



Chapter 9: Application Design and Development



Database System Concepts

■ Chapter 1: Introduction

■ Part 1: Relational databases

- Chapter 2: Introduction to the Relational Model
- Chapter 3: Introduction to SQL
- Chapter 4: Intermediate SQL
- Chapter 5: Advanced SQL
- Chapter 6: Formal Relational Query Languages

■ Part 2: Database Design

- Chapter 7: Database Design: The E-R Approach
- Chapter 8: Relational Database Design
- [Chapter 9: Application Design](#)

■ Part 3: Data storage and querying

- Chapter 10: Storage and File Structure
- Chapter 11: Indexing and Hashing
- Chapter 12: Query Processing
- Chapter 13: Query Optimization

■ Part 4: Transaction management

- Chapter 14: Transactions
- Chapter 15: Concurrency control
- Chapter 16: Recovery System

■ Part 5: System Architecture

- Chapter 17: Database System Architectures
- Chapter 18: Parallel Databases
- Chapter 19: Distributed Databases

■ Part 6: Data Warehousing, Mining, and IR

- Chapter 20: Data Mining
- Chapter 21: Information Retrieval

■ Part 7: Specialty Databases

- Chapter 22: Object-Based Databases
- Chapter 23: XML

■ Part 8: Advanced Topics

- Chapter 24: Advanced Application Development
- Chapter 25: Advanced Data Types
- Chapter 26: Advanced Transaction Processing

■ Part 9: Case studies

- Chapter 27: PostgreSQL
- Chapter 28: Oracle
- Chapter 29: IBM DB2 Universal Database
- Chapter 30: Microsoft SQL Server

■ Online Appendices

- Appendix A: Detailed University Schema
- Appendix B: Advanced Relational Database Model
- Appendix C: Other Relational Query Languages
- Appendix D: Network Model
- Appendix E: Hierarchical Model



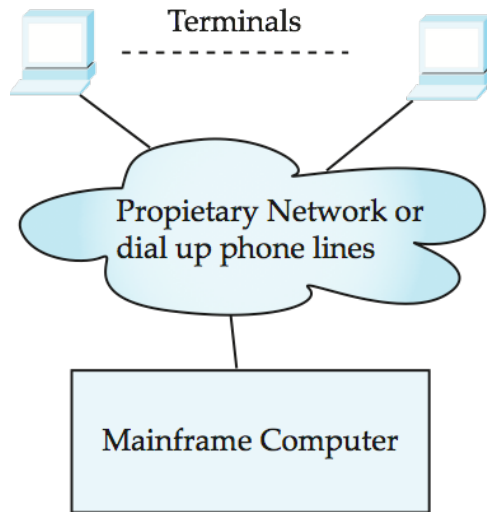
Chapter 9: Application Design and Development

- 9.1 Application Programs and User Interfaces
- 9.2 Web Fundamentals
- 9.3 Servlets and JSP
- 9.4 Application Architectures
- 9.5 Rapid Application Development
- 9.6 Application Performance
- 9.7 Application Security
- 9.8 Encryption and Its Applications

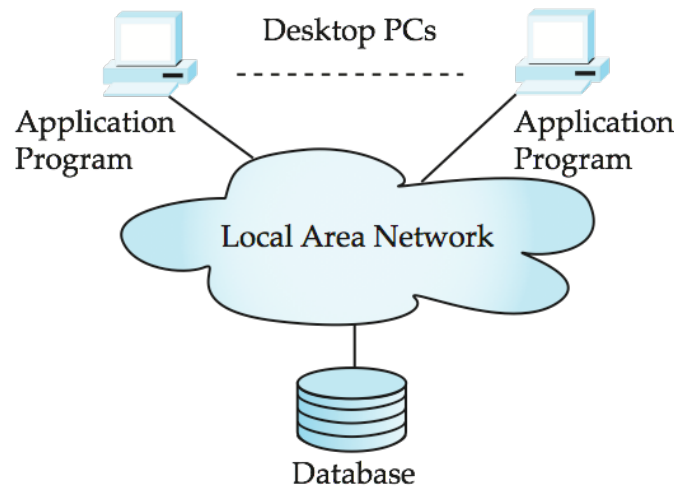


Application Architecture Evolution

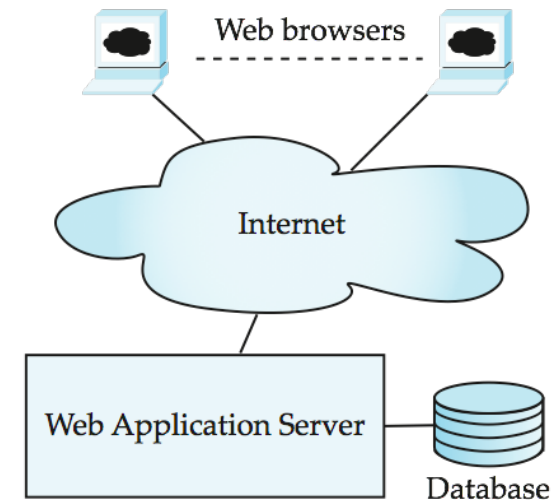
- Three distinct era's of application architecture
 - mainframe (1960's and 70's)
 - personal computer era (1980's)
 - Web era (1990's onwards)



(a) Mainframe Era



(b) Personal Computer Era



(c) Web era



Java Application with JDBC Code in Chapter 5

```
public static void JDBCexample(String userid, String passwd)
{ try {
    Class.forName ("oracle.jdbc.driver.OracleDriver"); // JDBC driver loading
    Connection conn = DriverManager.getConnection(
        "jdbc:oracle:thin:@db.yale.edu:2000:univdb", userid, passwd);

    Statement stmt = conn.createStatement();

    ResultSet rset = stmt.executeQuery(
        "select dept_name, avg (salary)
        from instructor
        group by dept_name");

    while (rset.next()) {
        System.out.println( rset.getString("dept_name") + " " + rset.getFloat(2) );
    }

    stmt.close();
    conn.close();

}

catch (Exception e) {   System.out.println("Exception : " + e); }

}
```



User



Web Browser: **Applet**



User



Web Application

WEB Interface
: **Servlet
(JSP)**

Monitoring DBMS
using **Trigger**

Limited Access
by **Authorization**

Protecting DBMS
by **Security**



JDBC API Calls



DBMS



Application Programs and User Interfaces

- Most database users do *not* use a query language like SQL
- An application program acts as **the intermediary** between users and the database
 - Applications split into
 - ▶ front-end
 - ▶ middle layer
 - ▶ Backend
- **Front-end:** User interface
 - Forms
 - Graphical user interfaces
 - Many interfaces are Web-based
- **Midle Layer:** Application Server + Web Server or Web-Application Server
- **Backend:** Database server



Web Interface

- Web browsers have become the **de-facto standard user interface** to databases
 - Enable large numbers of users to access databases from anywhere
 - Javascript, Flash and other scripting languages run in browser, but are **downloaded transparently**
 - Examples: banks, airline and rental car reservations, university course registration and grading, an so on.
- The WWW is **a distributed information system based on hypertext**.
- Most Web documents are hypertext documents formatted via the **HyperText Markup Language (HTML)**
- HTML documents contain
 - text along with font specifications, and other formatting instructions
 - hypertext links to other documents, which can be associated with regions of the text.
 - **forms**, enabling users to enter data which can then be sent back to the Web server



Web Interfaces to Database

■ Dynamic generation of documents

- Limitations of static HTML documents
 - ▶ Cannot customize fixed Web documents for individual users.
 - ▶ Problematic to update Web documents, especially if multiple Web documents replicate data.
- Solution: Generate Web documents **dynamically** from data stored in a database.
 - ▶ Can tailor the display based on user information stored in the database.
 - E.g., tailored ads, tailored weather and local news, ...
 - ▶ Displayed information is up-to-date, unlike the static Web pages
 - E.g., stock market information, ..



Chapter 9: Application Design and Development

- 9.1 Application Programs and User Interfaces
- 9.2 Web Fundamentals
- 9.3 Servlets and JSP
- 9.4 Application Architectures
- 9.5 Rapid Application Development
- 9.6 Application Performance
- 9.7 Application Security
- 9.8 Encryption and Its Applications



Uniform Resources Locators

- In the Web, pointing to a file is provided by **Uniform Resource Locators (URLs)**.
- URL example:
<http://www.acm.org/sigmod>
 - The first part indicates how the document is to be accessed (http: the Hyper Text Transfer Protocol.)
 - The second part gives the unique name of a machine on the Internet.
 - The rest of the URL identifies the document within the machine.
- The local identification can be:
 - ▶ **The path name** of a file on the machine, or
 - ▶ **An identifier (path name) of a program, plus arguments** to be passed to the program
 - E.g., <http://www.google.com/search?q=silberschatz>



HTML and HTTP

- HTML provides formatting, hypertext link, and image display features
 - including tables, stylesheets (to alter default formatting), etc.
- HTML is a collection of built-in tags
- HTML also provides tags for **input features** (accepting data and events from the user)
 - ▶ Select from a set of options
 - Pop-up menus, radio buttons, check lists
 - ▶ Enter values
 - Text boxes
 - Filled in input sent back to the server, **to be acted upon by an executable** at the server
- HyperText Transfer Protocol (HTTP) used for communication **with the Web server**



Sample HTML Source Text

<html>

<body>

<table border>

<tr> <th>ID</th> <th>Name</th> <th>Department</th> </tr>

<tr> <td>00128</td> <td>Zhang</td> <td>Comp. Sci.</td> </tr>

....

