Objectives

- Servlets Review
- JSPs
- Web Application Organization
- Version Control

Servlets Review

- How do we access a servlet's init parameter?
- Why do we use init parameters?
- Where are init parameters defined?
- How can we save state across multiple requests from a user?
  - What are the pros and cons of each?

Review: Servlet Life Cycle

- Web application server creates **one** instance of servlet
  - Calls `init` method of servlet created
- As requests come in, WAS calls `service` method of appropriate servlet
  - In turn, servlet calls appropriate `doMethod`
- When web application server shuts down, calls `destroy` method of each servlet

The World's First Web Page

World Wide Web

The WorldWideWeb (W3) is a wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an executive summary of the project, Mailing lists , Policy , November's W3 news , Frequently Asked Questions .

What's out there?
Pointers to the world's online information, subjects , W3 servers, etc.
Help
on the browser you are using
Software Products
A list of W3 project components and their current state. (e.g. Line Mode ,X11 Viola , NeXTStep , Servers , Tools , Mail robot , Library)
Technical
Details of protocols, formats, program internals etc
Bibliography
Paper documentation on W3 and references.
People
A list of some people involved in the project.
History
A summary of the history of the project.
How can I help ?
If you would like to support the web..
Getting code
Getting the code by anonymous FTP , etc.

The Page: `http://info.cern.ch/hypertext/WWW/TheProject.html`
A Story: `http://www.businessinsider.com/the-worlds-first-web-page-is-back-online-2013-4`

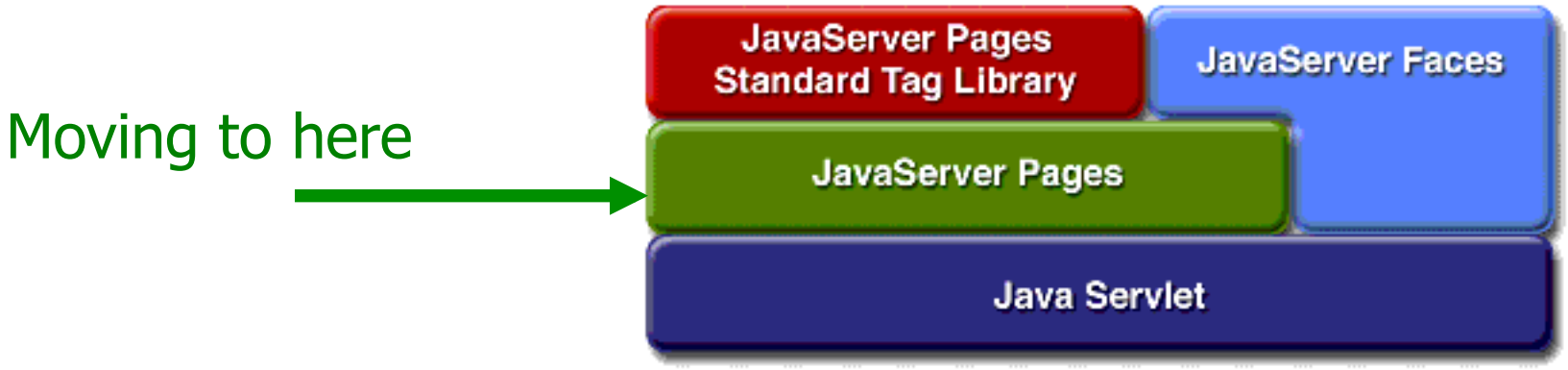
JAVASERVER PAGES (JSPS)

Discussion

What made writing servlets difficult?

Motivation: JavaServer Pages (JSPs)

- Simplify web application development
- Separate UI from backend code
  - Separate presentation layer
- Difficult to write HTML in print statements



JavaServer Pages (JSPs)

- Merge HTML and Java
  - Separate static HTML from dynamic
  - Make HTML templates, fill in dynamic content
  - Encourages separation of tasks
- Web application server compiles JSPs into servlet code
  - Clean and efficient
- Easier to develop, deploy, modify scripted pages
  - How much trouble did you have with HTML in Strings?

