Sessions

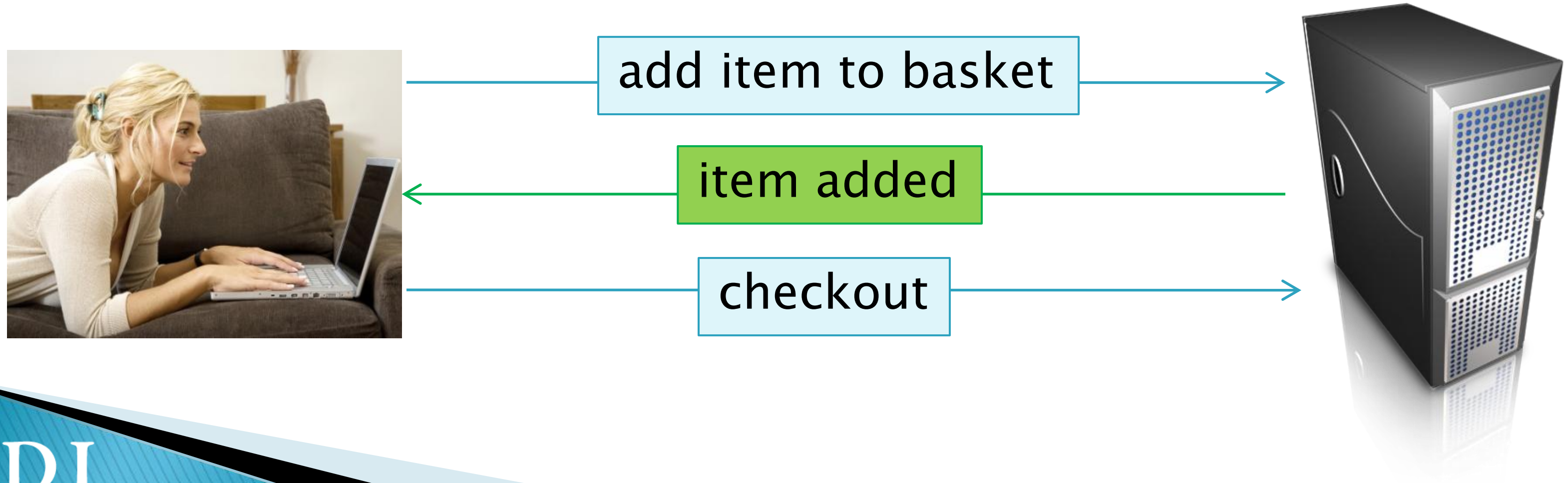
Managing state information

EHSDI

eBuzima

Conversational State

- A servlet on its own has no mechanism for associating consecutive requests from the same client
- In order to implement servlet for things like shopping carts we need a way of sharing data between the requests of a client



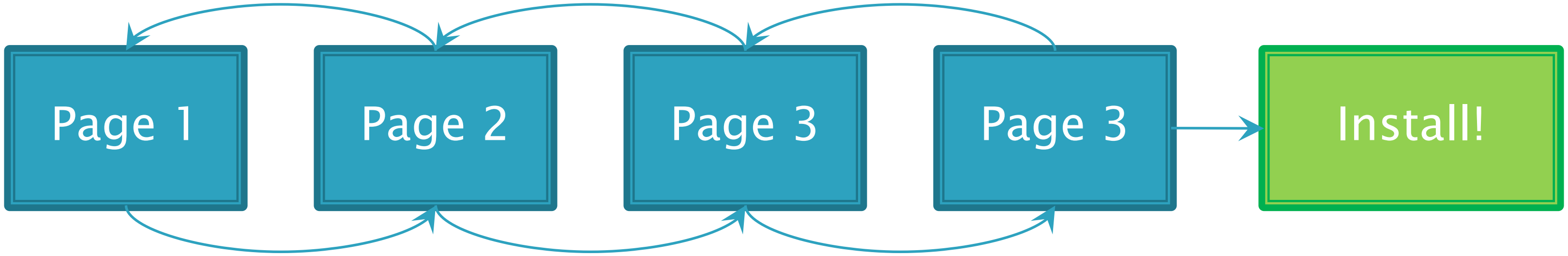
add item to basket

item added

checkout

Conversational State

- Another example is an installation wizard such as the one in OpenMRS 1.5+
- We need to remember the answers for each page...

