Comet
(Reverse Ajax)

Sang Shin
http://www.javapassion.com
“Learn with Passion!”

Agenda

- Web 2.0 Phenomenon
- Introduction to Comet
- Issues of supporting Comet
  - Client side
  - Server side
- Bayeux and Cometd
- Step by step process of building Comet application using Cometd
- Cometd server side configuration

Web 2.0 Phenomenon

Web 2.0 – Driven by Participation

A Web by the people, for the people.

- Documents on the web increasingly generated by users

- "Information Age" -> "Participation Age"
- More time-sensitive, collaborative participation
  - Chatting
  - Multi-user gaming

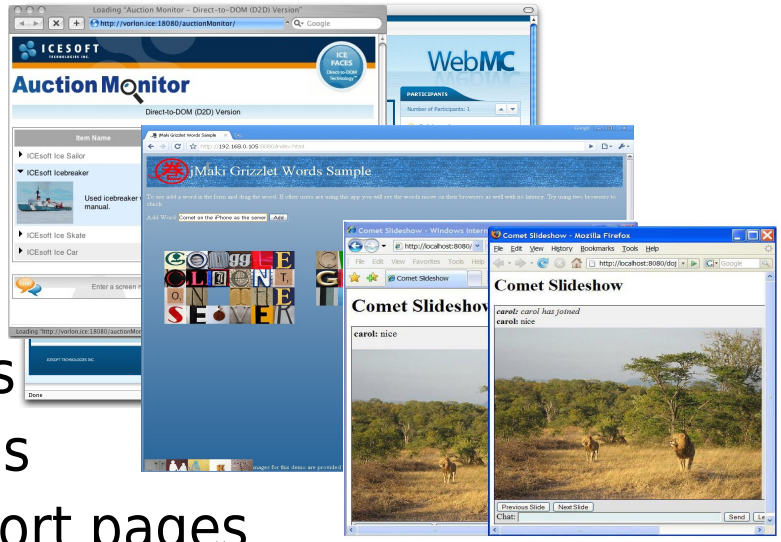
The Asynchronous Web Revolution

The Web enters the Participation Age.

- The first-generation Ajax is not “asynchronous” enough
  - It handles a single user client-side asynchronicity only
- Full asynchronicity includes “updates pushed from server to the clients” at any time
  - Server-driven page update
- Allow users to communicate and collaborate within the web application
- Called “Comet”, “Server-side Push”, “Ajax Push”, or “Reverse Ajax”

Applications in the Participation Age

- Distance learning
- Collaborative authoring
- Auctions
- Shared WebDAV filesystem
- Blogging and reader comments
- SIP-coordinated mobile applications
- Hybrid chat/email/discussion forums
- Customer assistance on sales/support pages
- Multi-step business process made collaborative
- Shared trip planner or restaurant selector with maps
- Shared calendar, “to do” list, project plan
- Games