JSP Syntax: Expression

- Enclose code in <%= %>

```
<html>
<body>
<p>
Hello!  The time is now <%= new java.util.Date() %>
</p>
</body>
</html>
```

Expression

Evaluated, turned into a `String`

JSP Syntax: Scriptlet

```
<html>
<body>
<%
    // This is a scriptlet.  The "date" variable
    // we declare here is available in the
    // embedded expression later on.
    java.util.Date date = new java.util.Date();
%>
<p>
Hello!  The time is now <%= date %>
</p>
</body>
</html>
```

Example: SurveyServlet as a JSP

```
<%
 for (int i = 0; i < animalNames.length; i++) {
 %>
 <tr>
        <td><%=animalNames[i]%></td>
        <td><%=votes[i]%></td>
        <td><%=formattedPercentages[i]%></td>
        <%
                totalVotes += votes[i];
        %>
 </tr>
 <%
        }
 %>
```

To be displayed at end

JSP Directives

- Page Directive
  - Java files to import (like import statement in Java)

```
<%@ page import="java.util.*,java.text.*" %>
<%@ page import="ourcode.MyClass"%>
```

- Include Directive
  - Include contents of another file: JSP, HTML, or text
  - Example: include site's common headers or footers

```
<%@ include file="header.jsp" %>
```

JSP Declarations

- For instance variables and methods

```
<%!
   private ArrayList users;

   public void jspInit() {
      // on start up: set up
   }
   public void jspDestroy() {
      // on shut down: clean up
   }
%>
```

