Programmatic WebSockets

JavaOne 2014 – Steven Lemmo

ObjectRiver
METADATA COMPILERS

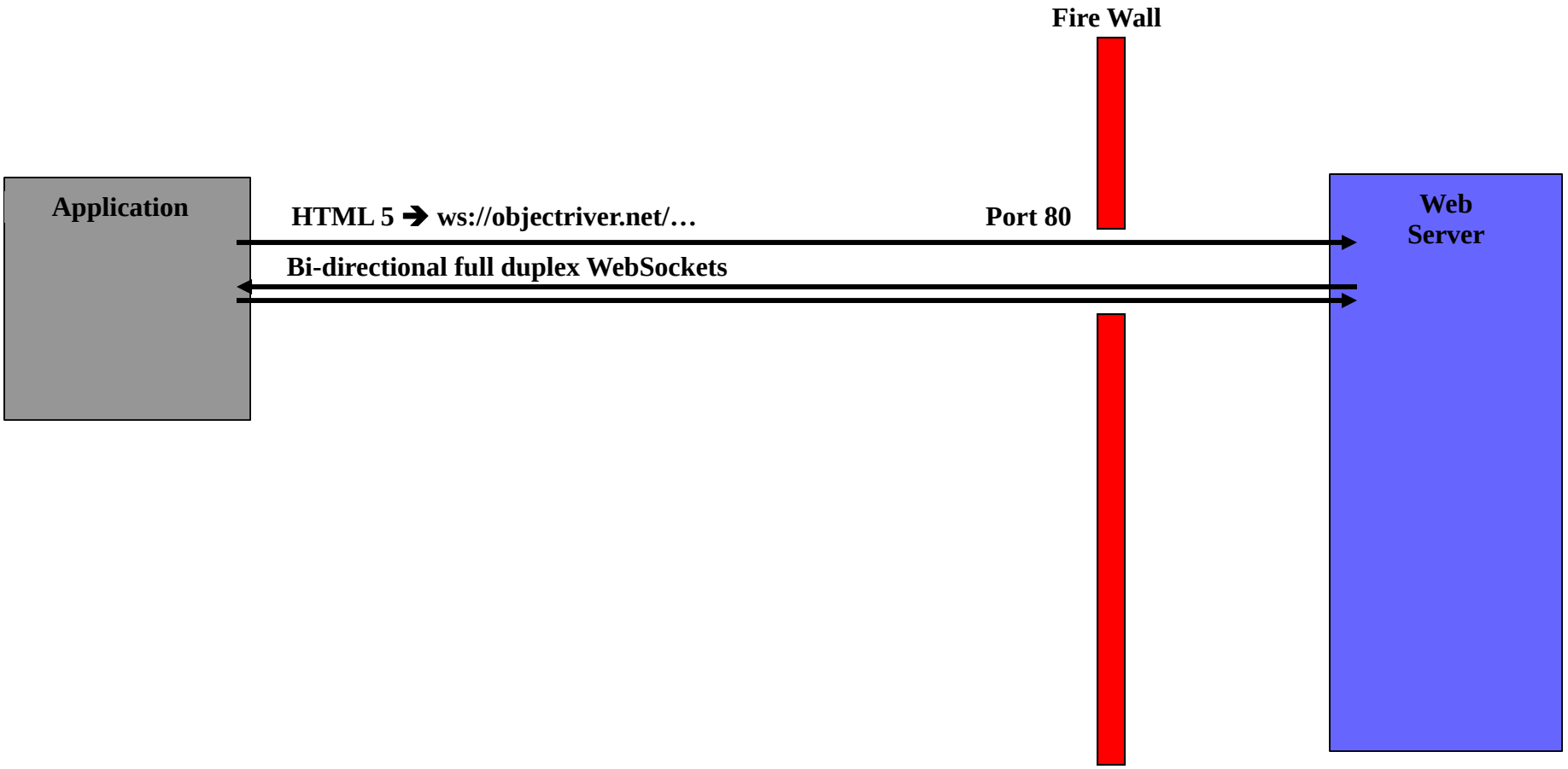
Sockets for the Web

Finally!

- Before the Web (Internal applications behind the firewall.
  - Sockets
  - RPC (Sun ONC/RPC)
  - DCE Open Software Foundation
  - CORBA
    - Binary protocols
    - State-full
    - Low latency
- Web Now
  - HTTP, XML, JSON
    - String based protocols
    - Stateless
- HTML5 WebSockets
  - State-full
  - Full duplex peer to peer.
  - Low latency
  - String and binary protocols.
  - All through port 80!