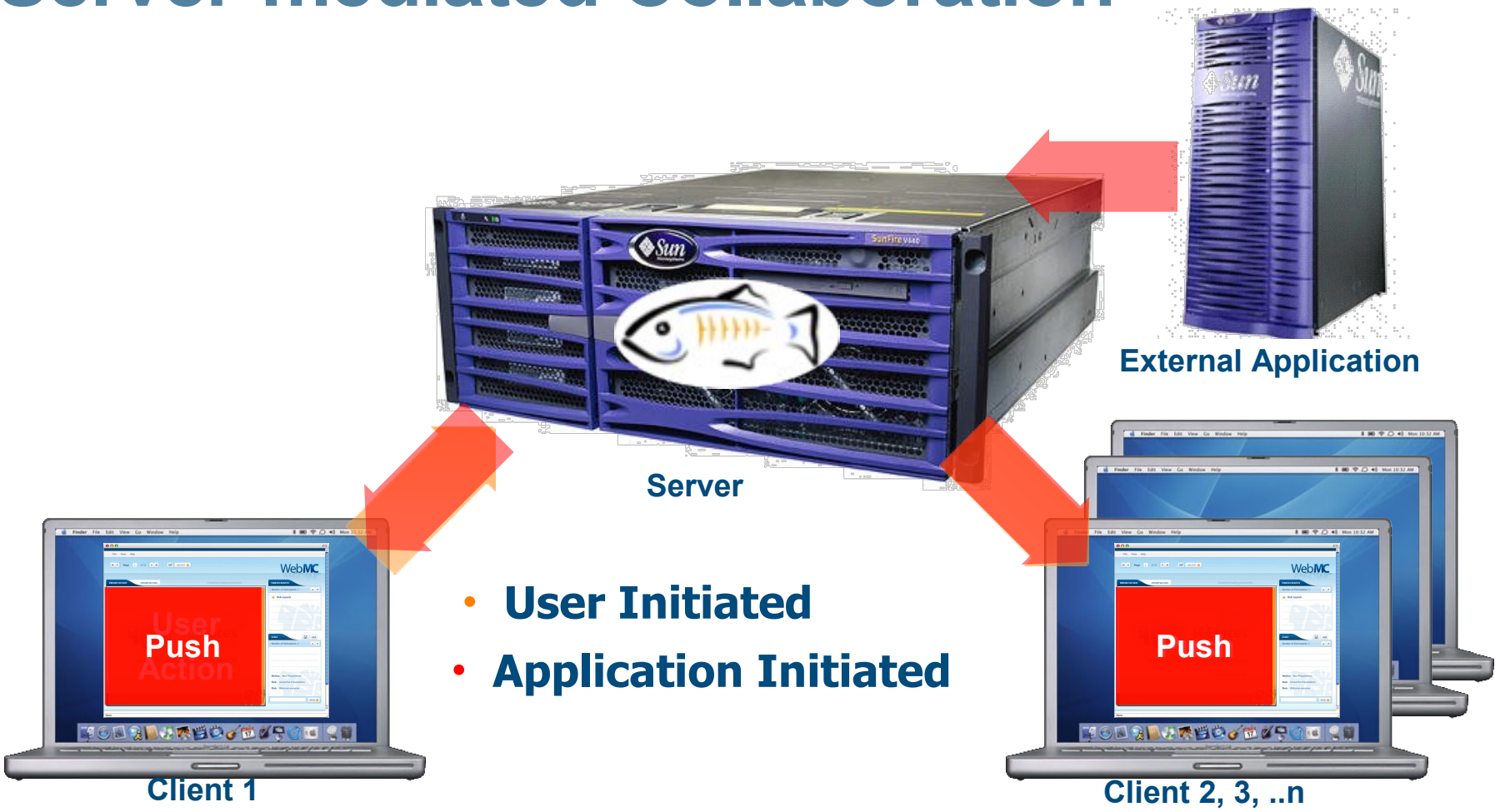


Demo:
Running Slideshow Comet Application

Exercise_3 of "4293_ajaxcomet.zip"

Introduction to Comet

Server-mediated Collaboration



- User Initiated
- Application Initiated

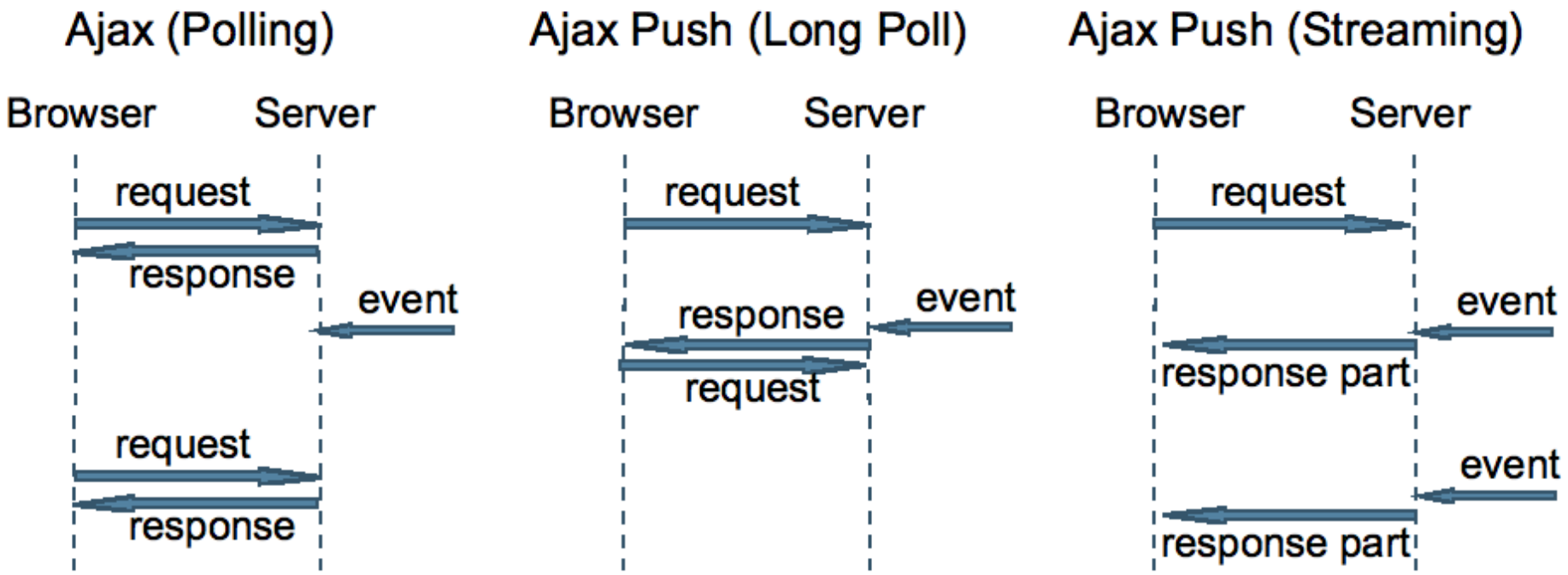
What is Comet (Ajax Push)?