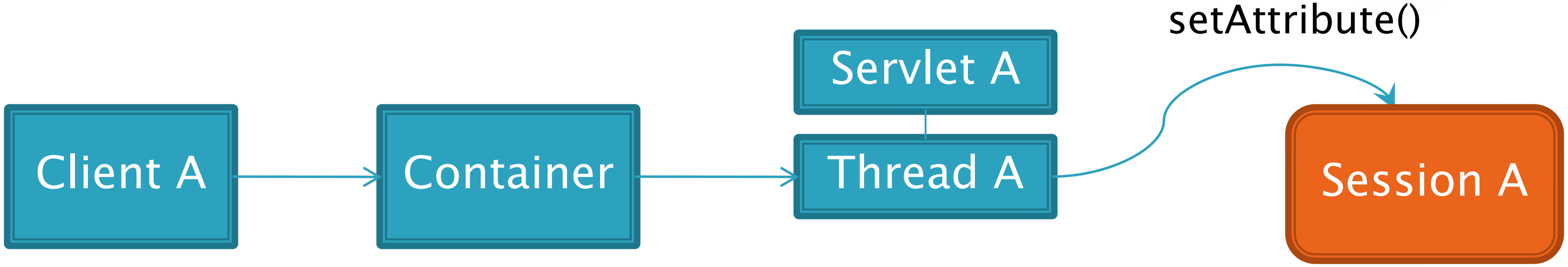


Session Objects

- To share data between client requests, we use the HttpSession object, which has these members:
  - `setAttribute(String name, Object value)` – stores a named value
  - `getAttribute(String name)` – retrieves the named value
- We can store anything in a session object, though it is not recommended to store large amount of data

Session Objects

add iPod to shopping basket



setAttribute()

Servlet A

Client A

Container

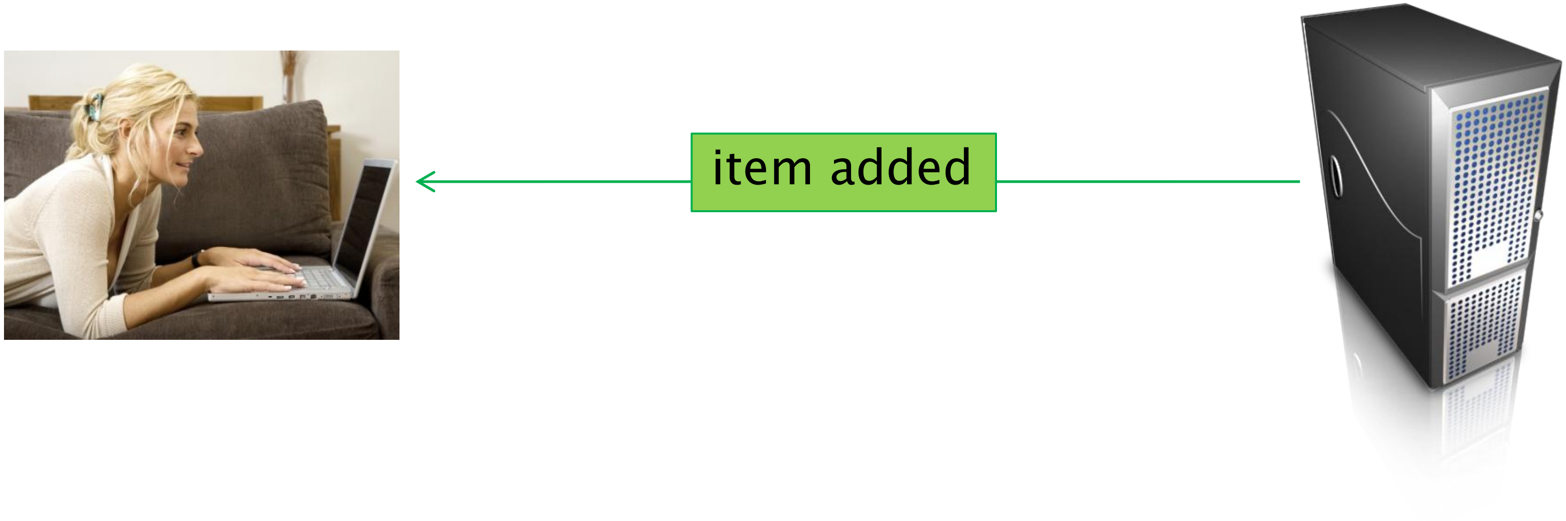
Thread A

Session A

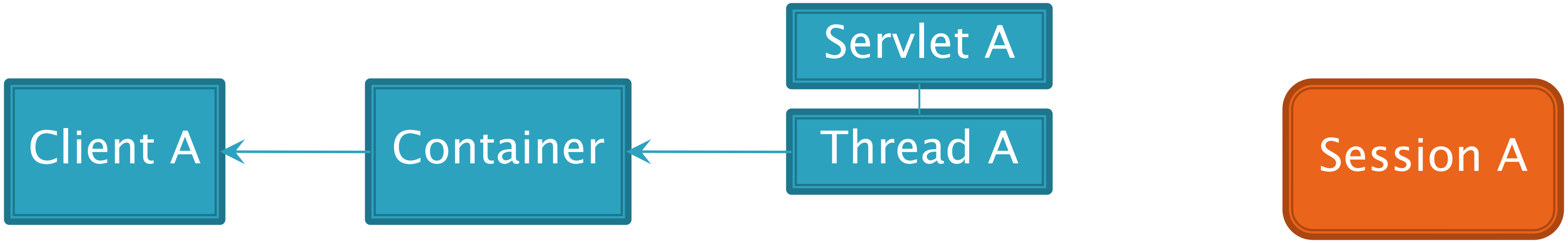
client sends request to
container to add iPod
to basket

