Implementing Asynchronous Web Application using Grizzly's Comet.

Jeanfrancois Arcand
Staff Engineer
Java WebTier

Agenda

- Introduction
  - What is Grizzly
  - What is Comet Request Processing
- Comet support in Grizzly
  - Definitions
  - How it works
  - Comet implementation
- What's next
- Questions

What is Grizzly

- Grizzly is a generic NIO framework for building scalable **server** application.
- Grizzly support blocking and non blocking socket operations, over plain or ssl connection.
- Grizzly extensions currently used by several internal projects (Alaska, GlassFish, Tango, etc.) and external projects (Jetty 6.1)
- Grizzly v2 is targetted for GlassFish 9.1, and will be used by the ORB, MQ and possibly JAX-WS.

What is Comet Request Processing

- Definition from Wikipedia:

Comet is a programming technique that enables web servers to send data to the client without having any need for the client to request for it. It allows creation of event-driven web applications which are hosted in the browser.

What makes Comet Application different from traditional application (from http://alex.dojotoolkit.org/?p=545)?

- Fundamentally, they all use long-lived HTTP connections to reduce the latency with which messages are passed to the server.
- In essence, they do not poll the server occasionally. Instead the server has an open line of communication with which it can push data to the client.

What makes Comet Application different from traditional application? (from http://alex.dojotoolkit.org/?p=545)

Comet applications can deliver data to the client at any time, not only in response to user input. The data is delivered over a single, previously-opened connection. This approach reduces the latency for data delivery significantly.

