#ibminterconnect

Introduction to WebSockets

Session 1641

Erin Schnabel
Liberty Runtime Development Lead, IBM
@ebullientworks

IBM

InterConnect 2015
The Premier Cloud & Mobile Conference

February 22 – 26
MGM Grand & Mandalay Bay | Las Vegas, Nevada

About WebSockets…

- WebSocket network protocol
  - Data format
- WebSocket API (Javascript & Java EE)
  - Endpoint configuration
  - Session open/close
  - Message read/write
  - Error handling
  - Annotations
- Network architecture
  - Proxies / load balancers / routers…

#ibminterconnect

<tripDownMemoryLane>
Before WebSockets...

IBM

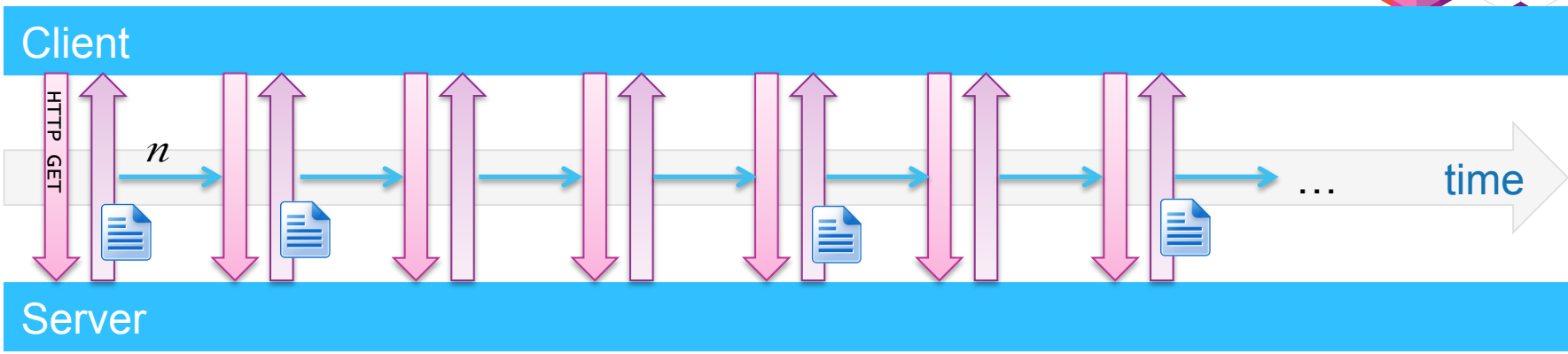
InterConnect 2015

The Premier Cloud & Mobile Conference

Options for two-way communication

- Polling
- Long polling
- Streaming / forever response
- Multiple connections

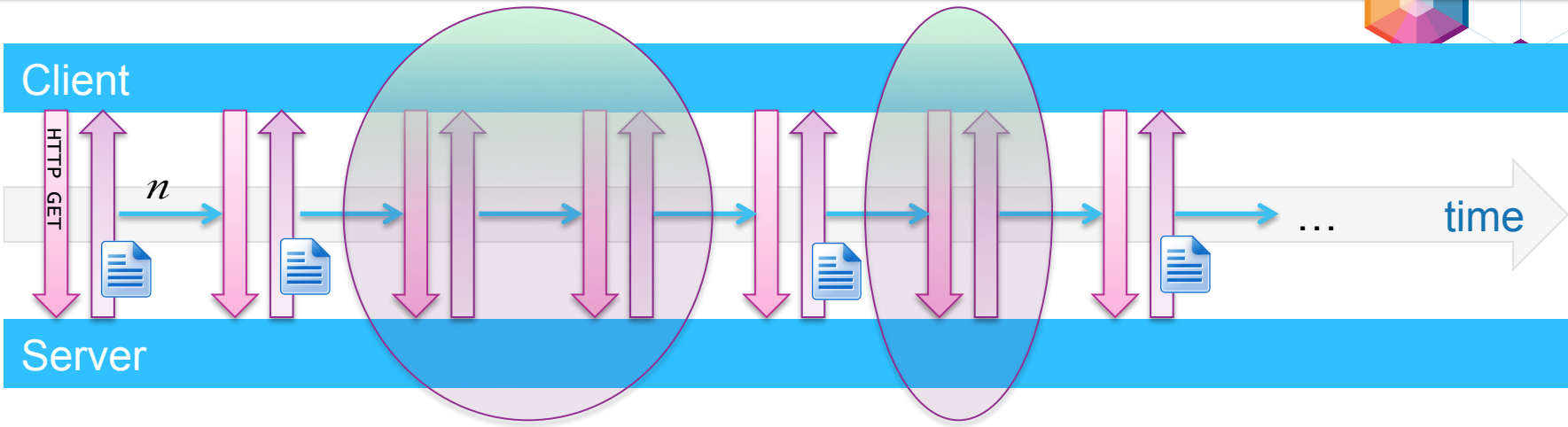
(1): Polling



- Client polls the server every n …
- Server always immediately responds (with or without data)
- Might work for periodic data where the period is known/constant

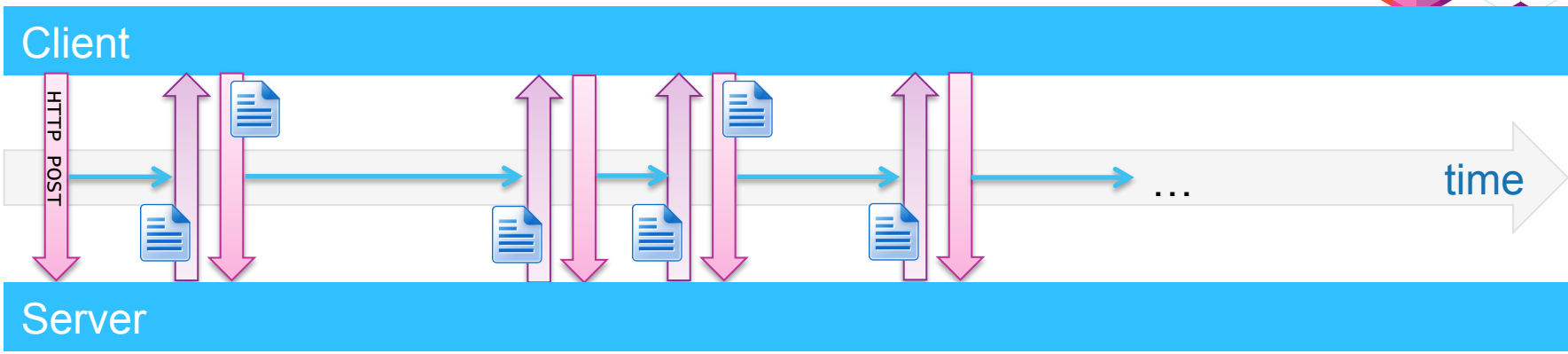
BUT…

(1): Polling



- Client polls the server every n …
- Server always immediately responds (with or without data)
- Might work for periodic data where the period is known/constant
- *Obvious waste (CPU and bandwidth) when there is no data*

