relatively small
- Uses numerous new technologies, often mixed together

Deploying Software

- Bundled : Pre-installed on computer
- Shrink-wrap : Bought and installed by end-users
- Contract : Purchaser pays developer to develop and install, usually for a fixed price
- Embedded : Installed on a hardware device, usually with no direct communication with user
- Web : Executed across the Internet through HTTP

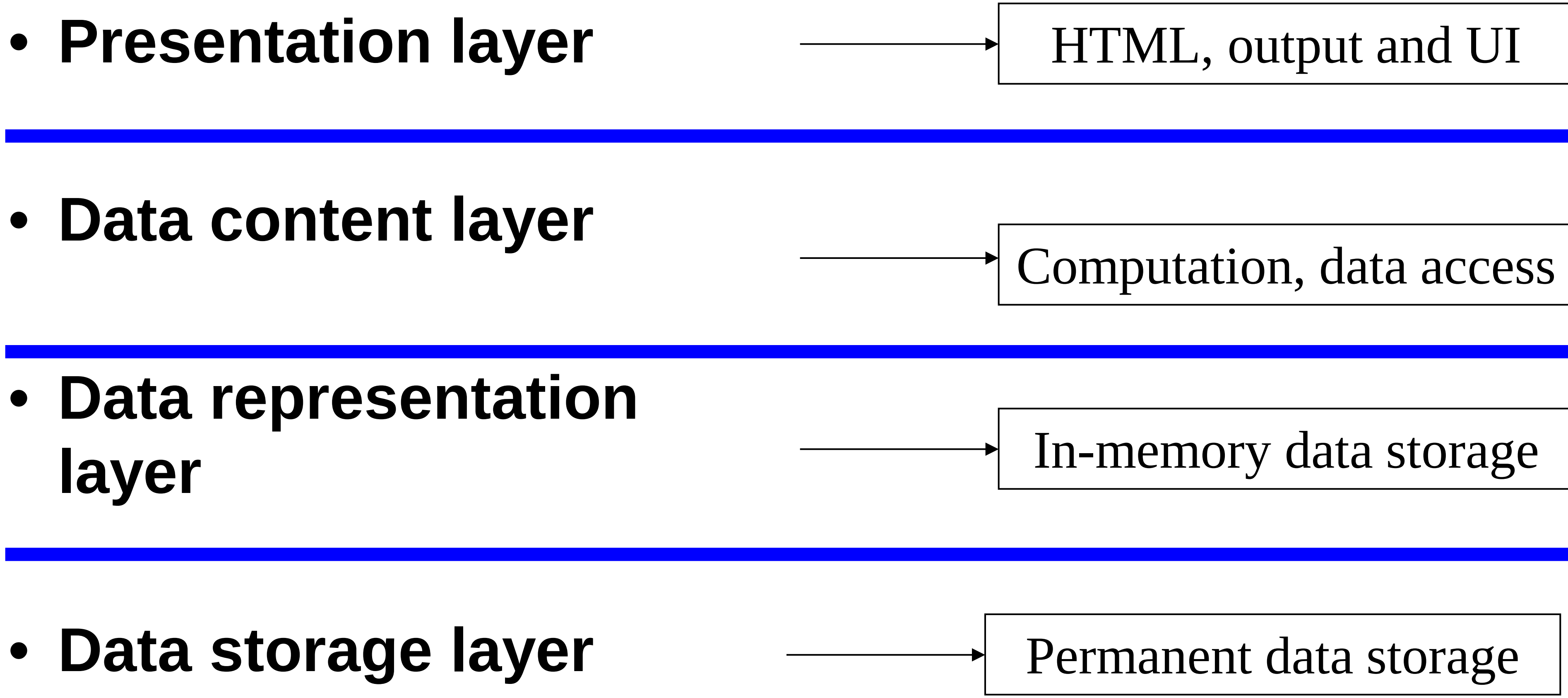
General Problem

- Web applications are heterogeneous, dynamic and must satisfy very high quality attributes
- Use of the Web is hindered by low quality Web sites and applications
- Web applications need to be built better and tested more

Problem Parameters

- HTTP is a stateless protocol
  - Each request is independent of previous request
- Servers have little information about where a request comes from
- Web site software is extremely loosely coupled
  - Coupled through the Internet – separated by space
  - Coupled to diverse hardware devices
  - Written in diverse software languages

Separation of Concerns in Web Apps



- **Presentation layer**

HTML, output and UI

- **Data content layer**

Computation, data access

- **Data representation layer**

In-memory data storage

- **Data storage layer**

Permanent data storage

Differences in Testing Web Software

- Traditional graphs do not apply
  - Control flow graph
  - Call graph
- State behavior is hard to model and describe
- All inputs go through the HTML UI – low controllability
- Hard to get access to server-side state (memory, files, database) – low observability
- Not clear what logic predicates can be effectively used
- No model for mutation operators on web software

New Essential Problems of Web Apps

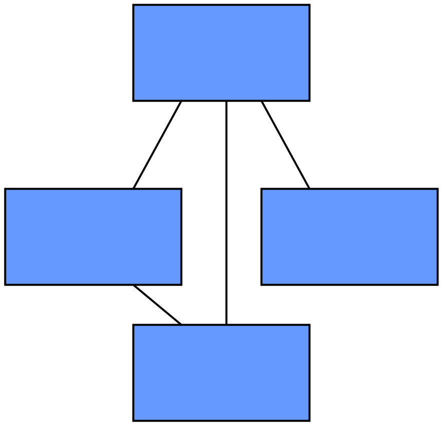
1. Web site applications feature distributed integration and are extremely loosely coupled
   - Internet and diverse hardware / software

2. HTML forms are created dynamically by web applications
   - UI created on demand and can vary by user and time

3. Users can change the flow of control arbitrarily
   - back button, forward button, URL rewriting, refresh

4. Dynamic integration of new software components
   - new components can be added during execution

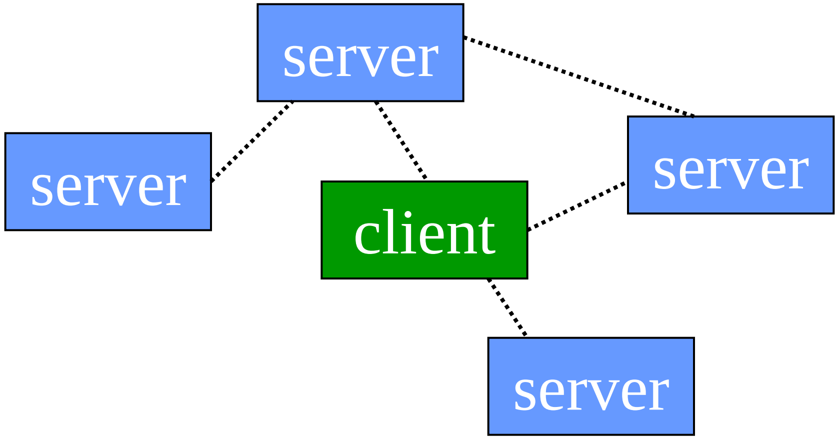
Problem 1: Loosely Coupled



Traditional software

Connected by calls and message passing

High and moderate coupling



Web-based software

Connected with HTTP and XML

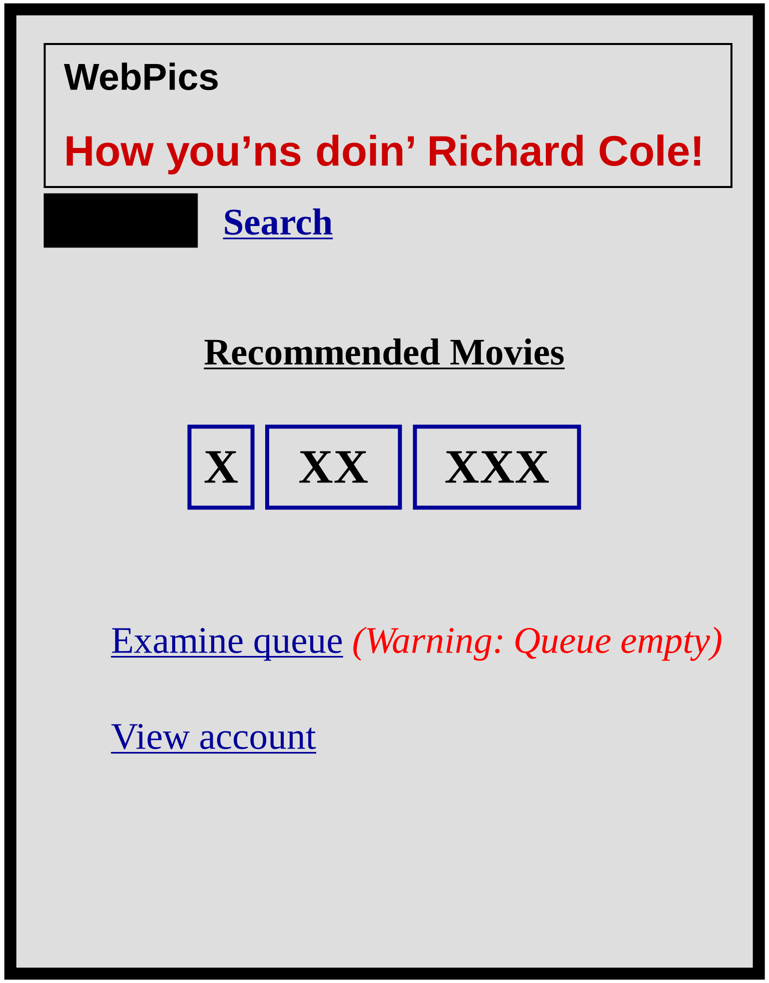
Loose, *extremely* loose, distributed coupling

How can we ensure the reliability of this type of software?