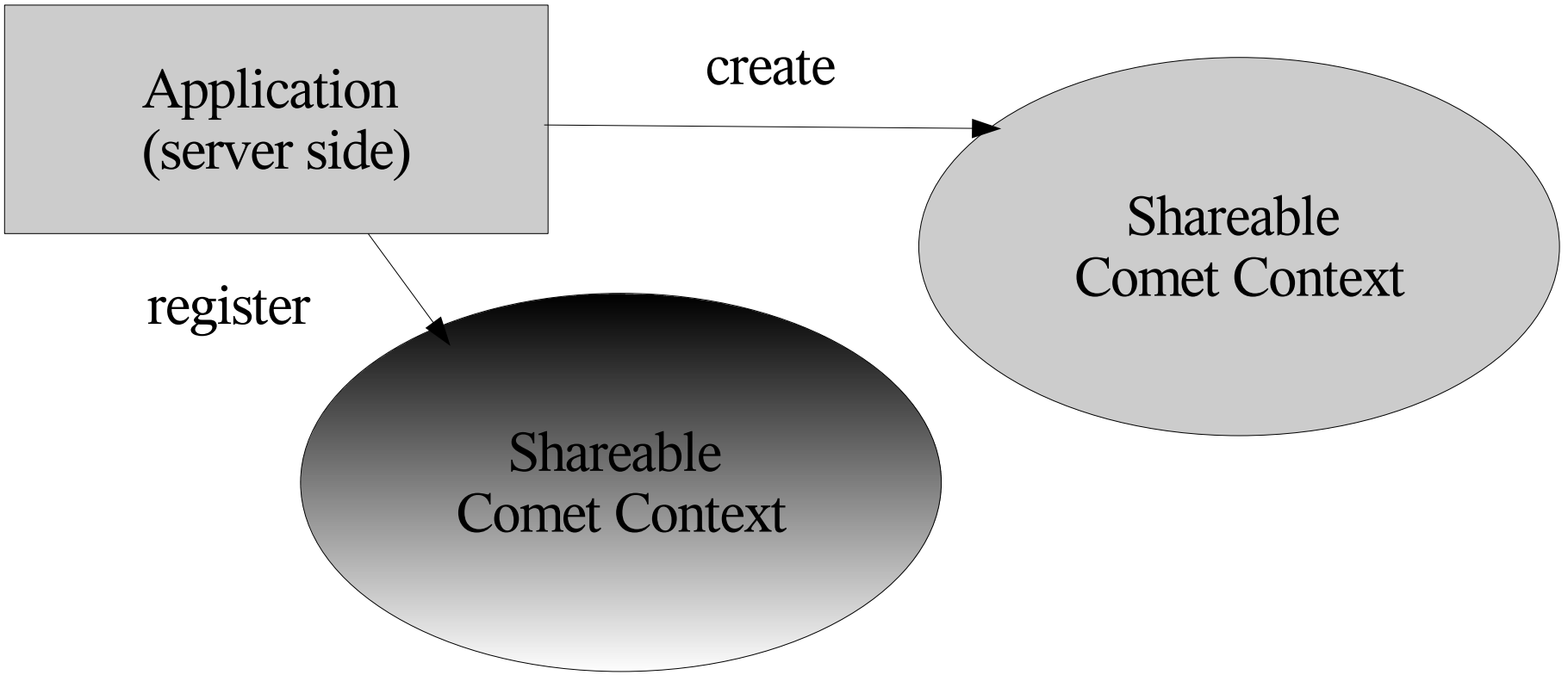
Comet support in Grizzly: Details

- Implemented on top of the Grizzly Asynchronous Request Processing extension [1].
- Hide the complexity of NIO/Asynchronous Request Processing.
- Support clean and ssl connection.
- Make it available to JSF, JSP, Servlet, POJO, JavaScript (Phobos)
- **Main Goal: Make it simple!**