WebSockets into port 80

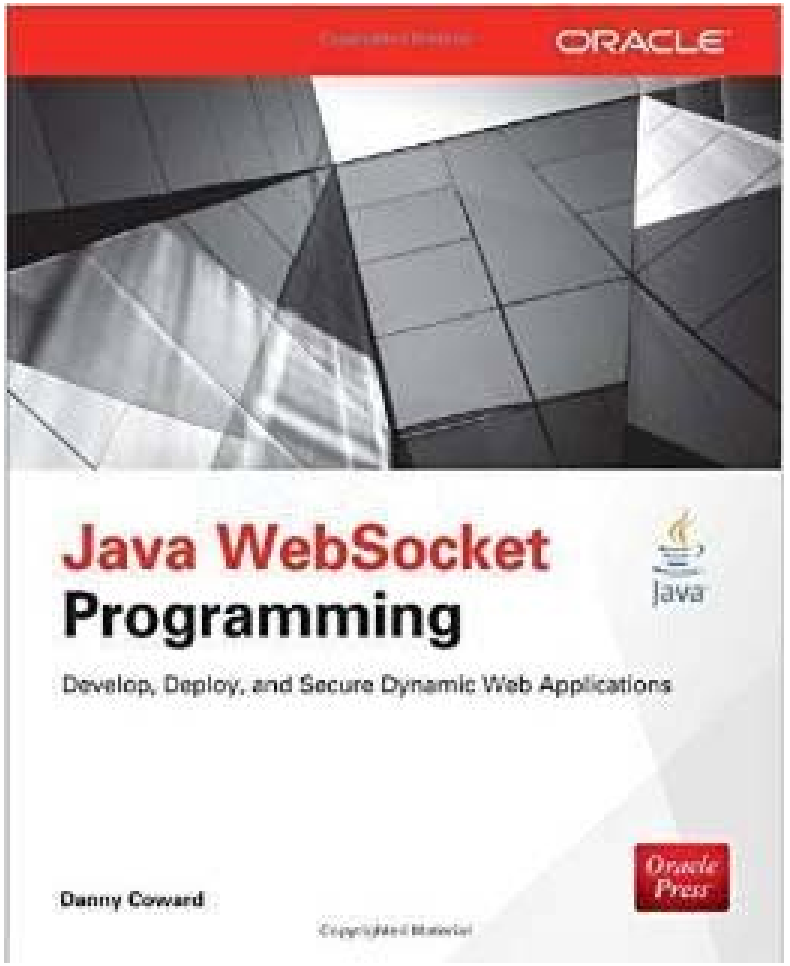
HTML 5 request gets promoted to WebSockets



ObjectRiver
METADATA COMPILERS

JSR356 WebSocket API

- Buy this book.
- This book describes the JSR356 Java WebSocket API specification.
- Today we are going to look at an example of a professional document style RPC mapping for WebSockets.



ObjectRiver
METADATA COMPILERS

Annotation

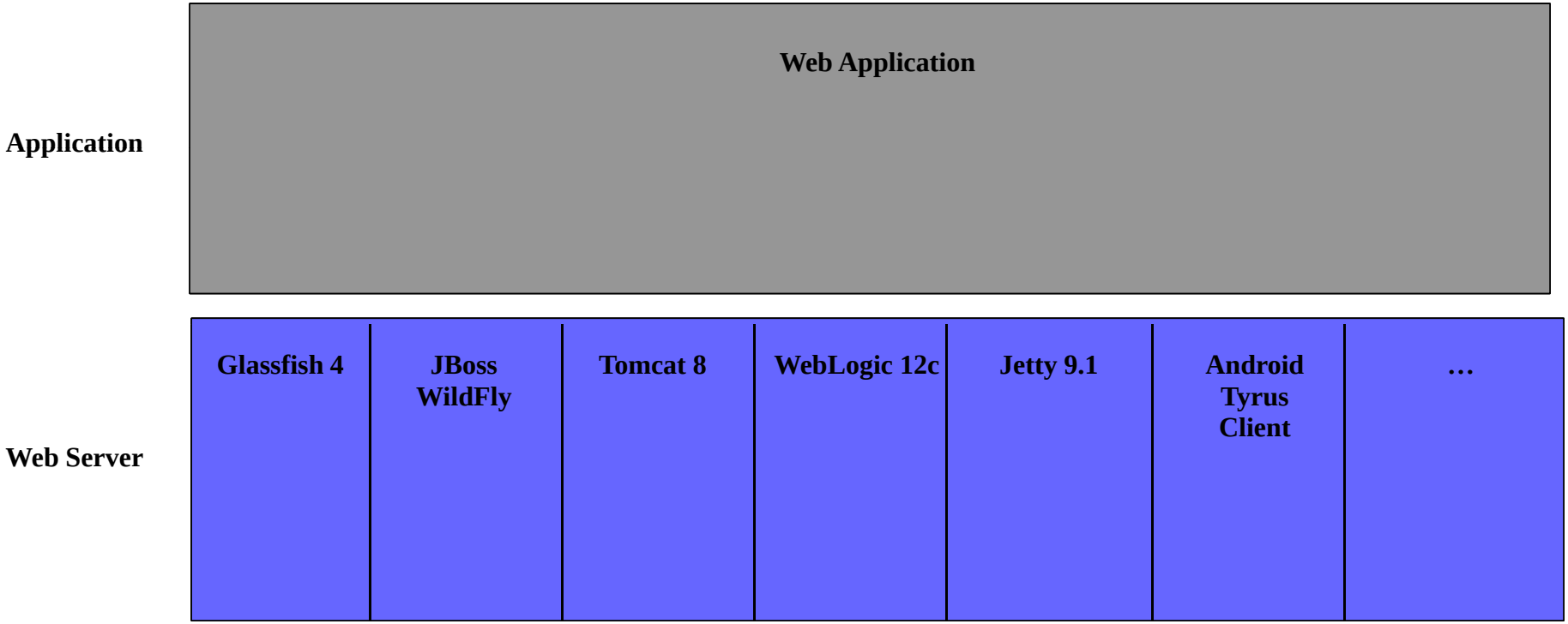
Programmatic vs. Annotation

- @ServerEndpoint(“/echo”) Public class EchoServer {
  @OnMessage public String echo(String msg) {

- Annotation is good for quick simple demos etc…
- Small programs like chat, logging, etc.
- But what is happening underneath?