Extremely Loose Coupling

- *Tight Coupling* : Dependencies among the methods are encoded in their logic
  - Changes in A may require changing logic in B
- *Loose Coupling* : Dependencies among the methods are encoded in the structure and data flows
  - Changes in A may require changing data uses in B
- *Extremely Loose Coupling* (*ELC*) : Dependencies are encoded only in the data contents
  - Changes in A only affects the contents of B’s data

Problem 2: Dynamic Flow of Control



WebPics

How you’ns doin’ Richard Cole!

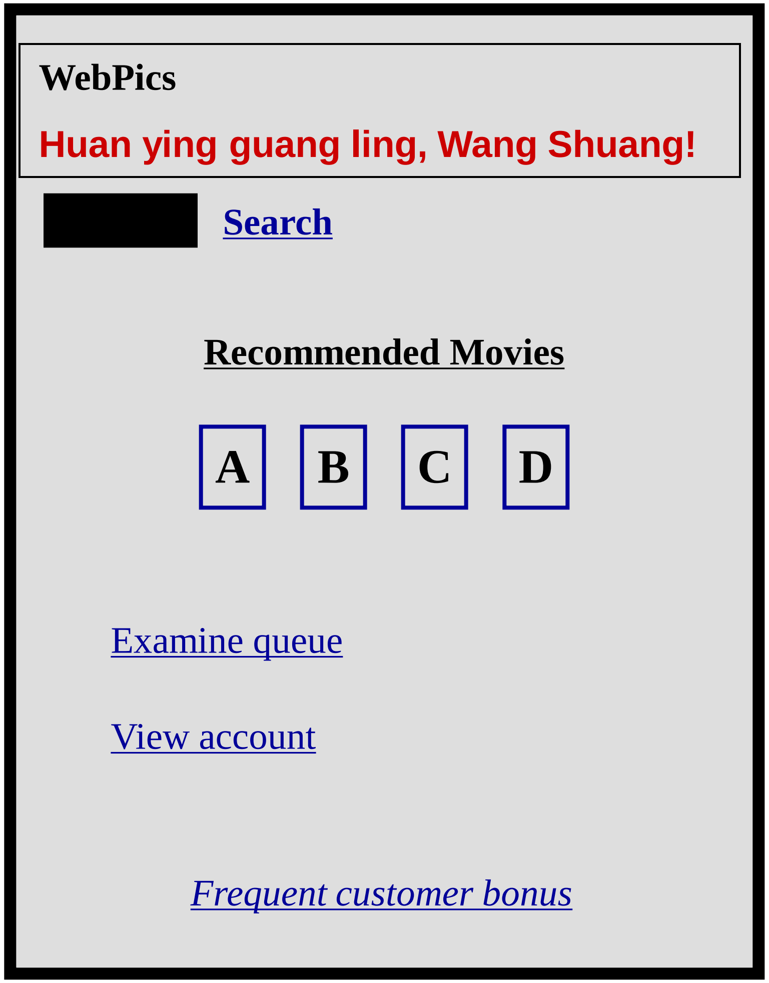
Search

Recommended Movies

X XX XXX

Examine queue *(Warning: Queue empty)*

View account



WebPics

Huan ying guang ling, Wang Shuang!

Search

Recommended Movies

A B C D

Examine queue

View account

Frequent customer bonus

How can we ensure the reliability of this type of system?

Dynamic Execution of Web Apps

- Parts of the program are generated dynamically
- Dynamic web pages are created when users make requests
- Different users will see different programs !
- The potential control, ala the traditional control flow graph, cannot be known ahead of time

The potential flow of control cannot be known statically

Problem 3: User Control Flow

- Users can make unexpected changes to the flow of control
  - Back buttons, refreshing, caching, forward button, URL hacking
- State is stored in the server and in the HTML in the client's browser
- *Operational transitions* : Transitions NOT based on an HTML link: back, forward, URL rewriting, refresh
- These transitions can cause unanticipated changes to the state of the web application

How can we ensure the reliability of this type of software?

Problem 4: Dynamic Integration

- Software modules can dynamically integrate with others if they use the same data structures
- EJBs can be inserted into web applications, which can immediately start using them
- Web services find and bind to other web services dynamically

Outline

1. Overview
2. **Static Hyper Text Web Sites**
3. Dynamic Web Applications
    1. Client-side testing
    2. Server-side testing
4. Testing Web Services

Testing Static Hyper Text Web Sites

- This is not program testing, but checking that all the HTML connections are valid
- The main issue to test for is dead links
- We should also evaluate
  - Load testing
  - Performance evaluation
  - Access control issues
- The usual model is that of a graph
  - Nodes are web pages
  - Edges are HTML links

Outline

1. Overview
2. Static Hyper Text Web Sites
3. **Dynamic Web Applications**
    1. Client-side testing
    2. Server-side testing
4. Testing Web Services

Testing Dynamic Hyper Text Web Sites

- The user interface is on the client
- Some software is on the client (scripts such as Javascript)
- Most software is on the server
- Client-side testing does not access source or state on the server
- Server-side testing can use the source or the server state

Outline

1. Overview
2. Testing Static Hyper Text Web Sites
3. Testing Dynamic Web Applications
    1. **Client-side testing**
    2. Server-side testing
4. Testing Web Services

