Module 8

HttpSession

Course Objectives

- After completing this module, you should be able to:
  - Discuss the task of managing client application data
    - Session management with the WebSphere Application Server Session Manager tool
  - Describe the use of HttpSession to maintain a user session
  - Explain how object sharing is implemented in the servlet environment
  - Describe the various ways to manage application state

Session Management (1 of 2)

- Sessions provide a way to identify a user across more than one page request or visit to a Web site, and to store information about that user
- Web applications must manage state information:
  - Current customer, shopping cart, and so forth
  - Application involves several servlets
  - Servlets need to be stateless
- Multiple implementation technologies including:
  - HttpSession
  - HTTP Cookies
  - HTML Hidden Field
  - URL Rewriting

Session Management (2 of 2)

- The HttpSession interface, part of the Servlet API, provides an interface for managing application state on the server
- In applications that are marked as *distributable*, the session data objects placed into the HttpSession object must be serializable (they must implement the Serializable interface)
- A session:
  - Represents a client-server HTTP connection
  - Lifetime spans multiple servlets and page requests
  - Is identified within requests via a Session identifier

Session Usage

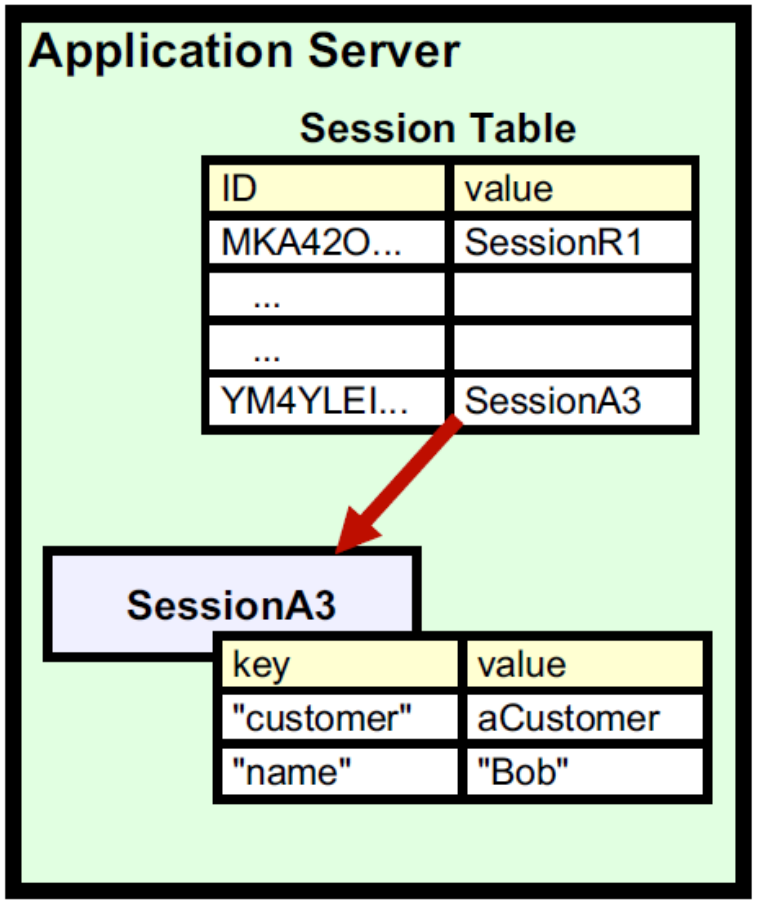
- Servlet asks to bind to the Session object representing the current session:
  - A session is requested – request.getSession(boolean create)
  - The method returns the current HttpSession, if it exists
  - If create is true (or no parameter is specified) AND no current Session exists, a newly created session is returned
- The session is unavailable when:
  - The client browser is closed
  - The session is explicitly invalidated
  - The session times out

HttpSession data store

- HttpSessions store application-specific information
  - Stored as <"key", object> pairs
    - `void setAttribute(String, Object)`
    - `Object getAttribute(String)`