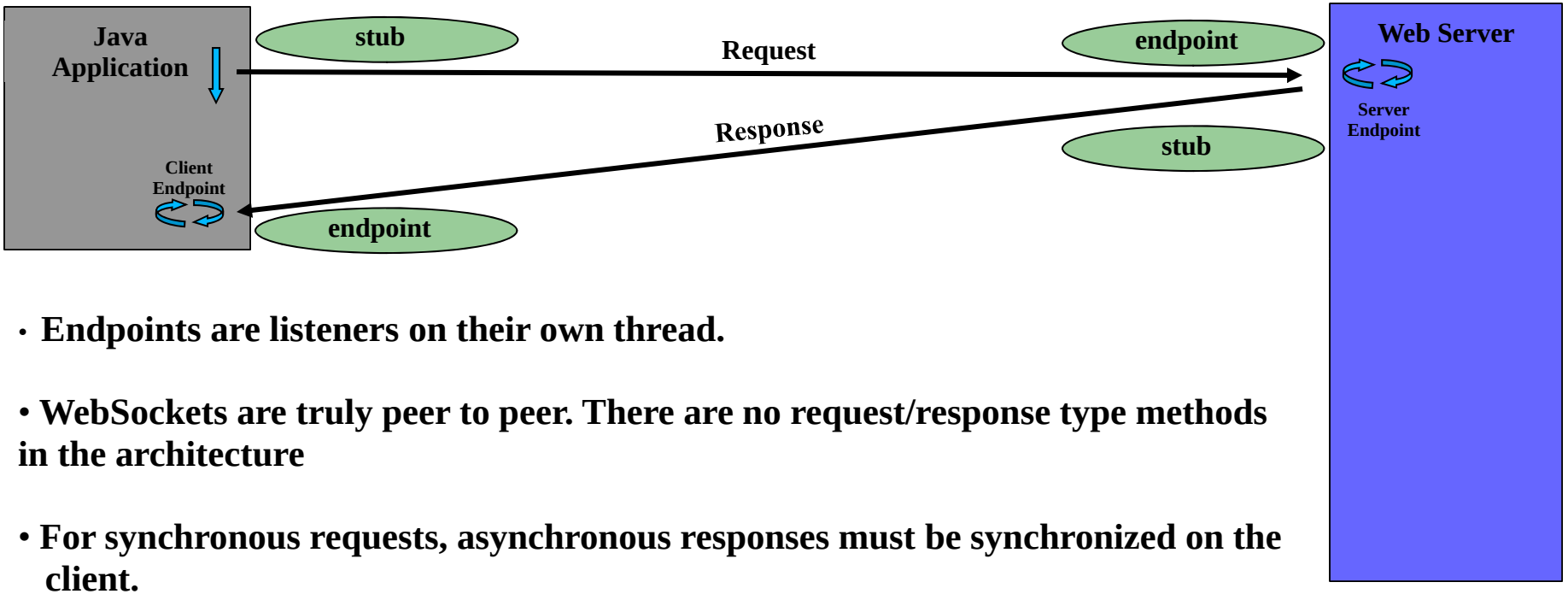
WebSocket Server Stacks

It's here now!