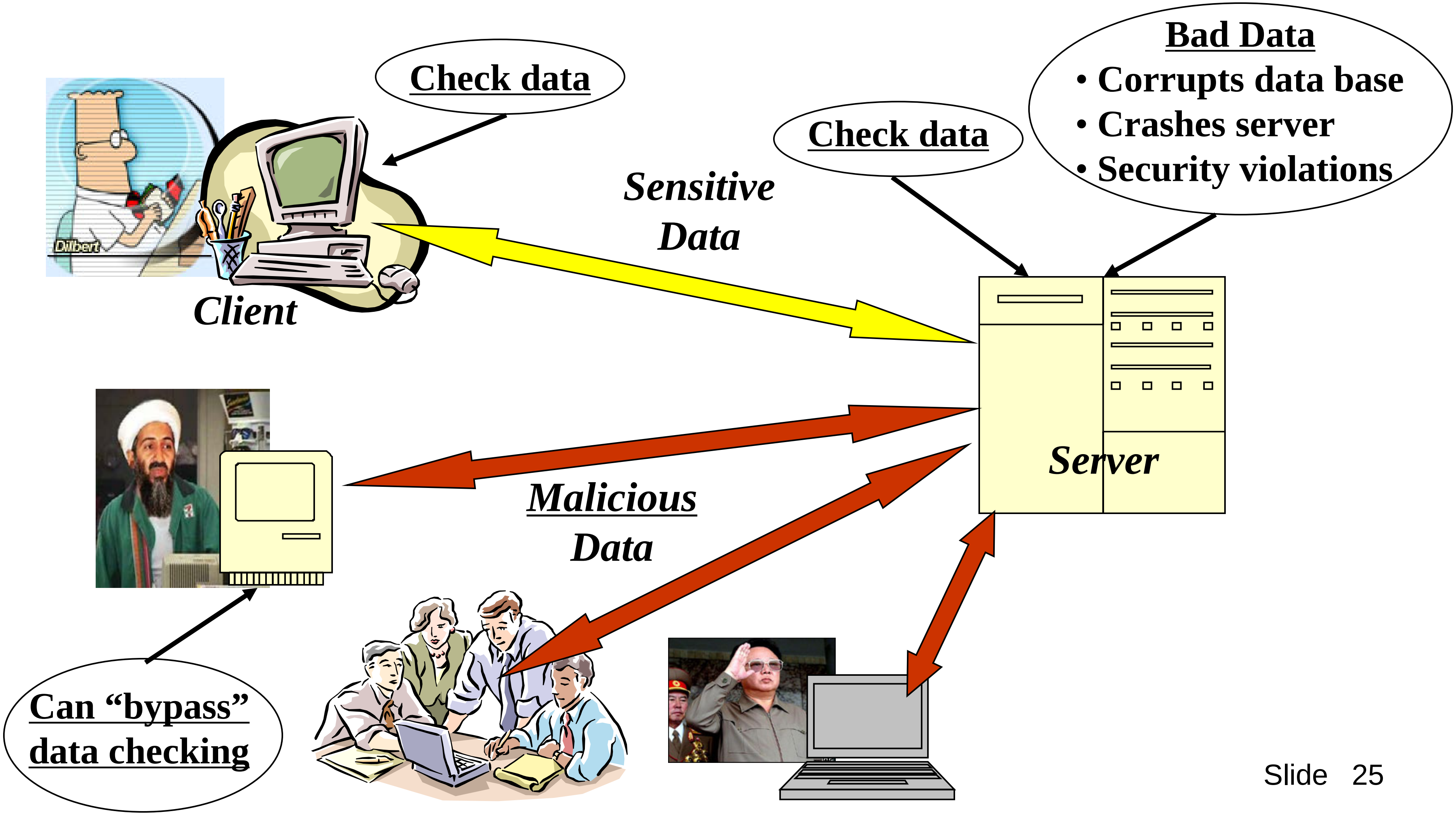
Client-Side (Black-Box) Testing

- The UI and the software are on separate computers
- The inputs to web software are defined by the HTML form elements
  - Text boxes, buttons, dropdown lists, links, etc
- Techniques for generating values
  - Supplied by the tester
  - Generated randomly
  - User session data – data collected from previous users of the software
- Choosing values
  - Bypass testing – values that violate constraints on the inputs, as defined by client-side information
- The problem of finding all screens in a web application is undecidable

Test Value Selection

- Challenge
  - How to automatically provide effective test values ?
- *Semantic Domain Problem* (SDP)
  - Values within the application domain are needed
  - Enumeration of all possible test values is inefficient
- Possible solutions
  - Random values (ineffective)
  - Automatically generated values (very hard!)
  - User data (incomplete)
  - Study application and construct a set of values (feasible)
  - Tester-supplied inputs (feasible but expensive)

Web Application Input Validation



Check data

Client

Sensitive Data

Check data

Bad Data

- **Corrupts data base**
- **Crashes server**
- **Security violations**

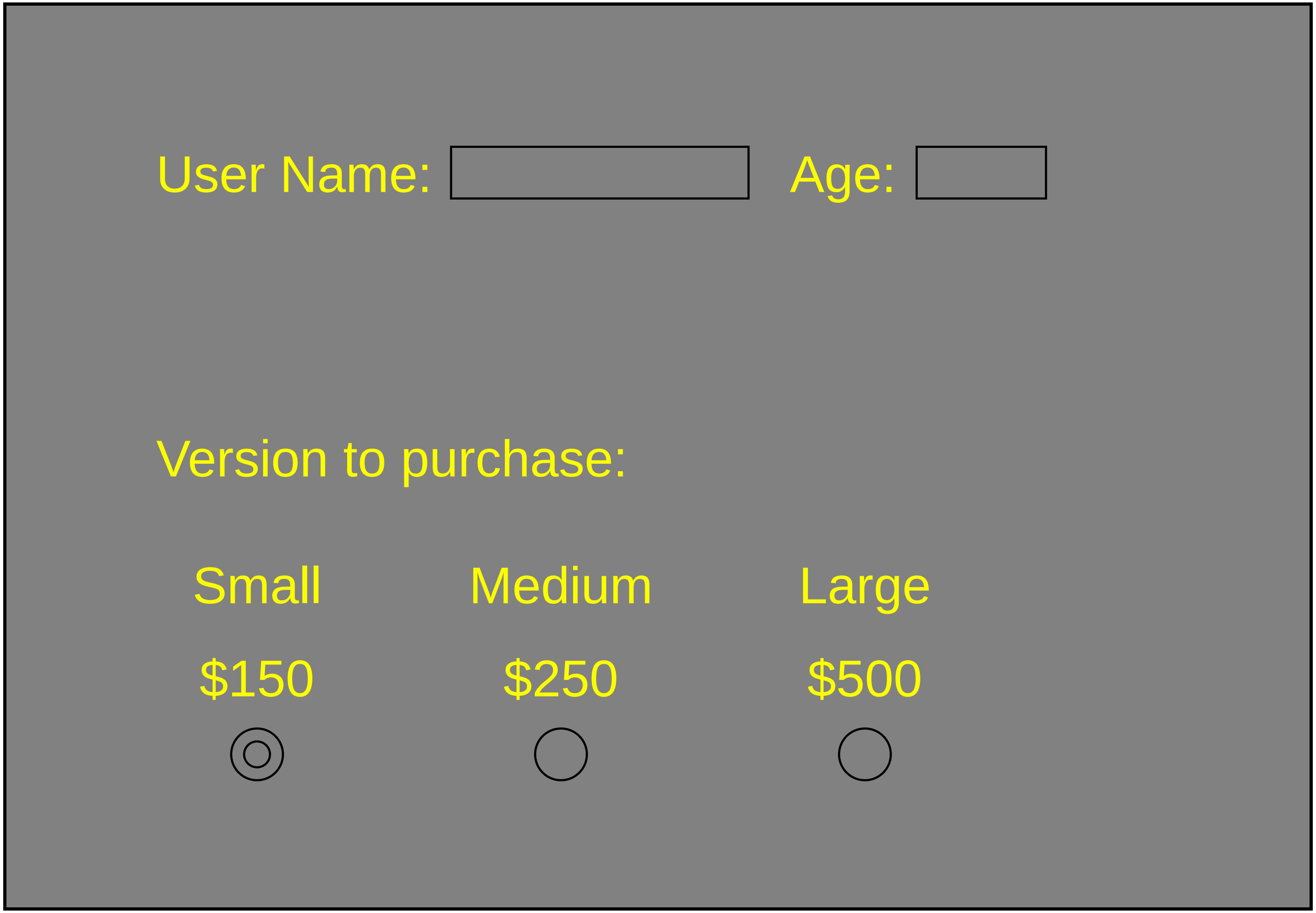
Server

***Malicious* Data**

Can "bypass" data checking

- “*bypass*” client-side constraint enforcement

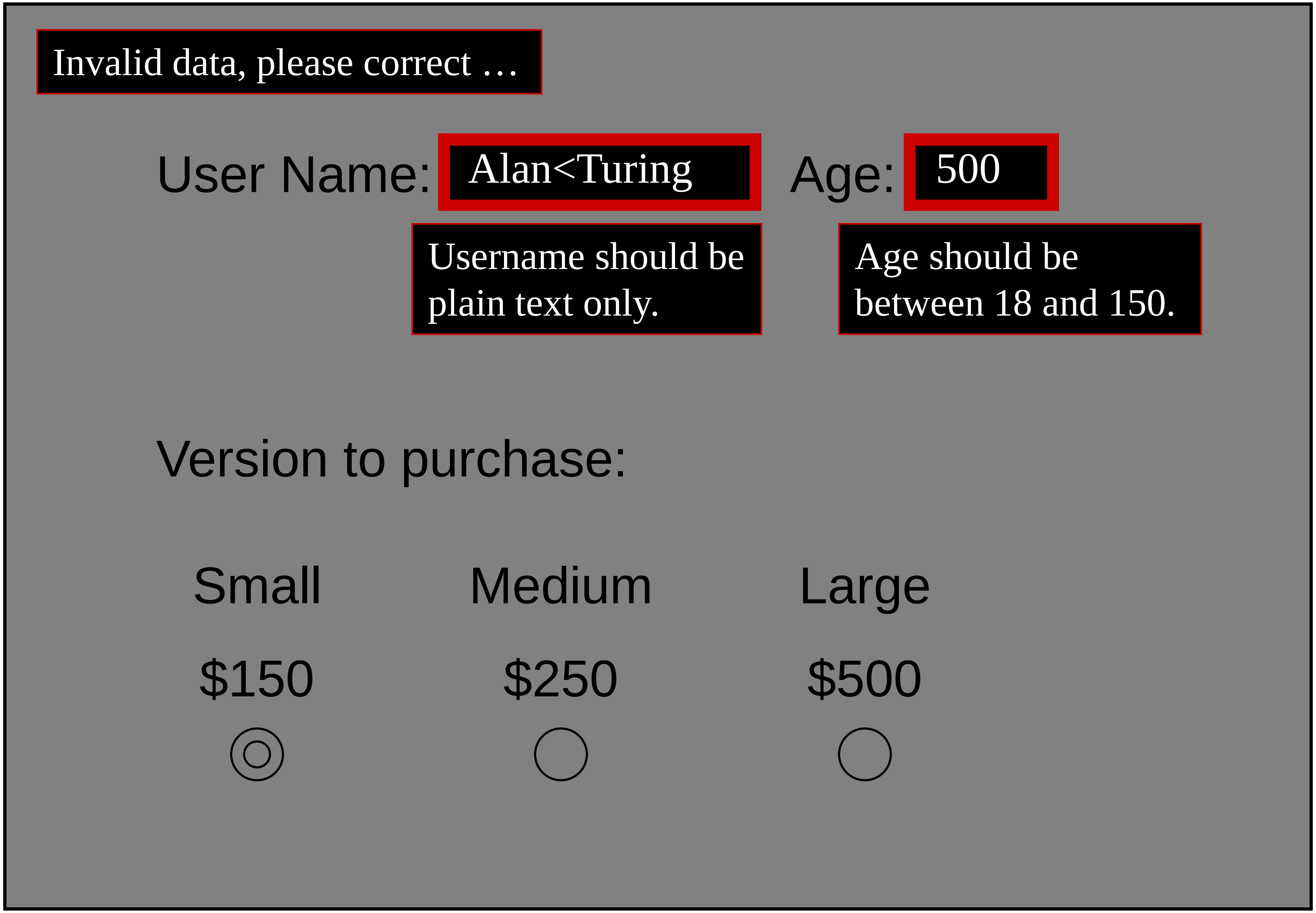
- Bypass testing constructs tests to intentionally violate constraints :
  - Eases test automation
  - Validates input validation
  - Checks robustness
  - Evaluates security