- We won't do too much of this
  - Let servlets do the work

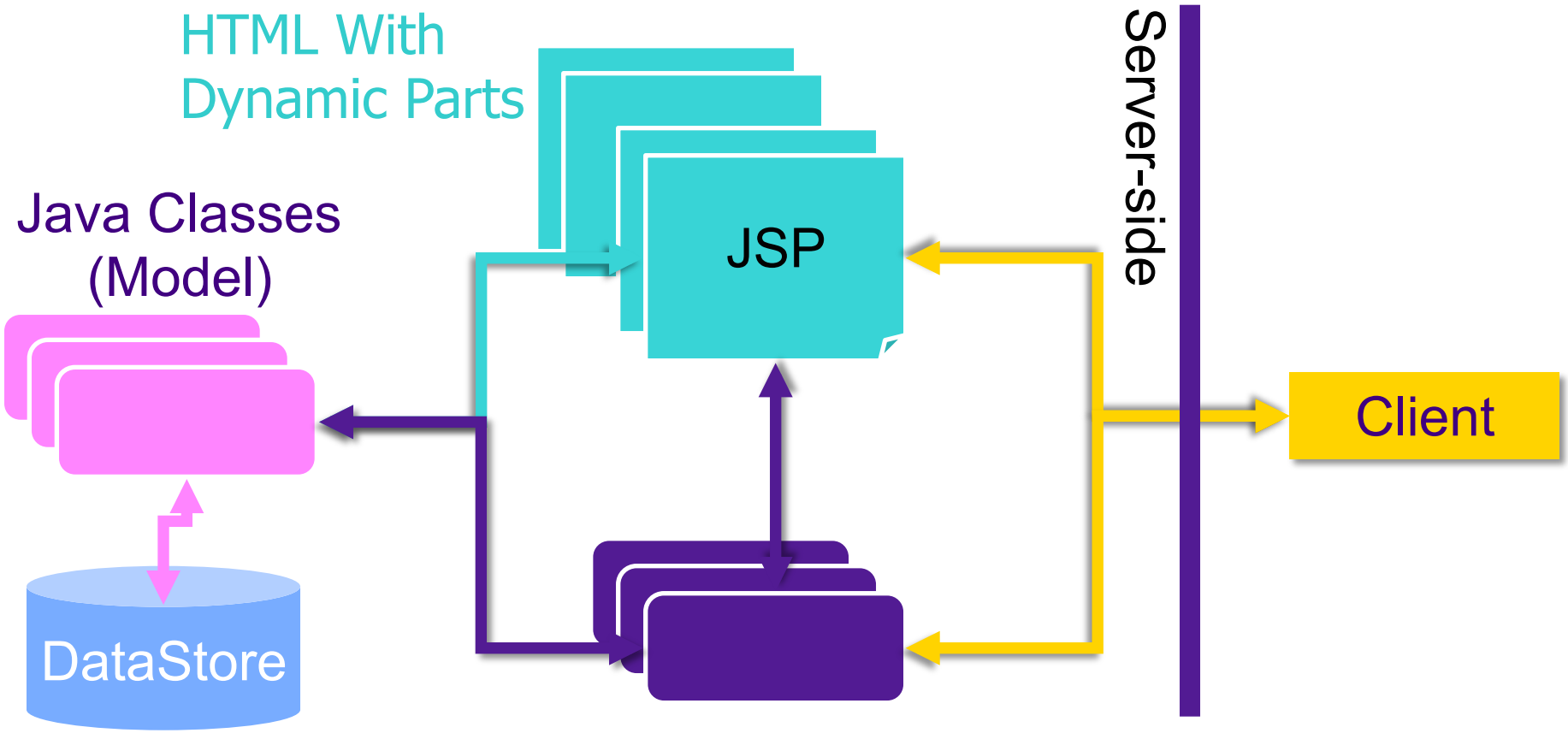
JSP Variables

- By default, JSPs have some variables
  - **Not explicitly declared in the file**
  - `HttpServletRequest request`
  - `HttpServletResponse response`
  - `HttpSession session`