</table>

ID	Name	Department
00128	Zhang	Comp. Sci.
12345	Shankar	Comp. Sci.
19991	Brandt	History

<form action="PersonQuery" method=get>

Search for:

<select name="persontype">

<option value="student" selected> Student </option>

<option value="instructor"> Instructor </option>

</select>

Name: <input type="text" size=20 name="name">

<input type="submit" value="submit">

</form>

</body>

</html>

Search for:

Name:



Web Servers

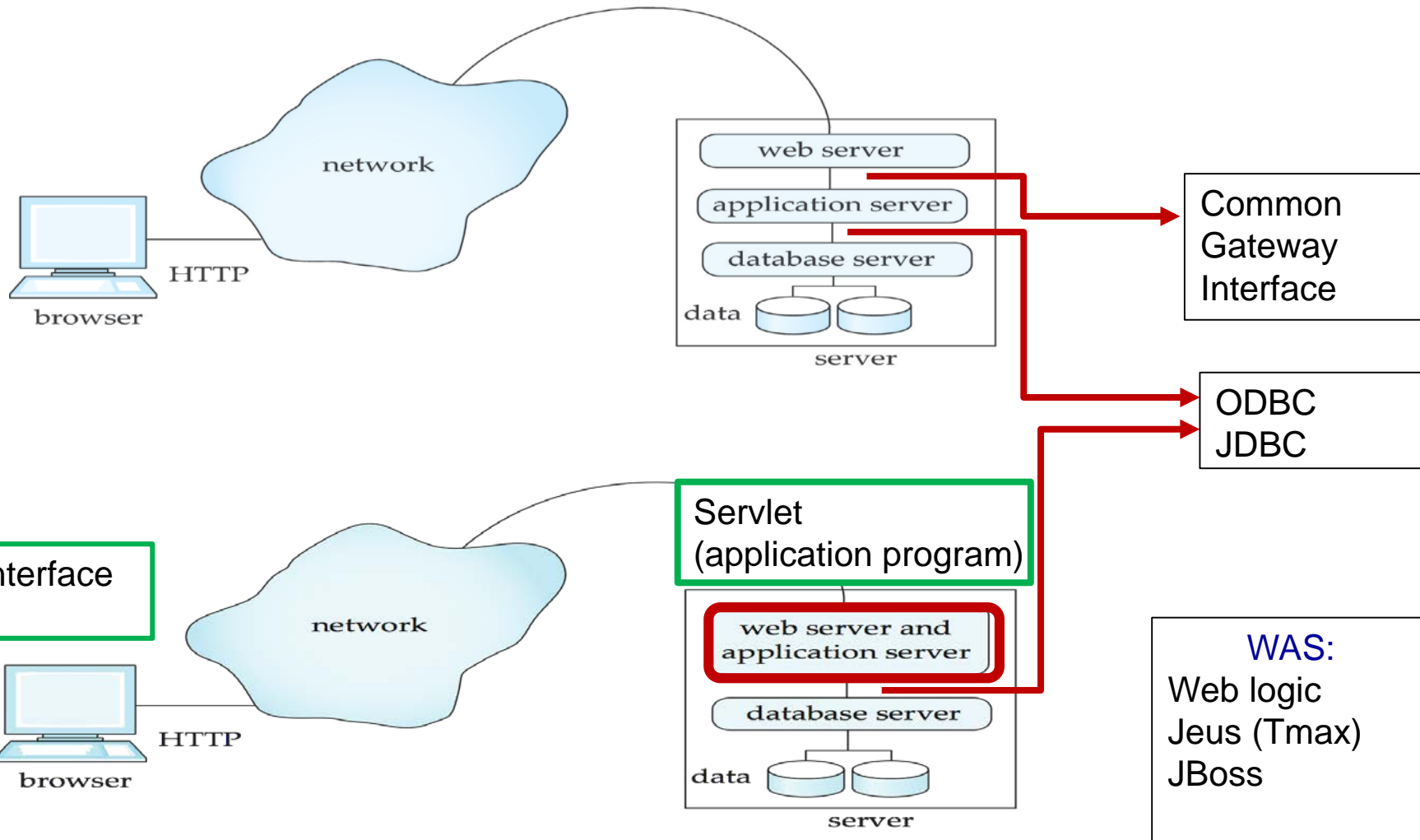
- A Web server can easily serve as a **front end** to a variety of information services.
 - **The document name** in a URL may identify an executable program, that, when run, generates a HTML document.
 - When an HTTP server receives a request for such a document, it executes the program, and sends back **the HTML document that is generated**.
 - The Web client can pass **extra arguments** with the name of the document.
- The Web browser provides a graphical user interface to the information service.
- To install a new service on the Web, one simply needs to create and install an executable code that provides that service.

- 3-layer Web Application Architecture
 - **Common Gateway Interface (CGI)**: a standard interface between **web server and application server**
- 2-layer Web Application Architecture
 - Application program runs within the web server which we call “WAS” (Web Application Server)



2-Layer & 3-Layer Web Architecture

- Multiple levels of indirection have overheads





HTTP and Sessions

- The HTTP protocol is **connectionless**
 - That is, once the server replies to a request, the server closes the connection with the client, and forgets all about the request
 - In contrast, **Unix logins, and JDBC/ODBC connections** stay connected until the client disconnects
 - ▶ retaining user authentication and other information
 - Motivation: **reduces load on server**
 - ▶ operating systems have tight limits on number of open connections on a machine
- Information services need session information
 - E.g., user authentication should be done only once per session
- Solution: use a **cookie**



Sessions and Cookies

- A **cookie** is a small piece of text containing identifying information
 - Sent by server to browser
 - ▶ Sent on first interaction, to identify session
 - Sent by browser to the server that created the cookie on further interactions
 - ▶ part of the HTTP protocol
 - Server saves information about cookies it issued, and can use it when serving a request
 - ▶ E.g., **authentication information**, and **user preferences**
- Cookies can be stored permanently or for a limited time



Chapter 9: Application Design and Development

- 9.1 Application Programs and User Interfaces
- 9.2 Web Fundamentals
- 9.3 Servlets and JSP
- 9.4 Application Architectures
- 9.5 Rapid Application Development
- 9.6 Application Performance
- 9.7 Application Security
- 9.8 Encryption and Its Applications



Servlets

- **Java Servlet** specification defines **an API** for communication between **the Web-Application Server** and **application program** running in the server
 - E.g., methods to get parameter values from Web forms, and to send HTML text back to client
- Application program (also called **a servlet**) is loaded into the server
 - **Each request spawns a new thread in the server**
 - The thread is closed once the request is serviced
- Servlet API provides a **getSession()** method
 - Sets a cookie on first interaction with browser, and uses it to identify session on further interactions
 - Provides methods to store and look-up per-session information
 - ▶ E.g. user name, preferences, ..
- **Servlet API calls and JDBC API calls are mixed in a servlet program**



HTML code with a form tag with PersonQuery in the Client Side Brower

```
<form action="PersonQuery" method=get>
```

Search for:

```
<select name="persontype">
```

```
<option value="student" selected>Student </option>
```

```
<option value="instructor"> Instructor </option>
```

```
</select> <br>
```

Name: <input type=text size=20 name="name">

```
<input type=submit value="submit">
```

```
</form>
```

```
</body>
```

```
</html>
```

Search for:

Name:

- HTML로만 구성된 Client-side application program
- HTML 안에 Java Applet이나 JavaScript등으로 필요한 client-side programming을 할수 있다
- Server-side에 있는 PersonQuery servlet을 match해주는 역할은 Web Application Server (WAS)이 한다



PersonQueryServlet Code in the Server Side (Java Servlet)

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class PersonQueryServlet extends HttpServlet {
    public void doGet (HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HEAD><TITLE> Query Result</TITLE></HEAD>");
        out.println("<BODY>");
        ..... BODY OF SERVLET (next slide) ...
        out.println("</BODY>");
        out.close();
    }
}
```



PersonQueryServlet Code (cont.)

```
String persontype = request.getParameter("persontype");
String number = request.getParameter("name");
if(persontype.equals("student")) {
    ... code to find students with the specified name ...
    ... using JDBC to communicate with the database ..
    out.println( "<table BORDER COLS=3>" );
    out.println( " <tr> <td>ID</td> <td>Name: </td>" + " <td>Department</td> </tr>" );
    for(... each result ...){
        ... retrieve ID, name and dept name
        ... into variables ID, name and deptname
        out.println( "<tr> <td>" + ID + "</td>" + "<td>" + name + "</td>" + "<td>" + deptname
            + "</td></tr>" );
    };
    out.println( "</table>" );
}
else {
    ... as above, but for instructors ...
}
```

Client-side의 request를 받아서
JDBC call을 DBMS에 pass하고
DBMS에서 return된 data를 HTML에 담아서
Client-side Browser로 pass한다



Servlet Sessions

- Servlet API supports **handling of sessions**
 - Sets a cookie on first interaction with browser, and uses it to identify session on further interactions
- To check if session is already active:
 - `if (request.getSession(false) == true)`
 - ▶ `then` existing session
 - ▶ `else` .. redirect to authentication page
 - authentication page
 - ▶ check login/password
 - ▶ `request.getSession(true)`: creates new session
- Store/retrieve attribute value pairs for a particular session
 - `session.setAttribute("userid", userid)`
 - `session.getAttribute("userid")`



Servlet Support in WAS

- Servlets run inside application servers (WAS) such as
 - [Apache Tomcat](#), [Glassfish](#), [Jboss](#), [Jeus \(Tmax\)](#)
 - [BEA Weblogic](#), IBM WebSphere and Oracle Application Servers
- [Application servers \(WAS\) support](#)
 - deployment and monitoring of servlets
 - Java 2 Enterprise Edition (J2EE) platform
 - ▶ supporting objects
 - ▶ parallel processing across multiple application servers
 - ▶ etc



Server-Side Scripting

- **Server-side scripting** simplifies the task of connecting a database to the Web
 - Define an HTML document with embedded executable code/SQL queries.
 - Input values from HTML forms can be used directly in the embedded code/SQL queries.
 - When the document is requested, the Web server executes the embedded code/SQL queries to generate the **actual HTML document**.
- Numerous server-side scripting languages
 - **Web-programming purpose script language**
 - ▶ JSP (JavaServer Page, 1999)
 - ▶ ASP (ActiveServer Page, 2000)
 - ▶ PHP (*PHP: Hypertext Preprocessor*, 1995)
 - ▶ ColdFusion's Markup Language (cfml), Jscript, and so on
 - **General purpose scripting languages**
 - ▶ Python, VBScript, Perl, Python



JSP (JavaServer Pages)

- A JSP page with embedded Java code (very simple for programmer)