Endpoints

Endpoints are listeners on a thread



- **Endpoints are listeners on their own thread.**

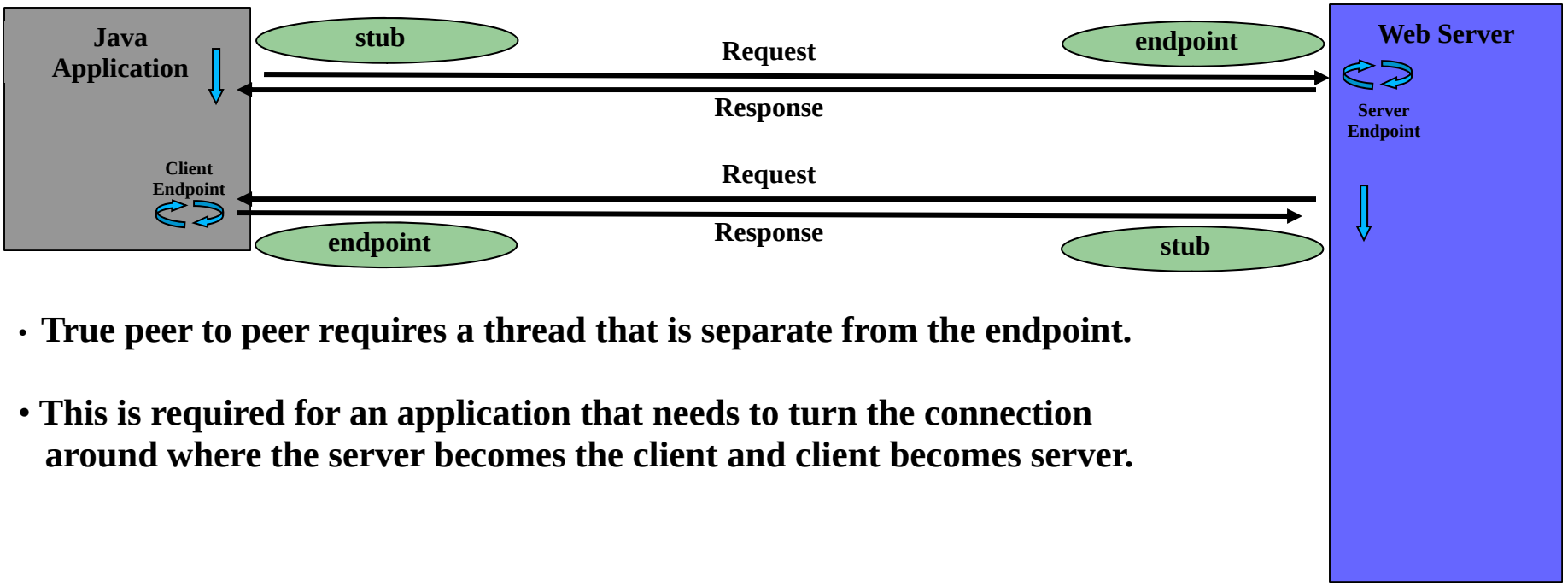
- **WebSockets are truly peer to peer. There are no request/response type methods in the architecture**

- **For synchronous requests, asynchronous responses must be synchronized on the client.**

- **Each client has it's own connection and thread on the server.**

Full duplex

Separate execution thread on server



- **True peer to peer requires a thread that is separate from the endpoint.**
- **This is required for an application that needs to turn the connection around where the server becomes the client and client becomes server.**

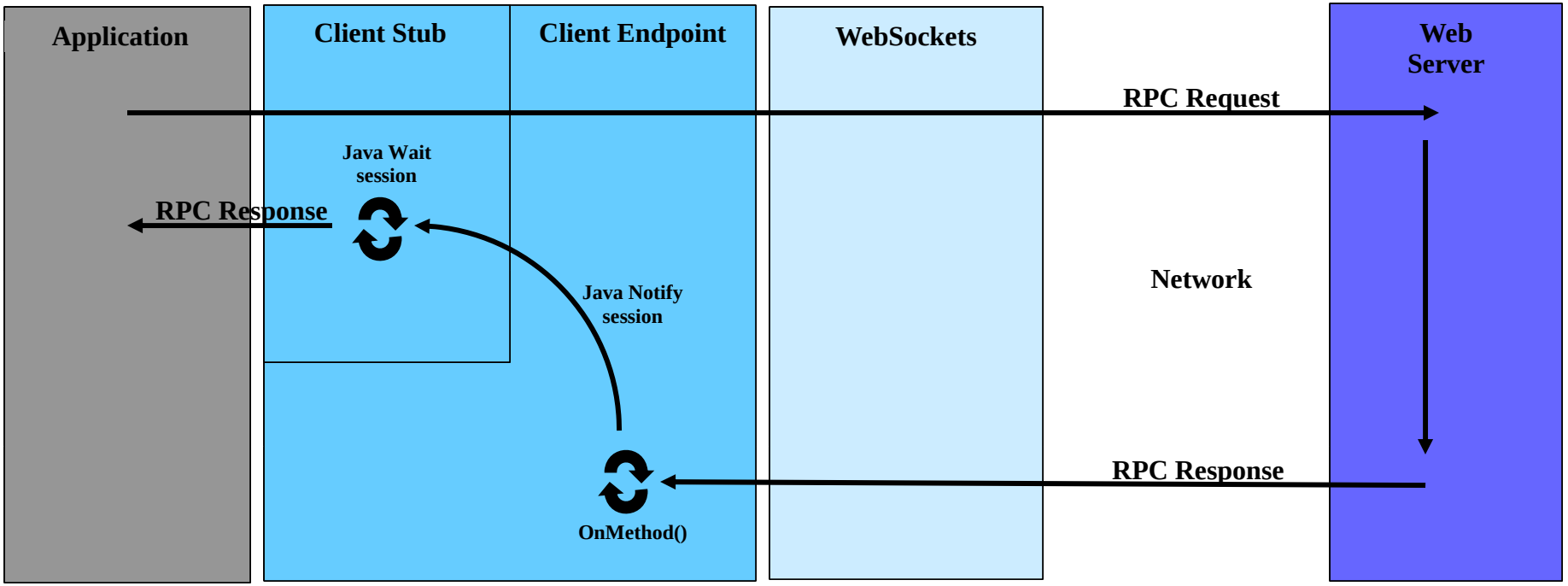
Hello World

Not your father's Hello World

- Hello World interface definition
  - @Sync String[] hello(@in String hello, @in Boolean throwSomething) throws HelloException;
- Breakdown of request/responses
  - void hello(String hello, Boolean throwSomething);
  - void helloResponse(String[] result);
  - void raiseHelloException(HelloException ex);
  - void raiseWebSocketException(WebSocketException ex);

hello_wait()