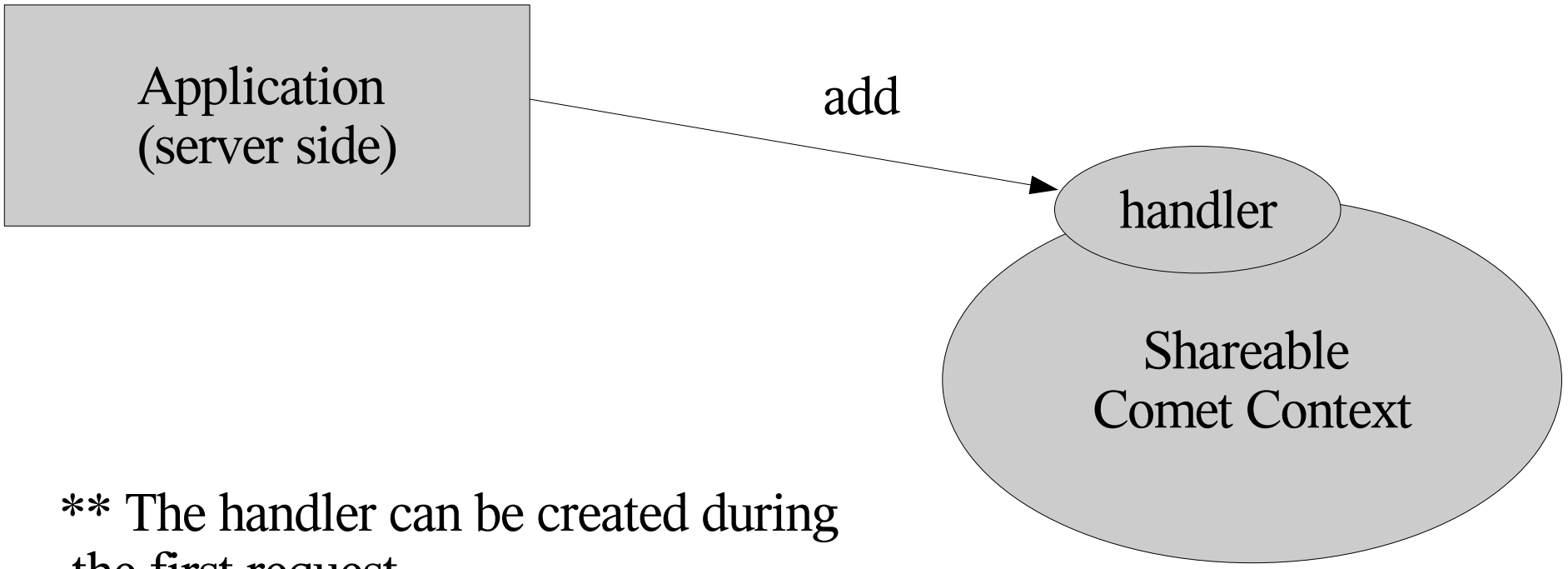
The Grizzly Comet: Definitions

- CometContext: A shareable “space” applications can subscribe and get updated when the context is updated.
- CometEvent: An object containing the state of the Comet Context (content updated, client connected/disconnected, etc.).
- CometHandler: The interface an application must implement in order to be part of one or several CometContext.

First, the application creates a new Comet context (or register to an existing one)



Next, add your Comet request handler to the context. You usually create one handler per connection (not mandatory).