```
<html>
```

```
<head> <title> Hello </title> </head>
```

```
<body>
```

```
<% if (request.getParameter("name") == null)
    { out.println("Hello World"); }
    else { out.println("Hello, " + request.getParameter("name")); }
%>
```

```
</body>
```

```
</html>
```

- When a JSP page is requested by a browser
 - WAS first executes the Java code inside the red box
 - And generates HTML output from the JSP page
 - The generated HTML output is sent back to the browser

- JSP allows new tags to be defined in tag libraries
 - to build rich user interfaces such as paginated display of large datasets



PHP (PHP: Hypertext Preprocessor)

- PHP is widely used for Web server scripting
- Extensive libraries including for database access using ODBC

```
<html>
```

```
<head> <title> Hello </title> </head>
```

```
<body>
```

```
<?php  if (!isset($_REQUEST['name']))
```

```
    { echo "Hello World"; }
```

```
    else { echo "Hello, " + $_REQUEST['name']; }
```

```
?>
```

```
</body>
```

```
</html>
```



Client Side Scripting

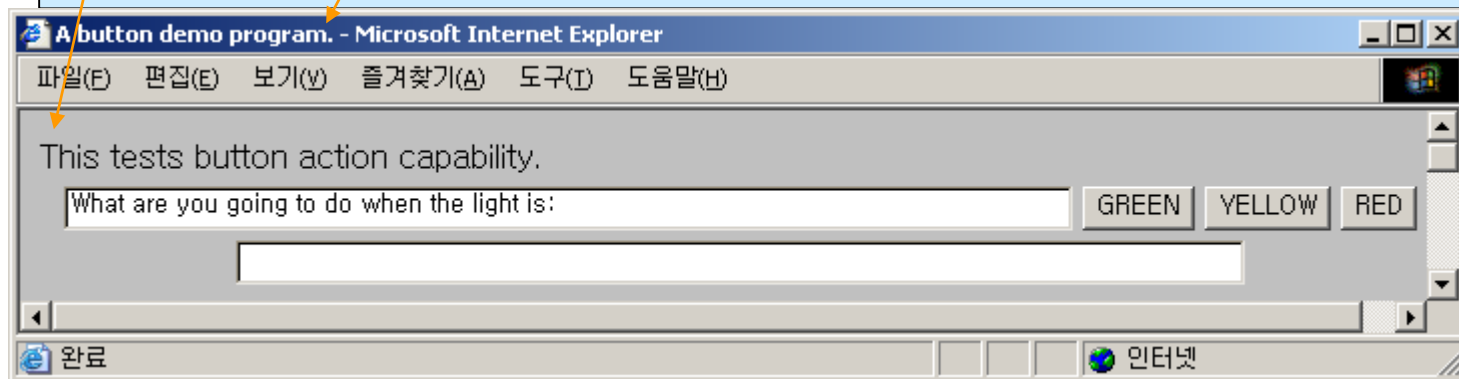
- Browsers can fetch certain scripts (**client-side scripts**) or programs along with documents, and execute them in “**safe mode**” at the client site
 - Javascript
 - Applets
 - Macromedia Flash and Shockwave (for animation/games)
 - VRML
- **Client-side scripts/programs** allow **documents to be active**
 - E.g., animation by executing programs at the local site
 - E.g., ensure that values entered by users satisfy some correctness checks
 - Permit flexible interaction with the user.
 - ▶ Executing programs at the client site speeds up interaction by avoiding many round trips to server

Sample Java Applet (1)

- Show **Green, Red, Yello** button on the screen
- If the user clicks any button, trigger the action method by showing the corresponding message ("**stop**", "**go**", "**ready to stop**")

TrafficLight.htm

```
<HTML>
<HEAD> <TITLE> A button demo program. </TITLE> </HEAD>
<BODY>
  This tests button action capability.
  <APPLET code="TrafficLight.class" WIDTH=700 HEIGHT=325> </APPLET>
</BODY>
</HTML>
```



Sample Java Applet (2)

```
import java.awt.*;
import java.awt.event.*;

public class TrafficLight extends java.applet.Applet
    implements ActionListener
{
    TextField  m1, m2;
    Button     b1, b2, b3;

    public void init ()
    {
        m1 = new TextField(80);
        m1.setText("What are you going to do when the light is:");
        b1 = new Button("GREEN");
        b2 = new Button("YELLOW");
        b3 = new Button("RED");
        m2 = new TextField(80);
        add(m1) ;
        add(b1) ;
        add(b2) ;
        add(b3) ;
        add(m2) ;
        b1.addActionListener(this);
        b2.addActionListener(this);
        b3.addActionListener(this);
    }
}
```

```
public void actionPerformed(ActionEvent event)
{
    Object cause = event.getSource();

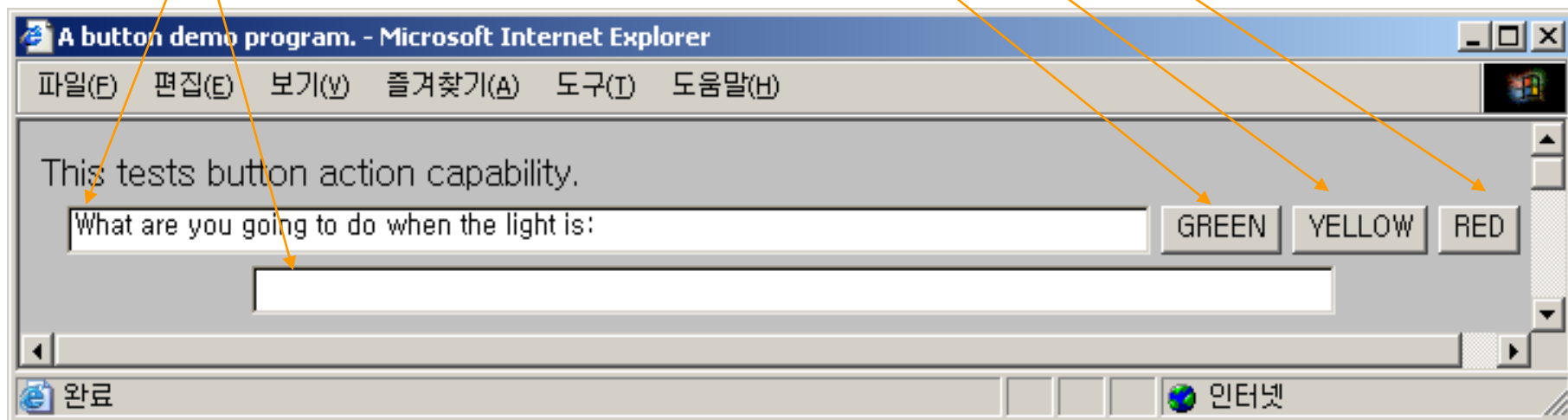
    if (cause == b1)
    {
        m2.setText("Keep on rolling.");
    }
    if (cause == b2)
    {
        m2.setText("Step on it!  You can make it!");
    }
    if (cause == b3)
    {
        m2.setText("I suppose you'll have to stop.");
    }
}
}
```

TrafficLight.java

Sample Java Applet (3)

public class **TrafficLight** extends java.applet.Applet
implements **ActionListener**

```
{  
    TextField m1, m2;  
    Button b1, b2, b3;  
    public void init ()  
    { m1 = new TextField(80);  
      m1.setText("What are you going to do when the light is:");  
      b1 = new Button("GREEN");  
      b2 = new Button("YELLOW");  
      b3 = new Button("RED");  
      m2 = new TextField(80);  
      add(m1) ;  
      add(b1) ;  
      add(b2) ;  
      add(b3) ;  
      add(m2) ;  
    }
```





Client Side Scripting and Security

- Security mechanisms needed to ensure that malicious scripts do not cause damage to the client machine
 - Easy for limited capability scripting languages
 - Harder for general purpose programming languages like Java

- E.g., Java's security system ensures that the Java applet code does not make any system calls directly
 - Disallows dangerous actions such as file writes
 - Notifies the user about potentially dangerous actions, and allows the option to abort the program or to continue execution.



Javascript

- Javascript very widely used
 - forms basis of new generation of Web applications (called Web 2.0 applications) offering rich user interfaces
- Javascript functions can
 - check input for validity
 - modify the displayed Web page, by altering the underlying **document object model (DOM)** tree representation of the displayed HTML text
 - communicate with a Web server to fetch data and modify the current page using fetched data, without needing to reload/refresh the page
 - ▶ forms basis of AJAX technology used widely in Web 2.0 applications
 - ▶ E.g. on selecting a country in a drop-down menu, the list of states in that country is automatically populated in a linked drop-down menu



Javascript inside HTML

- Example of Javascript used to validate form input