These names must be used

- JSPs can access request parameters, session data

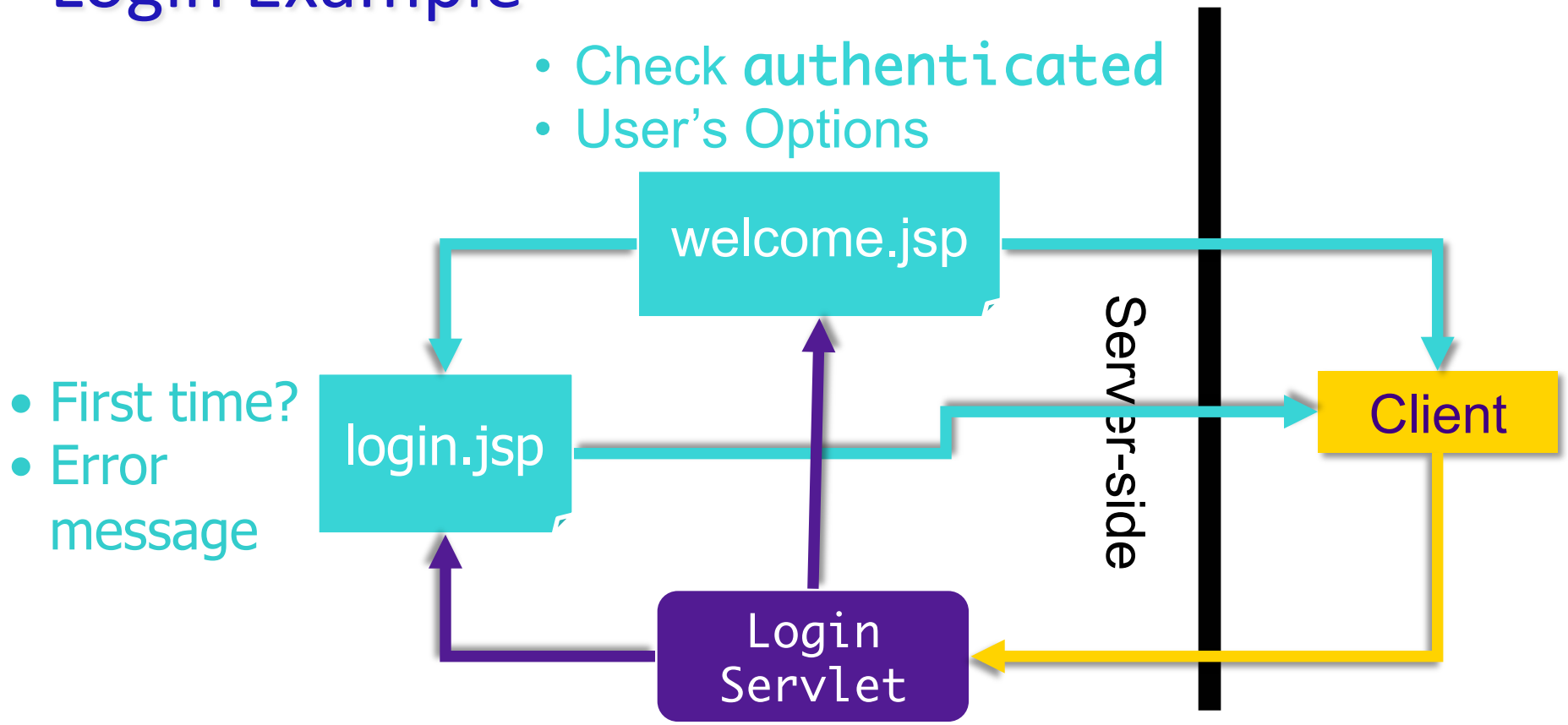
Web Application Architecture



Java Servlets

- Heavy lifting of requests
- Forward to JSPs

Communicating Between Servlets and JSPs: Login Example



- Check user name/password
- Set `authenticated` or `error` attribute
- Forward to welcome.jsp or login.jsp

Communicating Between JSPs and Servlets

- ***Attributes***
  - Name/Value pairs
  - Values are **Objects**
  - Can get/set attributes on the **`HttpServletRequest`** object
    - Similar to *session* attributes but call methods on a *request* object
      - Different lifetime
- Parameters
  - Names and Values are **`Strings`**
  - From forms or in URLs
  - Also in the **`HttpServletRequest`** object

Forwarding Requests

- **HttpServletRequest**'s **getRequestDispatcher** method
  - Returns a **RequestDispatcher** object

The name of the resource to forward to

```
request.getRequestDispatcher("welcome.jsp").
    forward(request, response);
```

- Can use **RequestDispatcher**'s **include** method similarly

Adding a JSP to SurveyServlet

- Separate heavy lifting from the HTML
- Think of JSP as a template
  - What is static about the response page?
  - What is dynamic?
- Servlet will handle most of the work

Look at code

Protecting JSPs

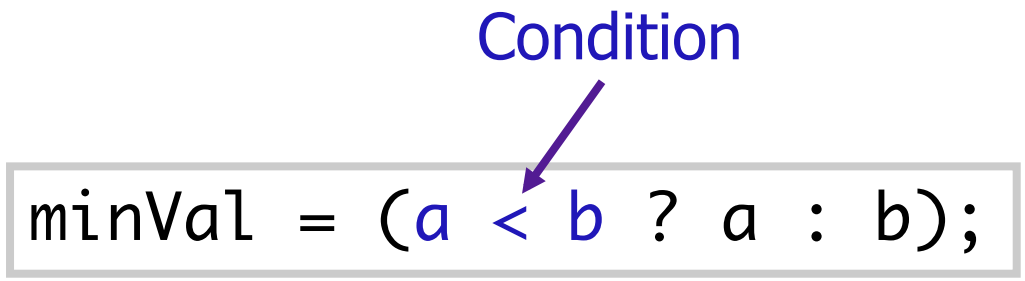
- If there are JSPs that you don't want users to be able to access from typing in the URL, put them in the `WEB-INF` directory
  - Web application server blocks access to the JSP
  - Don't need code to check authorization again
    - Only get to JSP through a servlet that checks authorization
- Forward requests from a servlet to the JSP by including `WEB-INF` in the URI

WEB-INF Directory

- Put “response” pages that you don’t want users to directly access
  - Example: User shouldn’t be able to access petResponse.jsp directly

Trick: Ternary Operator

- Alternative `if-then-else` syntax
- Returns a value
- Example of assigning a variable the minimum:

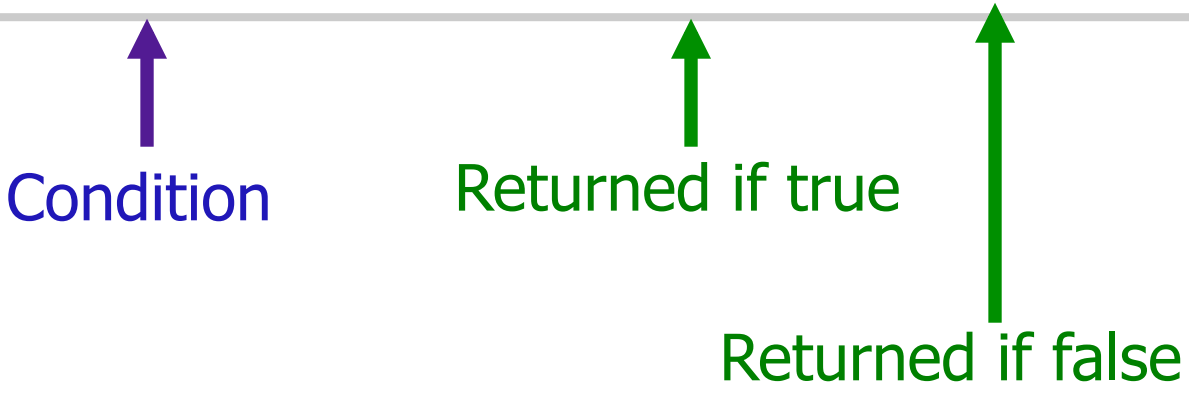


  - Assign `minVal` value `a` if condition is `true`, `b` if condition is `false`

Ternary Operator in JSP

- Allows for more concise code

```
<input type=text name="username"
value="<%= userName != null ? userName : ""%>">
```



HttpServletRequest

- getContextPath()
  - Returns the portion of the request URI that indicates the context of the request.
- Example with various Request methods

http://example.com:8080/app/dirpath/index.jsp?cat=2&cat=5

getScheme() → "http"
getServerName() → "example.com"
getServerPort() → 8080

getContextPath() → "/app"
getServletPath() → "/dirpath"
getPathInfo() → "/index.jsp"
getParameter("cat") → "2"
getParameterValues("cat") → {"2", "5"}

Use in JSP

```
<a href="<%=request.getContextPath()%>">
Main Page</a>
```

VERSION CONTROL

Review: Version Control

- Why do we need version control?
  - What is it good for?
- What are examples of version control systems?
- How do we "get" code from the repository in svn?
- How do we put our code into the public version of the code?
- How do we pull code from the public version of the code?

Backups and Coordination Issues

- Maybe you wrote a paper and had several versions
  - Good practice to iterate over it!
  - Keep track of versions using names, e.g., `paper1.pdf`, `paper.draft.pdf`, `paper.outline.pdf`, `paper.mar7.pdf`
- Coordinate a group
  - One person's account has *the* version
  - Make conflicting changes to files
    - Figure out fix, Merge files