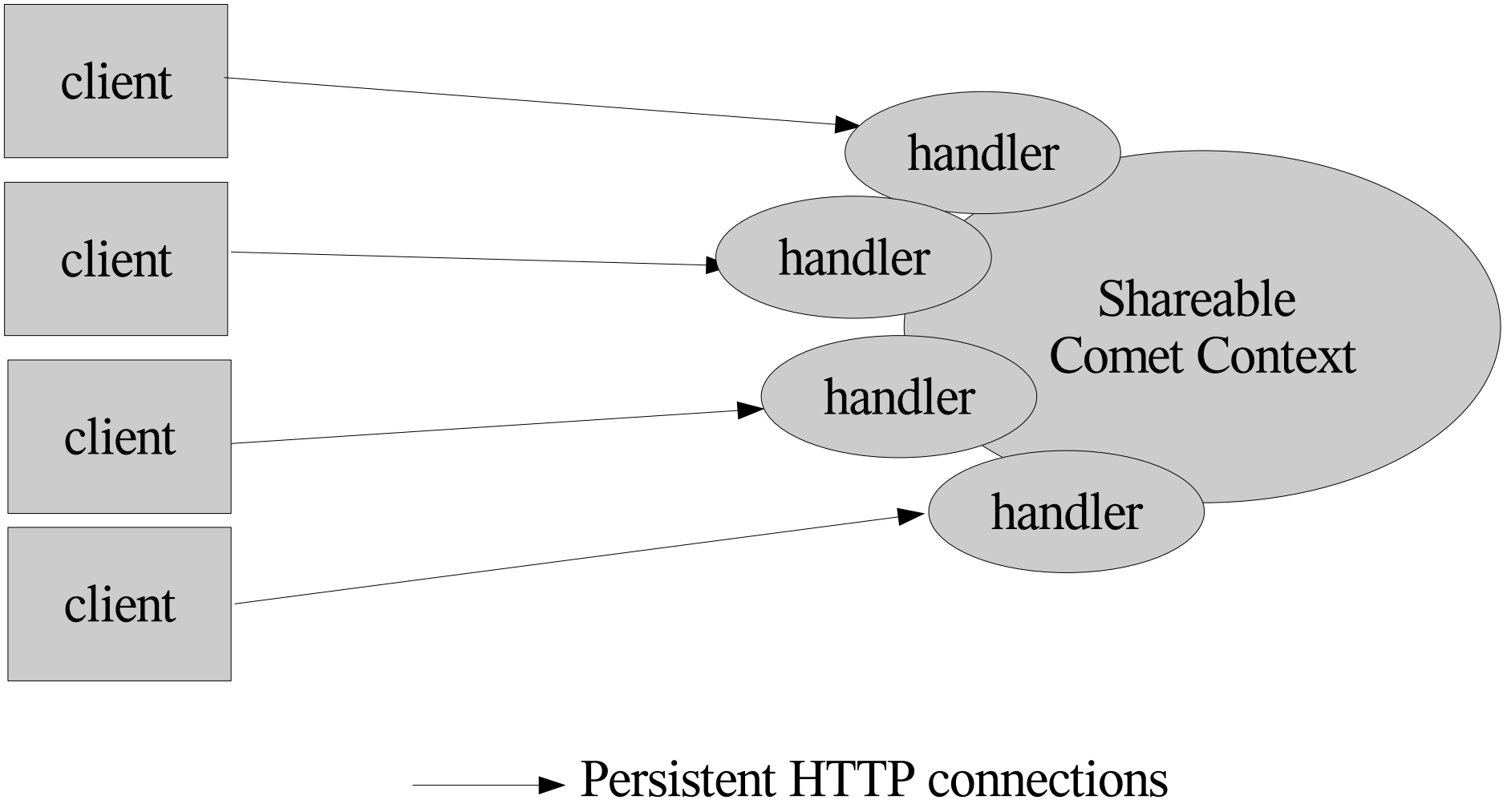


** The handler can be created during the first request.

→ Persistent HTTP connections

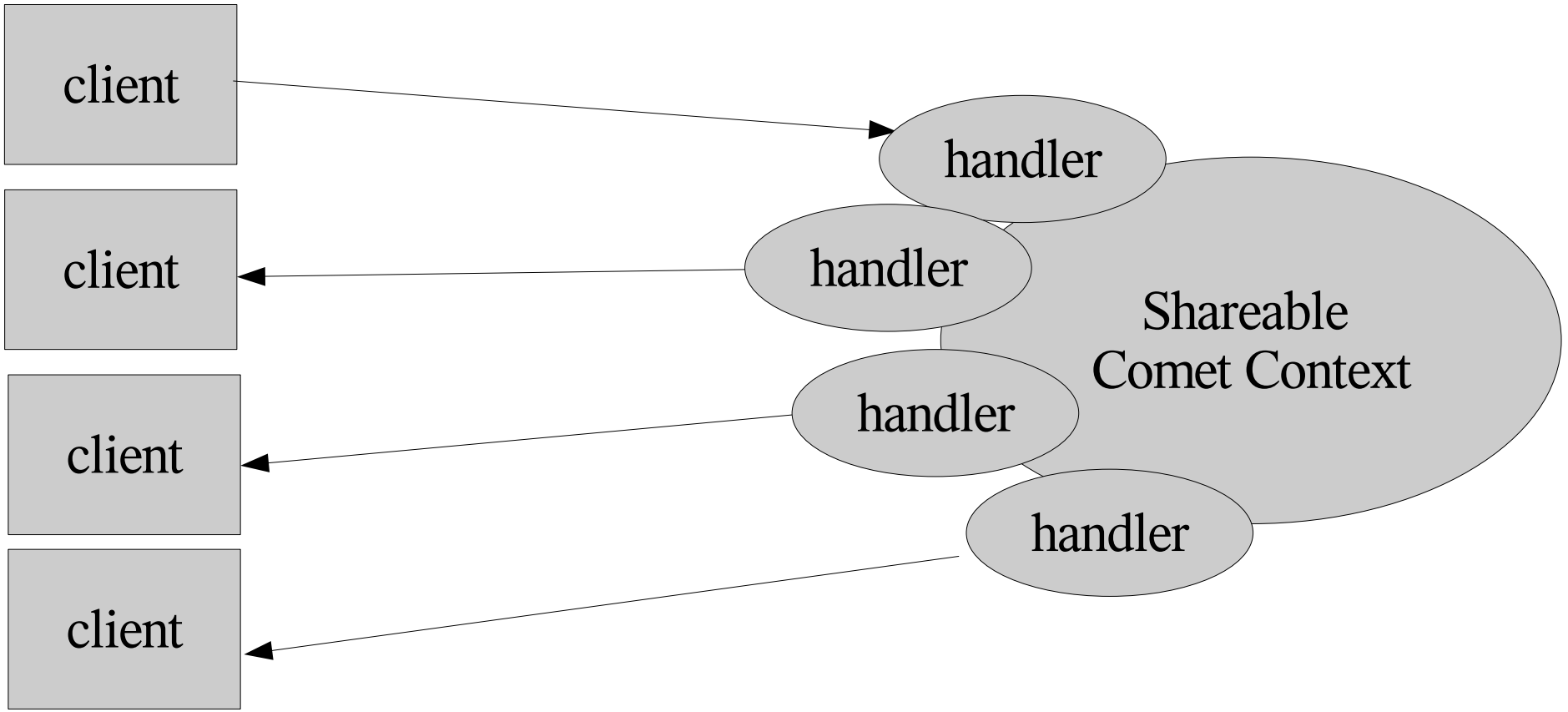
Next, clients open a **single connection** to a shareable context.