Java wait/notify synchronizes request



- Client sends request to server
- Client blocks on session with Java wait() with timeout from client handle
- Client endpoint in onMessage() wakes up client with Java notify synchronizing two asynchronous methods

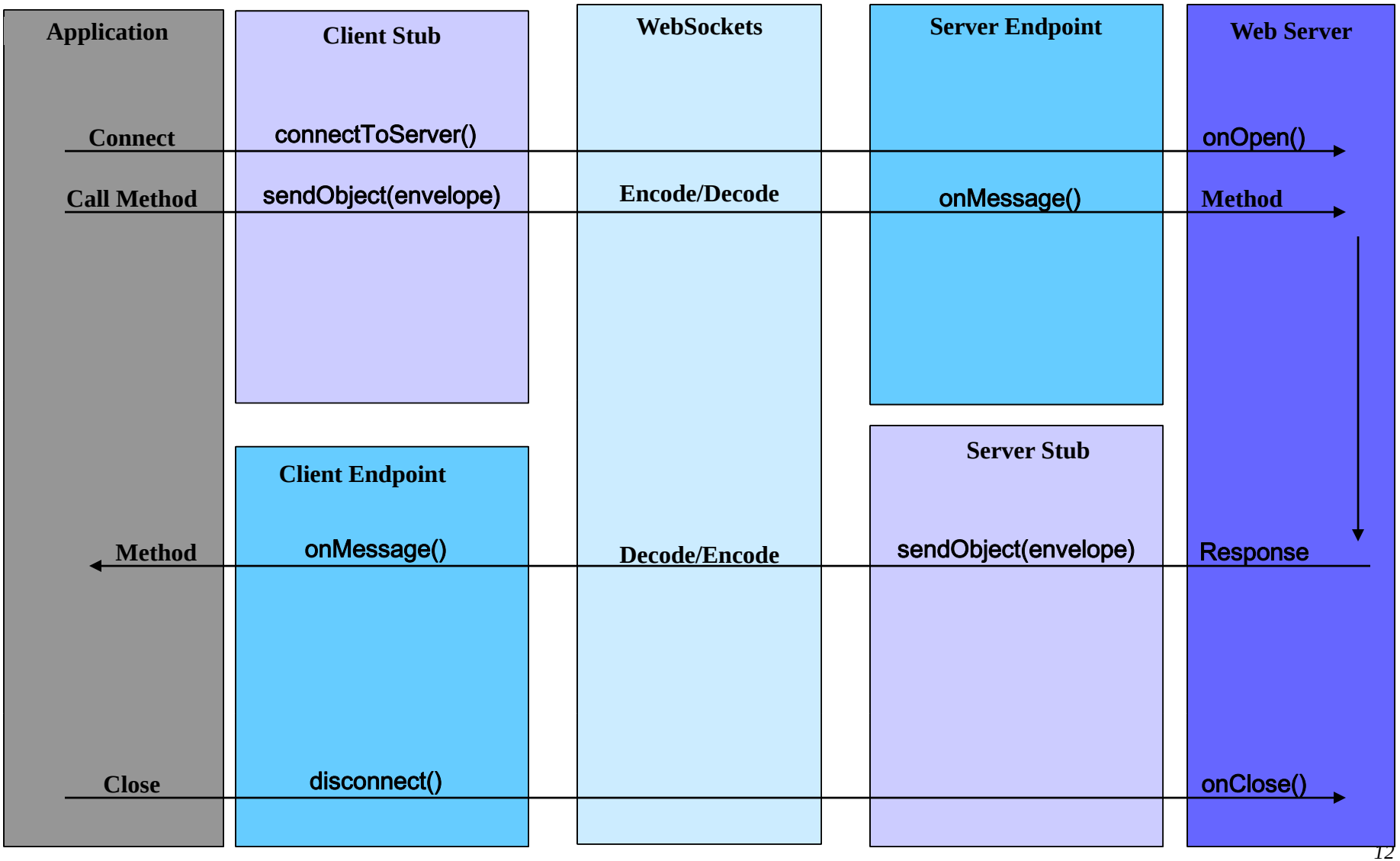
ObjectRiver
METADATA COMPILERS

Binary & Text Transport

- Sending Objects
  - **sendObject(envelope)**
    - Today we will be using the object abstraction for marshalling data between the client and server.
- Fixed length Clob/Blob support
  - **sendText()**
  - **sendBinary()**
- Give me the pipe
  - **getSendStream()** – Java InputStream
  - **getSendWriter()** – Java Writer

Basic Code layout

Core WebSocket methods



ObjectRiver
METADATA COMPILERS

URL Path

ws://localhost/HelloApp/Test/Hello

- ws://<host>:<port>/<web-apps>/<context-root>/<uri-path>
  - web-apps
    - In Tomcat this is the name of the war.
    - Glassfish does not have a web-apps
  - context-root
    - This is the path leading to the application specific
  - uri-path
    - Application specific portion. Can contain templates, and rest style urls.
- Example:
  - ws://example.org/employee/tools/chat
    - web-apps=employee
    - root-context=tools
    - uri-path=chat
  - ws://bank.com/stock/price?ticker=CRM
    - web-apps=stock
    - root-context=price
    - uri-path=?ticker=CRM
  - ws://localhost:8080/HelloApp/Test/Hello