servlet adds iPod item to session
associated with this client,
as an attribute

Session Objects



item added



Servlet A

Thread A

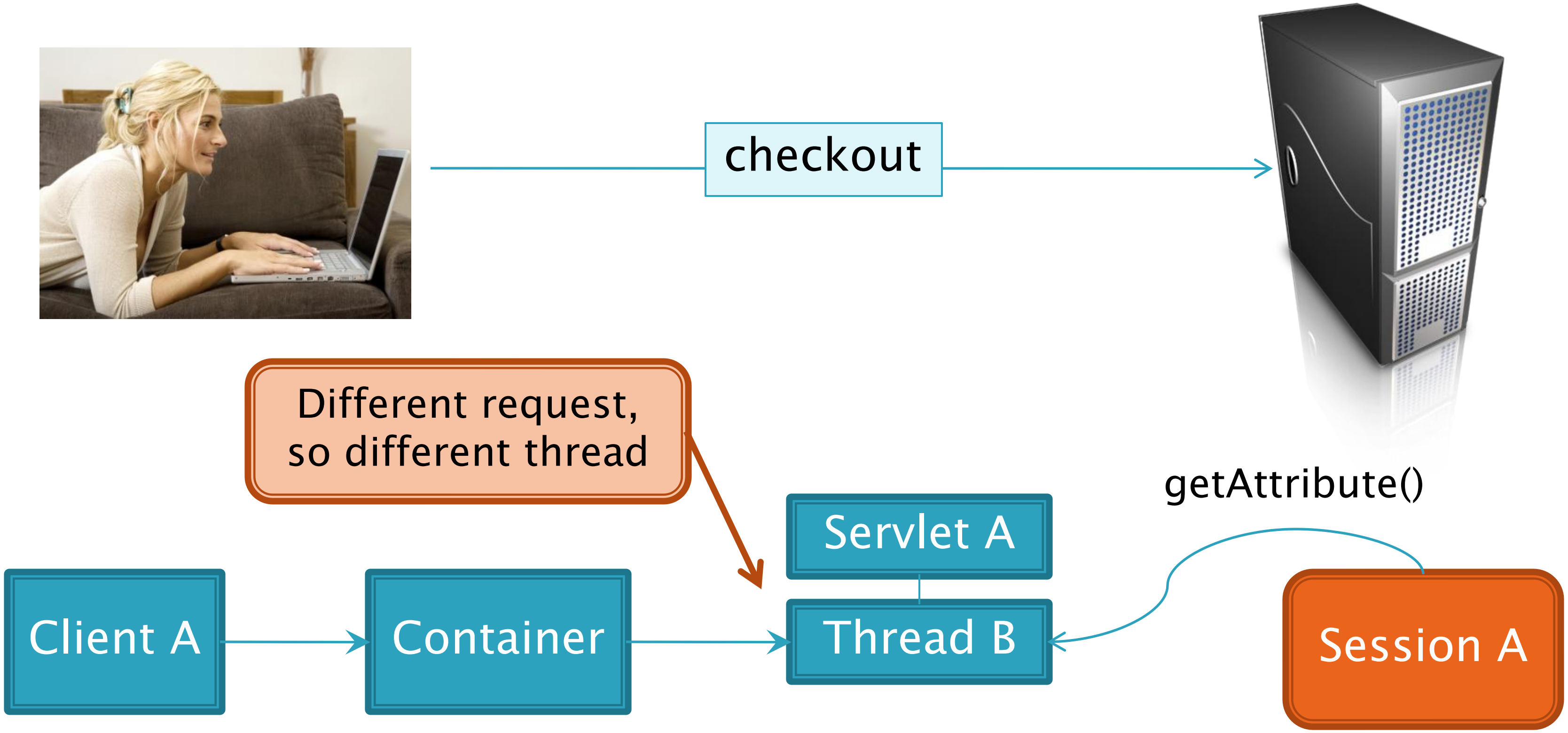
Client A

Container

Session A

servlet returns a response to the client
to confirm that the iPod was added to their
shopping basket