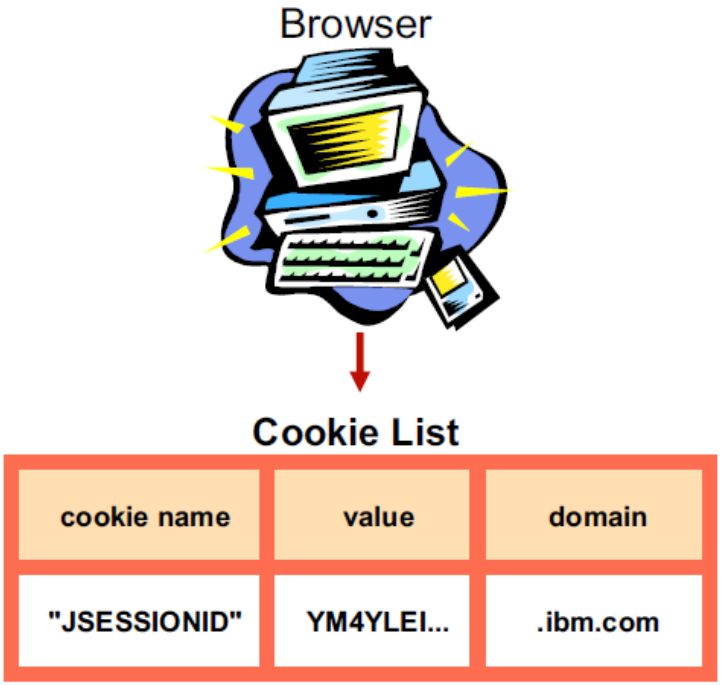
Session at runtime - server

- HttpSession objects are managed by the Web container
- They are registered by ID
- The ID must be delivered to the client initially, and presented back to the server on subsequent requests