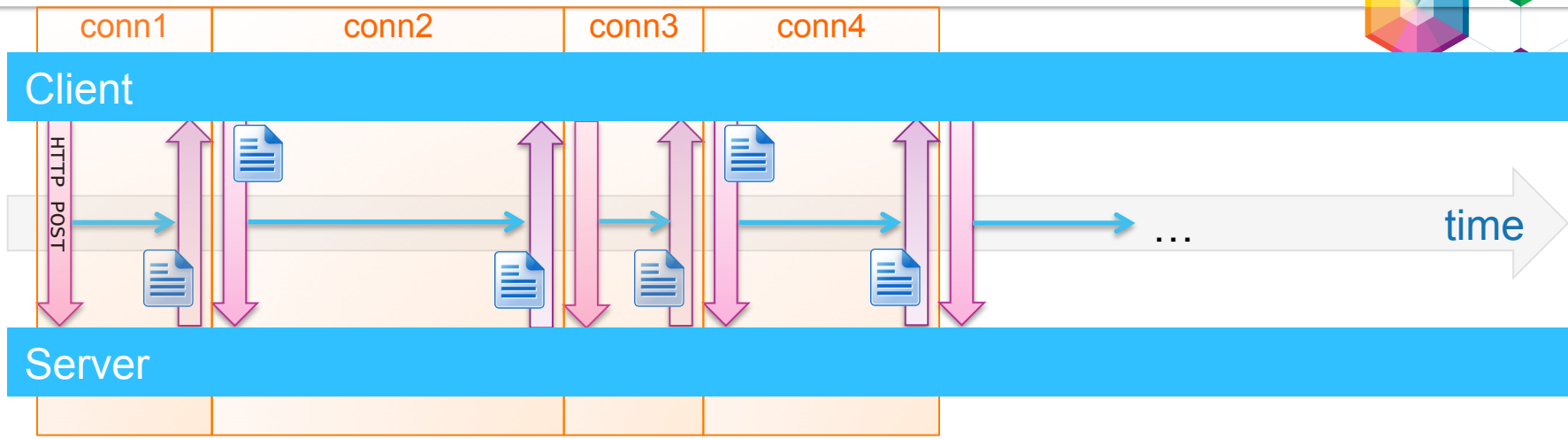
(2): Long Polling



- Client sends initial request
- Server waits until it has data to respond
- Client receives response, and immediately creates new request
- Obvious improvement over plain polling

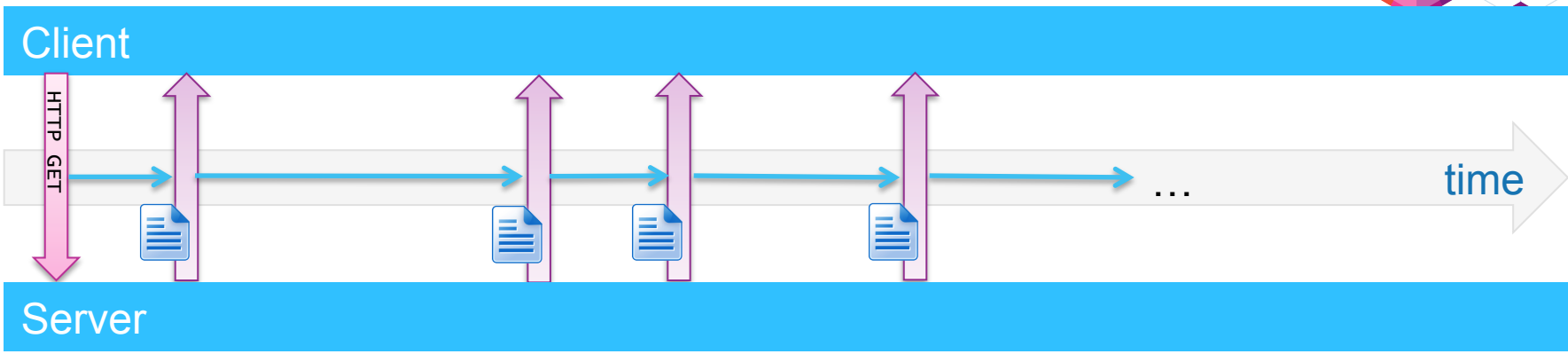
BUT…

(2): Long Polling



- Client sends initial request
- Server waits until it has data to respond
- Client receives response, and immediately creates new request
- Obvious improvement over plain polling
- *Each request/response creates and closes a connection*
- *Client has to wait to send new data until the server responds*

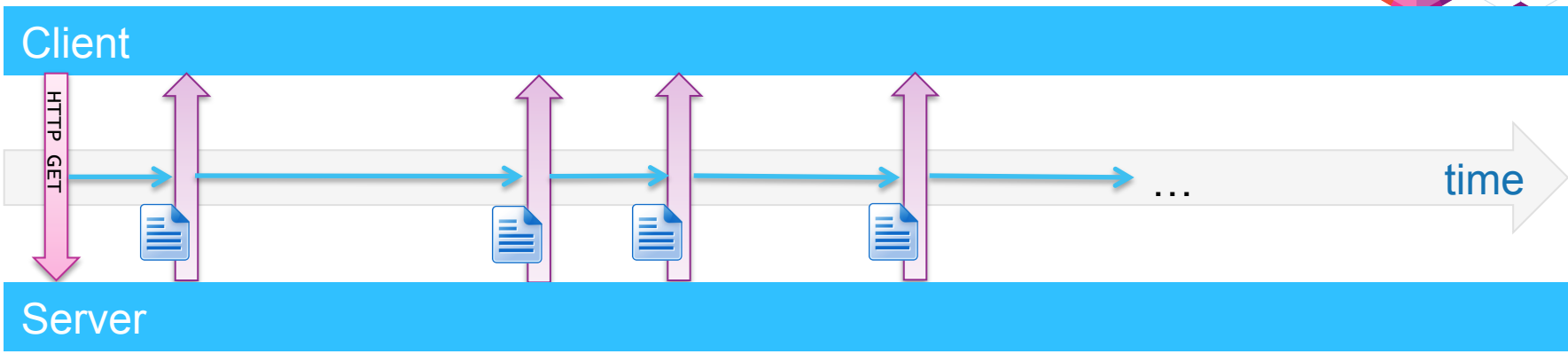
(3): Streaming / forever response



- Client sends initial request
- Server waits until it has data to respond
- Server responds by streaming data
  - Server has an open connection to *push* updates
- Connection is maintained

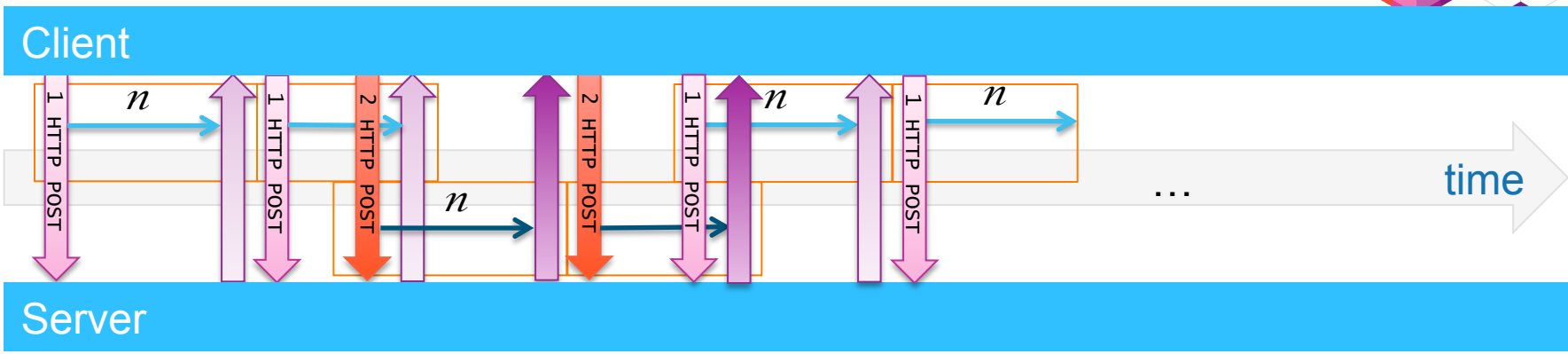
BUT…

(3): Streaming / forever response



- Client sends initial request
- Server waits until it has data to respond
- Server responds by streaming data
  - Server has an open connection to *push* updates
- Connection is maintained
- *It is half-duplex: only server to client*
- *User agents and proxies might not like partial responses*