Responsive, low-latency interaction for the web

- A programming technique that enables web servers to send data to the client without having any need for the client to request for it
- Allows creation of highly responsive, event-driven web applications
  - Keep clients up-to-date with data arriving or changing on the server, without frequent polling
- Advantages
  - Lower latency, not dependent on polling frequency
  - Server and network do not have to deal with frequent polling requests to check for updates

Ajax Poll vs Ajax Push

Bending the rules of HTTP.



- **Comet** Refers to both the Long Polling and Streaming methods of web programming

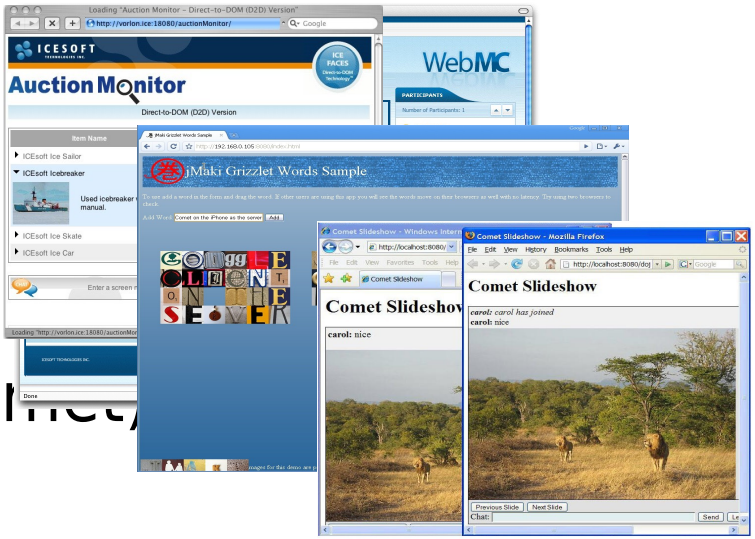
Ajax Poll vs Ajax Push

Bending the rules of HTTP.

- Poll:
  - Send a request to the server every X seconds
  - The response is “empty” if there is no update
- Long Poll: (most popular)
  - Client sends a request to the server, server waits for an event to happen, then send the response
  - The response is never empty
  - No client side hack needed – the server functions like a slow server
- Http Streaming:
  - Client sends a request, server waits for events, stream multi-part/chunked responses, and then wait for the events
  - The response is continually appended to

Comet Examples

- GMail and GTalk
- Meebo
- 4homemedia.com
  (using GlassFish project's Comet)
- JotLive
- KnowNow
- Many more ...



Issues of Supporting Comet

Limit on the Browser Side

- In most web browsers, the number of concurrent HTTP connections to a given domain is limited to two (to six)
- For common synchronous HTTP requests, this is not a problem
  - The connection is quickly created and released
- But in Comet environment, where the connection is kept open for long time, this causes a problem
  - The browser suspends all requests until a new connection is available