Version Control

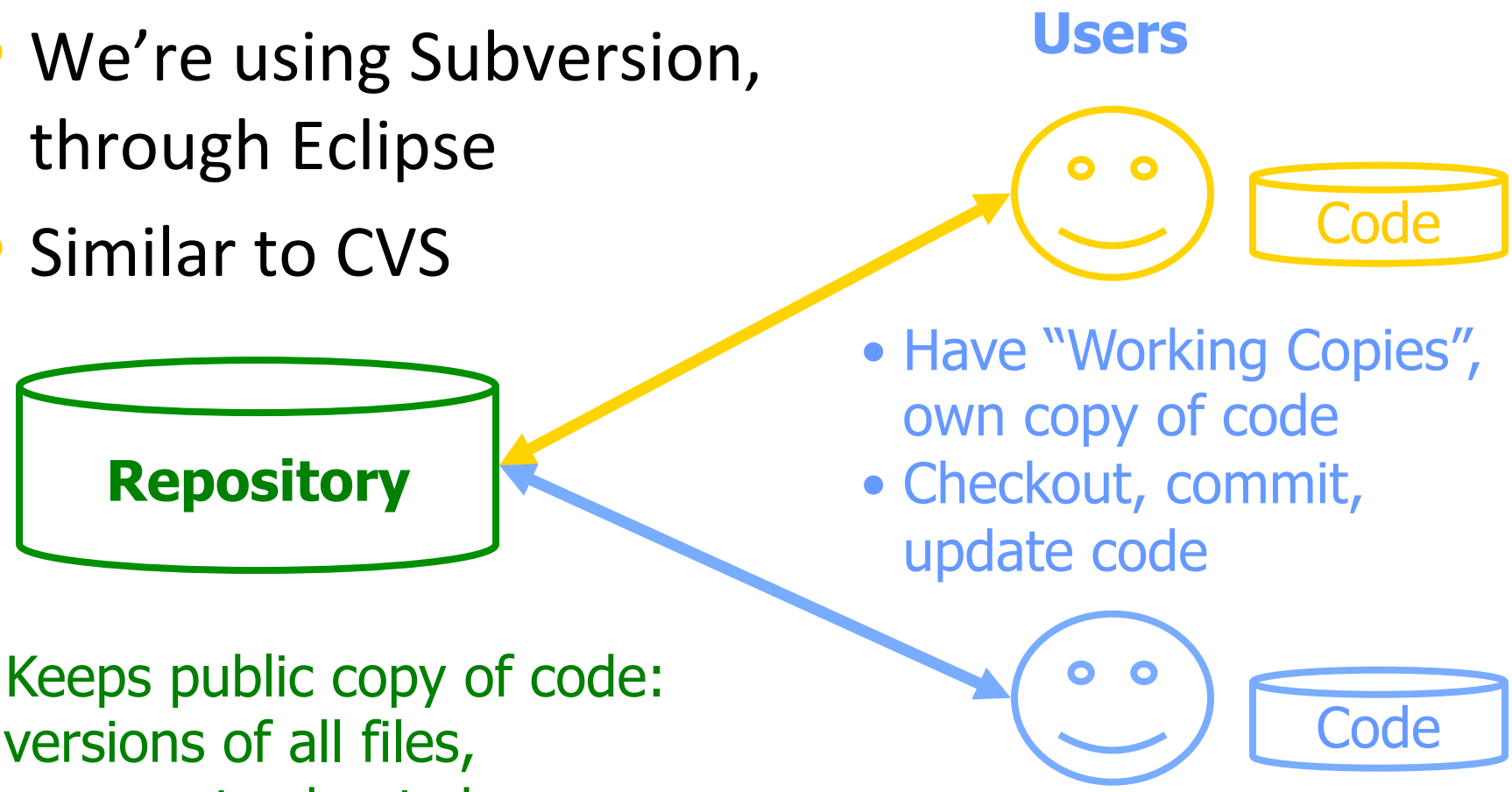
- Backup and Restore
  - Files are saved as they are edited
  - Revert to a specific version/checkpoint
- Synchronization
  - Lets people share files
  - Stay up-to-date with the latest version
- Track changes to code
  - Save comments explaining why change happened
  - Stored in the VCS, not the file
  - Track how, why a file evolves over time
- Track Ownership
  - Tags every change with the name of the person who made it

Version Control

- Short-term undo
  - Messed up a file? Go back to the last **good** version
- Long-term undo
  - Created a bug a year ago? Jump back to see change you made.
- Sandboxing
  - Making a big change? Make temporary changes in isolated area, test, work out kinks before "checking in" your changes
- Branching and merging
  - Branch a copy of your code into a separate area, modify it in isolation (tracking changes separately)
  - Later, merge work into common area.