User Name: ▭ Age: ▭

Version to purchase:

Small	Medium	Large
$150	$250	$500
◉	○	○



Invalid data, please correct …

User Name: Alan<Turing

Username should be plain text only.

Age: 500

Age should be between 18 and 150.

Version to purchase:

Small	Medium	Large
$150	$250	$500
◉	○	○

Bypass Testing

- This example illustrates how users can “bypass” client-side constraint enforcement
- Bypass testing constructs tests to intentionally violate constraints
  - Eases test automation
  - Checks robustness
  - Evaluates security
- Preliminary results
  - Rules for constructing tests
  - Successfully found errors in numerous Web apps

Applying Bypass Testing

Validating input data on the client is like asking your opponent to hold your shield in a sword fight

- Analyze HTML to extract each form element
- Model constraints imposed by HTML and JavaScript
- Rules for data generation :
  - From client-side constraints
  - Typical security violations
  - Common input mistakes

Types of Client Input Validation

- Client side input validation is performed by HTML form controls, their attributes, and client side scripts that access DOM
- Validation types are categorized as HTML and scripting
  - HTML supports syntactic validation
  - Client scripting can perform both syntactic and semantic validation

HTML Constraints

- **Length *(max input characters)***
- **Value *(preset values)***
- **Transfer Mode *(GET or POST)***
- **Field Element *(preset fields)***
- **Target URL *(links with values)***

Scripting Constraints

- **Data Type *(e.g. integer check)***
- **Data Format *(e.g. ZIP code format)***
- **Data Value *(e.g. age value range)***
- **Inter-Value *(e.g. credit # + exp. date)***
- **Invalid Characters *(e.g. <,../,&)***

Example Client-Side Constraint Rules

- Violate size restrictions on strings
- Introduce values not included in static choices
  - Radio boxes
  - Select (drop-down) lists
- Violate hard-coded values
- Use values that JavaScripts flag as errors
- Change “transfer mode” (get, post, …)
- Change destination URLs

Example Server-Side Constraint Rules

- Data type conversion
- Data format validation
- Inter-field constraint validation
- Inter-request data fields (cookies, hidden)

Example Security Violation Rules

Potential Illegal Character	Symbol
Empty String	
Commas	**,**
Single and double quotes	**’ or ”**
Tag symbols	**Tag symbols < and >**
Directory paths	**.. ../**
Strings starting with forward slash	**/**
Strings starting with a period	**.**
Ampersands	**&**
Control character	**NIL, newline**
Characters with high bit set	**254 and 255**
Script symbols	**<javascript> or <vbscript>**

Test Value Selection

- Challenge:
  - How to automatically provide effective test values?
- *Semantic Domain Problem* (SDP)
  - Values within the application domain are needed
  - Enumeration of all possible test values is inefficient
- Possible Solutions
  - Random Values (ineffective – lots of junk)
  - Automatically generated values (very hard)
  - Taking values from session log files (feasible but incomplete)
  - Tester input (feasible)
- Tool (designed by Jeff Offutt & collegues) used an input domain created by parsing the interface and tester input