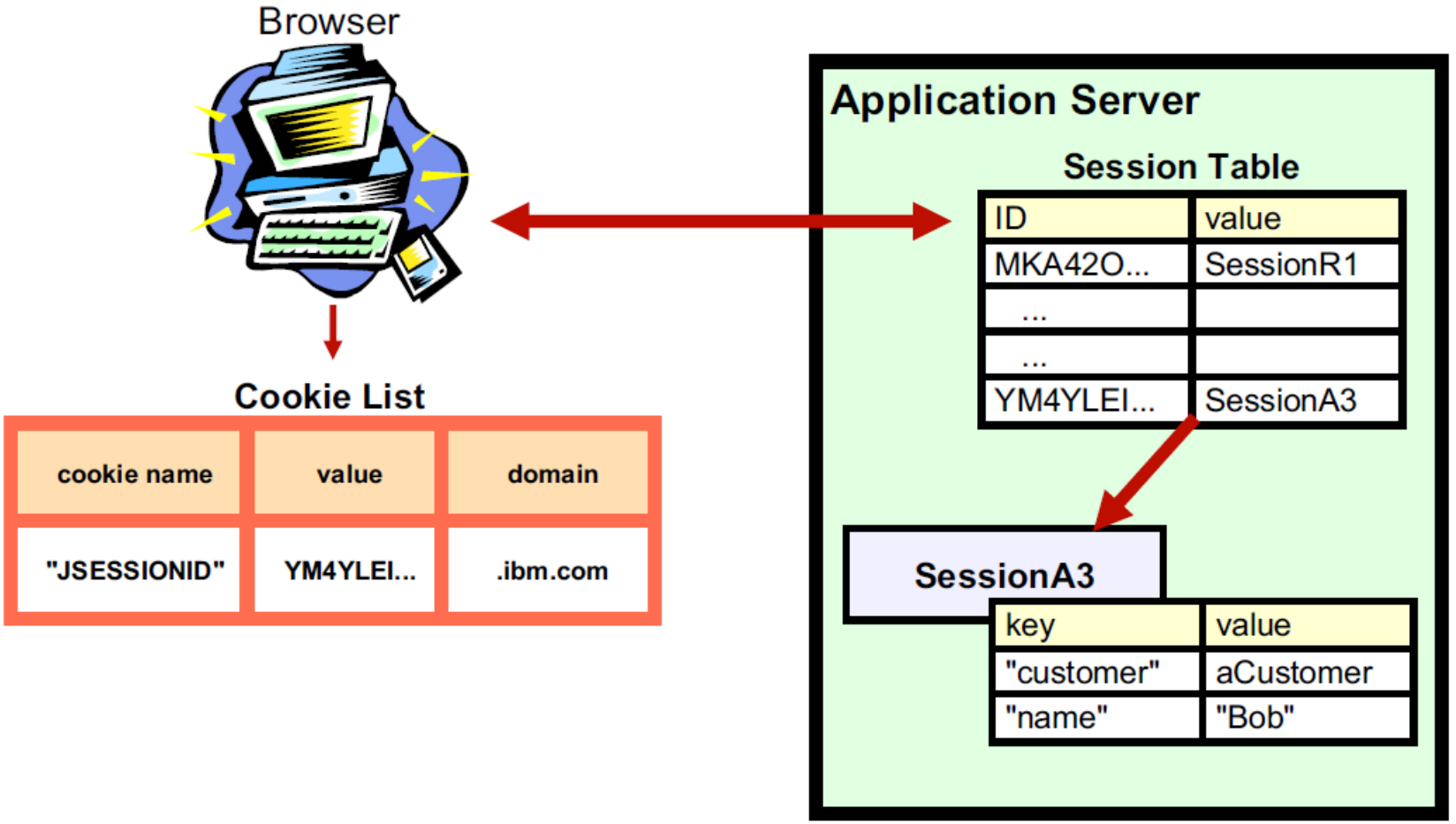


Sessions at runtime: client

- The preferred (default) delivery vehicle for session ID is a transient cookie
- Alternative URL rewriting is supported by HttpServletResponse
  - No automatic support in JSP pages
  - Requires ad hoc support for client-side script generated URLs



Sessions at runtime



Session invalidation

- Release HttpSession objects when finished
  - An Application Server can only maintain a certain number of HttpSession objects in memory
- Sessions can be invalidated either programmatically or through a timeout
  - session.invalidate
  - Removes all values from the session
- The session timeout (inactive interval) can be set for the application server as a whole
  - The default timeout is 30 minutes
- Also session.setMaxInactiveInterval(int) can provide session-specific timeout value