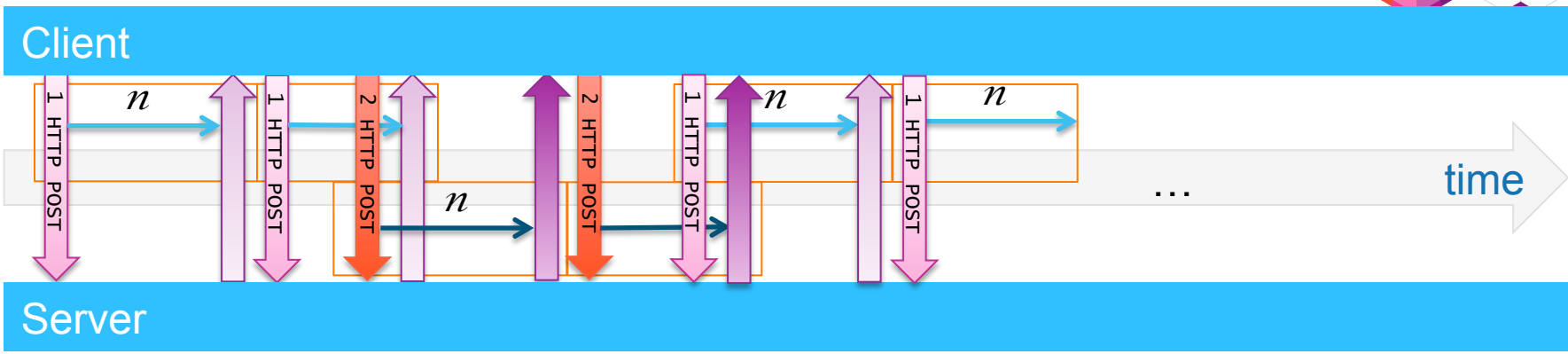
(4): Multiple connections



- Long polling over two separate HTTP connections
  - Approximation of bi-directional connection
  - Two connections are used (HTTP recommended max)
    - long polling
    - second connection allows client to send data to the server

BUT...

(4): Multiple connections



- Long polling over two separate HTTP connections
  - Approximation of bi-directional connection
  - Two connections are used (HTTP recommended max)
    - long polling
    - second connection allows client to send data to the server
- *Non-trivial connection coordination and management*
- *Two connections for every client*

Hidden cost of HTTP…

- TCP handshake when establishing new connection
  - Even worse for SSL...
- HTTP headers on every message
  - Always present, can vary in size and quantity

http://www.websocket.org/quantum.html

```
GET /PollingStock//PollingStock HTTP/1.1

Host: localhost:8080

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.1.5)
Gecko/20091102 Firefox/3.5.5

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-Language: en-us

Accept-Encoding: gzip,deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7

Keep-Alive: 300

Connection: keep-alive

Referer: http://www.example.com/PollingStock/
```

For small messages, you may end up pushing around more HTTP headers than data!

#ibminterconnect

</tripDownMemoryLane>

IBM

InterConnect 2015

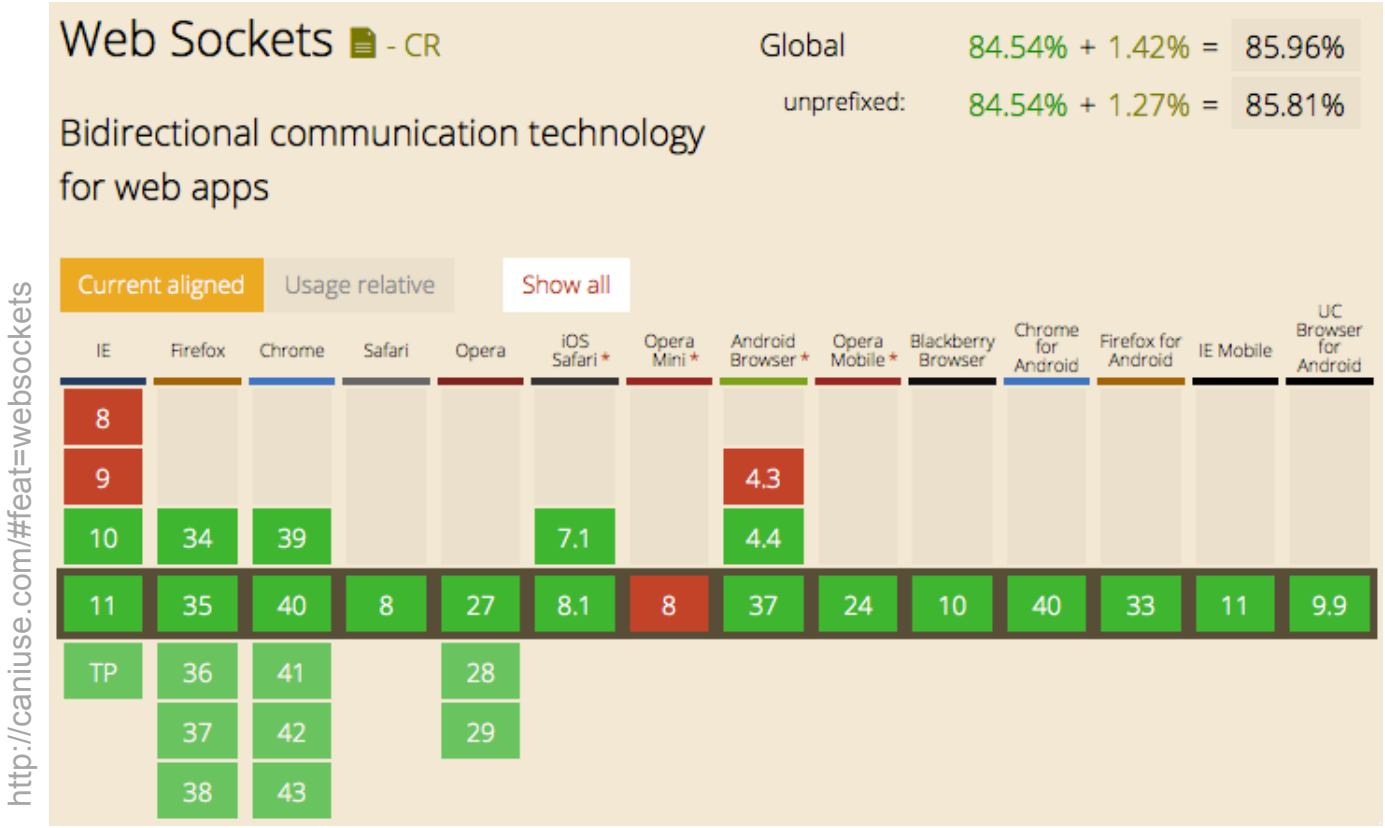
The Premier Cloud & Mobile Conference

There is a better way: WebSockets

- Bi-directional
  - Client and server can send messages at any time
- Full duplex
  - Client and server can send updates at the same time
  - No requirement for request/response pair or message ordering
- Single long running connection with established context
  - No connection management/coordination
- Connection upgraded from HTTP
  - No new connection protocol to build infrastructure for
- Efficient use of bandwidth and CPU
  - **Messages can focus on application data**