Real-World Examples

atutor.ca
- Atalker

demo.joomla.or
- Poll, Users

phpMyAdmin
- Main page,
- Set Theme,
- SQL Query,
- DB Stats

brainbench.com
- Submit Request Info, New user

myspace.com
- Events & Music Search

nytimes.com
- Us-markets

mutex.gmu.edu
- Login form

yahoo.com
- Notepad, Composer,
- Search reminder,
- Weather Search

barnesandnoble.com
- Cart manager,
- Book search/results

amazon.com
- Item dispatch,
- Handle buy

bankofamerica.com
- ATM locator, Site search

comcast.com
- Service availability

ecost.com
- Detail submit,
- Shopping cart control

google.com
- Froogle, Language tools

pageflakes.com
- Registration

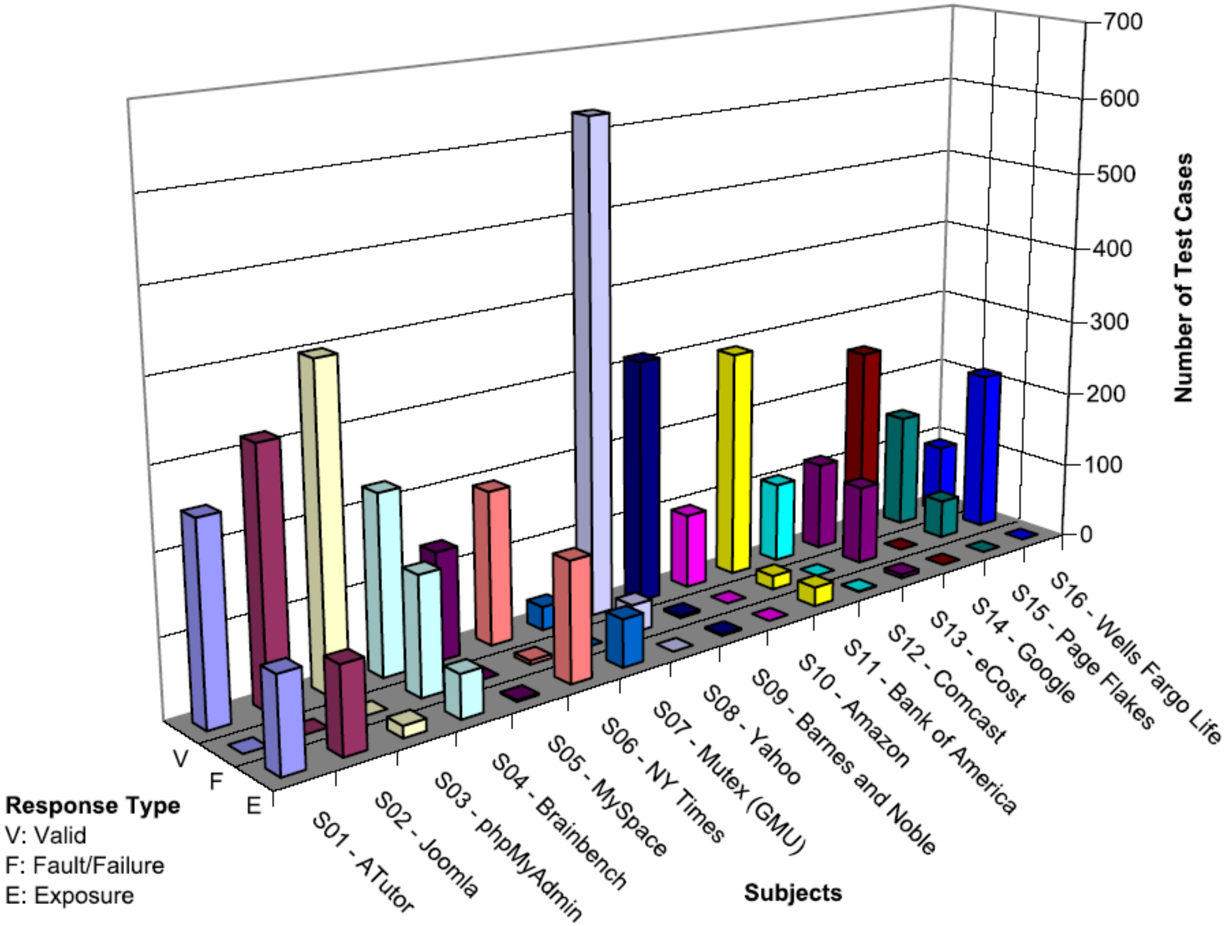
wellsfargolife.com
- Quote search

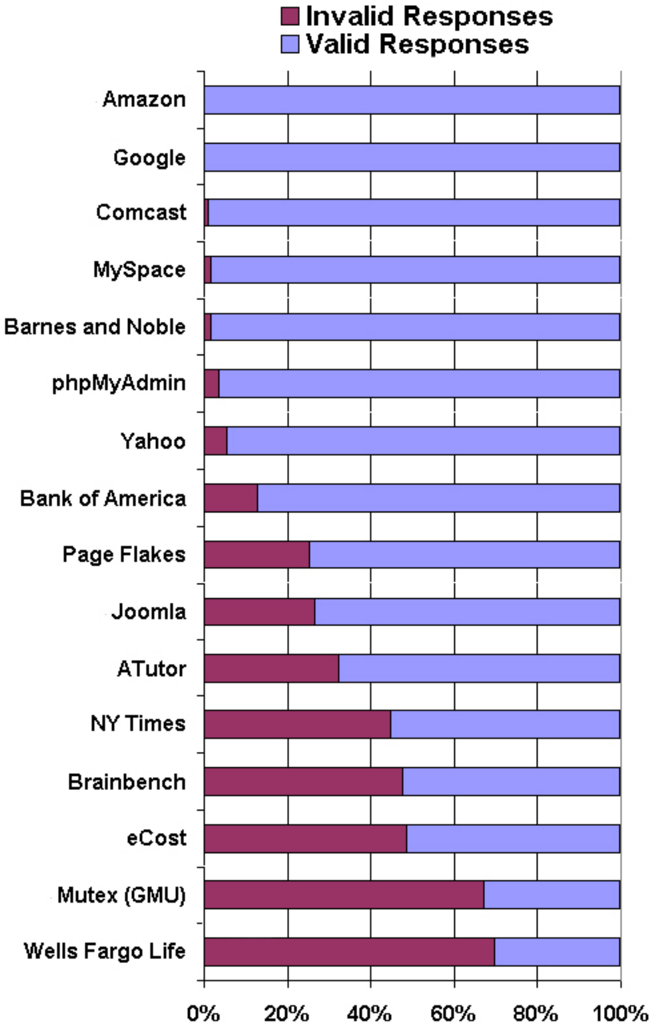
Pure black-box testing means no source (or permission) needed !

Output Checking

- (V) Valid Responses : invalid inputs are adequately processed by the server
  - **(V1) Server acknowledges the invalid request and provides an explicit message regarding the violation**
  - **(V2) Server produces a generic error message**
  - **(V3) Server apparently ignores the invalid request and produces an appropriate response**
  - **(V4) Server apparently ignores the request completely**
- (F) Faults & Failures : invalid inputs that cause abnormal server behavior *(typically caught by web server when application fails to handle the error)*
- (E) Exposure : invalid input is not recognized by the server and abnormal software behavior is exposed to the users
- These do not capture whether the valid responses corrupted data on the server

Results


700
600
500
400
300
200
100
0
Number of Test Cases
V
F
E
Response Type
V: Valid
F: Fault/Failure
E: Exposure
S01 - ATutor
S02 - Joomla
S03 - phpMyAdmin
S04 - Brainbench
S05 - MySpace
S06 - NY Times
S07 - Mutex (GMU)
S08 - Yahoo
S09 - Barnes and Noble
S10 - Amazon
S11 - Bank of America
S12 - Comcast
S13 - eCost
S14 - Google
S15 - Page Flakes
S16 - Wells Fargo Life
Subjects


Invalid Responses
Valid Responses
Amazon
Google
Comcast
MySpace
Barnes and Noble
phpMyAdmin
Yahoo
Bank of America
Page Flakes
Joomla
ATutor
NY Times
Brainbench
eCost
Mutex (GMU)
Wells Fargo Life
0%
20%
40%
60%
80%
100%

Outline

1. Overview
2. Testing Static Hyper Text Web Sites
3. Testing Dynamic Web Applications
    1. Client-side testing
    2. **Server-side testing**
4. Testing Web Services

Server-Side (White-Box) Testing

- If we have access to the source on the server, we can try to model the web software
- Many testing criteria on non-web software rely on a static control flow graph
  - Edge testing, data flow, logic coverage, …
  - Also slicing, change impact analysis, …
- The standard control flow graph cannot be computed for web applications !
- But all the pieces of the web pages and the programs are contained in the software presentation layer …

Modeling Web Applications

- Restricted to the presentation layer only
- Two levels of abstraction

1. Component Interaction Model (CIM)
   - Models individual components
   - Combines atomic sections
   - Intra-component
2. Application Transition Graph (ATG)
   - Each node is one CIM
   - Edges are transitions among CIMs
   - Inter-component

Atomic Sections

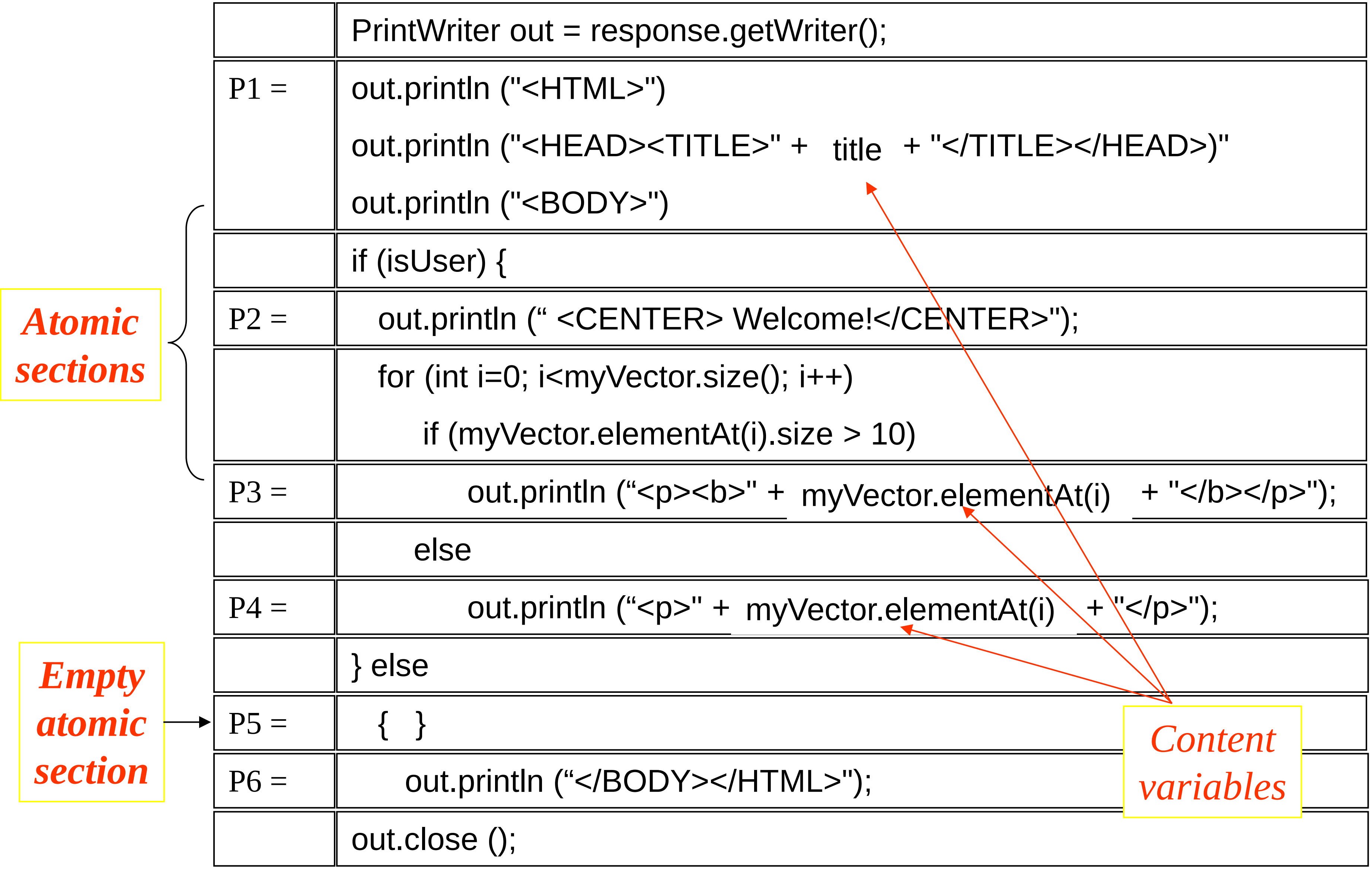
- An *atomic section (AtS) is a section of HTML* (possibly including scripting language routines such as JavaScript) that has the property that if part of the section is sent to a client, the entire section is
- This is called an “all-or-nothing property”
- Atomic sections are analogous to basic blocks in traditional programs (although the focus is on data presentation, not execution, and many executable statements are ignored)

