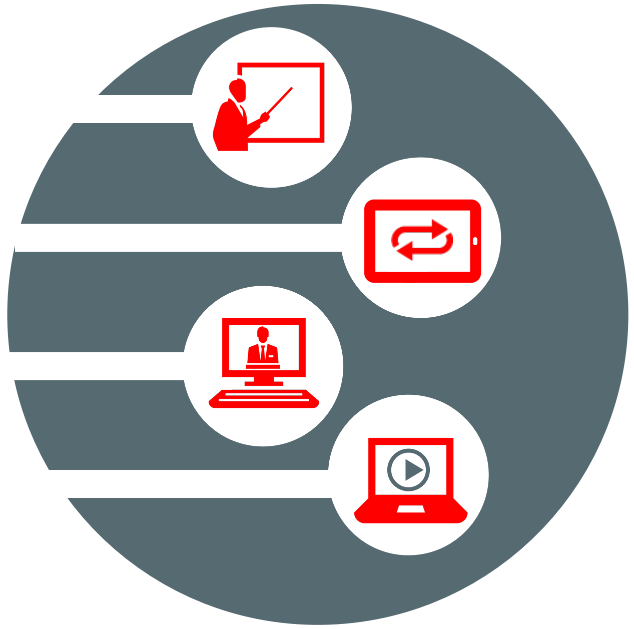
ORACLE®

JavaOne™

ORACLE®

Keep Learning with Oracle University

ORACLE
UNIVERSITY



Classroom Training

Learning Subscription

Live Virtual Class

Training On Demand

Cloud

Technology

Applications

Industries

education.oracle.com

Session Surveys

Help us help you!!

- Oracle would like to invite you to take a moment to give us your session feedback. Your feedback will help us to improve your conference.
- Please be sure to add your feedback for your attended sessions by using the Mobile Survey or in Schedule Builder.

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

WebSocket in Enterprise apps

Pavel Bucek (pavel.bucek@oracle.com)

Oracle
October 28, 2015

Program Agenda

1. Evolution – REST, Polling, SSE, WebSocket
2. Does HTTP/2 make WebSocket obsolete?
3. When to use WebSocket
4. Java EE – WebSocket API
5. Advanced architectures/usecases