```
<html>
<head>
  <script type = "text/javascript">
    function validate() {
      var credits = document.getElementById("credits").value;
      if ( isNaN(credits) || credits<=0 || credits>=16) {
        alert("Credits must be a number greater than 0 and less than 16");
        return false
      }
    }
  </script>
</head>
<body>
  <form action= "createCourse" onsubmit="return validate()">
    Title: <input type="text" id="title" size="20"><br />
    Credits: <input type="text" id="credits" size="2"><br />
    <Input type="submit" value="Submit">
  </form>
</body>
</html>
```



Chapter 9: Application Design and Development

- 9.1 Application Programs and User Interfaces
- 9.2 Web Fundamentals
- 9.3 Servlets and JSP
- 9.4 Application Architectures
- 9.5 Rapid Application Development
- 9.6 Application Performance
- 9.7 Application Security
- 9.8 Encryption and Its Applications



Application Architectures

■ Application layers

● **Presentation or user interface** layer

▶ **model-view-controller (MVC) architecture**

- **model**: business logic
- **view**: presentation of data, depends on display device
- **controller**: receives events, executes actions, and returns a view to the user

● **business-logic** layer

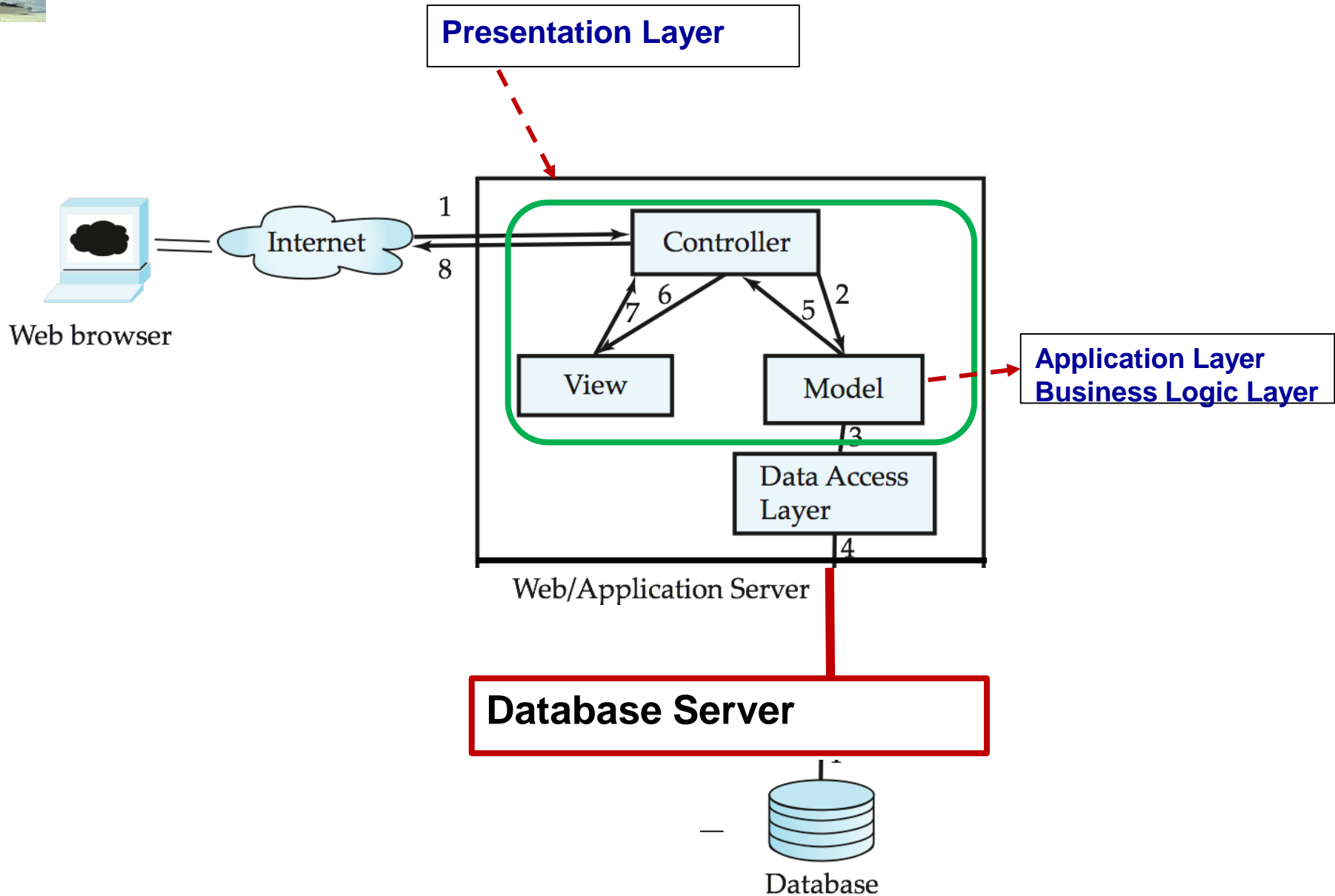
- ▶ provides high level view of data and actions on data
 - often using an object data model
- ▶ hides details of data storage schema

● **data access** layer

- ▶ interfaces between business logic layer and the underlying database
- ▶ provides mapping from object model of business layer to relational model of database



Fig 9.11: Application Architecture





Business Logic Layer

- Provides abstractions of entities
 - e.g. students, instructors, courses, etc
- Enforces **business rules** for carrying out actions
 - E.g. student can enroll in a class only if she has completed prerequisites, and has paid her tuition fees
- Supports **workflows** which define how a task involving multiple participants is to be carried out
 - E.g. how to process application by a student applying to a university
 - Sequence of steps to carry out task
 - Error handling
 - ▶ e.g. what to do if recommendation letters not received on time
 - Workflows discussed in Section 26.2



The Data-Access Layer

(Object-Relational Mapping)

- Most application code written in OOPL (Java, Python), while storing data in a traditional relational database
- Schema designer has to provide a mapping between object data and relational schema
 - e.g. Java class *Student* mapped to relation *student*, with corresponding mapping of attributes
 - An object can map to multiple tuples in multiple relations
- Application opens a session, which connects to the database
- Objects can be created and saved to the database using `session.save(object)`
 - mapping used to create appropriate tuples in the database
- Query can be run to retrieve objects satisfying specified predicates



Object-Relational Mapping Systems

- The **Hibernate ORM** object-relational mapping system is widely used ([open source](#))
 - public domain system, runs on a variety of database systems
 - supports a query language that can express complex queries involving joins
 - ▶ translates queries into SQL queries
 - allows relationships to be mapped to sets associated with objects
 - ▶ e.g. courses taken by a student can be a set in Student object
 - See book for Hibernate code example

- The **Entity Data Model** developed by [Microsoft](#)
 - provides an entity-relationship model directly to application
 - maps data between entity data model and underlying storage, which can be relational
 - **Entity SQL** language operates directly on Entity Data Model



Web Services

- Allow data on Web to be accessed using **remote procedure call mechanism**
- Two approaches are widely used
 - **Representation State Transfer (REST)**: allows use of standard HTTP request to a URL to execute a request and return data
 - ▶ returned data is encoded either in **XML**, or in **JavaScript Object Notation (JSON)**
 - **Big Web Services**:
 - ▶ uses **XML representation** for sending request data, as well as for returning results
 - ▶ standard protocol layer built on top of HTTP
 - ▶ See Section 23.7.3



Disconnected Operations

- Tools for applications to use the Web when connected, but operate locally when disconnected from the Web
- Even in disconnected state, parallel execution of Client SW and Server SW
 - E.g. Google Gears (a browser plugin)
 - ▶ Provide a local database, a local Web server and support for execution of JavaScript at the client
 - ▶ JavaScript code using Gears can function identically on any OS/browser platform
 - Adobe AIR software provides similar functionality outside of Web browser



Chapter 9: Application Design and Development

- 9.1 Application Programs and User Interfaces
- 9.2 Web Fundamentals
- 9.3 Servlets and JSP
- 9.4 Application Architectures
- 9.5 Rapid Application Development
- 9.6 Application Performance
- 9.7 Application Security
- 9.8 Encryption and Its Applications



Rapid Application Development

- A lot of effort is required to develop Web application interfaces
 - more so, to support rich interaction functionality associated with Web 2.0 applications
- Several approaches to speed up application development
 - Function library to generate user-interface elements
 - Drag-and-drop features in an IDE to create user-interface elements
 - Automatically generate code for user interface from a declarative specification
- Above features have been in used as part of **rapid application development (RAD)** tools even before advent of Web
- GUI Building Tools
- Web Application Frameworks



GUI Building Tools

- **Microsoft ASP.NET** provides a variety of controls that are interpreted at server, and generate HTML code
- **Visual Studio** provides drag-and-drop development using these controls
 - E.g. menus and list boxes can be associated with DataSet object
 - Validator controls (constraints) can be added to form input fields
 - ▶ JavaScript to enforce constraints at client, and separately enforced at server
 - User actions such as selecting a value from a menu can be associated with actions at server
 - DataGrid provides convenient way of displaying SQL query results in tabular format



Web application frameworks

■ Major Features

- OO Data Model with Object-Relational Mapping
- Declarative way of specifying a form
- Template script system

■ Java Server Faces (JSF) framework

- includes JSP tag library

■ Jboss Seam

■ Ruby on Rails

- Allows easy creation of simple **CRUD** (create, read, update and delete) interfaces by code generation from database schema or object model