Technology Solution

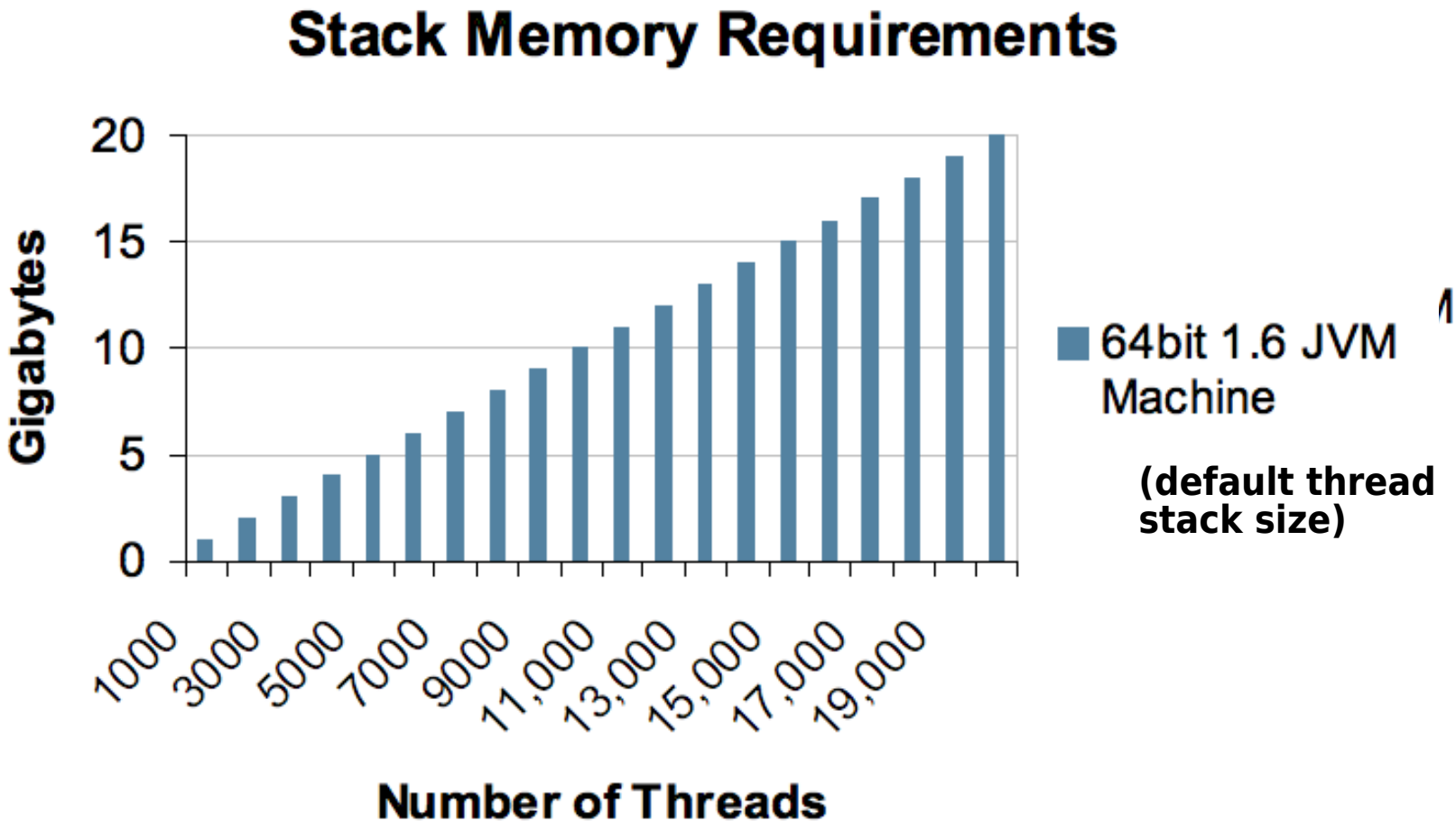
- Is there a way to share a single HTTP connection among multiple clients?
- This is how *Bayeux* protocol comes in

Server Architecture Challenge – Can it Scale?

- Using blocking, synchronous technology will result in a blocked thread for each open connection that is “waiting”
  - Every blocked thread will consume memory
  - This lowers scalability and can affect performance
- To get the Java Virtual Machine (JVM™) to scale to 10,000 threads and up needs specific tuning and is not an efficient way of solving this
- Servlet 2.5 are an example of blocking, synchronous technology

Architecture Challenges

The serious effect of blocking threads



Technology Solution: NIO

- Use new I/O (NIO) non-blocking sockets to avoid blocking a thread per connection
  - Release the original request thread while waiting for an event
  - May process the event/response on another thread than the original request
  - Can handle large number of clients (e.g. 10’000) “waiting” without any threads processing or blocked
  - Most modern web servers (GlassFish, Jetty, Tomcat, ..) take advantage of the NIO

Bayeux and Cometd

What Is Bayeux Protocol?

- Supports multiple “channels” over a single HTTP connection
  - Solves the two HTTP connection limit on the browser problem
  - Each channel represents a separate conversation
- Each channel has a unique name
  - /my/channel-name, /my/*
- Based on publish/subscribe model
  - You subscribe a channel to receive messages
  - You publish a message onto a channel - Message router on the server routes the message to anybody who subscribed the channel
- Server functions as a message router (event router)
  - Client to client communication through the server

/meta/* Channels

- The "/meta/*" channel is reserved for communications with the event router itself (including connection setup, tear-down, and re-connection), and all conformant clients and servers must implement the following meta-channel verbs:
  - /meta/handshake
  - /meta/connect
  - /meta/subscribe
  - /meta/unsubscribe
  - /meta/disconnect

Message Format in Bayeux Protocol

- Message format is in the format of JSON

```
{
   "channel": "/some/name",
   "clientId": "83js73jsh29sjd92",
   "data": { "myapp" : "specific data", value: 100 }
}
```

What Is Cometd?

- Implementation of Bayeux protocol
- Easy programming model
  - Client - consists JavaScript technology (DOJO toolkit) or Java libraries that implement Bayeux
  - Server – message (event) router on the server side
- Server-side implementation is typically platform-specific servlet
  - This servlet functions as a event (message) router
  - Typically built-on the top of Web server-specific Comet API
  - The standardization effort in Servlet 3.1 (Asynchronous Servlet)

Handshaking

- /meta/handshake channel - request message

```
[{"version":"1.0","minimumVersion":"0.9",
"channel":"/meta/handshake","id":"0","ext":{"json-
comment-filtered":true},"supportedConnectionTypes":
["long-polling","callback-polling"]}]
```

- /meta/handshake channel - response message

```
[{"channel":"/meta/handshake","version":"1.0","suppor
tedConnectionTypes":["long-polling","callback-
polling"],"minimumVersion":"0.9","id":"0","clientId":
"b48be050fed5d347","successful":true,"advice":
{"reconnect":"retry","interval":0,"multiple-
clients":false},"authSuccessful":true}]
```