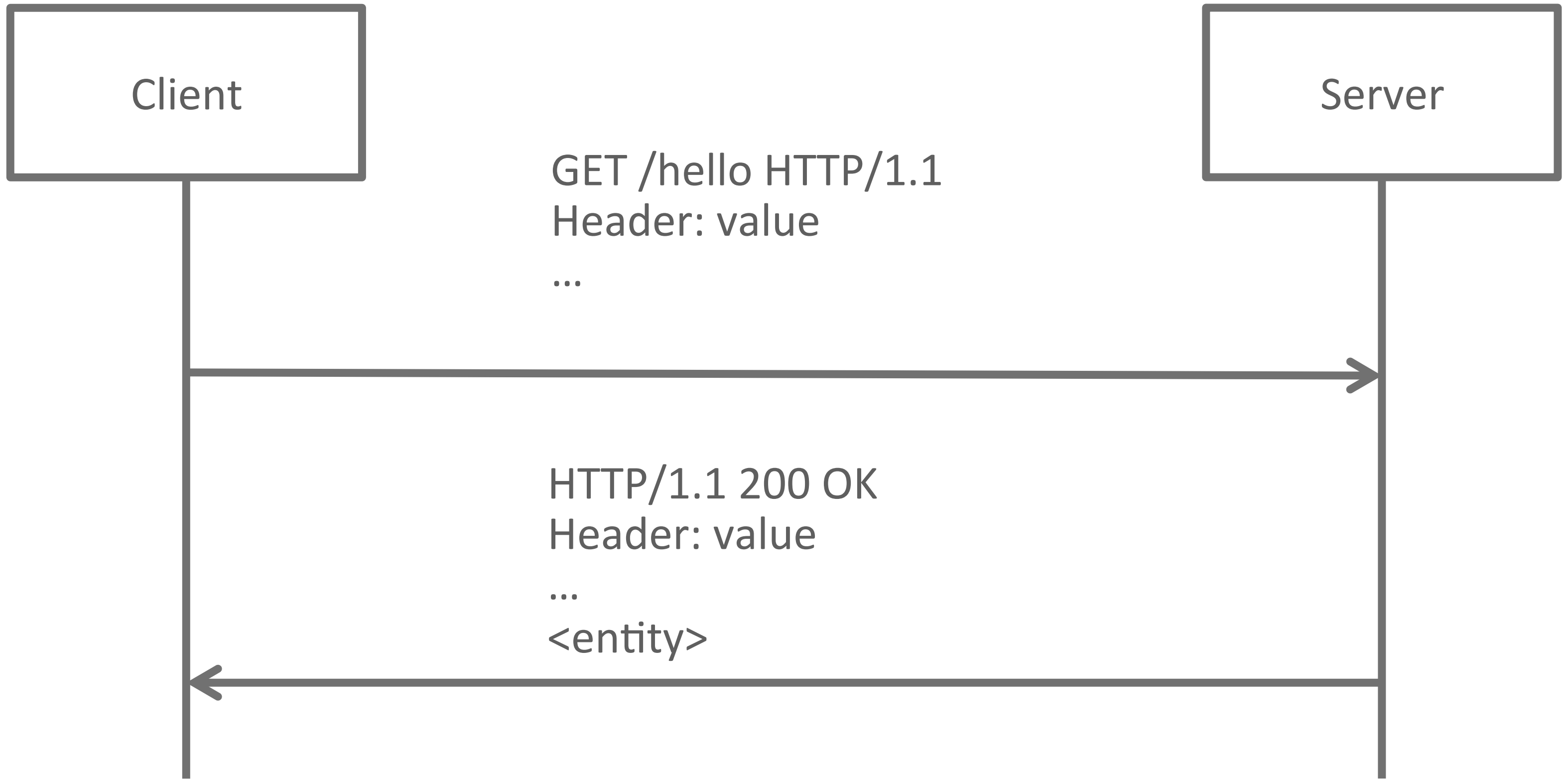
Evolution – REST

Representational State Transfer

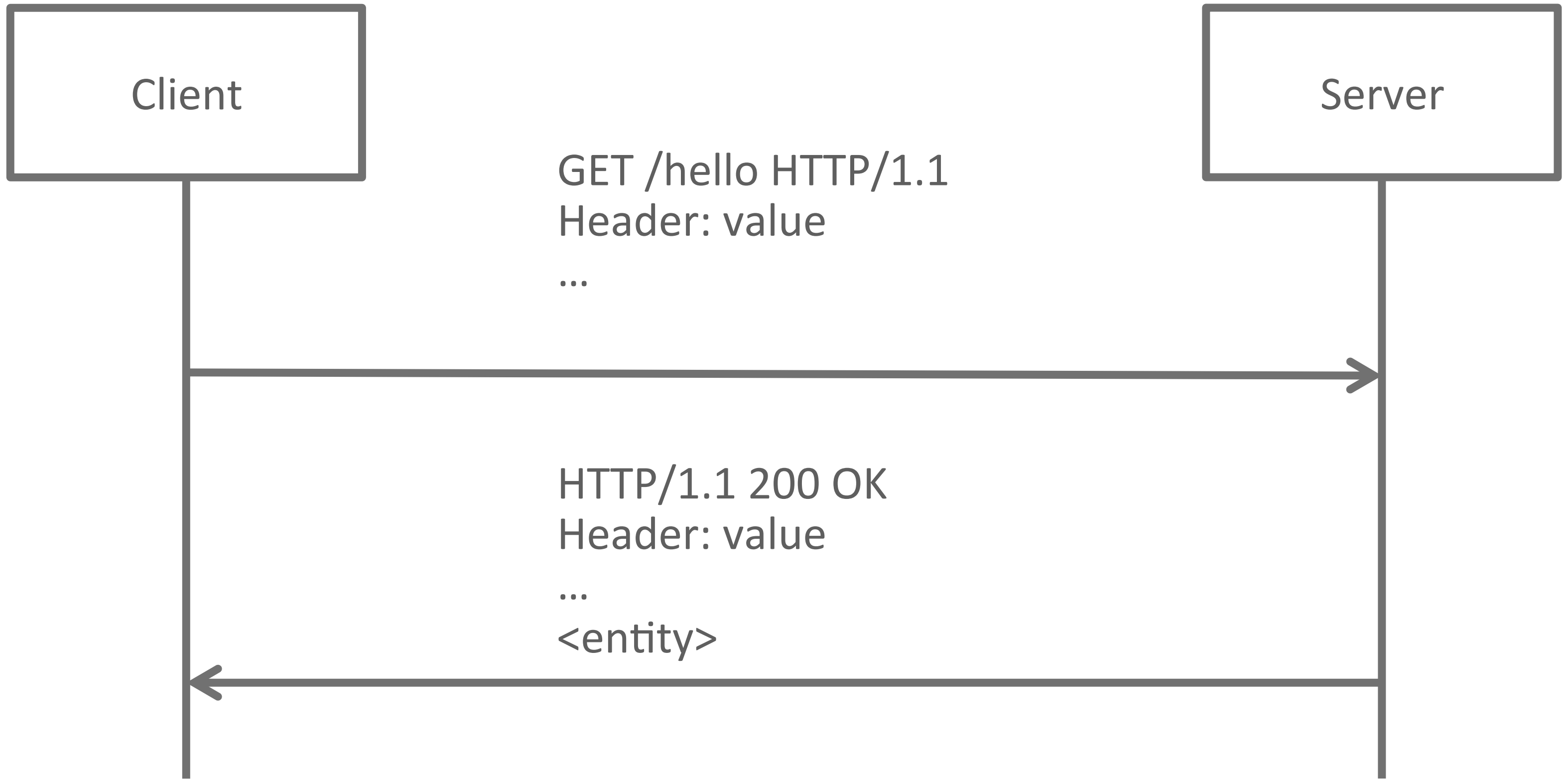
- Basic software architecture / scheme
- Exposes Resources (URIs), which handle METHODs
  - GET/PUT/POST/DELETE/HEAD/OPTIONS/TRACE/PATCH/…
- MediaType (Accept/Content-Type)
  - text/plain, text/html, application/json, …
- Caching (GET, HEAD), Hyperlinking, …
- REST effectively replaces SOAP
  - Simple Object Access Protocol