Report Generator for A Formatted Report

Acme Supply Company, Inc. Quarterly Sales Report

Period: Jan. 1 to March 31, 2009

Region	Category	Sales	Subtotal
North	Computer Hardware	1,000,000	1,500,000
	Computer Software	500,000	
	All categories		
South	Computer Hardware	200,000	600,000
	Computer Software	400,000	
	All categories		
		Total Sales	2,100,000



Chapter 9: Application Design and Development

- 9.1 Application Programs and User Interfaces
- 9.2 Web Fundamentals
- 9.3 Servlets and JSP
- 9.4 Application Architectures
- 9.5 Rapid Application Development
- 9.6 Application Performance
- 9.7 Application Security
- 9.8 Encryption and Its Applications

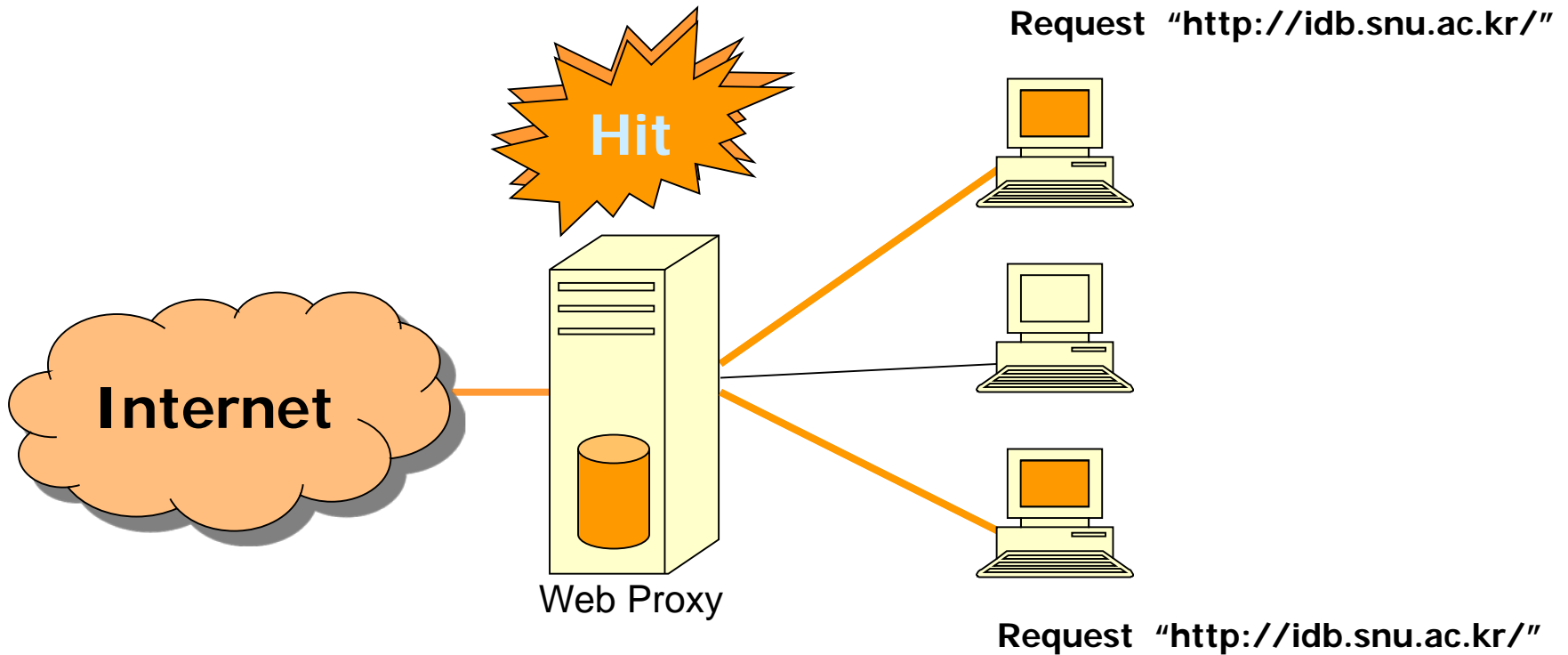


Improving Web Server Performance

- Performance is an issue for popular Web sites
 - May be accessed by millions of users every day, thousands of requests per second at peak time
- **Caching techniques** used to reduce cost of serving pages by exploiting commonalities between requests
 - **At the server site:**
 - ▶ Caching of JDBC connections between servlet requests
 - a.k.a. **connection pooling**
 - ▶ Caching results of database queries
 - Cached results must be updated if underlying database changes
 - ▶ Caching of generated HTML
 - **At the client's network**
 - ▶ Caching of pages by **Web proxy**



Caching of Pages by Web Proxy





Chapter 9: Application Design and Development

- 9.1 Application Programs and User Interfaces
- 9.2 Web Fundamentals
- 9.3 Servlets and JSP
- 9.4 Application Architectures
- 9.5 Rapid Application Development
- 9.6 Application Performance
- 9.7 Application Security
- 9.8 Encryption and Its Applications



SQL Injection

- Suppose query is constructed using
 - "select * from instructor where name = '" + name + "'"
- Suppose the user, instead of entering a name, enters:
 - X' or 'Y' = 'Y
- then the resulting statement becomes:
 - "select * from instructor where name = '" + "X' or 'Y' = 'Y" + "'"
 - which is:
 - ▶ select * from instructor where name = 'X' or 'Y' = 'Y'
 - User could have even used
 - ▶ X'; update instructor set salary = salary + 10000; --
- Prepared statement internally uses:
"select * from instructor where name = 'X\'' or \'Y\' = \'Y'"
- **Always use prepared statements, with user inputs as parameters**
- Is the following prepared statement secure?
 - conn.prepareStatement("select * from instructor where name = '" + name + "'")



Cross Site Scripting(XSS) and Cross-Site Request Forgery (CSRF)

- HTML code on one page executes action on another page

E.g. ``

- Risk: if user viewing page with above code is currently logged into mybank, the transfer may succeed
- Above example simplistic, since GET method is normally not used for updates, but if the code were instead a script, it could execute POST methods

- Prevent your web site from being used to launch XSS or CSRF attacks

- Disallow HTML tags in text input provided by users, using functions to detect and strip such tags

- Protect your web site from XSS/XSRF attacks launched from other sites

- ..next slide



Protecting XSS/XSRF attacks

- **Protect your web site from XSS/XSRF attacks launched from other sites**
 - Use **referer** value (URL of page from where a link was clicked) provided by the HTTP protocol, to check that the link was followed from a valid page served from same site, not another site
 - Ensure IP of request is same as IP from where the user was authenticated
 - ▶ prevents hijacking of cookie by malicious user
 - Never use a GET method to perform any updates
 - ▶ This is actually recommended by HTTP standard



Password Leakage

- **Never store passwords**, such as database passwords, in clear text in scripts that may be accessible to users
 - E.g. in files in a directory accessible to a web server
 - ▶ Normally, web server will execute, but not provide source of script files such as file.jsp or file.php, but source of editor backup files such as file.jsp~, or .file.jsp.swp may be served
- **Restrict access to database server** from IPs of machines running application servers
 - Most databases allow restriction of access by source IP address



Application Authentication

- **Single factor authentication** such as passwords too risky for critical applications
 - guessing of passwords, sniffing of packets if passwords are not encrypted
 - passwords reused by user across sites
 - spyware which captures password
- **Two-factor authentication**
 - e.g. password plus one-time password sent by SMS
 - e.g. password plus one-time password devices
 - ▶ device generates a new pseudo-random number every minute, and displays to user
 - ▶ user enters the current number as password
 - ▶ application server generates same sequence of pseudo-random numbers to check that the number is correct.



Application Authentication (cont.)

■ **Man-in-the-middle** attack

- E.g. web site that pretends to be mybank.com, and passes on requests from user to mybank.com, and passes results back to user
- Even two-factor authentication cannot prevent such attacks
- Solution: authenticate Web site to user, using **digital certificates**, along with **secure http protocol**

■ **Central authentication** within an organization

- application redirects to central authentication service for authentication
- avoids multiplicity of sites having access to user's password
- **LDAP** or **Active Directory** used for authentication



Application Authentication (cont.)

- **Single sign-on** allows user to be authenticated once, and applications can communicate with authentication service to verify user's identity without repeatedly entering passwords
 - **Security Assertion Markup Language (SAML)** standard for exchanging authentication and authorization information across security domains
 - ▶ e.g. user from Yale signs on to external application such as acm.org using userid joe@yale.edu
 - ▶ application communicates with Web-based authentication service at Yale to authenticate user, and find what the user is authorized to do by Yale (e.g. access certain journals)
 - **OpenID** standard allows sharing of authentication across organizations
 - ▶ e.g. application allows user to choose Yahoo! as OpenID authentication provider, and redirects user to Yahoo! for authentication



Application-Level Authorization

- Current SQL standard does not allow **fine-grained authorization** such as “students can see their own grades, but not other’s grades”
 - Problem 1: Database is lack of end-user information
 - Problem 2: SQL authorization is at the level of tables, or columns of tables, but not to specific rows of a table
- One workaround: use views such as