Hello Implementation

Client-side Hello World

```
public class HelloMain {
    public static void main(String[] args) {
        HelloClient client = null;
        try {
          client = HelloClient.connect("ws://localhost:8101/HelloApp/Test/Hello“);
          String[] worlds = client.hello("Hello",false);
          for(String world: worlds) {
               System.out.println(world);
          }
          client.hello("Hello",true);
       }
       catch(HelloException ex) {
                System.out.println(ex.getMessage());
        }
        catch(DeploymentException ex) { System.err.println(ex.getMessage()); }
        catch(WebSocketException ex) { ex.printCauseTrace(); }
         finally {
             if(client!=null) client.disconnect();
        }
    }
}
```

Hello Implementation

Server-side Hello World

```
public class HelloEndpointServerSessionImpl_V1 implements HelloEndpointServerInterface  {
    protected Session session__ = null;
    public void onOpen(Session session, EndpointConfig endpointConfig) {
        this.session__ = session;
    }
    public void onClose(Session session, CloseReason closeReason) {
    }
    public void onError(Session session, java.lang.Throwable thr) {
        System.out.println(thr.getMessage());
        thr.printStackTrace();
    }
    /**
     * Implementation
     */
    public String[] hello( String hello, Boolean throwSomething) throws HelloException {
        System.out.println(hello);
        if(throwSomething.booleanValue())
                throw new HelloException("World");
        return new String[]{"World","World","World","World"};
    }
}
```

sendObject(Envelope)

Envelope contains a single document

```
public class HelloEnvelope_V1 extends HelloEnvelope {
        public HelloMethods method; // enumeration

        /** Documents */
        protected HelloDocument_V1 helloDocument_V1;
        protected HelloResponseDocument_V1 helloResponseDocument_V1;
        protected HelloException helloException;

       /** getters / setters */
       public HelloException getHelloException()
       public void setHelloException(HelloException helloException)
       public HelloDocument_V1 getHelloDocument_V1()
       public void setHelloDocument_V1(HelloDocument_V1 helloDocument_V1)
       public HelloResponseDocument_V1 getHelloResponseDocument_V1()
       public void setHelloResponseDocument_V1(HelloResponseDocument_V1
}
```

ObjectRiver
METADATA COMPILERS

Document

Each method has documents

```
public class HelloDocument_V1 {
    public HelloDocument_V1(String hello, Boolean throwSomething) { … }
    protected String hello;
    protected Boolean throwSomething;
    /** Getters /Setters */
}
public class HelloResponseDocument_V1 {
    public HelloResponseDocument_V1(String[] result) { … }
    protected String[] result;
    /** Getters /Setters */
}
public class HelloException extends WebSocketException {
    public HelloException(String message,Throwable cause) {
        super(message,cause);
     }
}
```

Connect & Client Config

ClientEndpointConfig & connectToServer

```
public static <T extends HelloClient> T connect(T client, String path)
        throws IOException, DeploymentException, URISyntaxException {
    WebSocketContainer wsc = ContainerProvider.getWebSocketContainer();
    ClientEndpointConfig clientEndpointConfig = ClientEndpointConfig.Builder
            .create()
            .configurator(new HelloClientConfigurator())
            .encoders(Arrays.<Class<? extends Encoder>>asList(
               HelloBinaryStreamEnvelopeEncodeDecode.class,
               HelloTextStreamEnvelopeEncodeDecode.class
               ))
            .decoders(Arrays.<Class<? extends Decoder>>asList(
               HelloBinaryStreamEnvelopeEncodeDecode.class,
               HelloTextStreamEnvelopeEncodeDecode.class
               ))
            .build();
    client.session = wsc.connectToServer(client, clientEndpointConfig, new URI(path));
    return client;
}
```

ObjectRiver METADATA COMPILERS

Server-side Config

ServerEndpointConfig

```
public class HelloServerApplicationConfig implements ServerApplicationConfig {
    public Set<ServerEndpointConfig> getEndpointConfigs(Set<Class<? extends Endpoint>> set) {
        return new HashSet<ServerEndpointConfig>() {
            {
                add(ServerEndpointConfig.Builder
                        .create(HelloServerEndpoint.class, EndpointPathName)
                        .configurator(new HelloServerConfigurator())
                        .encoders(Arrays.<Class<? extends Encoder>>asList(
                            HelloBinaryStreamEnvelopeEncodeDecode.class ,
                            HelloTextStreamEnvelopeEncodeDecode.class
                            ))
                        .decoders(Arrays.<Class<? extends Decoder>>asList(
                            HelloBinaryStreamEnvelopeEncodeDecode.class ,
                            HelloTextStreamEnvelopeEncodeDecode.class
                            ))
                        .build());
            }
        };
    }
    public Set<Class<?>> getAnnotatedEndpointClasses(Set<Class<?>> set) {
        return Collections.emptySet();
    }
}
```