Atomic Sections (2)

- The simplest AtS is a complete static HTML file
- Dynamically generated HTML pages are typically comprised of several atomic sections from a server program that generates HTML
- *Content variable* (CV): A program variable that provides data to an atomic section (or HTML page) but not structure
- Atomic sections may be empty

Atomic Sections :

HTML with *static structure* and *content variables*



	PrintWriter out = response.getWriter();
P1 =	out.println ("<HTML>") out.println ("<HEAD><TITLE>" + title + "</TITLE></HEAD>)" out.println ("<BODY>")
	if (isUser) {
P2 =	out.println (“ <CENTER> Welcome!</CENTER>");

	for (int i=0; i<myVector.size(); i++)
	if (myVector.elementAt(i).size > 10)

P3 =	out.println (“<p><b>" + myVector.elementAt(i) + "</b></p>");
	else
P4 =	out.println (“<p>" + myVector.elementAt(i) + "</p>");
	} else
P5 =	{ }
P6 =	out.println (“</BODY></HTML>");
	out.close ();

Atomic sections

Empty atomic section

Content variables

Component Expressions

- Atomic sections are combined to model dynamically generated web pages
- Four ways to combine:
  1. Sequence : $p1 \bullet p2$
  2. Selection : $(p1 \mid p2)$
  3. Iteration : $p1^*$
  4. Aggregation : $p1 \{p2\}$
     - *p2* is included inside of *p1*
- The previous example produces:

$$p \rightarrow p1 \bullet (p2 \bullet (p3 \mid p4)^* \mid p5) \bullet p6$$

- Composite sections can be <u>produced automatically</u>

Five types of transitions

1. Simple Link Transition : An HTML link (<A> tag)

– *($p \rightarrow c$):* Invoking an <*A*> link in *p* causes a transition from the client to a component *c* on the server

– If *p* has more than one <*A*> link and can thus invoke one of several static or dynamic pages, c_1, c_2, . . . , c_k, then the destination is represented as $c_1 \mid c_2 \mid . . . \mid c_k$

– *Note:* *p* and *q* are component expressions of atomic sections and c is a software component that generates HTML or *c* is an HTML file

Modeling Component Transitions (2)

Five types of transitions

1. Simple Link Transition : An HTML link (<A> tag)
2. Form Link Transition : Form submission link
3. Component Expression Transition : Execution of a software component causes a component expression to be sent to the client
4. Operational Transition : A transition out of the software’s control
   - Back button, Forward button, Refresh button, User edits the URL, Browser reloads from cache
5. Redirect Transition : Server side transition, invisible to user

Component Interaction Model (CIM)

- Each web software component is modeled as a quadruple *Component Interaction Model,*
- *CIM* = <*S, A,CE, T*>
  - Where *S is the start page,*
  - *A is the set of atomic sections,*
  - *CE is the* component expression, and
  - *T is a set of transitions*
- These sets are fixed and static, based on the source of the presentation layer software
- It is assumed that each software component has a unique start page

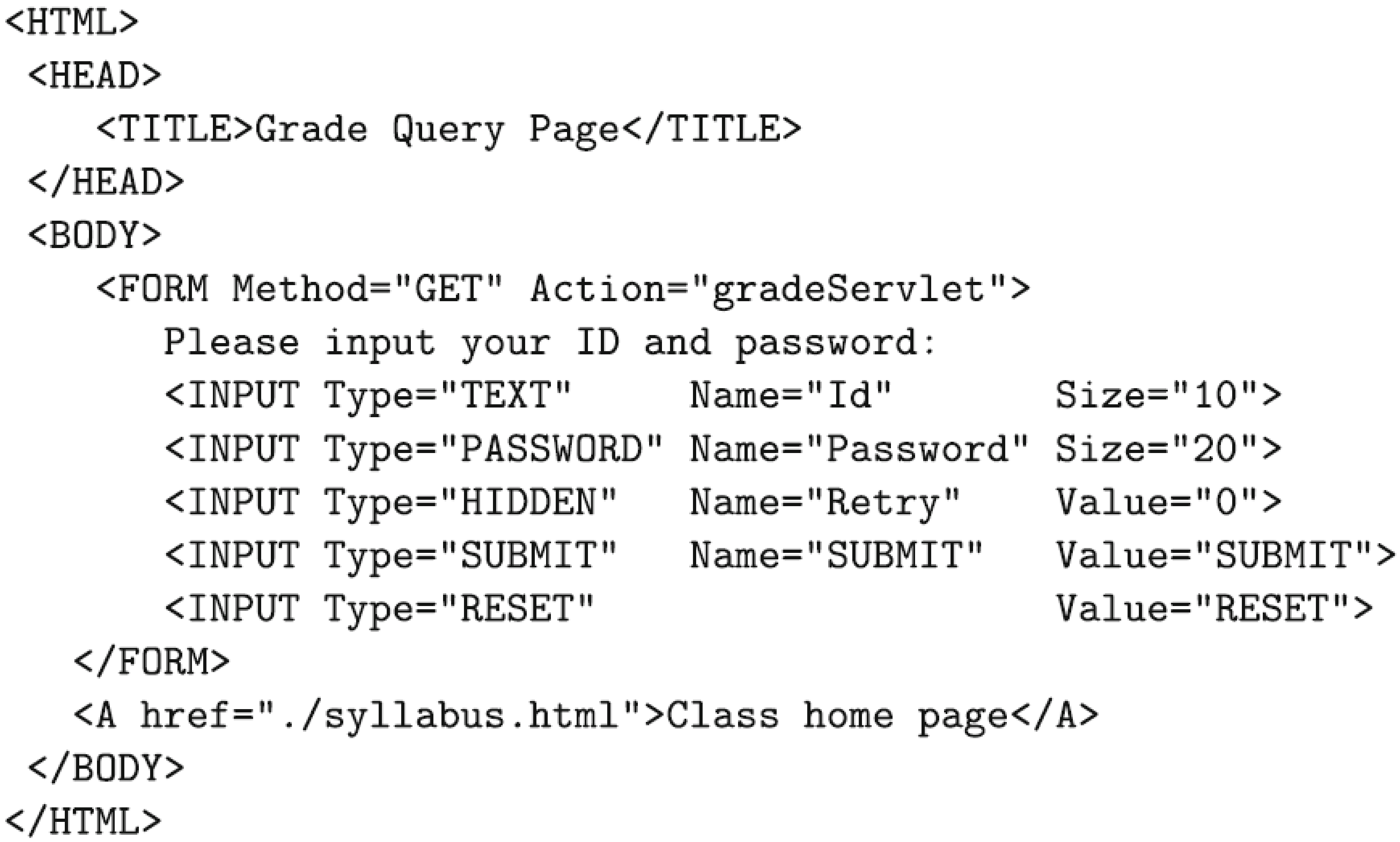
Component Interaction Model (CIM) – An example

- Example presented here is an HTML page that uses the Java servlet to provide online grade queries to students
- A student must access the main page first to enter an id and password
- Then a servlet validates the id and password; if successful, the servlet retrieves the grade information and sends it back to the student
- If unsuccessful, an error message is returned to the student asking the student to either retry or send an email to the instructor for further assistance.