Connect

- /meta/connect channel - request message

```
[{"channel":"/meta/connect","clientId":"b48be050fed5d
347","connectionType":"long-polling","id":"1"}]
```

- /meta/connect channel - response message

```
[{"channel":"/meta/connect","clientId":"b48be050fed5d
347","successful":true,"id":"1","advice":{"reconnect"
:"retry","interval":0,"multiple-
clients":false},"timestamp":"Fri, 13 Feb 2009
03:45:38 GMT"}]
```

Subscribe

- /meta/subscribe channel - request message

```
[{"channel":"/meta/subscribe","subscription":"/chat/d
emo","clientId":"b48be050fed5d347","id":"2"},{"data"
:{"user":"g","join":true,"chat":"g has
joined"},"channel":"/chat/demo","clientId":"b48be050f
ed5d347"
,"id":"3"}]
```

- /meta/subscribe channel - response message

```
[{"channel":"/meta/subscribe","successful":true,"clie
ntId":"b48be050fed5d347","subscription":"/chat
/demo","id":"2"},
{"channel":"/chat/demo","successful":true,"clientId":
"b48be050fed5d347","id":"3"},
 {"channel":"/chat/demo","data":
{"user":"g","join":true,"chat":"g has
joined"},"id":"3","clientId":"b48be050fed5d347"
}]
```

Data

- data channel - request message

```
[{"data":{"slide":"/cometSlideshow/images/image0.jpg"},
"channel":"/chat/demo", "clientId":"b48be050fed5d347"
,"id":"5"}]
```

- data channel - response message

```
[{"channel":"/chat/demo","successful":true,"clientId":"
b48be050fed5d347","id":"5"}, {"channel":"/chat/demo",
"data":
{"slide":"/cometSlideshow/images/image0.jpg"},"id":"5",
"clientId":"b48be050fed5d347"}]
```

Step by Step Process of Building Comet Application using Cometd (Client side)

Steps

1. Configure Cometd servlet in the web.xml
2. Initialize (Connect to Cometd server)
3. Subscribe a channel
4. Write callback function
5. Publish to the channel
6. Unsubscribe the channel
7. Disconnect

Step1: Configure Cometd Servlet in your Application's web.xml

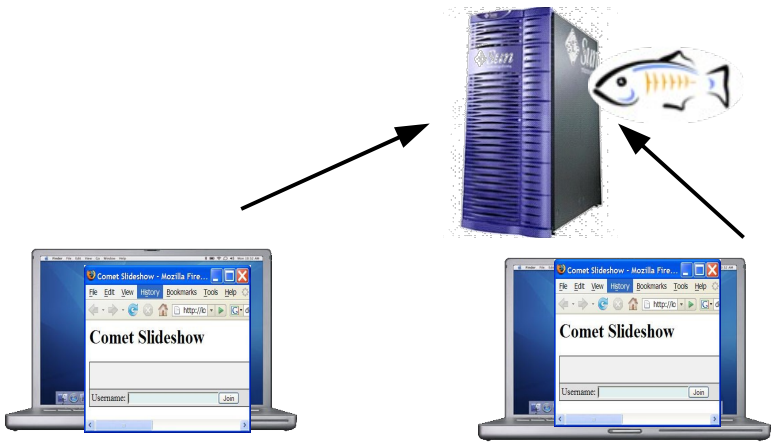
```
<servlet>
    <servlet-name>Grizzly Cometd Servlet</servlet-name>
    <servlet-class>
        com.sun.grizzly.cometd.servlet.CometdServlet
    </servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>Grizzly Cometd Servlet</servlet-name>
    <url-pattern>/cometd/*</url-pattern>
</servlet-mapping>
```

Step2 : Initialize (Connect to Cometd Server)

```
dojo.require("dojox.cometd");

// Initialize a connection to the given Cometd server:
// the GlassFish Grizzly Bayeux servlet
dojox.cometd.init("cometd");
```

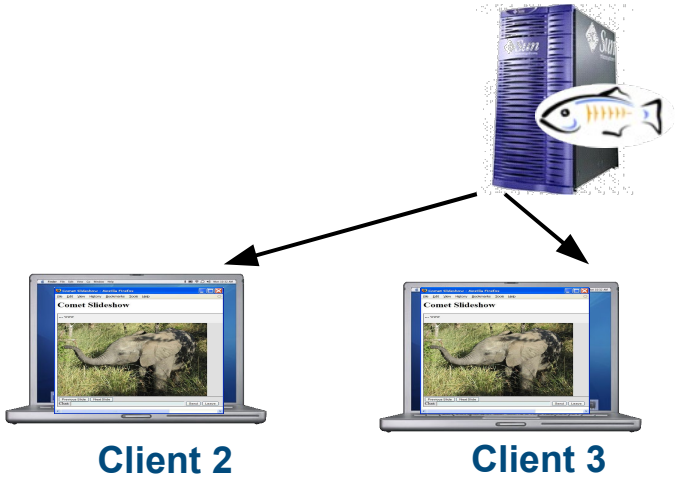
Step3 : Subscribe a Channel

```
// Subscribe a channel with callback function.
// Every time a message is published, the callback function
// (of all clients who subscribed the channel) gets invoked.
dojox.cometd.subscribe(“channel”, "callback receiver",
                                  "callbackFunction");
```