Client Stub

Stub packages document in envelopes

```
public String[] hello( String hello, Boolean throwSomething ) throws HelloException {
    HelloDocument_V1 inDoc = new HelloDocument_V1(hello,throwSomething);
    try {
        HelloEnvelope_V1 envelope = new HelloEnvelope_V1();
        envelope.setMethod(HelloMethods.hello_1_instance);
        envelope.setHelloDocument_V1(inDoc);
        session.getBasicRemote().sendObject(envelope);
    }
    catch (EncodeException ex) {
        throw new WebSocketException("Encode Exception encoding hello", ex);
    }
    catch (IOException ex) {
        throw new WebSocketException("IOException Encoding Hello , ex);
    }
    HelloResponseDocument_V1 outDoc = hello_wait();
    String[] result = outDoc.getResult();
    return result;
}
```

Server Endpoint OnMessage

Endpoint listeners for envelope

```
public class HelloMessageHandler implements MessageHandler.Whole<HelloEnvelope> {
    protected HelloEndpointServerInterface_V1 impl = null;
    public HelloMessageHandler(HelloEndpointServerInterface_V1 impl) {
        this.impl = impl;
    }
    public void onMessage(HelloEnvelope envelope) {
        try {
            dispatch((HelloEnvelope_V1)envelope);
        }
        catch (Exception ex) {
            try {
                envelope.setMethod(HelloMethods.raiseWebSocketException_1_instance);
                envelope.setWebSocketException( new WebSocketException("onMessage() ", ex));
                session.getBasicRemote().sendObject(envelope);
            }
            catch(IOException ignore) {}
            catch(EncodeException ignore) {}
            return;
        }
    }
```

Server Endpoint Dispatch

Dispatch calls remote method

```
private void dispatch(HelloEnvelope_V1 envelope) {
    switch(envelope.getMethod().getEnum()) {
    case hello_1: {
        HelloDocument_V1 request = envelope.getHelloDocument_V1();
        String hello = request.getHello();
        Boolean throwSomething = request.getThrowSomething();
        try {
            String[] result = impl.hello( hello, throwSomething );
            HelloResponseDocument helloResponse = new HelloResponseDocument();
            helloResponse.setResult(result);
            envelope.setMethod(HelloMethods.helloResponse_1_instance);
            envelope.setHelloResponseDocument_V1(helloResponse);
        }
        catch(HelloException ex) {
            envelope.setMethod(HelloMethods.raiseHelloException_1_instance);
            envelope.setHelloException(ex);
        }
     }
    session.getBasicRemote().sendObject(envelope);     }
```

Encode/Decode handlers

sendObject(envelope)

- sendObject(envelope) takes envelope object.
- Encode/Decode routines are added to session's configuration, on client and server
- Envelope object is a choice object containing documents for all request and response.
- Registered handler -> MessageHandler.Whole<HelloEnvelope>

Binary XDR Encode/Decode

HelloDocument Binary Encode/Decode

```
public class XdrHelloDocument_V1 extends XdrMarshal {
        public boolean execute(XDR xdrs, XdrThing thing1)  throws WebRpcMarshallingException {
                HelloDocument_V1 request_response = (HelloDocument_V1) thing1.getThing();
                XdrThing thing2 = new XdrThing();
                if(xdrs.x_op== XDR.op.XDR_DECODE) {
                        request_response = new HelloDocument_V1();
                        thing1.setThing(request_response);
                 }
                 try {
                        thing2.setThing(request_response.getHello());
                        if(!xdr_string.instance.execute(xdrs, thing2)) { return false; }
                        request_response.setHello((String)thing2.getThing());

                        thing2.setThing(request_response.getThrowSomething());
                        if(!xdr_pointer(xdrs, thing2, xdr_bool.instance)) { return false; }
                        request_response.setThrowSomething((Boolean)thing2.getThing());
                }
        }
```

Text JSON Encode/Decode

HelloDocument Text Encode/Decode

```
public class JsonHelloDocument_V1 extends JsonMarshal {
        public boolean execute(JSON jsons, JsonThing thing1)  throws WebRpcMarshallingException {
            HelloDocument_V1 request_response = (HelloDocument_V1) thing1.getThing();
            JsonThing thing2 = new JsonThing();
               if(jsons.x_op== JSON.op.JSON_DECODE) {
                  request_response = new HelloDocument_V1();
                  thing1.setThing(request_response);
               }
               try {
                  thing2.setJvalue("hello",request_response.getHello());
                  if(!json_string.instance.execute(jsons, thing2)) { return false; }
                  request_response.setHello((String)thing2.getThing());
                  jsons.separator();

                  thing2.setJvalue("throwSomething",request_response.getThrowSomething());
                  if(!json_pointer(jsons, thing2, json_bool.instance)) { return false; }
                  request_response.setThrowSomething((Boolean)thing2.getThing());
               }
               catch (JsonParsingException ex) {
                  throw new WebRpcMarshallingException("JsonHelloDocument_V1",ex);
               }
```