Component Interaction Model (CIM) – An example (2)

- This small application includes a static HTML file, a query servlet, and another servlet that sends the email to the instructor (details about sendMail are omitted for brevity)
- The atomic sections for gradeServlet are marked in next slide gradeServlet uses three methods, Validate(), CourseName() and CourseGrade()
- These methods are part of the data content layer of the web application, not the presentation layer, and do not directly affect the response page that gradeServlet produces
- They generate data content, but do not effect the atomic sections, thus they are not necessary for modeling

HTML login page



```
<HTML>
 <HEAD>
    <TITLE>Grade Query Page</TITLE>
 </HEAD>
 <BODY>
    <FORM Method="GET" Action="gradeServlet">
       Please input your ID and password:
       <INPUT Type="TEXT"     Name="Id"       Size="10">
       <INPUT Type="PASSWORD" Name="Password" Size="20">
       <INPUT Type="HIDDEN"   Name="Retry"    Value="0">
       <INPUT Type="SUBMIT"   Name="SUBMIT"   Value="SUBMIT">
       <INPUT Type="RESET"                    Value="RESET">
   </FORM>
   <A href="./syllabus.html">Class home page</A>
 </BODY>
</HTML>
```

Component Interaction Model : gradeServlet

	ID = request.getParameter ("Id");
	passWord = request.getParameter ("Password");
	retry = request.getParameter ("Retry");
	PrintWriter out = response.getWriter();

P1 =	out.println (“<HTML> <HEAD><TITLE>" + title + "</TITLE></HEAD><BODY>)"
	if ((Validate (ID, passWord)) {
P2 =	out.println (“ <B> Grade Report </B>");
	for (int I=0; I < numberOfCourse; I++)
P3 =	out.println(“<p><b>" + courseName (I) + "</b>“ + courseGrade (I) + “</p>”);
	} else if (retry < 3) {

P4 =	retry++;
	out.println ("Wrong ID or wrong password");
	out.println ("<FORM Method=\“get\" Action=\"gradeServlet\">);
	out.println ("<INPUT Type=\“text\" Name=\"Id\" Size=10>");
	out.println ("<INPUT Type=\“password\" Name=\"Password\" Width=20>");
	out.println ("<INPUT Type=\“hidden\" Name=\"Retry\" Value=" + (retry) + ">");
	out.println ("<INPUT Type=\“submit\" Name=\“Submit\" Value=\“submit\">");
	out.println ("<A Href=\"sendMail\">Send mail to the professor</A>");

	} else if (retry < 3) {
P5 =	out.println (“<p>Wrong ID or password, retry limit reached. Good bye.") }
P6 =	out.println(“</BODY></HTML>");

CIM for gradeServlet

- ***S*** **= login.html**
- $A = \{p_1, p_2, p_3, p_4, p_5, p_6\}$
- CE **= gradeServlet =** $p_1 \bullet ((p_2 \bullet p_3^{*}) \mid p_4 \mid p_5) \bullet p_6$
- $T = \{$ ***login.html*** $\twoheadrightarrow$ ***gradeServlet [get, (Id, Password, Retry)],***

  gradeServlet.$p_4 \longrightarrow$ ***sendMail [get, ()],***

  gradeServlet.$p_4 \twoheadrightarrow$ ***gradeServlet [get, (Retry)]*** $\}$

Application Transition Graph (ATG)

- A higher level of abstraction is needed to model the entire web application
- The web *Application Transition Graph (ATG)* combines component interaction models to model an entire application
- In an ATG, nodes are web software components and edges are links and other types of transitions among the nodes

Application Transition Graph (2)

- Formally, a web application is modeled as a quadruple *ATG* = < Γ, Θ*,* Σ*,* α >
  - Γ : Finite set of web components
  - Θ : Set of transitions among web software components
    - Includes type of HTTP request and data
  - Σ : Set of variables that define the web application state
  - α : Set of start pages

ATG for gradeServlet

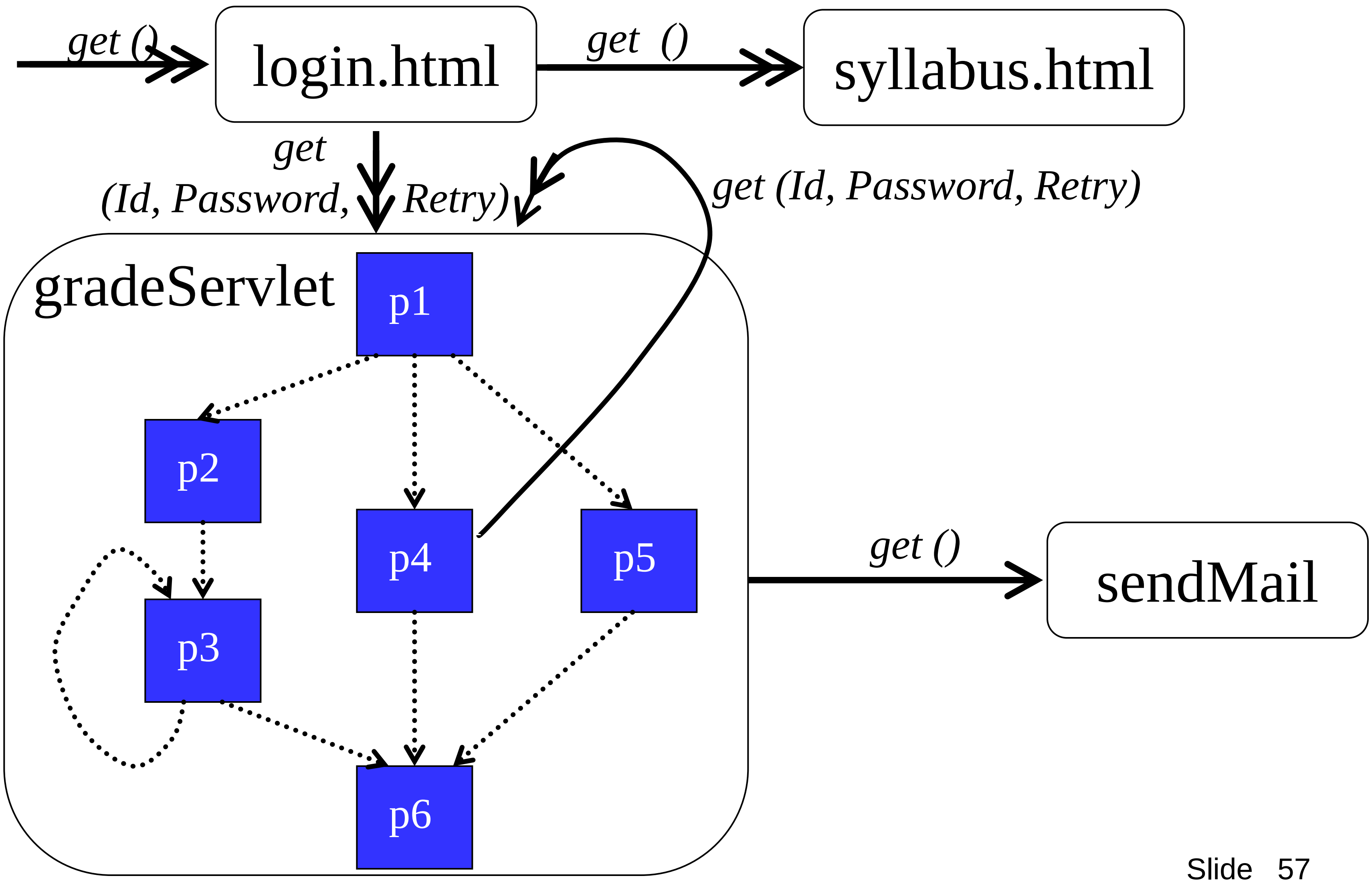
- **Γ = { login.html, gradeServlet, sendMail, syllabus.html }**
- **Θ = {*login.html*** $\longrightarrow$ ***syllabus.html [get, ()]*,**

  login.html $\twoheadrightarrow$ ***gradeServlet [get, (Id, Password, Retry)]*,**

  gradeServlet.p$_4$ $\longrightarrow$ ***sendMail [get, ()]*,**

  gradeServlet.p$_4$ $\twoheadrightarrow$ ***gradeServlet [get, (Retry)]* }**
- **Σ = *{ Id, Password, Retry }***
- **α = { login.html }**

ATG for gradeServlet



Atomic Section Modeling

- Atomic sections provide a fundamental mechanism to model Web applications presentation layer
- Can handle :
  - Distributed integration
  - Dynamically created HTML pages
  - Operational transitions
- Requires deep analysis of software source

Some Current Open Questions

- How to define data flow?
  - DU-pairs cannot be determined statically – uses cannot always be found
- Automatically generating ATG
- Issues not handled:
  - Session data
  - Multiple users
  - Concurrency
  - Input data
  - Output validation
  - Dynamic integration