Same Client, Same Session



checkout

Different request, so different thread

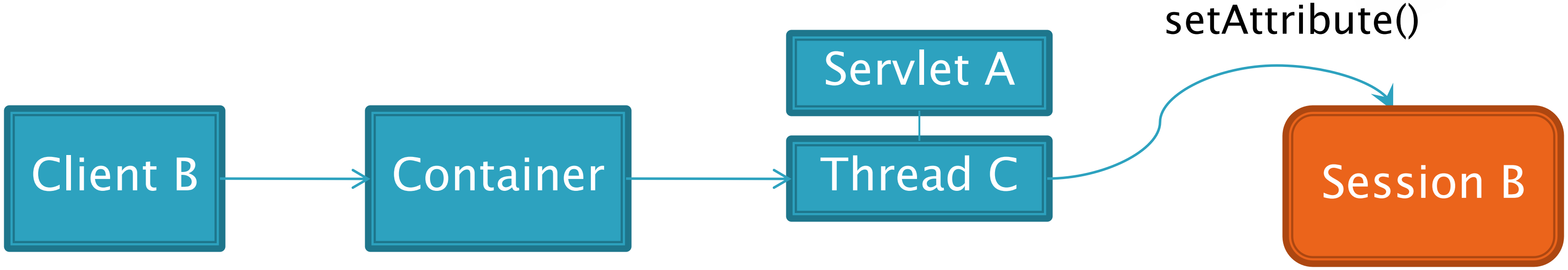
getAttribute()

client requests to purchase the items in their shopping basket

servlet gets the iPod item from the session associated with this client

Different Client, Different Session

add XBox to shopping basket



setAttribute()

Client B

Container

Servlet A

Thread C

Session B

client sends request to
container to add XBox
to basket

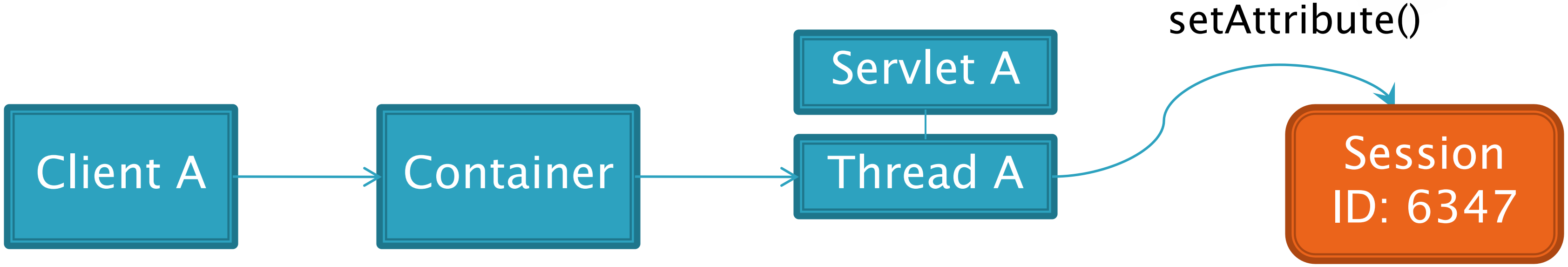
servlet adds XBox item to session
associated with *this* client,
as an attribute

Who Owns Which Session?

- HTTP is a stateless protocol so every request appears to come from a new client
- IP addresses aren't necessarily unique, so they can't be used
- Instead, when a client makes their first request, the server generates a unique session ID, and sends this back to the client
- When the client makes another request, they identify themselves using the session ID