Session vs Singleton Server

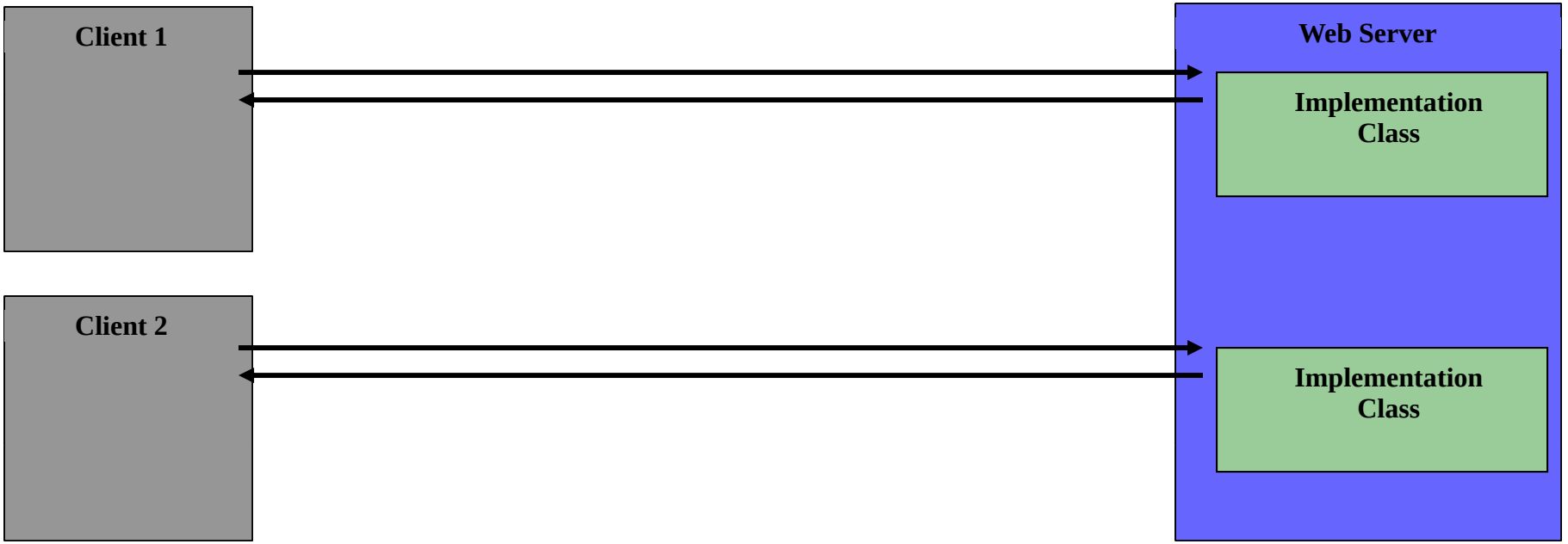
Server Architecture

- Session Server
  - This is where each client has its own state-full copy of the server's implementing class.
  - The state lasts for the duration of the WebSockets session with the client
  - It's exactly like each client having it's own copy of the server.

- Singleton Server
  - This is where there is a single instance of the implementing class, and the server's state is shared.
  - Therefore multiple client threads of execution are executing on the single singleton class.
  - Implementing class must be thread safe.
  - Individual client state can be stored in the session's user properties.

Session Server

- Session Server
  - This is where each client has it's own state-full copy of the server's implementing class.
  - The state lasts for the duration of the WebSockets session with the client
  - It's exactly like each client having it's own copy of the server.



ObjectRiver
METADATA COMPILERS

Session Server

WebSocket Endpoints are session objects

```
public class HelloServerEndpoint extends Endpoint {
    protected HelloEndpointServerInterface_V1 impl = null;
    …
    public void onOpen(final Session session, EndpointConfig config) {
        impl = new HelloEndpointServerSessionImpl();
        …
```

Singleton Server

- Singleton Server
  - This is where there is a single instance of the implementing class, and the server's state is shared.
  - Therefore multiple client threads of execution are executing on the single singleton class.
  - Implementing class must be thread safe.
  - Individual client state can be stored in the session's user properties.

Singleton Server

Singleton design pattern

```
public class HelloServerEndpoint extends Endpoint {
        protected HelloEndpointServerInterface_V1 impl = null;
        …
        public void onOpen(final Session session, EndpointConfig config) {
             HelloEndpointServerInterface_V1 impl =
                HelloEndpointServerInterface_V1.getInstance();
             …
        }    }

public class HelloEndpointServerInterface_V1 implements ServerEndPointInterface {
        private static HelloEndpointServerInterface_V1 ourInstance;
        public synchronized static HelloEndpointServerInterface_V1 getInstance() {
              if (ourInstance == null) {
                  ourInstance = new HelloEndpointServerInterface_V1 ();
              }
              return ourInstance;
        }
```

ObjectRiver
METADATA COMPILERS

Summary

- Standard Java sockets interface for the web.
- Full duplex
- Low latency
- Complex data
- Example from presentation at
  - www.objectriver.net/cloudcompiler/hello.zip
- Checkout our free WebSocket Compiler Toolkit at
  - www.objectriver.net/products.htm

Steven Lemmo
CTO & Founder
ObjectRiver Inc
steve@objectriver.net
www.objectriver.net