Session invalidation example

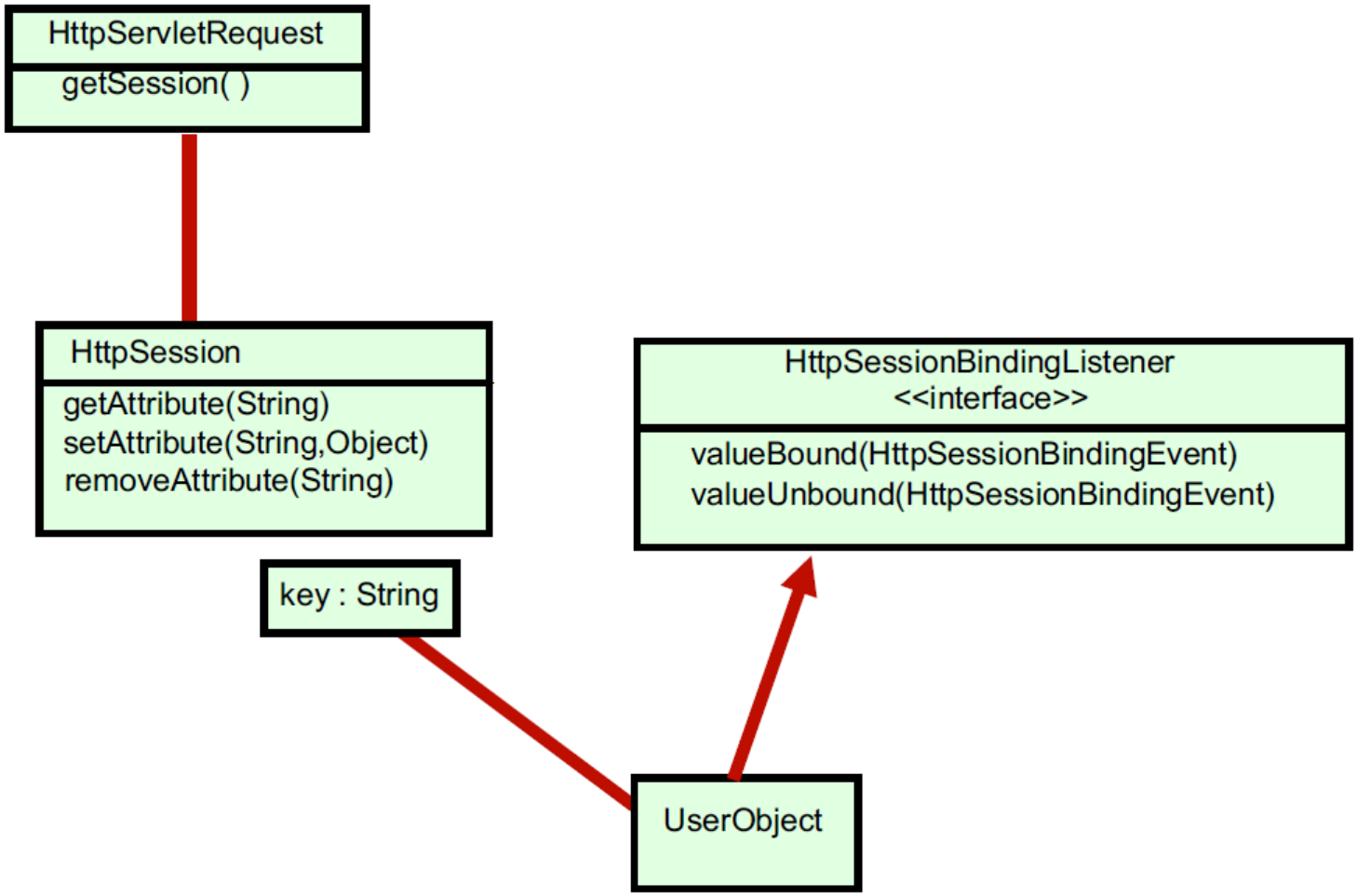
```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class ApplicationLogoutServlet extends HttpServlet {

     public void doGet(HttpServletRequest req, HttpServletResponse resp)
       throws ServletException, IOException {

       HttpSession mySession = req.getSession(false);

       // Invalidate session
      if (mySession != null) {
          mySession.invalidate();
       }
       // Perform additional application logoff processing
       // and send output response to browser here
    }
}
```

Thread safety

- The HttpSession object is a shared resource
  - Access to shared objects should be synchronized
  - Do not synchronize indirectly (for example, synchronizing various servlets' **doPost()** methods)
  - Instead, wrap sets of **setAttribute()** and **getAttribute()** in a synchronized block

```
Customer cust = (Customer)
session.getAttribute("customer");
synchronized (cust) {
      // work with the customer object
}
```