JGD



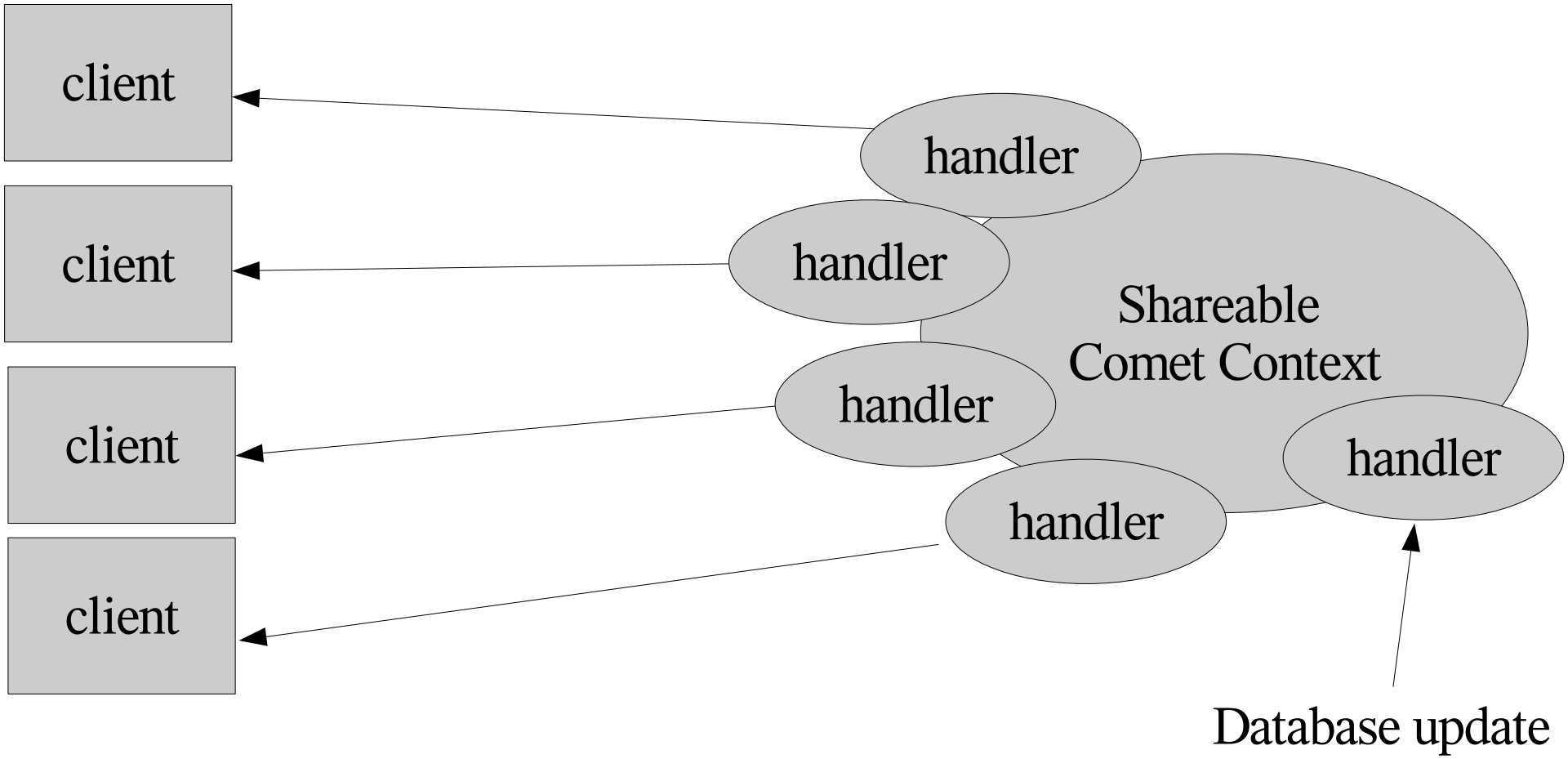
When one client push data, all other clients are updated.