WebSockets have been standardized

- IETF RFC-6455: WebSocket Protocol Specification, 2011
- JSR 356: WebSocket API Specification, 2013
    - Part of Java EE 7



But…

- What about plain HTTP requests?
  - HTTP is still great for loading static resources!
- What about REST?
  - REST is still great for client-initiated CRUD operations
- What about MQTT?
  - Pub/Sub model, different QOS, different protocol
  - *Runs atop WebSockets!* (Liberty/Rtcomm, Paho.js, …)

#ibminterconnect

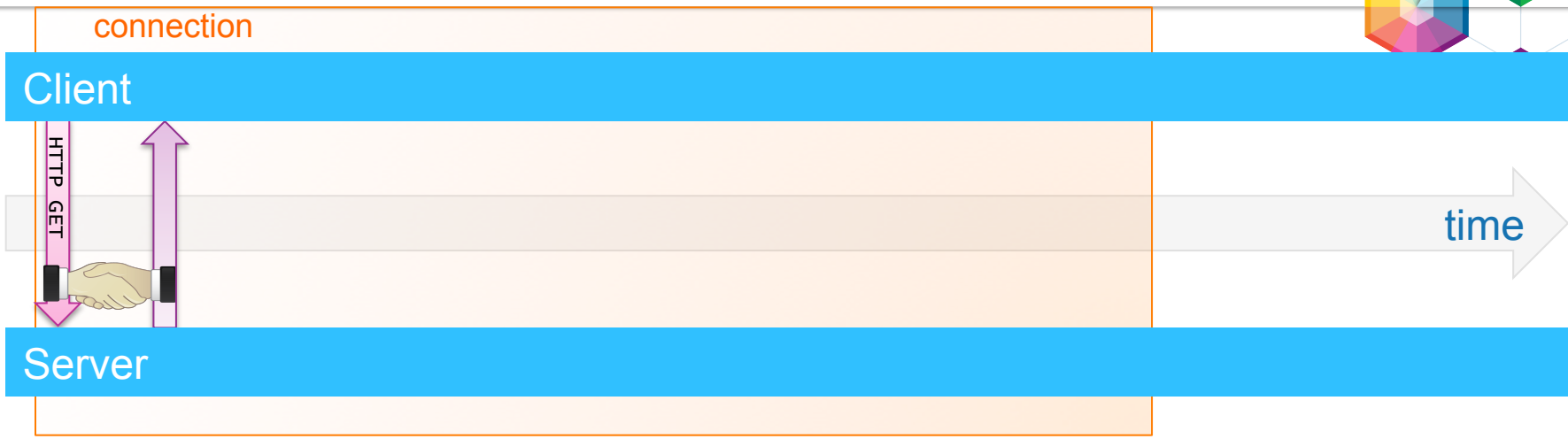
<background> How WebSockets work..

IBM

InterConnect 2015

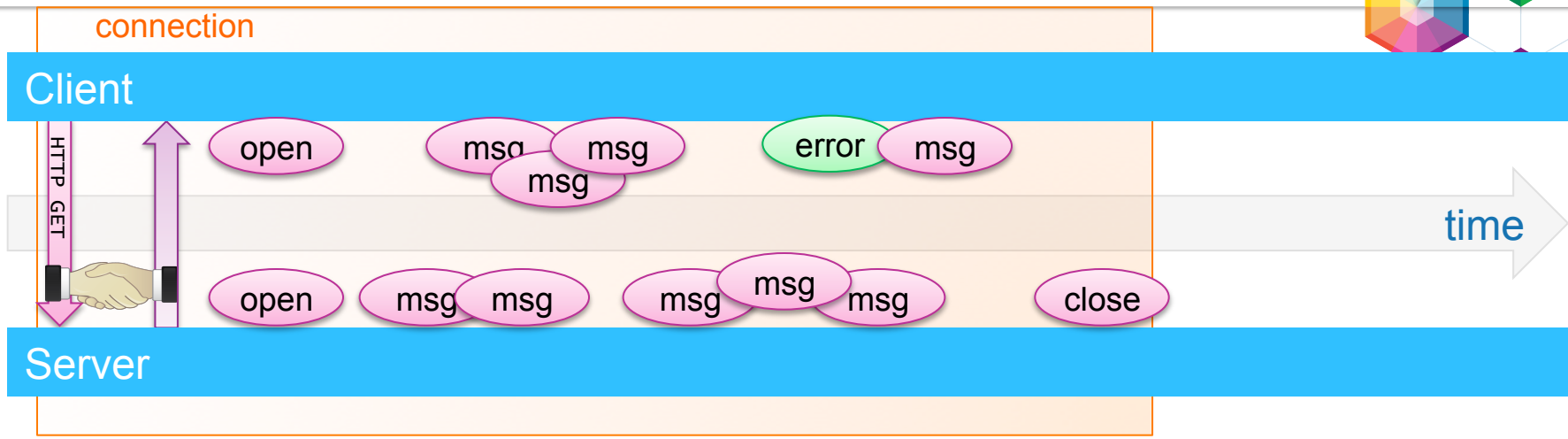
The Premier Cloud & Mobile Conference

WebSocket connection



- Handshake:
  - Client initiates connection
  - Server responds (accepts the upgrade)

WebSocket connection



- Handshake:
  - Client initiates connection
  - Server responds (accepts the upgrade)
- Once the WebSocket is established
  - both sides notified that socket is open
  - either side can send messages at any time
  - either side can close the socket

WebSocket Protocol: it starts with a handshake...

request

```
GET /myapp HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: GhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Version: 13
Sec-WebSocket-Protocol: custom
Sec-WebSocket-Extensions: compress
Origin: http://example.com
...
```