```
create view studentTakes as  
select *  
from takes  
where takes.ID = syscontext.user_id()
```

 - where **syscontext.user_id()** provides end user identity
 - ▶ end user identity must be provided to the database by the application
 - Having multiple such views is cumbersome



Application-Level Authorization (Cont.)

- Currently, authorization is done entirely in application
- Entire application code has access to entire database
 - large surface area, making protection harder
- Alternative: **fine-grained (row-level) authorization** schemes
 - extensions to SQL authorization proposed but not currently implemented
 - **Oracle Virtual Private Database (VPD)** allows predicates to be added transparently to all SQL queries, to enforce fine-grained authorization
 - ▶ e.g. add `ID= sys_context.user_id()` to all queries on student relation if user is a student



Audit Trails

- An audit trail is **a log of all changes** to the application data, along with information such as **which user** performed the change and **when** the change was performed.
- Applications must log actions to **an audit trail**, to detect who carried out an update, or accessed some sensitive data
- Audit trails used **after-the-fact (사후에)** to
 - detect security breaches
 - repair damage caused by security breach
 - trace who carried out the breach
- Audit trails needed
 - at Database level
 - at Application level



Chapter 9: Application Design and Development

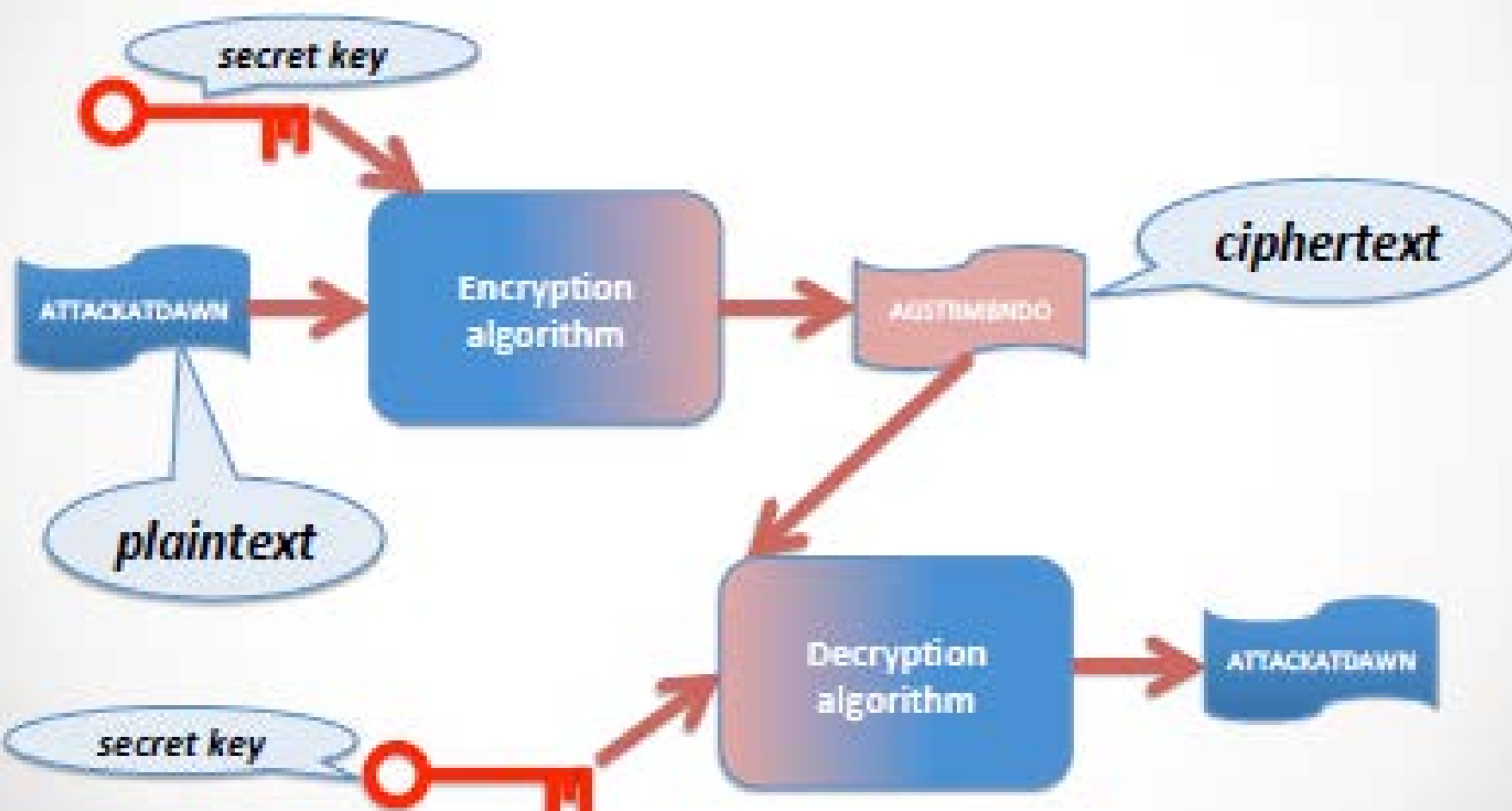
- 9.1 Application Programs and User Interfaces
- 9.2 Web Fundamentals
- 9.3 Servlets and JSP
- 9.4 Application Architectures
- 9.5 Rapid Application Development
- 9.6 Application Performance
- 9.7 Application Security
- 9.8 Encryption and Its Applications

Encryption

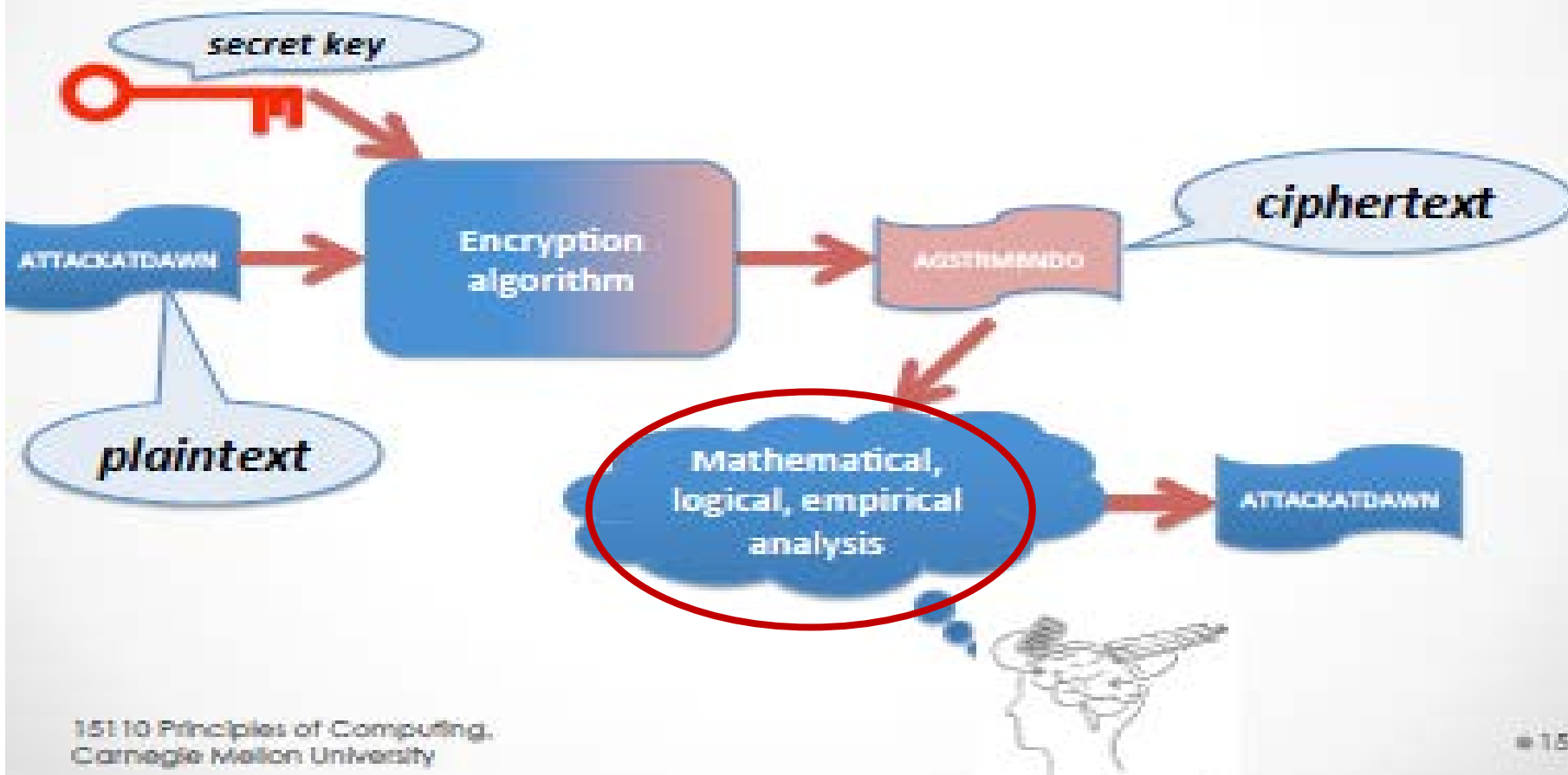
- We encrypt (encode) our data so others can't understand it (easily) except for the person who is supposed to receive it.
- We call the data to encode **plaintext** and the encoded data the **ciphertext**.
- Encoding and decoding are *inverse functions* of each other.

Encryption in computing

Encryption/decryption



Cryptanalysis



Cryptanalysis = 크립테넬러시스 = 암호해독

Two basic ways of altering text to encrypt/decrypt

- Substitute one letter for another using some kind of rule

Substitution
cipher

- Scramble the order of the letters using some kind of rule

Transposition
cipher



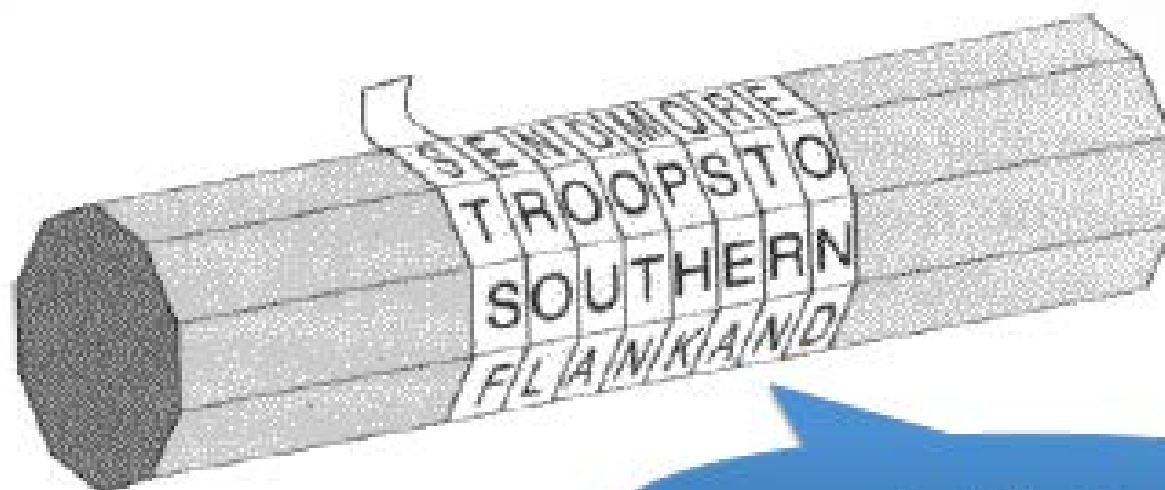
Substitution Ciphers

- Simple encryption scheme using a substitution cipher:
 - Shift every letter forward by 1:
 $A \rightarrow B, B \rightarrow C, \dots, Z \rightarrow A$
- Example:
MESSAGE \rightarrow NFTTBHF
- Can you decrypt TFDSFU?

Caesar Cipher

- Shift forward n letters; n is the secret key
- For example, shift forward 3 letters:
 $A \rightarrow D, B \rightarrow E, \dots, Z \rightarrow C$
 - This is a Caesar cipher using a key of 3.
- MESSAGE \rightarrow PHVVDJH
- How can we crack this encrypted message if we don't know the key?
DEEDUSEKBTFEIIYRBOTUSETUJXYI

Transposition ciphers



an ancient Greek method

STSF...EROL...NOUA...DOTN...MPHK...OSEA...RTRN...EOND...



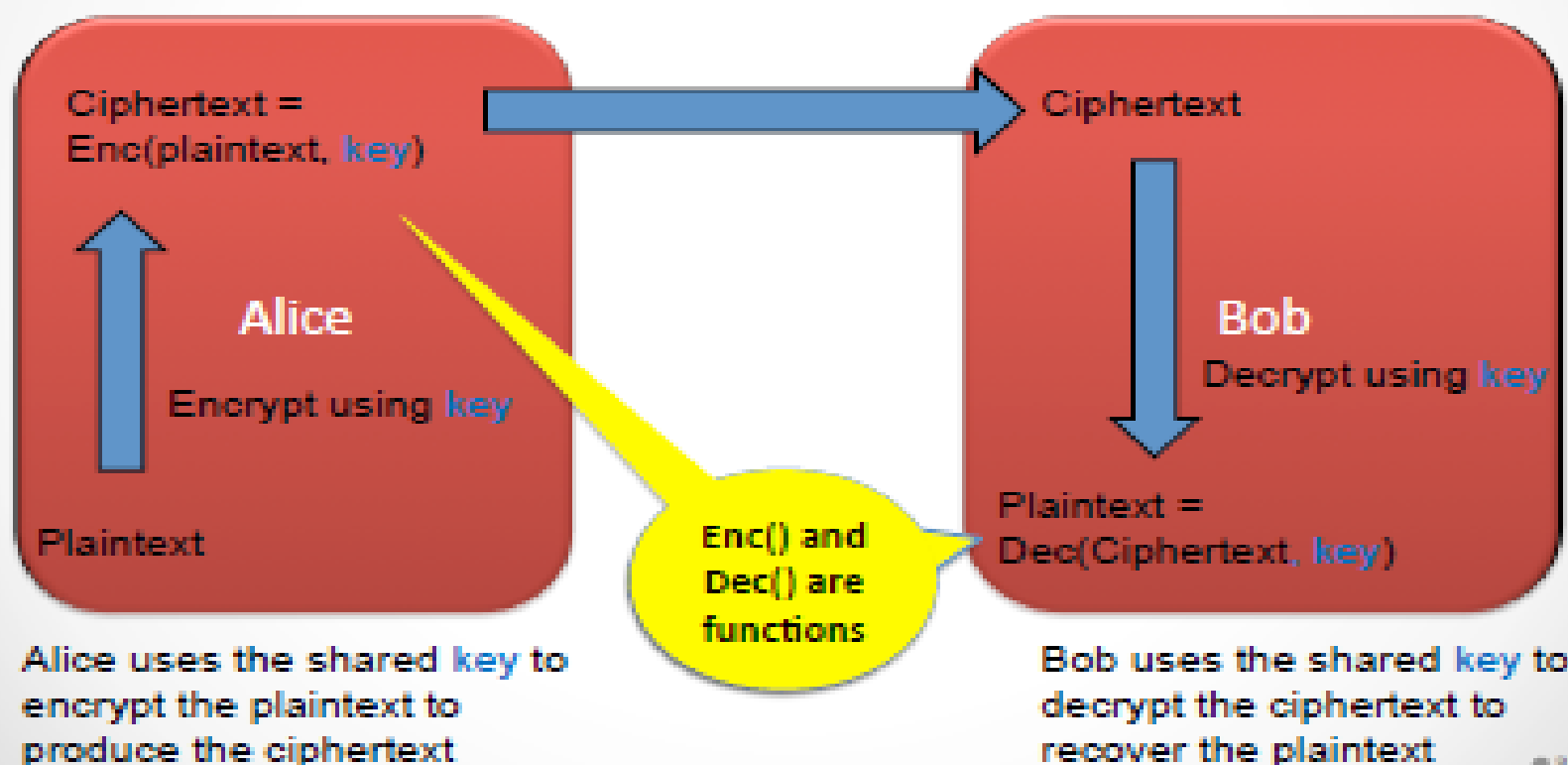
Symmetric vs. asymmetric encryption

- **Symmetric (shared-key) encryption:** commonly used for long messages
 - Often a complicated mix of substitution and transposition encipherment
 - Reasonably fast to compute
 - Requires a shared secret key usually communicated using (slower) *asymmetric encryption*
- **Asymmetric encryption:** different keys are used to encrypt and to decrypt

Keyspace

- *Keyspace* is jargon for the number of possible secret keys, for a particular encryption/decryption algorithm
- Number of bits per key determines *size of keyspace*
 - important because we want to make *brute force attacks* infeasible
 - brute force attack: run the (known) decryption algorithm repeatedly with every possible key until a sensible plaintext appears
- Typical key sizes: several hundred bits

Symmetric (Shared Key) Encryption



● 35

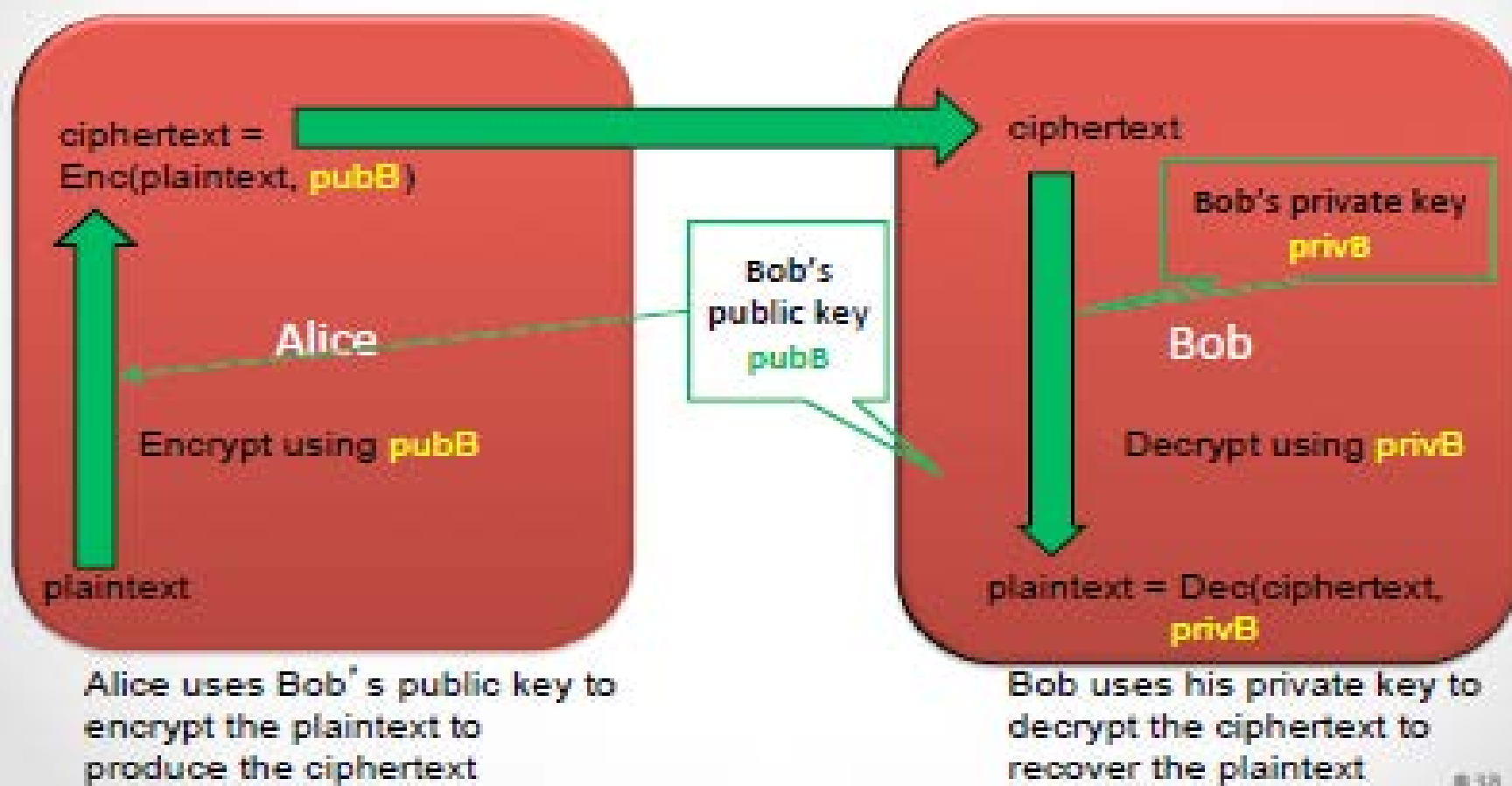
Establishing Shared Keys

- Problem: how can Alice and Bob secretly agree on a key, using a public communication system?
- Solution: asymmetric encryption based on *number theory*