First Request, New Session ID

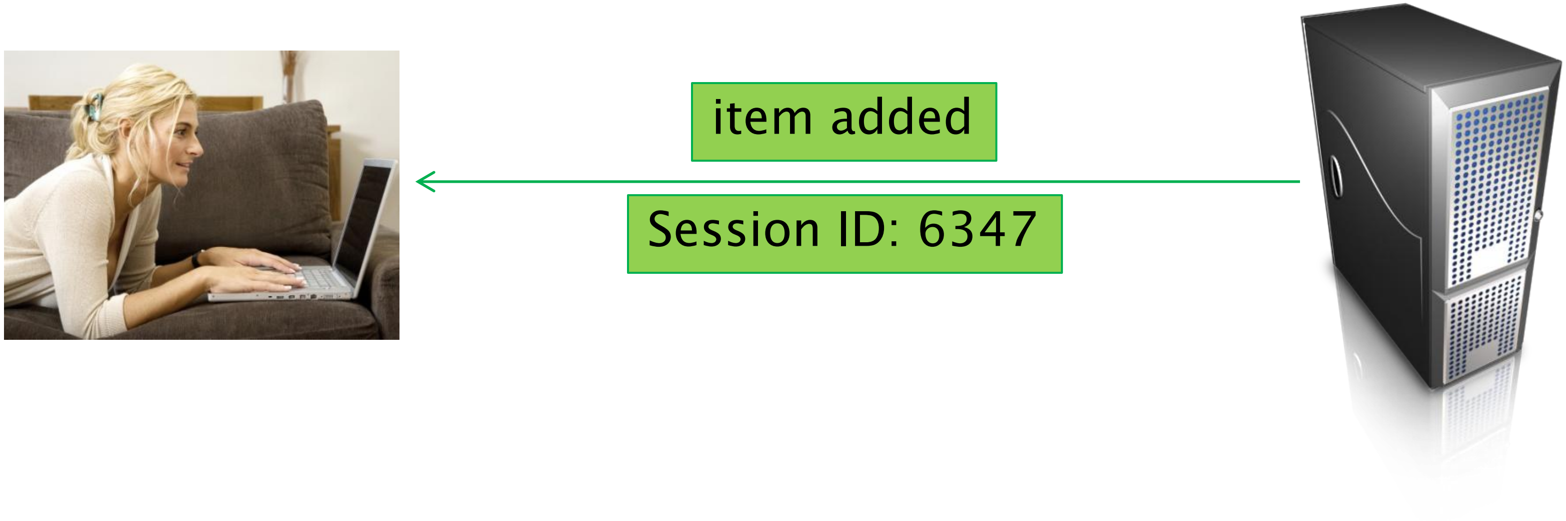
add iPod to shopping basket



client sends request to
container to add iPod
to basket

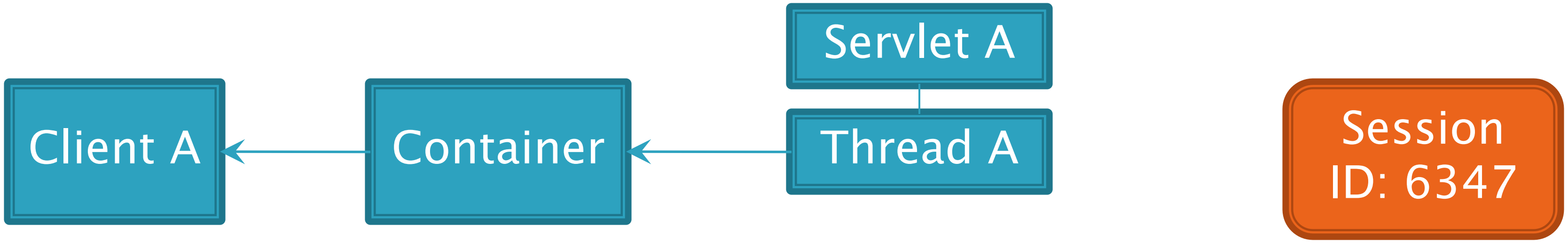
container creates new HttpSession
object with a unique session ID