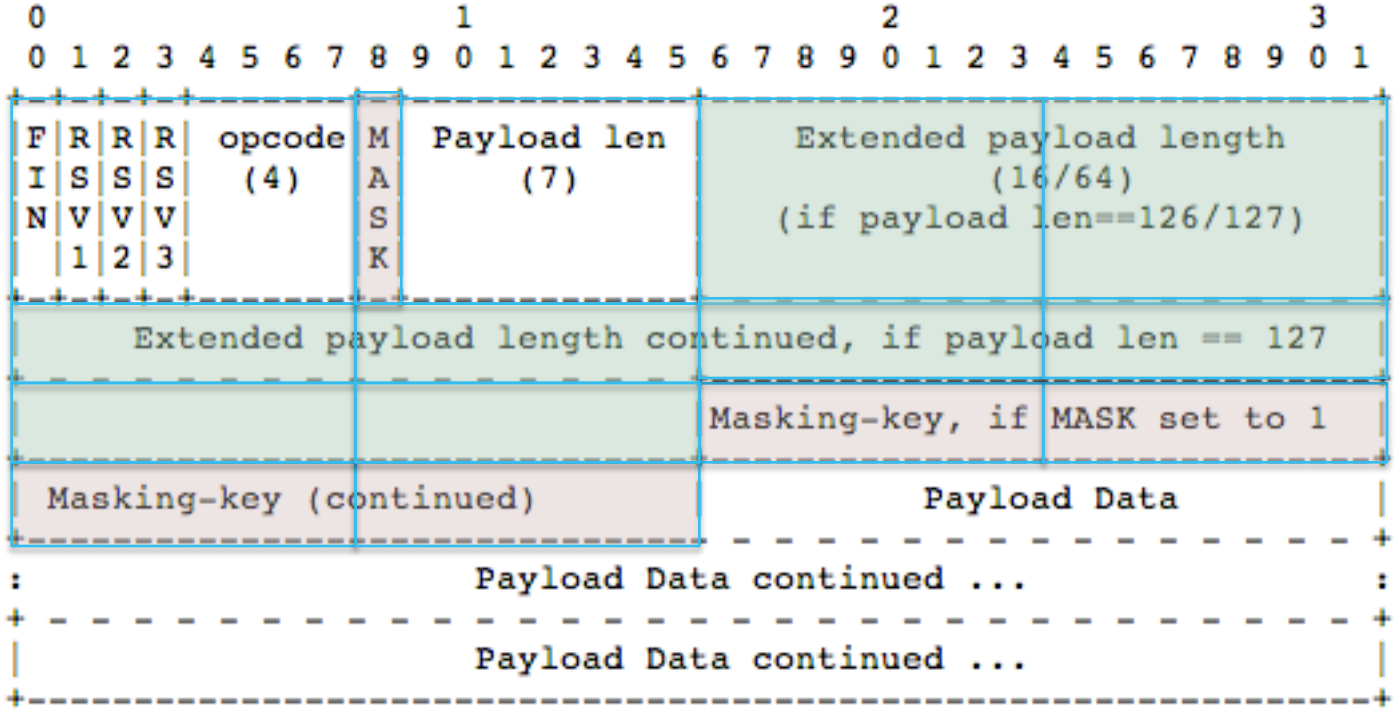
Request upgrade to WebSocket connection

WebSocket handshake headers

response

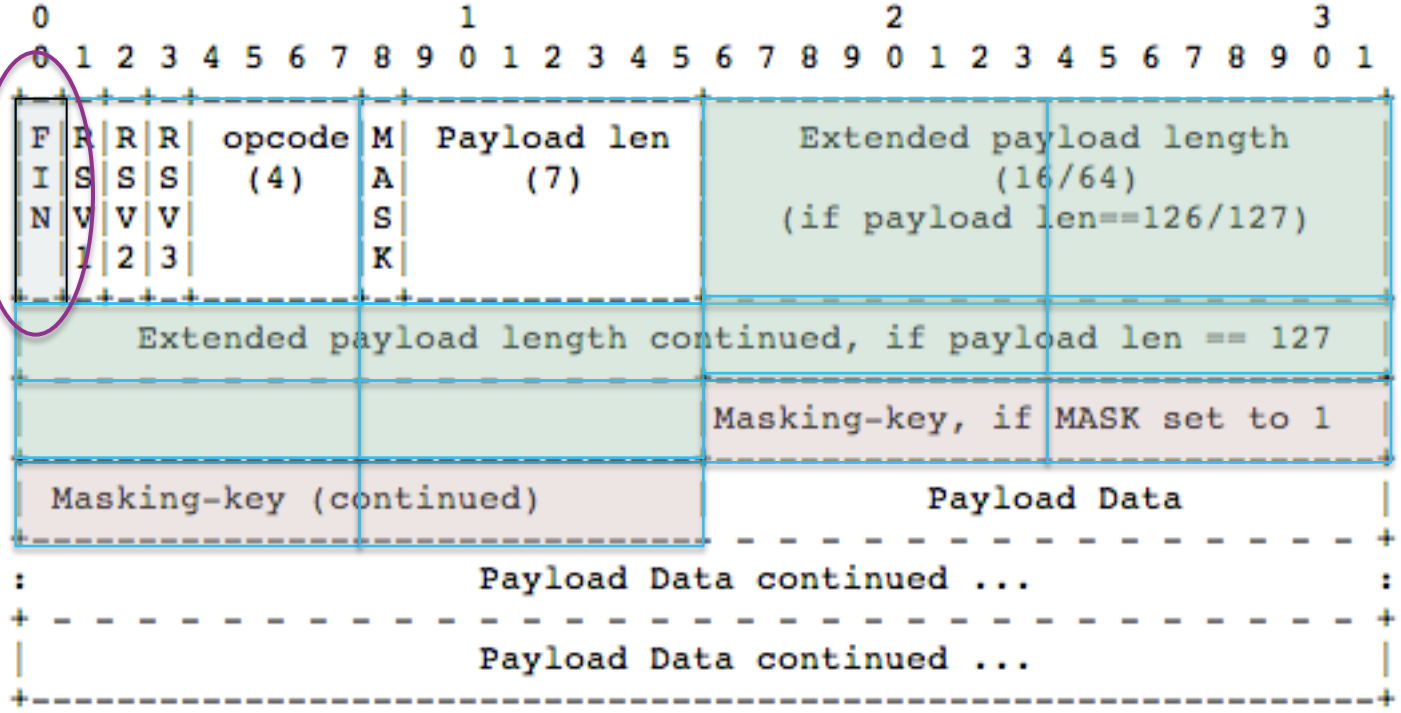
```
HTTP/1.1 101 Switching Protocols
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: custom
Sec-WebSocket-Extensions: compress
```

... and then transitions to frames



- Data or text is transmitted in frames
  - Minimally framed: small header, then payload

Messages can be fragmented across frames



- Message can be in one or more frames
  - Continue until FIN
  - A frame contains data for only one message
  - Extensions can be used to multiplex connections

Op Codes: identifying messages

- Control frames
  - Ping – 0xA
  - Pong – 0x9
  - Close – 0x8
- Data frames
  - Text – 0x1
    - UTF-8
  - Binary – 0x2
    - Arbitrary content: up to the application layer to determine
- Additional op codes are defined by negotiated extensions
  - Use reserved flags in the header