 - Alice has one secret, Bob has a different secret; working together they establish a shared secret
 - Examples: Diffie-Hellman key exchange, RSA public key encryption

One type of asymmetric encryption: RSA

- Common encryption technique for transmitting symmetric keys on the Internet (https, ssl/tls)
 - Named after its inventors: Rivest, Shamir and Adleman
 - Used in https (you know when you're using it because you see the URL in the address bar begins with https://)

Asymmetric Public Key Encryption

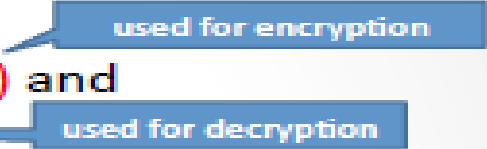


How RSA works

- First, we must be able to represent any message as a single number (it may already be a number as is usual for a symmetric key)
- For example:

A T T A C K A T D A W N
012020010311012004012314

Public and Private Keys

- Every receiver has a **public key** (e, n) and a **private key** (d, n).

- The transmitter encrypts a (numerical) message M into ciphertext C using the receiver's public key:

$$M^e \text{ modulo } n \rightarrow C \text{ (ciphertext)}$$

- The receiver decodes the encrypted message C to get the original message M using the private key (which no one else knows).

$$C^d \text{ modulo } n \rightarrow M \text{ (plaintext)}$$

RSA Example

- Alice's Public Key: $(3, 33)$ ($e = 3, n = 33$)
- Alice's Private Key: $(7, 33)$ ($d = 7, n = 33$)
 - Usually these are really huge numbers with many hundreds of digits!
- Bob wants to send the message 4
 - Bob encrypts the message using e and n :
 $4^3 \text{ modulo } 33 \rightarrow 31$... Bob sends 31
- Alice receives the encoded message 31
 - Alice decrypts the message using d and n :
 $31^7 \text{ modulo } 33 \rightarrow 4$

Generating n , e and d

- p and q are (big) random primes. $p = 3, q = 11$
- $n = p \times q$ $n = 3 \times 11 = 33$
- $\varphi = (p - 1)(q - 1)$ $\varphi = 2 \times 10 = 20$
- e is small and relatively prime to φ $e = 3$
- d , such that:
 $e \times d \bmod \varphi = 1$ $3 \times d \bmod 20 = 1$
 $d = 7$

Usually the primes are huge numbers—hundreds of digits long.

e , n 값은 알려진것이고 p , q 만 알아내면, φ 을 알고
 그러면 d 값을 알아낼수 있다!, But.....

Cracking RSA

- Everyone knows (e, n) . Only Alice knows d .
- If we know e and n , can we figure out d ?
 - If so, we can read secret messages to Alice.
- We can determine d from e and n .
 - Factor n into p and q .
$$n = p \times q$$
$$\varphi = (p - 1)(q - 1)$$
$$e \times d = 1 \pmod{\varphi}$$
 - We know e (which is public), so we can solve for d .
- But only if we can factor n

RSA is safe (for now)

- Suppose someone can factor my 5-digit n in 1 ms,
- At this rate, to factor a 10-digit number would take 2 minutes.
- ... to factor a 15-digit number would take 4 months.
- ... 20-digit number ... 30,000 years.
- ... 25-digit number... 3 billion years.
- We're safe with RSA! (at least, from factoring with digital computers)

Certificate Authorities

- How do we know we have the right public key for someone?
- *Certificate Authorities* sign digital certificates indicating authenticity of a sender who they have checked out in the real world.
- Senders provide copies of their certificates along with their message or software.
- But can we trust the certificate authorities? (only some)



Encryption

- Data may be *encrypted* when database authorization provisions do not offer sufficient protection.
- Properties of good encryption technique:
 - Relatively simple for authorized users to encrypt and decrypt data.
 - Encryption scheme depends not on the secrecy of the algorithm but on the secrecy of a parameter of the algorithm called the encryption key.
 - Extremely difficult for an intruder to determine the encryption key.
- **Symmetric-key encryption**: same key used for encryption and for decryption
- **Public-key encryption** (a.k.a. **asymmetric-key encryption**): use different keys for encryption and decryption
 - encryption key can be public, decryption key secret



Encryption (Cont.)

- *Data Encryption Standard (DES)* substitutes characters and rearranges their order on the basis of an encryption key which is provided to authorized users via a secure mechanism. Scheme is no more secure than the key transmission mechanism since the key has to be shared.