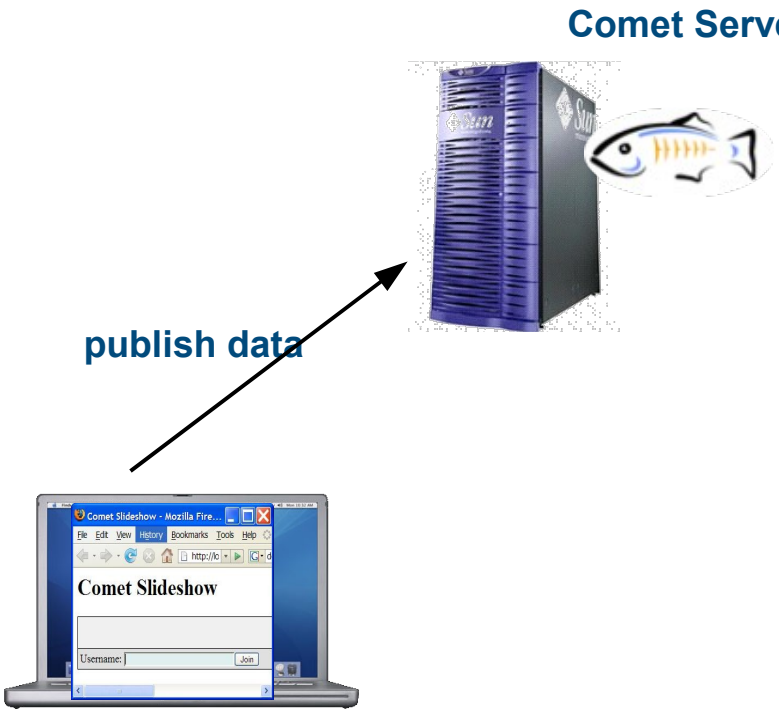
Step4 : Write callback function

```
// The callback function gets invoked when the client receives
// a “published” message from a Cometd server. Typically you
// update the contents on the page.
callbackHandler: function(msg) {
     alert("msg.data.testMessage = " + msg.data.testMessage)
}
```



Step5 : Publish data onto the channel

```
// Publish data (in JSON format) onto the channel.
dojox.cometd.publish("channel", {jsonname: jsonvalue});
```



Step6 : Unsubscribe, Disconnect

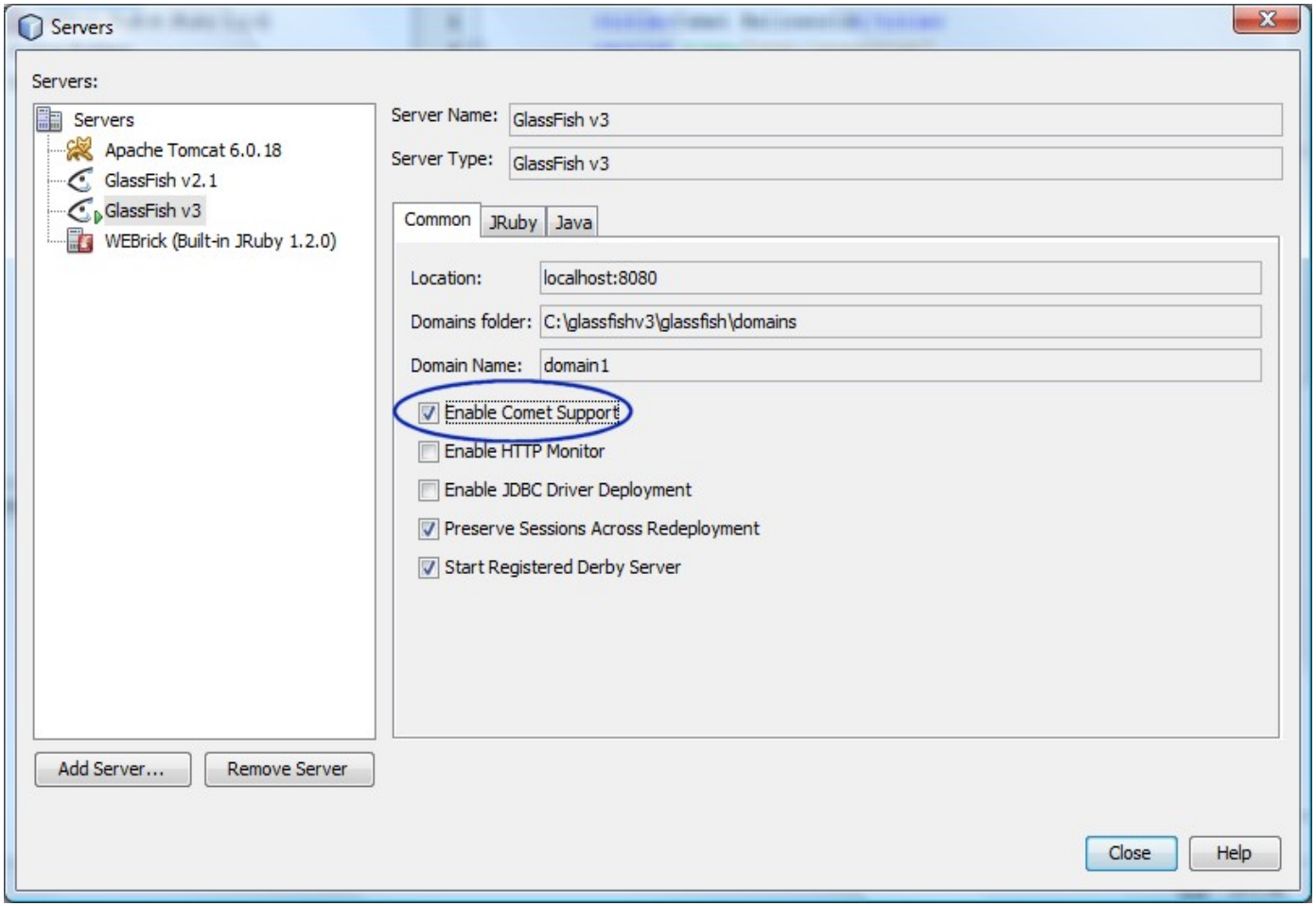
```
// Unsubscribe the channel
dojox.cometd.unsubscribe(“channel”,
                                    myObject, "callbackHandler");

// Disconnect from the Cometd server
dojox.cometd.disconnect();
```