HttpSession classes

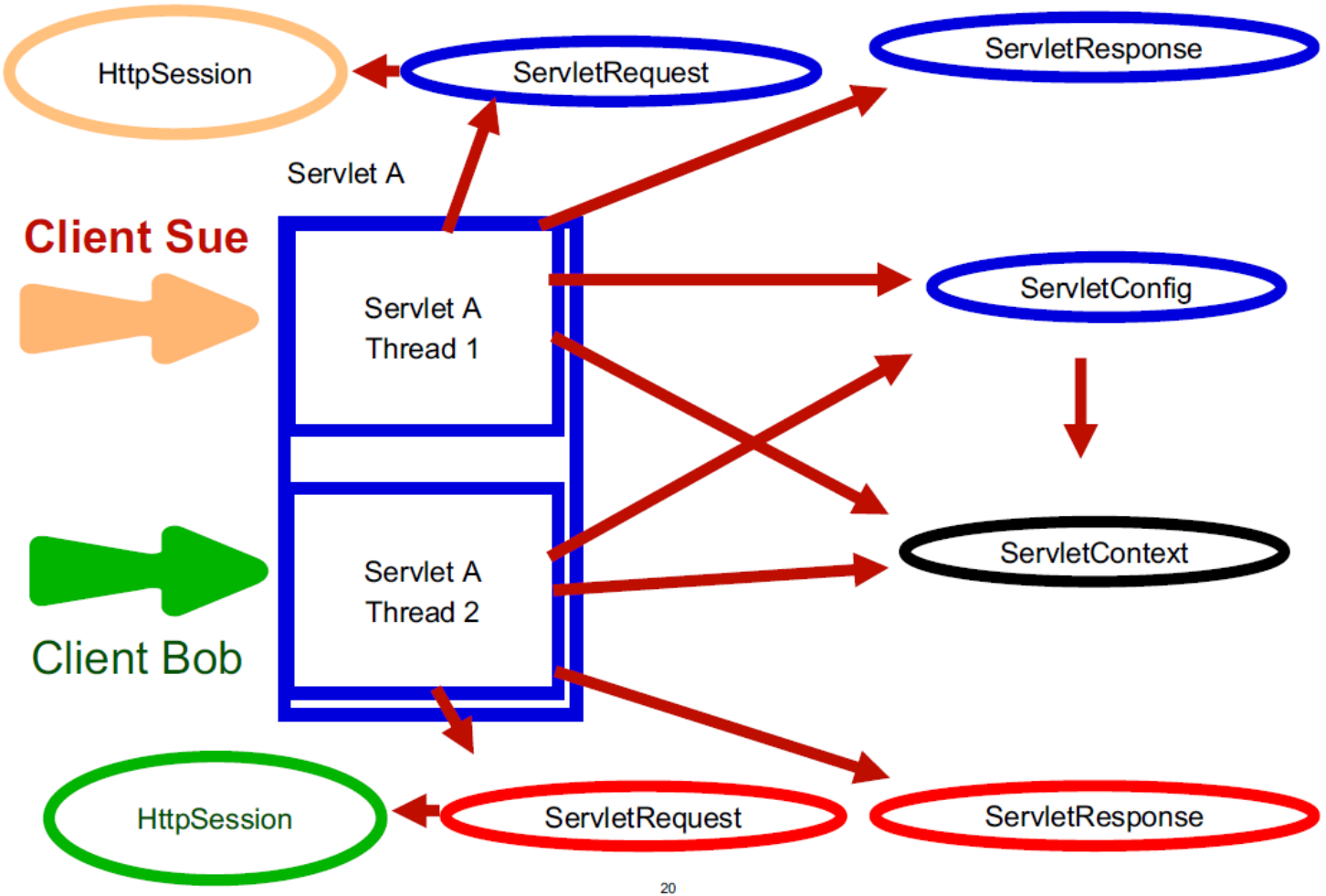


Session serialization

- Objects stored in a session must be serializable:
  - To share between servers in a clustered server configuration
  - For persistence to work
- Make sure that objects reachable from the session are also serializable
- When creating objects to be stored in the session, implement the serializable interface as follows :

```
public class NewObject implements java.io.Serializable { ... }
```

Servlet objects



Checkpoint

1. Explain how to invalidate a session.
2. Why do you need to be concerned with thread safety?
3. Why would you need to serialize a session?

Module Summary

- This unit covered the following topics:
  - Discuss the task of managing client application data
  - Describe the use of HttpSession to maintain a user session
  - Explain how object sharing is implemented in the servlet environment
  - Describe the various ways to manage application state