Using Version Control

- We're using Subversion, through Eclipse
- Similar to CVS

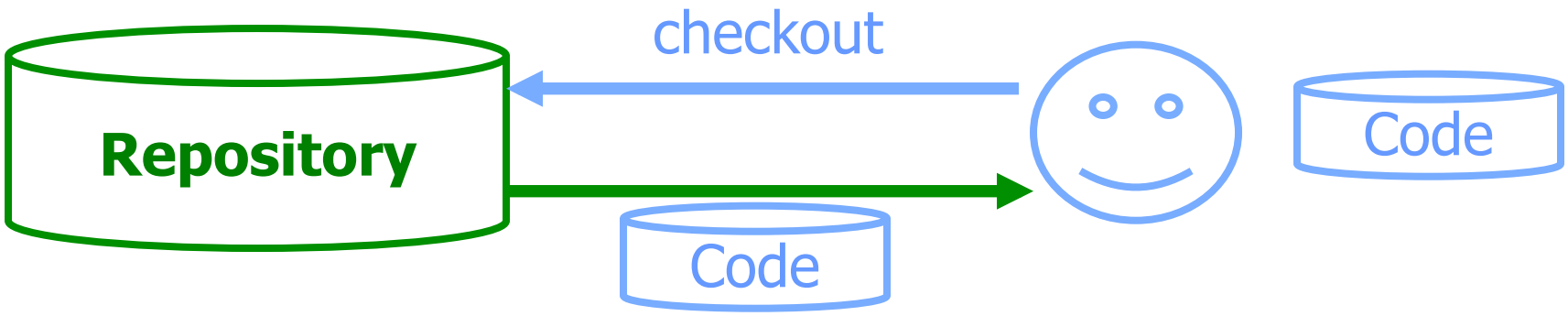


- Have "Working Copies", own copy of code
- Checkout, commit, update code

- Keeps public copy of code: versions of all files, comments about changes, who made changes

Using Version Control: **checkout**

- To start, need to **checkout** your working copy of the code



Using Version Control: `commit`

- After you make changes that you want others to see, **commit** your version

- Checks for conflicts -- code conflicts with recent changes in the public copy

- Update code, fix conflicts
- Try commit again

Using Version Control: `commit`

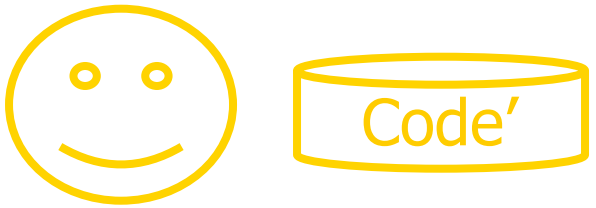
- After you make changes that you want others to see, **commit** your version
  - Include comments about what changes you made and why

- Checks for conflicts
- Updates each modified file
- Records comments with updated files

Using Version Control: `commit`

- After you make changes that you want others to see, **commit** your version
  - Include comments about what changes you made and why

- Checks for conflicts
- Updates each modified file
- Records comments with updated files



Other people's code doesn't change

Using Version Control: `update`

- To see the *current* version of the code, **update** your repository
  - Resolve conflicts

Using Version Control: `add, delete`

- You need to **add** and **delete** files and directories to the repository, then **commit**

- Create new records for added files
- Close records for deleted files
  - Files not deleted from repository

- Add, delete files and directories
- Commit

Version Control Advice

- Does not eliminate need for communication
  - Process becomes much more difficult if developers do not communicate
- Before picking up again, **update** your working copy
- **Commit** only after you've tested code and you're fairly sure it works
  - Write descriptive comments so your team members know what you did and why

Subclipse

- You'll use Subclipse (Subversion plugin for Eclipse) to manage your working copy and the public copy
- I'll checkout the public copy, create WAR file, deploy on servo
- Set up is part of Lab 5

TODO

- Lab 5 - Subversion and JSPs
  - Install Subclipse; not ready for check out yet
- Web Quality Attributes – Reading
- Exam – next Thursday, May 12 – 10:10 a.m.
- Project
  - So far
    - Clarification of context and vocabulary of the project
    - Clear on requirements for the remainder of the project
  - Revisions of requirements document, design document, work plan, static mockup
    - Thursday midnight
  - High-priority functionality: Next Wednesday midnight