Demo:
Building Helloworld Cometd Application

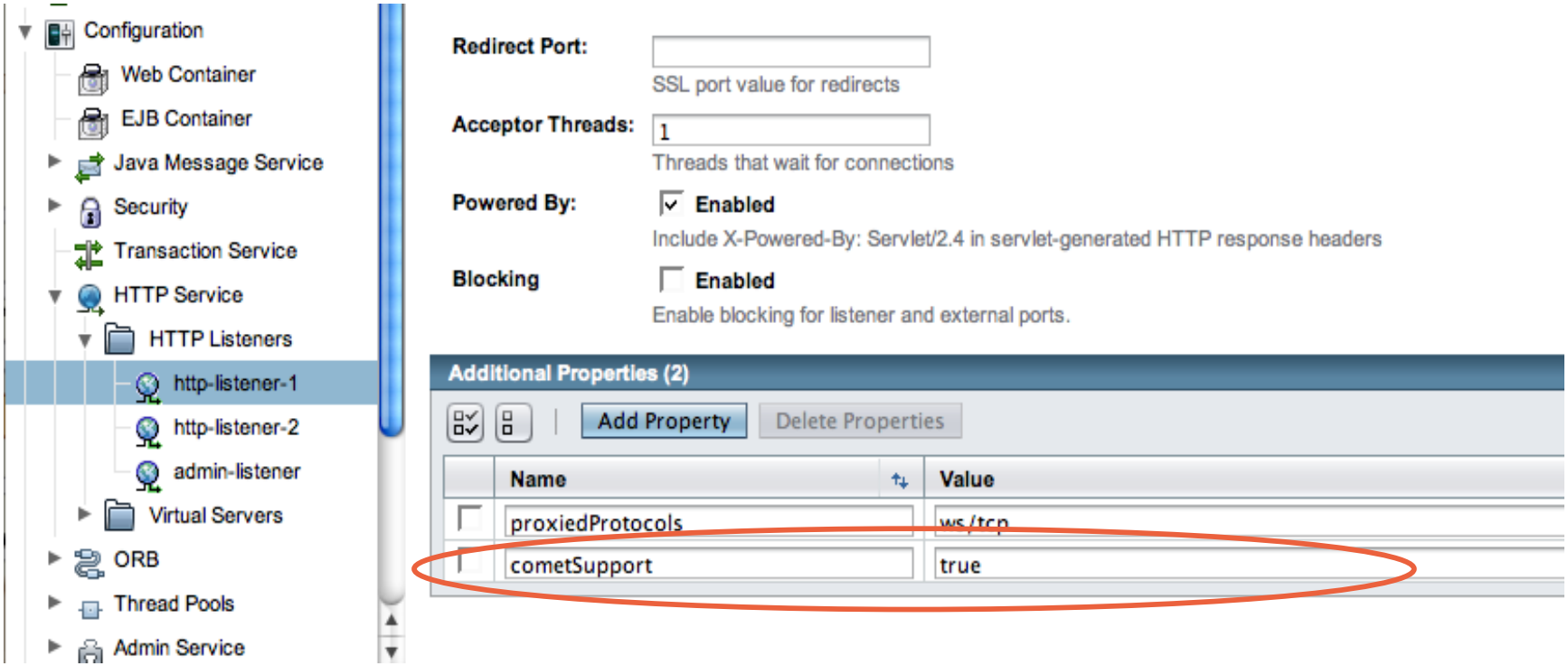
Exercise_1 of “4293_ajaxcomet.zip”