Session ID Returned to Client



item added

Session ID: 6347



Servlet A

Client A

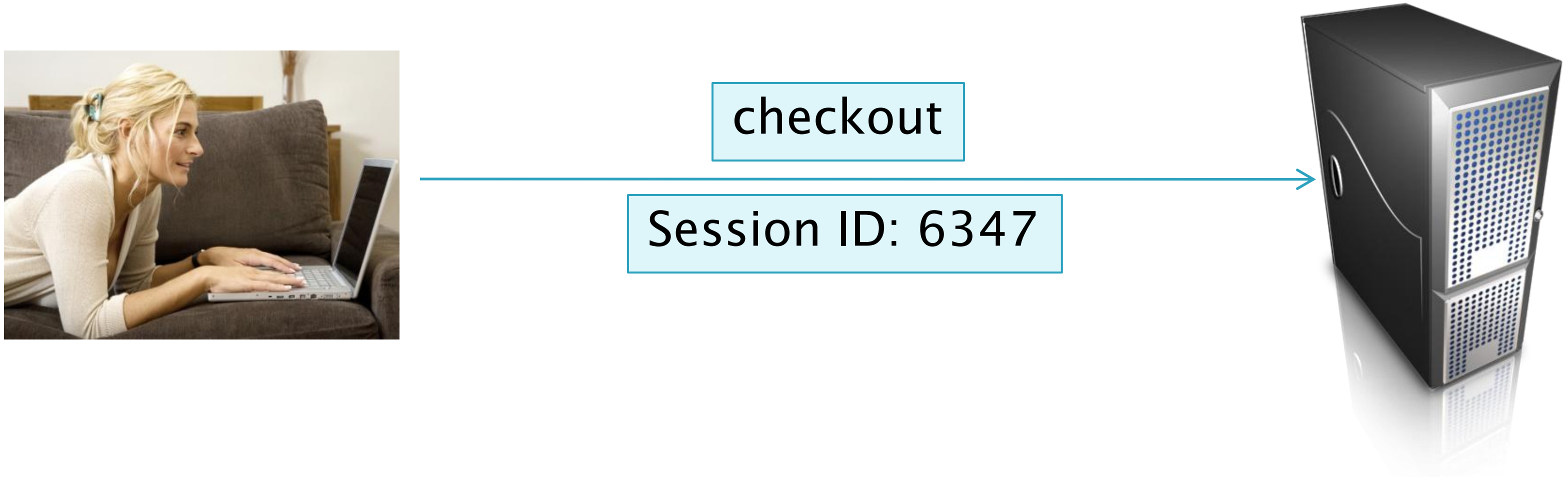
Container

Thread A

Session ID: 6347

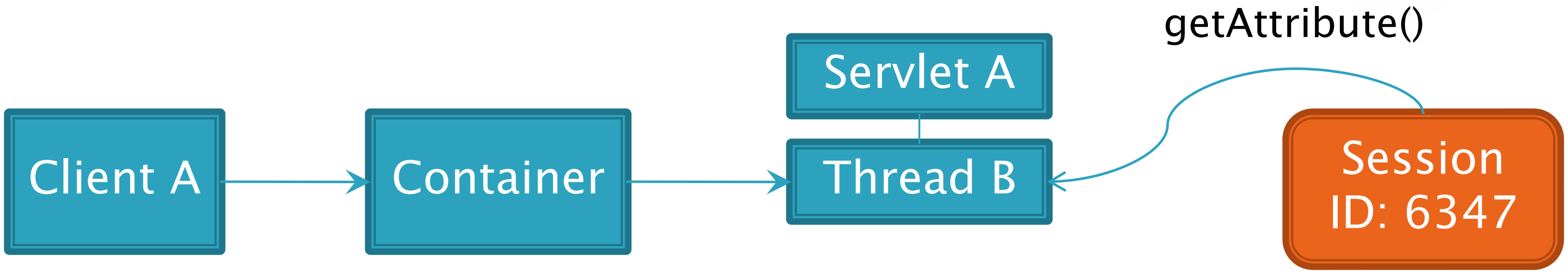
container returns a response which includes the new session ID

Session ID Now Sent By Client



checkout

Session ID: 6347



getAttribute()

Servlet A

Client A

Container

Thread B

Session ID: 6347

client sends the session ID
as part of its request

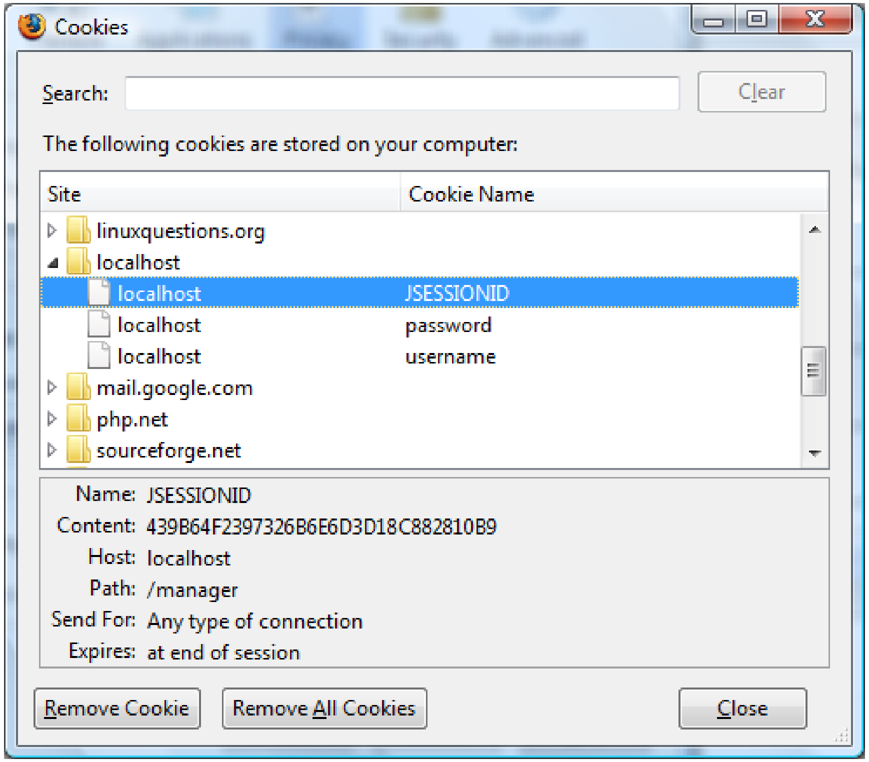
container gets the session object
associated with this client
using the session ID

Sending Session IDs

- The easiest way for the container and servlet to send and receive a session ID is using *cookies*
- A cookie is a name/value pair, it can be:
  - Transmitted in an HTTP request or response
  - Stored on the clients computer

Cookies

- Firefox allows us to view the cookies that are stored on our machine



The following cookies are stored on your computer:

Site	Cookie Name
linuxquestions.org	
localhost	
localhost	JSESSIONID
localhost	password
localhost	username
mail.google.com	
php.net	
sourceforge.net	

Name: JSESSIONID
Content: 439B64F2397326B6E6D3D18C882810B9
Host: localhost
Path: /manager
Send For: Any type of connection
Expires: at end of session