#ibminterconnect

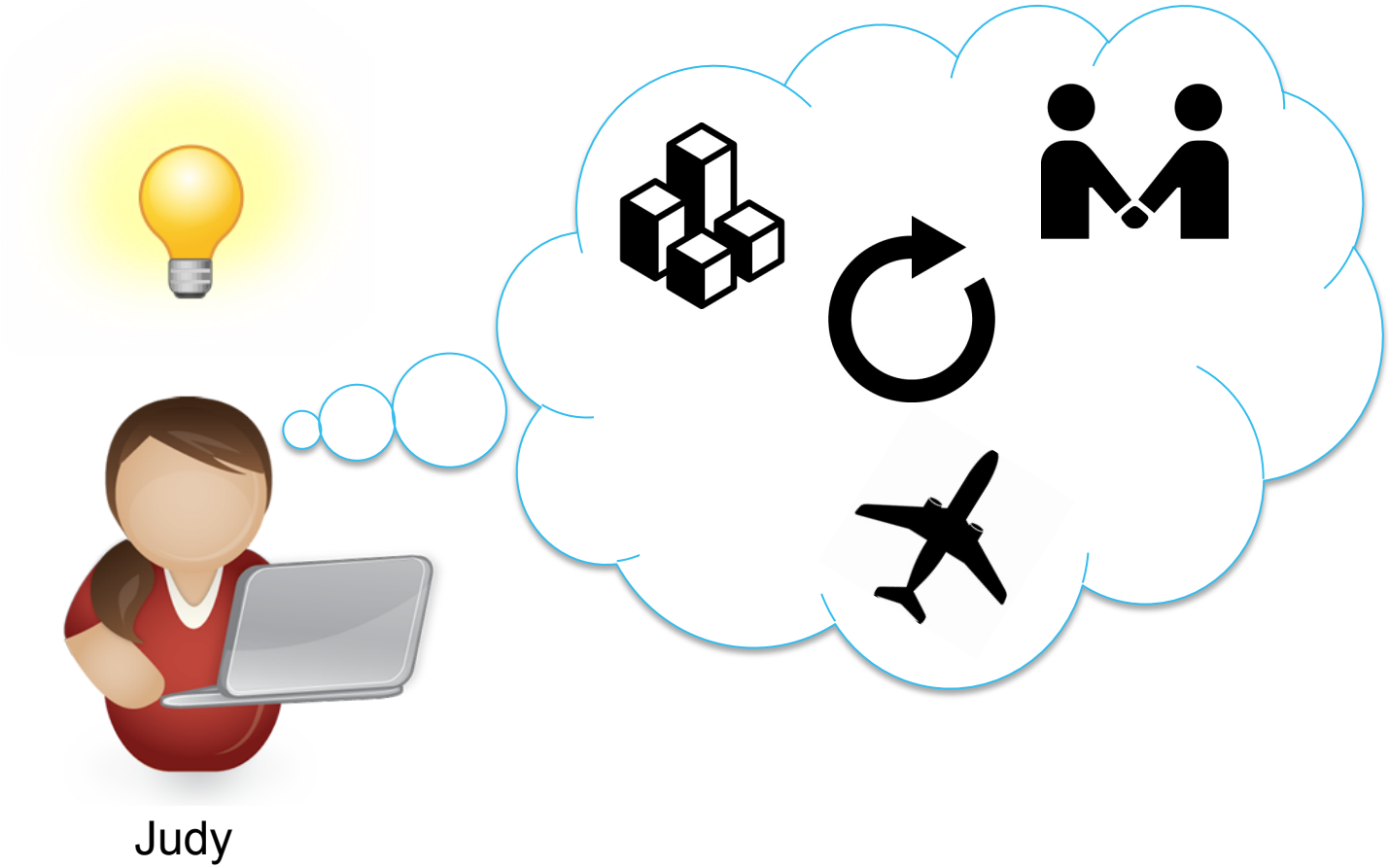
</background>

IBM

InterConnect 2015

The Premier Cloud & Mobile Conference

How do we use WebSockets in an application?



WebSockets API (JavaScript)

- Developed as part of HTML5:
  - http://dev.w3.org/html5/websockets/

```
interface WebSocket : EventTarget {
    readonly attribute DOMString url;
    // ready state
    const unsigned short CONNECTING = 0;
    const unsigned short OPEN = 1;
    const unsigned short CLOSING = 2;
    const unsigned short CLOSED = 3;
    readonly attribute unsigned short readyState;
    readonly attribute unsigned long bufferedAmount;

    // networking attribute EventHandler onopen;
    attribute EventHandler onerror;
    attribute EventHandler onclose;
    readonly attribute DOMString extensions;
    readonly attribute DOMString protocol;
    void close([Clamp] optional unsigned short code, optional DOMString reason);

    // messaging
    attribute EventHandler onmessage;
    attribute BinaryType binaryType;
    void send(DOMString data);
    void send(Blob data);
    void send(ArrayBuffer data);
    void send(ArrayBufferView data);
};
```

JavaScript client invocation...

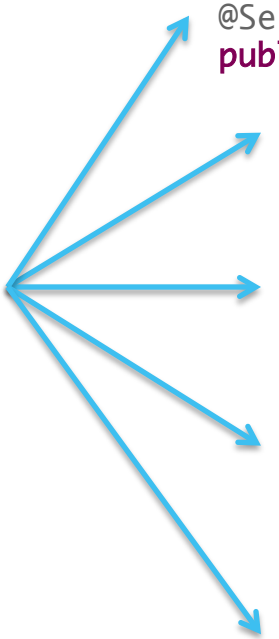
```
websocket = new WebSocket('ws://' +
                          window.document.location.host +
                          '/websocket/EchoEndpoint');

websocket.onerror = function(event) {
...
}

websocket.onopen = function(event) {
...
}

websocket.onclose = function(event) {
...
}

websocket.onmessage = function(event) {
...
}
```

WebSockets API (Java EE)

- Programmatic or annotation-based approach
- Client and Server Endpoints
  - Have a lifecycle
    - onOpen
    - onClose
    - onError
  - Communicate using Messages
    - onMessage
    - send
  - Use sessions
- Encoders and Decoders deal with data formatting
  - Messages ←→Java Objects
- SPI: extensions and data frames

Server Endpoint: Annotated

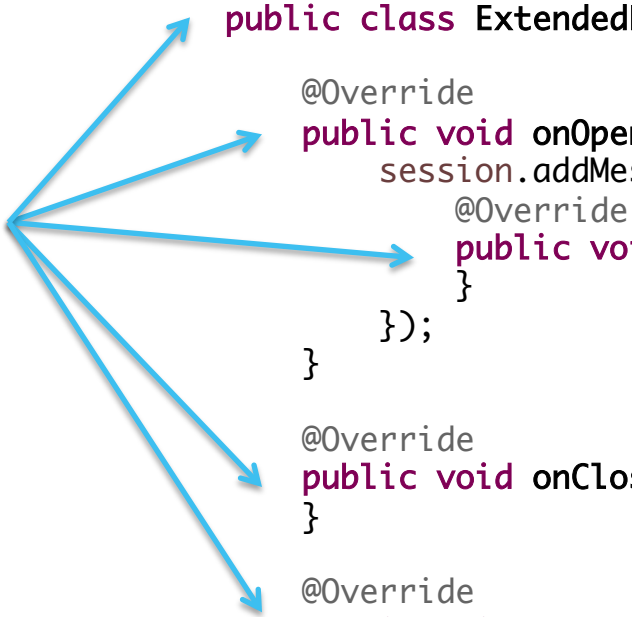
- Simple POJO with `@ServerEndpoint` annotation
  - value is the URI relative to your app's context root, e.g. ws://localhost/myapp/SimpleAnnotated
- Annotations for notifications: lifecycle and messages



```
@ServerEndpoint(value = "/SimpleAnnotated")
public class SimpleEndpoint {

    @OnOpen
    public void onOpen(Session session, EndpointConfig ec) {
    }

    @OnClose
    public void onClose(Session session, CloseReason reason) {
    }

    @OnMessage
    public void receiveMessage(String message, Session session) {

    }

    @OnError
    public void onError(Throwable t) {
    }
}
```

Server Endpoint: Programmatic

- Class extends `Endpoint`
- Callback methods for lifecycle event notifications
- Message notifications require a `MessageHandler`