Evolution – REST

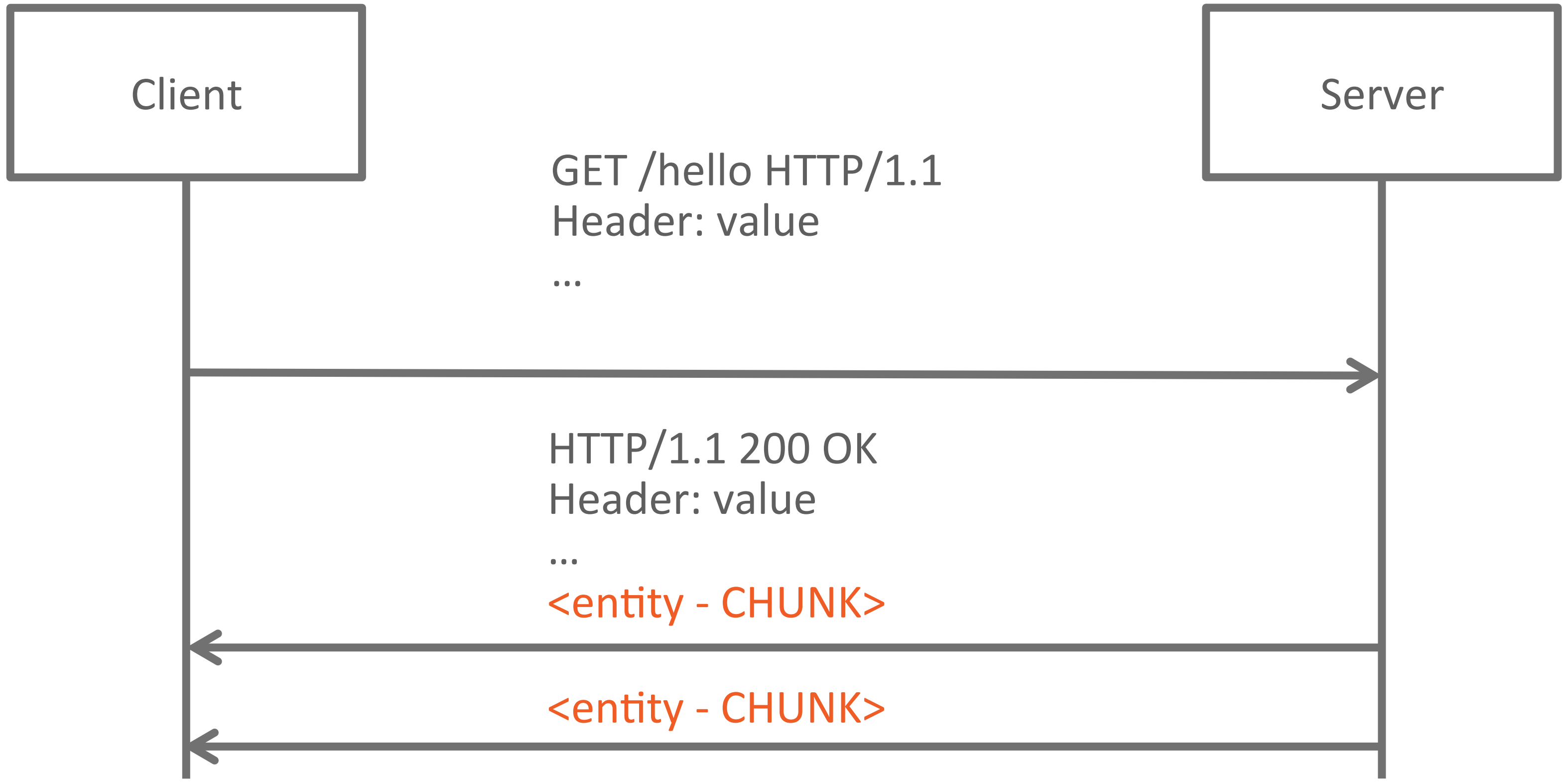
Representational State Transfer



Evolution – Polling



Evolution – Long-Polling



Evolution – SSE

Server Sent Events

- Channel for sending events from server to the client
- One-way, text protocol
- Very similar to long polling
  - (actually, it's exactly same, but this time it has own RFC)
- Accept: text/event-stream
- Semi-permanent connection
  - Clients are required to reconnect when the connection is lost
- Limited browser support (no IE, no Android)

Evolution – WebSocket

- RFC 6455 (December 2011)
- Bi-directional communication
- Uses HTTP/1.1 for initial handshake
  - Completely different protocol afterwards
- “Server” and “client” endpoints are equal after handshake
- Text or **binary** payload
- Supported in all modern browsers

Evolution – WebSocket

WebSocket Frame

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-------+-+-------------+-------------------------------+
|F|R|R|R| opcode|M| Payload len |    Extended payload length    |
|I|S|S|S|  (4)  |A|     (7)     |             (16/64)           |
|N|V|V|V|       |S|             |   (if payload len==126/127)   |
| |1|2|3|       |K|             |                               |
+-+-+-+-+-------+-+-------------+ - - - - - - - - - - - - - - - +
|     Extended payload length continued, if payload len == 127  |
+ - - - - - - - - - - - - - - - +-------------------------------+
|                               |Masking-key, if MASK set to 1  |
+-------------------------------+-------------------------------+
| Masking-key (continued)       |          Payload Data         |
+-------------------------------- - - - - - - - - - - - - - - - +
:                     Payload Data continued ...                :
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