Cometd Configuration on the Server Side

Option#1: Enable Cometd in GlassFish V3 (via NetBeans)

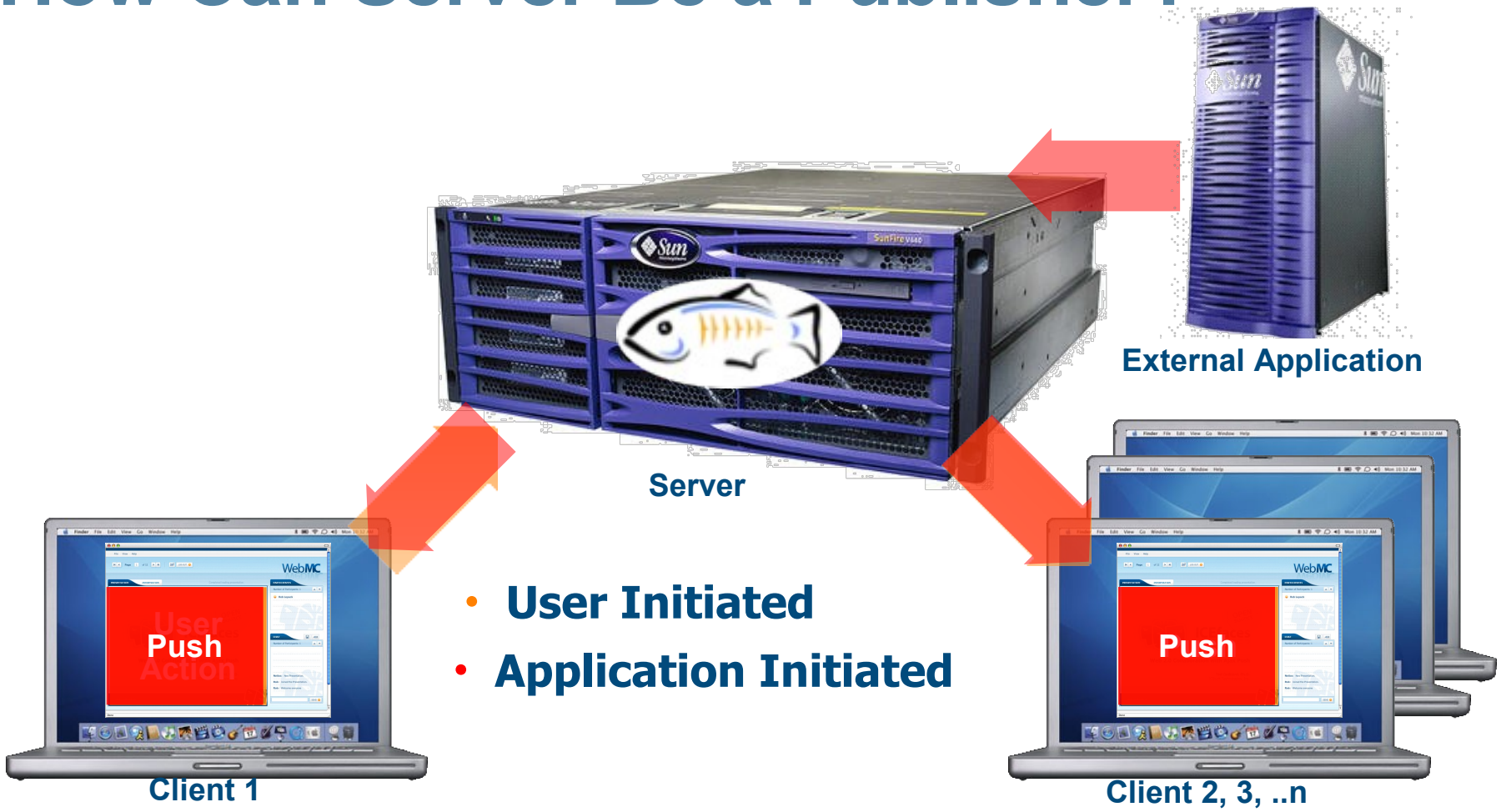


Option#2: Enable Cometd in GlassFish v3 (via GlassFish Admin Console)



Server Side Cometd Implementations

How Can Server Be a Publisher?



- User Initiated
- Application Initiated

Demo:
Server Publishing

Exercise_4 of “4293_ajaxcomet.zip”

Thank you!

Sang Shin
Michèle Garoche
http://www.javapassion.com
“Learn with Passion!”