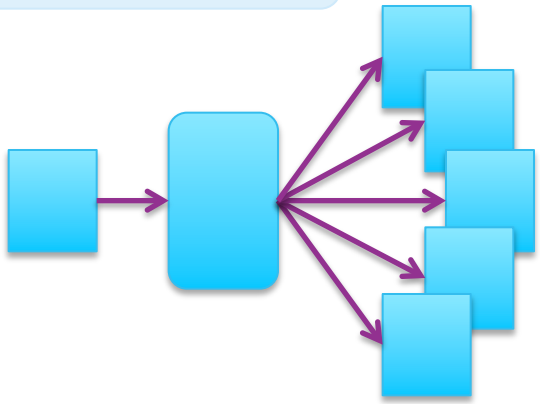
```
public class ExtendedEndpoint extends Endpoint {

    @Override
    public void onOpen(Session session, EndpointConfig ec) {
        session.addMessageHandler(new MessageHandler.Whole<String>() {
            @Override
            public void onMessage(String message) {
            }
        });
    }

    @Override
    public void onClose(Session session, CloseReason reason) {
    }

    @Override
    public void onError(Session session, Throwable t) {
    }
}
```

Simple echo + server provided data (using annotations)

- `@OnMessage` method is called when a message is received
  - If message is ‘stop’: close the session
  - Otherwise, echo the message along with a hit count

```
int count = 0;

@OnMessage
public void receiveMessage(String message, Session session) throws IOException {
    if ( "stop".equals(message) ) {
        session.close();
    } else {
        int id = count++;
        for (Session s : session.getOpenSessions() ) {
            s.getBasicRemote().sendText("Echo " + id + ":  " + message);
        }
    }
}
```

- Broadcast – iterate over open sessions



Invocation.. what happens?

- …. Let's see!

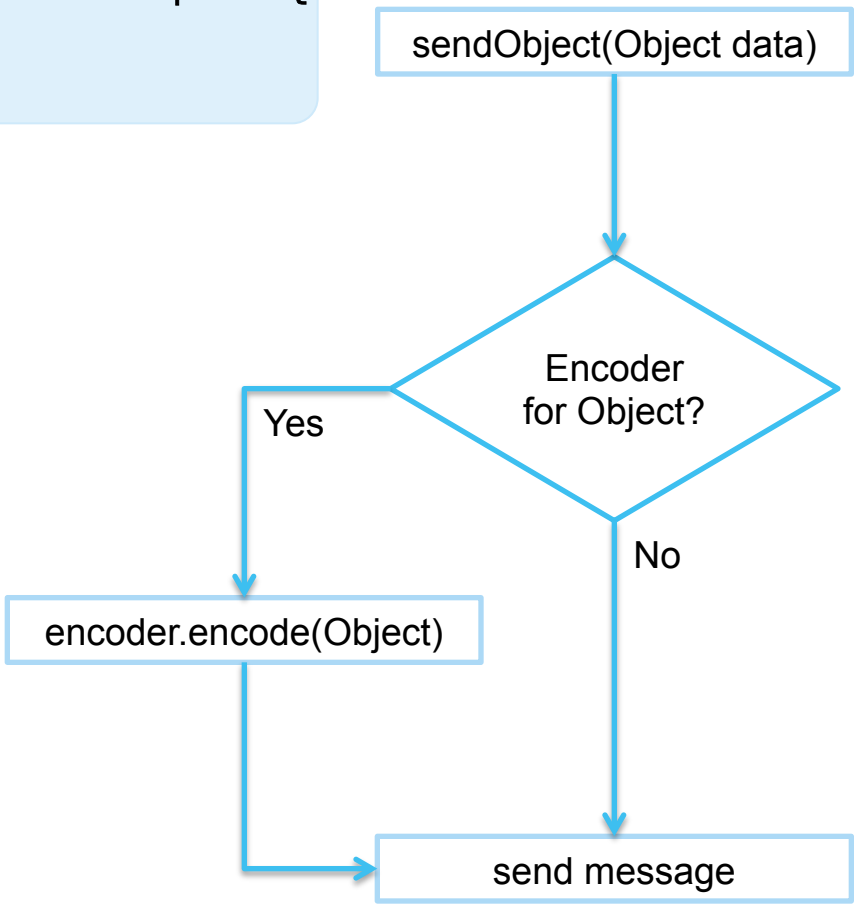
Encoder/Decoder: dealing with data

- Messages can be in text or binary format
- Encoders and Decoders translate between data on the socket and Java Objects

```
@ServerEndpoint(value = "/EchoEncoderEndpoint",
                decoders = EchoDecoder.class,
                encoders = EchoEncoder.class)
public class EchoEncoderEndpoint {


    @OnMessage
    public void receiveMessage(EchoObject o, Session session)
    throws IOException, EncodeException {
        if (o.stopRequest() ) {
            session.close();
        } else {
            for (Session s : session.getOpenSessions() ) {
                s.getBasicRemote().sendObject(o);
            }
        }
    }

}
```

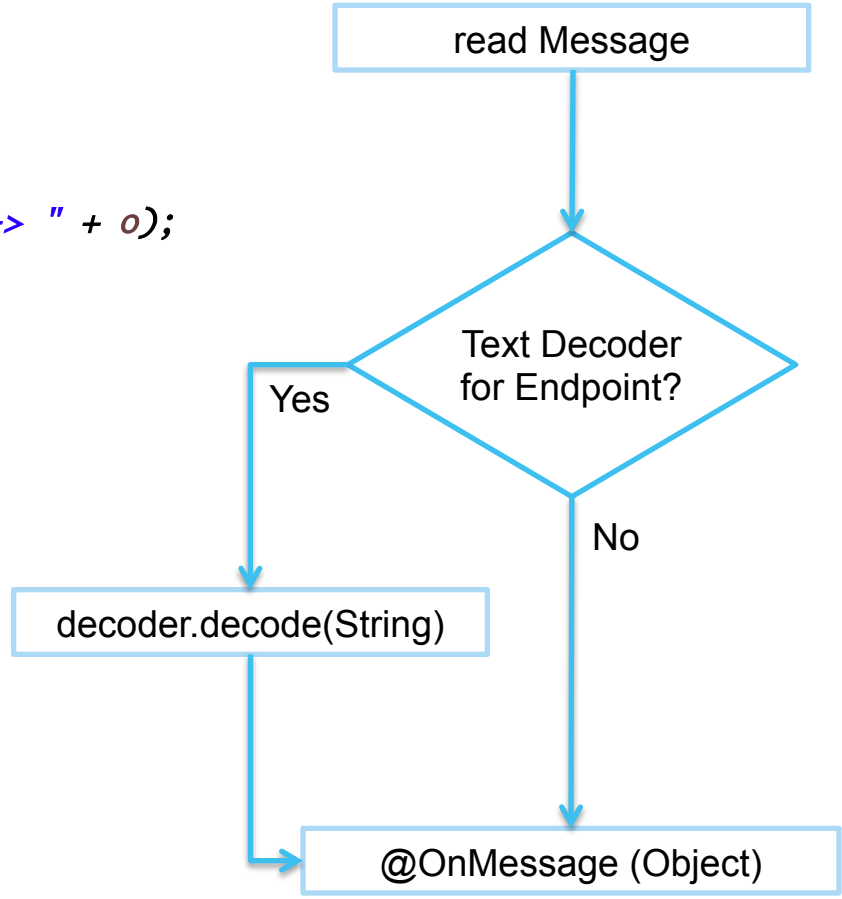
Encoder: dealing with data

```
public class EchoEncoder implements Encoder.Text<EchoObject> {

    @Override
    public String encode(EchoObject o) throws EncodeException {
        System.out.println("Encoding " + o);
        return o.toString();
    }

    @Override
    public void init(EndpointConfig ec) {}

    @Override
    public void destroy() {}
}
```