The First Response

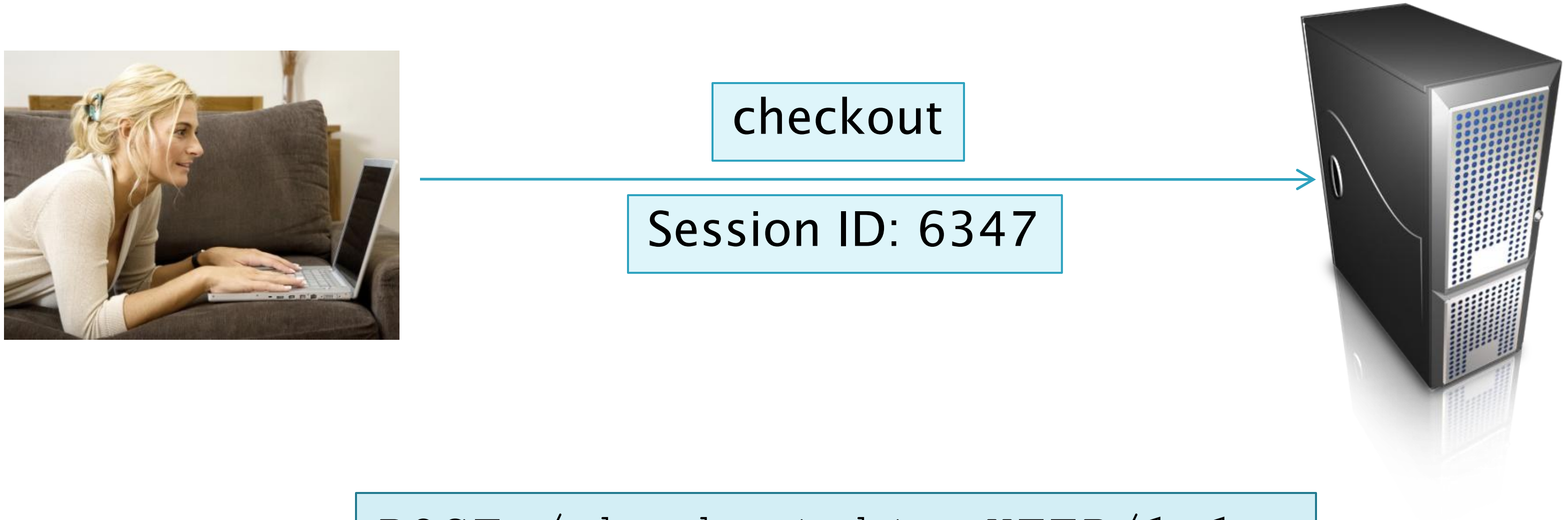
item added

Session ID: 6347

```
HTTP/1.1 200 OK
Set-Cookie: JSESSIONID=6347
Content-Type: text/html
Content-Length: 456
Connection: close

<html>
...
</html>
```

The Second Request



checkout

Session ID: 6347

```
POST /checkout.htm HTTP/1.1
Host: mysupercoolshop.com
User-Agent: Mozilla/5.0
Cookie: JSESSIONID=6347
Accept: text/xml,text/html
```

Custom Cookies

- Cookies are not just for session IDs – they can store any small piece of data
- We can create them as name/value pairs and add them to the server response…

```
Cookie cookie1 = new Cookie("username", "Bob");
Cookie cookie2 = new Cookie("password", "x23d2");

response.addCookie(cookie1);
response.addCookie(cookie2);
```

Custom Cookies

- They can then be read from the client request
- Unfortunately there is no method to get a named cookie, so we must search ourselves...

```
for (Cookie cookie : request.getCookies()) {
  if (cookie.getName().equals("username")) {
    username = cookie.getValue();
    break;
  }
}
```

Custom Cookies: Expiration

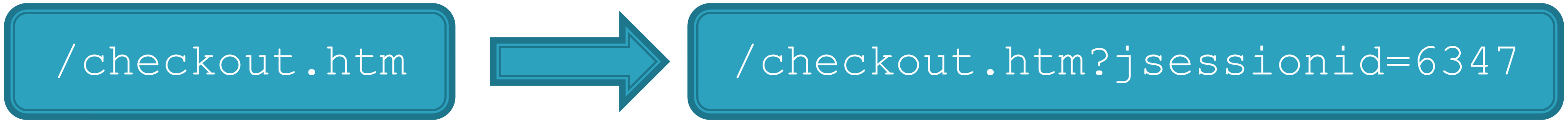
- By default cookies will expire (i.e. be deleted) when the client closes their browser (at the end of their "session")
- But we can specify a time in seconds for the cookie to be kept, so that next time the client opens their browser, it still exists...

```
Cookie cookie = new Cookie("username", "Bob");
cookie.setMaxAge(10000); // seconds

response.addCookie(cookie);
```

URL Rewriting

- Sometimes cookies will not work
  - The user may have disabled them in their browser
- In such a case the container will attempt to use URL rewriting instead
- This means the session ID is appended to every link that the user might click, e.g.