JGD



When something change on the context clients are updated.

JGD



Implementing Comet: Four simple steps

- Adding Comet support to an existing application is simple (no big refactoring).
- The next four slides will describe how to do it.
- More details can be found in [2].

Step 1 - Registration

- First, register to a new or existing Comet context. The context are usually created using a context path (but not mandatory)

```
CometEngine cometEngine =
            CometEngine.getEngine();
CometContext context =
    cometEngine.register(contextPath);
```

Step 2- Define your Comet handler

JGD

- The Comet handler interface is simple:

```
public void attach(E attachment);
public void onEvent(CometEvent event)
public void onInitialize(CometEvent event)
public void onTerminate(CometEvent event)
public void onInterrupt(CometEvent event)
```

Step 3- Adding Comet handler to the context

- Next, instantiate your Comet request handler and add it to the context

```
MyCometHandler handler =
                    new MyCometHandler();
cometContext.addCometHandler(handler)
```

Step 4 – Advertise changes.

JGD

- Once the Comet handler has been added to the Context, you can start pushing events. You can notify the context when the client push data or when something external has changed (ex: a database):

```
cometContext.notify("Comet is cool");
```

Comet handler example.

- For a Servlet, you will most probably implement the CometHandler using the HttpServletResponse:

```
public class CometResponseHandler implements
        CometHandler{

    public void attach(HttpServletResponse
                                httpServletResponse){
        this.httpServletResponse =
                    httpServletResponse;
```

Comet handler example.

- And the onEvent method will most probably looks like:

```
public void onEvent(CometEvent event) throws IOException{

    try{
        PrintWriter printWriter = httpServletResponse.getWriter();
        printWriter.println(event.attachment());
        printWriter.flush();
    } catch (Throwable t){
       t.printStackTrace();
    }
```

What's next.

- Ship a common set of handlers applications can re-use:
  - One for Servlet/Jsp/Jsf
  - One for Phobos
  - One for POJO
- Ship an AJAX|JMaki re-usable client component.
- Add cometd support

  (consists of a protocol spec called Bayeux, javacript libraries (dojo toolkit), and an event server (Grizzly))

  An external team is working on it right now.

What's next.

- Publish some performance numbers.
- Improve current implementation in order to compete with Jetty and Tomcat Comet support.

References

- [1]http://weblogs.java.net/blog/jfarcand/archive/2006/02/grizzly_part_ii.html
- [2] http://weblogs.java.net/blog/jfarcand/archive/2006/07/the_grizzly_com.html

Help Improve This Presentation

If you want to contribute, keep attribution and maintain license. We would also appreciate notifying us of the changes.

More related presentations are kept at the Presentations page of the GlassFish Wiki.

http://www.glassfishwiki.org/gfwiki/Wiki.jsp?page=Presentations

Implementing Asynchronous Web Application using Grizzly's Comet.