Decoder: dealing with data

```
public class EchoDecoder implements Decoder.Text<EchoObject> {

    @Override
    public EchoObject decode(String msg) throws DecodeException {
        EchoObject o;
        try {
            o = new EchoObject(msg);
        } catch(Exception e) {
            o = new EchoObject(e);
        }
        System.out.println("Decoded " + msg + " -> " + o);
        return o;
    }

    @Override
    public boolean willDecode(String msg) {
        return true;
    }

    @Override
    public void init(EndpointConfig ec) {}

    @Override
    public void destroy() {}
}
```



EchoObject

```
public class EchoObject {
    static final AtomicInteger count = new AtomicInteger();

    final JsonObject obj;

    public EchoObject(String msg)  {
        JsonReader r = Json.createReader(new StringReader(msg));
        JsonObject in = r.readObject();

        JsonObjectBuilder b = Json.createObjectBuilder();
        b.add("count", count.getAndIncrement());
        b.add("content", in.getString("content", "none provided"));
        obj = b.build();
    }

    public String toString() {
        return obj.toString();
    }

    public EchoObject(Exception e) {
        JsonObjectBuilder b = Json.createObjectBuilder();
        b.add("content", e.toString());
        b.add("count", -1);
        obj = b.build();
    }

    public boolean stopRequest() {
        return "stop".equals(obj.getString("content"));
    }
}
```

decode

encode

Invocation.. what happens?

- …. Let's see!

WebSockets have everything this application needs..

- Why wouldn't you use them?
  - Older devices / browsers don't support WebSockets
  - May be a challenge to degrade gracefully so older devices still have a decent experience
- Trouble with proxies..
  - wss:// recommended over ws://
    - Some proxy servers do not inspect encrypted traffic: just pass through

Challenges for Proxy servers

- WebSocket protocol is unaware of proxies / firewalls
- Relies on Upgrade header:
  - Hop-by-Hop Upgrade
    - Proxy server sends the request to the next hop
    - Upgrade header is only good for one link
- Proxy servers required to strip certain headers when forwarding
  - Some load balancers also mangle the Connection header
- HTTP CONNECT
  - Proxy to forward the TCP Connection to the destination
  - *Some proxies still analyze traffic: would choke on websocket frame*
  - *SSL tunnelling has a better shot:*
    - *Some proxies restrict CONNECT to SSL*
    - *BUT: SSL termination…*

WebSockets for Rich Clients

- Java API for a rich Client is similar to API for Server
- Annotations:

```
@ClientEndpoint
public class AnnotatedClient {
    @OnOpen
    public void onOpen(Session session, EndpointConfig ec) {
    }

    @OnClose
    public void onClose(Session session, CloseReason reason) {
    }

    @OnMessage
    public void processMessageFromServer(String message, Session session) {
        System.out.println("Message came from the server ! " + message);
    }

    @OnError
    public void onError(Throwable t) {
    }
}
```

WebSockets for Rich Clients

- Java API for a rich Client is similar to API for Server
- Programmatic:

```
final WebSocketContainer webSocketContainer = ContainerProvider.getWebSocketContainer();

Session session = webSocketContainer.connectToServer(new Endpoint() {
    @Override
    public void onOpen(Session session, EndpointConfig config) {
        session.addMessageHandler(new MessageHandler.Whole<String>() {
            @Override
            public void onMessage(String message) {
                System.out.println("Message came from the server ! " + message);
            }
        });
    }
}, URI.create("ws://some.uri"));
```

#ibminterconnect

Questions?

IBM

InterConnect 2015

The Premier Cloud & Mobile Conference

Notices and Disclaimers

U.S. Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM.

Information in these presentations (including information relating to products that have not yet been announced by IBM) has been reviewed for accuracy as of the date of initial publication and could include unintentional technical or typographical errors. IBM shall have no responsibility to update this information. THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IN NO EVENT SHALL IBM BE LIABLE FOR ANY DAMAGE ARISING FROM THE USE OF THIS INFORMATION, INCLUDING BUT NOT LIMITED TO, LOSS OF DATA, BUSINESS INTERRUPTION, LOSS OF PROFIT OR LOSS OF OPPORTUNITY. IBM products and services are warranted according to the terms and conditions of the agreements under which they are provided.

Any statements regarding IBM's future direction, intent or product plans are subject to change or withdrawal without notice.

Performance data contained herein was generally obtained in a controlled, isolated environments. Customer examples are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual performance, cost, savings or other results in other operating environments may vary.

References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business.

Workshops, sessions and associated materials may have been prepared by independent session speakers, and do not necessarily reflect the views of IBM. All materials and discussions are provided for informational purposes only, and are neither intended to, nor shall constitute legal or other guidance or advice to any individual participant or their specific situation.

It is the customer's responsibility to insure its own compliance with legal requirements and to obtain advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulatory requirements that may affect the customer's business and any actions the customer may need to take to comply with such laws. IBM does not provide legal advice or represent or warrant that its services or products will ensure that the customer is in compliance with any law.

Notices and Disclaimers (con't)

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products. IBM does not warrant the quality of any third-party products, or the ability of any such third-party products to interoperate with IBM's products. IBM EXPRESSLY DISCLAIMS ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents, copyrights, trademarks or other intellectual property right.

- IBM, the IBM logo, ibm.com, Bluemix, Blueworks Live, CICS, Clearcase, DOORS®, Enterprise Document Management System™, Global Business Services ®, Global Technology Services ®, Information on Demand, ILOG, Maximo®, MQIntegrator®, MQSeries®, Netcool®, OMEGAMON, OpenPower, PureAnalytics™, PureApplication®, pureCluster™, PureCoverage®, PureData®, PureExperience®, PureFlex®, pureQuery®, pureScale®, PureSystems®, QRadar®, Rational®, Rhapsody®, SoDA, SPSS, StoredIQ, Tivoli®, Trusteer®, urban{code}®, Watson, WebSphere®, Worklight®, X-Force® and System z® Z/OS, are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at: www.ibm.com/legal/copytrade.shtml.

#ibminterconnect

Thank You

Your Feedback is Important!

Access the InterConnect 2015 Conference CONNECT Attendee Portal to complete your session surveys from your smartphone, laptop or conference kiosk.

IBM

InterConnect 2015

The Premier Cloud & Mobile Conference

Other sessions you might like

1313A	Agile Development Using Java EE 7 with WebSphere Liberty Profile Tuesday: 08:00 AM - 10:00 AM - Mandalay Bay, South Seas Ballroom D
2977A (lab)	Accelerate Your Business with Liberty's New HTML5 Real-Time Features Tuesday: 11:00 AM - 01:00 PM - Mandalay Bay, South Seas Ballroom A
1304A	Technical Deep Dive into IBM WebSphere Liberty Tuesday: 11:00 AM - 12:00 PM - Mandalay Bay, Reef Ballroom E
3085A	Don't Wait! Develop Responsive Applications with Java EE7 Instead Thursday: 10:30 AM - 11:30 AM - Mandalay Bay, Lagoon L