- *Advanced Encryption Standard (AES)* is a new standard replacing DES, and is based on the Rijndael algorithm, but is also dependent on shared secret keys.
- *Public-key encryption* is based on each user having two keys:
 - *public key* – publicly published key used to encrypt data, but cannot be used to decrypt data
 - *private key* -- key known only to individual user, and used to decrypt data. Need not be transmitted to the site doing encryption.

Encryption scheme is such that it is impossible or extremely hard to decrypt data given only the public key.

- *The RSA public-key encryption scheme* is based on the hardness of factoring a very large number (100's of digits) into its prime components



DES (Data Encryption Standard)

- 미국 표준기술협회인 NIST가 1977년에 발표한 암호방식.
- 64bit Key와 개별 키 암호화 방식을 통해 텍스트를 암호화
- 메시지는 64비트블록으로 나뉘고 데이터 암호화 알고리즘을 통해 암호화
- 개별 키 암호화 방식에서는 발신자와 수신자만이 키를 공유
- 키를 모르면 코드를 풀기가 어렵지만, 해당키의 보안유지는 사용자의 몫
- 신용카드에 많이 사용



Encryption (Cont.)

- **Hybrid schemes** combining public key and private key encryption for efficient encryption of large amounts of data
- Encryption of small values such as identifiers or names vulnerable to **dictionary attacks**
 - especially if encryption key is publicly available
 - but even otherwise, statistical information such as frequency of occurrence can be used to reveal content of encrypted data
 - Can be deterred by adding extra random bits to the end of the value, before encryption, and removing them after decryption
 - ▶ same value will have different encrypted forms each time it is encrypted, preventing both above attacks
 - ▶ extra bits are called **salt bits**



Encryption in Databases

- Database widely support encryption
- Different levels of encryption:
 - **disk block**
 - ▶ every disk block encrypted using key available in database-system software.
 - ▶ Even if attacker gets access to database data, decryption cannot be done without access to the key.
 - **Entire relations, or specific attributes of relations**
 - ▶ non-sensitive relations, or non-sensitive attributes of relations need not be encrypted
 - ▶ however, attributes involved in primary/foreign key constraints cannot be encrypted.
- **Storage of encryption or decryption keys**
 - typically, single master key used to protect multiple encryption/decryption keys stored in database
- Alternative: encryption/decryption is done in application, before sending values to the database



Encryption and Authentication

- Password based authentication is widely used, but is susceptible to sniffing on a network.
- **Challenge-response** systems avoid transmission of passwords
 - DB sends a (randomly generated) challenge string to user.
 - User encrypts string and returns result.
 - DB verifies identity by decrypting result
 - Can use public-key encryption system by DB sending a message encrypted using user's public key, and user decrypting and sending the message back.
- **Digital signatures** are used to verify authenticity of data
 - E.g., use private key (in reverse) to encrypt data, and anyone can verify authenticity by using public key (in reverse) to decrypt data. Only holder of private key could have created the encrypted data.
 - Digital signatures also help ensure **nonrepudiation**: sender cannot later claim to have not created the data



Digital Certificates

- **Digital certificates** are used to verify authenticity of public keys.
- Problem: when you communicate with a web site, how do you know if you are talking with the genuine web site or an imposter?
 - Solution: use the public key of the web site
 - Problem: how to verify if the public key itself is genuine?
- Solution:
 - Every client (e.g., browser) has public keys of a few root-level **certification authorities**
 - A site can get its name/URL and public key signed by a certification authority: signed document is called a **certificate**
 - Client can use public key of certification authority to verify certificate
 - Multiple levels of certification authorities can exist. Each certification authority
 - ▶ presents its own public-key certificate signed by a higher level authority, and
 - ▶ uses its private key to sign the certificate of other web sites/authorities



End of Chapter