|                     Payload Data continued ...                |
+---------------------------------------------------------------+
```

WebSocket vs REST

- Might seem to be similar and some people even think about WebSocket as about another “Revolution” – like SOAP --> REST
- In reality, these are two different concepts which **COMPLEMENT** each other.
- REST can be still used for most of the implementation part – web, forms, …
- WebSocket provides bi-directional channel, suitable for exchanging “short” messages with the browser.
  - (the scope is not limited, you can re-implement all your communication with server, but there is no point in doing that..)

WebSocket vs HTTP/2

HTTP/2 Key Features

- Same semantics as HTTP/1.1
- Binary protocol
- Multiplexed protocol
  - Single TCP connection to single origin, shared for consequent/parallel requests
- Compressed headers
  - HTTP/2 introduces HPACK (compression algorithm)
- Server Push
  - Server can push (cacheable) content to the client before client asks

WebSocket vs HTTP/2

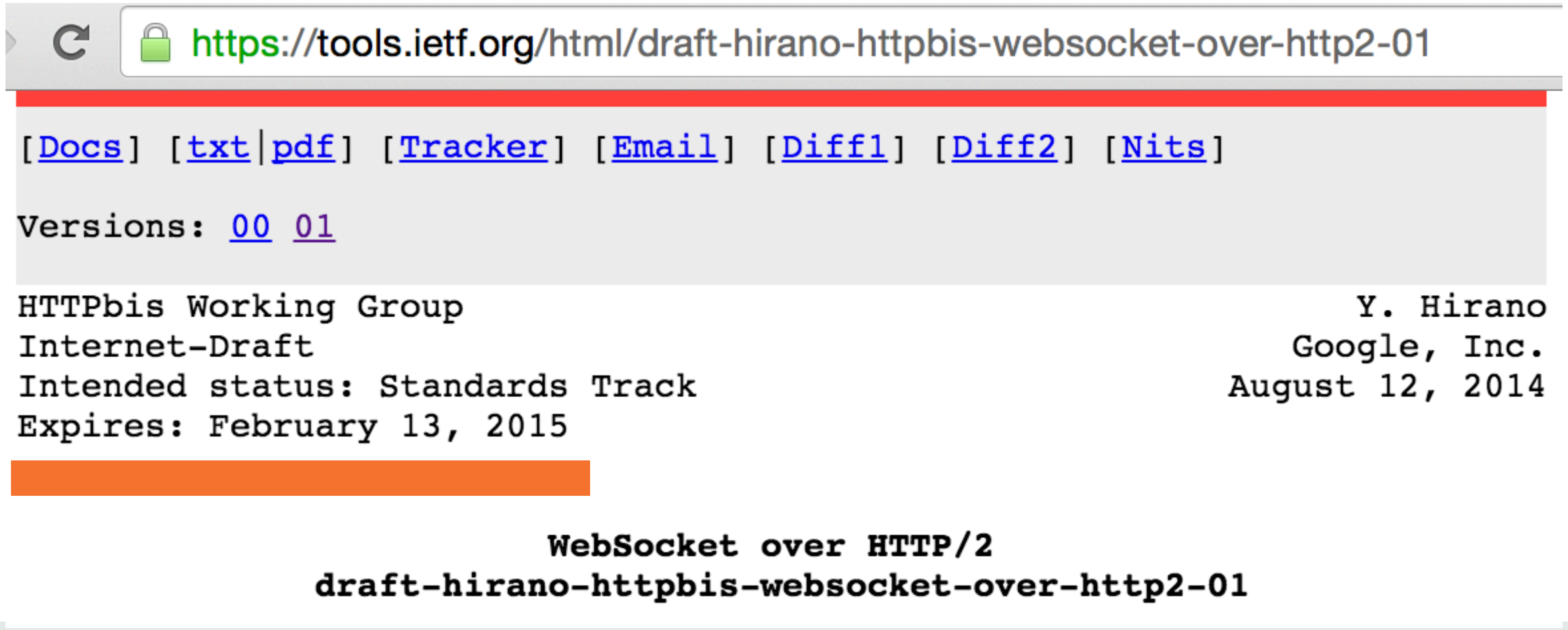
WebSocket vs Server push

- Pushed resources are cached on client side
- When client decides it needs something (image, ..), it looks into the cache
  - Doesn't need to be image, but all cool demos are using that (remember SPDY)
- Server push has its own issues
  - What if client does not need pushed resource? (there might be other caches, …)
- Push is not "interactive" message exchange

WebSocket and HTTP/2

Standards..

- Currently, WebSocket is not defined in HTTP/2 world
- WebSocket uses UPGRADE header, which gives complete control over TCP connection
- HTTP/2 Streams could support WS multiplex



https://tools.ietf.org/html/draft-hirano-httpbis-websocket-over-http2-01

[Docs] [txt|pdf] [Tracker] [Email] [Diff1] [Diff2] [Nits]

Versions: 00 01