/checkout.htm

/checkout.htm?jsessionid=6347

URL Rewriting: encodeURL

- This requires every link in your site to be encoded using `response.encodeURL()`

```
out.println("<a href=\"/checkout.htm\">"
 + "Go to checkout</a>");
```

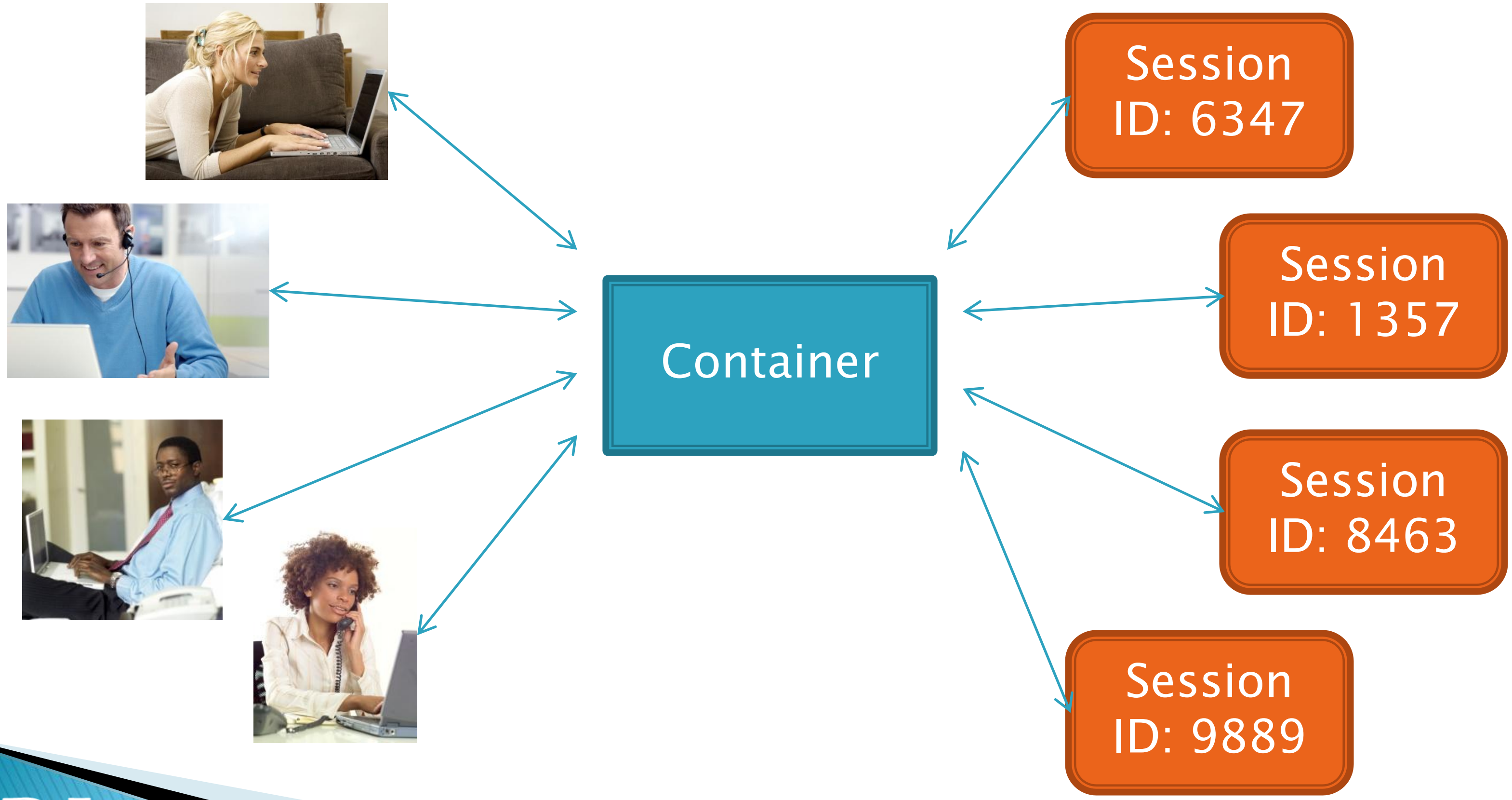
```
out.println("<a href=\""
 + response.encodeURL("/checkout.htm")
 + "\">Go to checkout</a>");
```

Getting The Session Object

- The session object is accessed through the request object, i.e
  - `HttpSession session = request.getSession();`
- If it is a client's first request, then `getSession()` will return a new session
- Otherwise, it will return an existing session object
- `session.isNew()` will tell us if it is a new session or an existing one

Managing Sessions

- Many users means many session objects


Container

Session ID: 6347

Session ID: 1357

Session ID: 8463

Session ID: 9889

Expiring Sessions

- Sessions objects use up memory so they should be deleted once a client has finished making requests
- But there is no mechanism in HTTP for knowing when a client has finished
- We can though tell the container to delete sessions after they haven't been used for a certain amount of time

Expiring Sessions

- When we create a session we can call `setMaxInactiveInterval()` to tell the container how long to wait before deleting the session
- Or we set the session timeout value in the DD

```
<web-app>
  ...
  <session-config>
    <session-timeout>15</session-timeout>
  </session-config>
</web-app>
```

Tells the container to delete sessions once they haven't been accessed for 15 minutes

References

- Books
  - Head First Servlets and JSP (O'Reilly)
- Websites
  - http://java.sun.com/javaee/reference/tutorials/