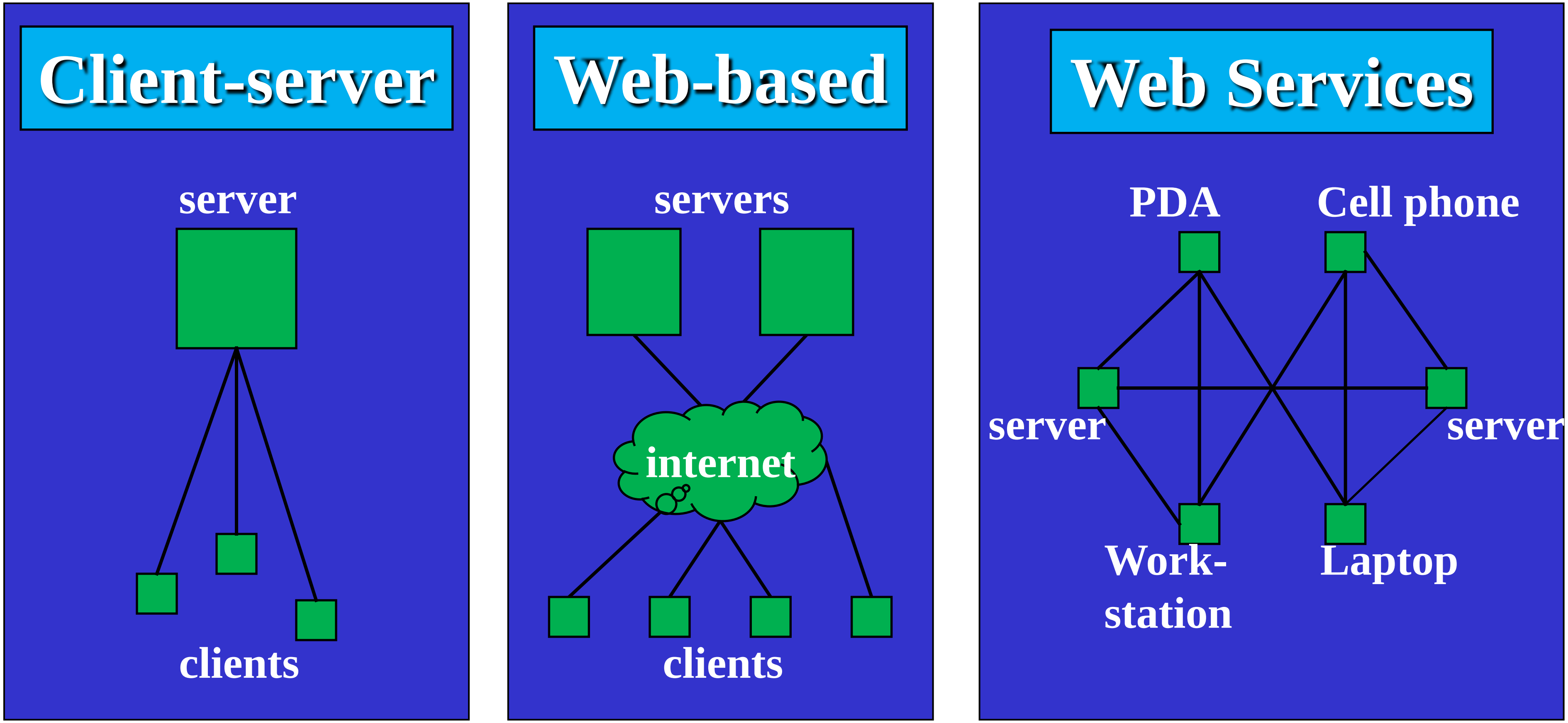
Outline

1. Overview
2. Static Hyper Text Web Sites
3. Dynamic Web Applications
    1. Client-side testing
    2. Server-side testing
4. **Testing Web Services**

Web Services

- A *Web Service* is a program that offers services over the Internet to other software programs
  - Internet-based
  - Uses SOAP and XML
  - Peer-to-peer communication
- Web service components can integrate dynamically, by finding other services during execution
- Web services transmit data that are formatted in XML

Web Service Architecture



Client-server

server

clients

Web-based

servers

internet

clients

Web Services

PDA

Cell phone

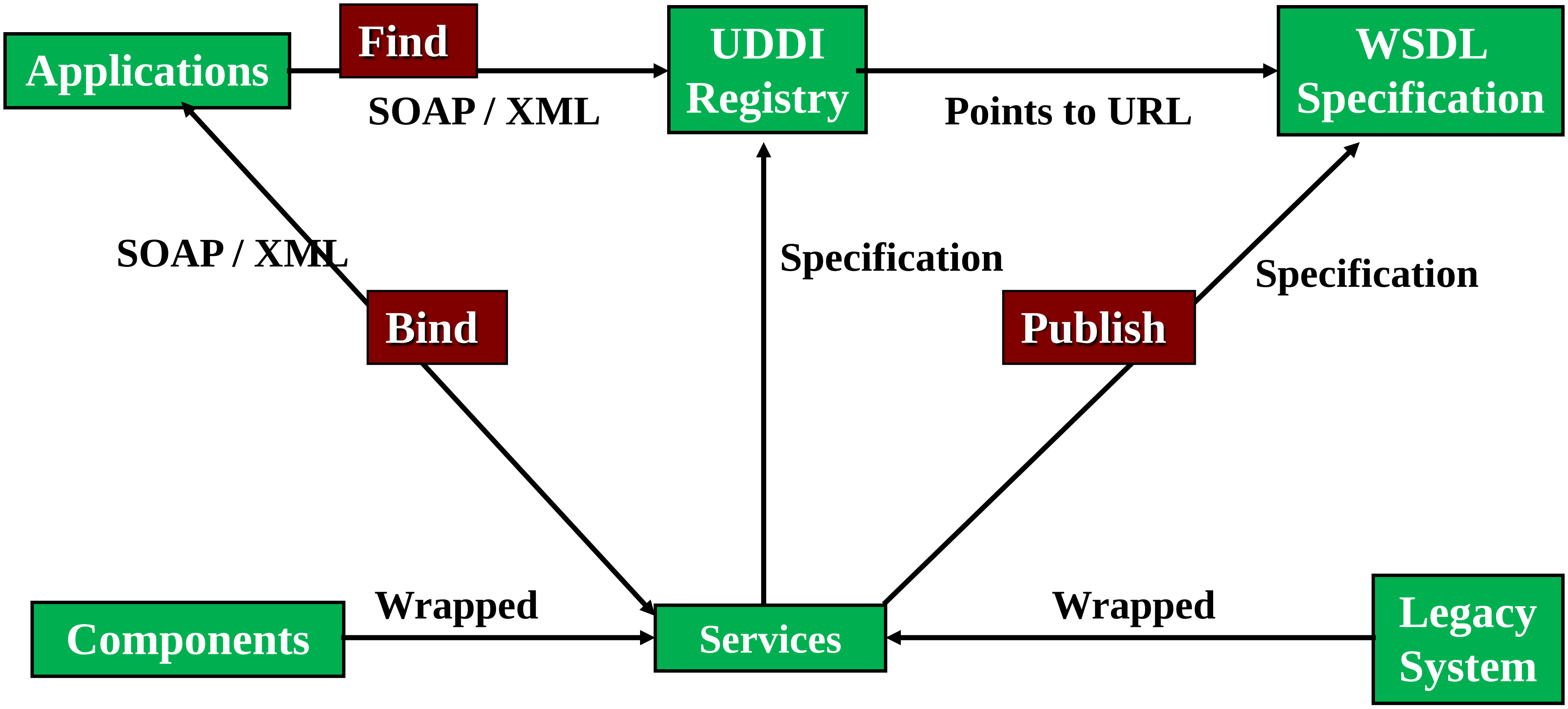
server

server

Work-
station

Laptop

Web Service Technologies



Difficulties of Testing Web Services

- Web services are always distributed
- Most “peer-to-peer” communication is between services published by different organizations
  - *Trust* is a major issue holding back the adoption of web services !
- Design and implementation are almost never available
- Structured messages are transmitted
- Most testing research so far has focused on messages
  - Syntax-based test criteria have been proposed for Web services

Conclusions

- The Web provides a new way to deploy software
- Web applications:
  - offer many advantages
  - use many new technologies
  - introduce fascinating new problems
- Web software engineering is just starting
- Two very useful techniques:
  - Atomic sections: A fundamental model
  - Bypass testing: Easy to automate – no source needed
- This is a very active research area

References

- Paul Ammann and Jeff Offutt, *Introduction to Software Testing*, Cambridge University Press, 2008
- Jeff Offutt, Ye Wu, *Modeling presentation layers of web applications for testing*, International Journal of Software and Systems Modeling, Springer, 2009