```
HTTPbis Working Group                                          Y. Hirano
Internet-Draft                                               Google, Inc.
Intended status: Standards Track                          August 12, 2014
Expires: February 13, 2015
```

WebSocket over HTTP/2
draft-hirano-httpbis-websocket-over-http2-01

When to use WebSocket

- Bi-directional communication
  - “messaging”
- Interactive applications
  - Any time, you need fast data exchange with the backend
- Time-critical data delivery
  - Stock quotes

WebSocket usecases

- Chat-like applications
  - Various implementations options
  - XMPP (Jabber) over WebSocket
- Trading and transactions
  - Fast feedback/execution
- Real-time monitoring
  - Depends on the data source
  - Interaction with monitored object
  - (SSE?)
- Remote control
  - Input with “real-time” feedback
  - From industry application to fun apps
- Games
  - HTML5 “native” transport
  - Supported by improvements in browsers 2D/3D canvas support
- General collaboration
  - Customer service, Social apps, …

WebSocket usecases

Subprotocols

- RFC 7118 – SIP over WebSocket
- RFC 7355 – SIP & CLF (Common Log Format) over WebSocket
- RFC 7395 – XMPP over WebSocket
- Drafts
  - MSRP (Message Session Relay Protocol) over WebSocket
  - SDP (Session Description Protocol) over WebSocket
  - Remote Framebuffer Protocol over WebSocket
  - <anything> over WebSocket

Java API for WebSocket

- JSR 356 – Part of Java EE 7
  - 1.0 (May 2013)
  - 1.1 (August 2014)
- Annotated and programmatic way how to deploy and access WebSocket endpoints
- Event-driven model - @OnOpen, @OnMessage, @OnError, @OnClose
- Encoders/Decoders, Path/Query parameter handling, Handshake headers interceptors, CDI integration, …

Java API for WebSocket – Annotated Endpoint

```
@ServerEndpoint("/echo")
public class EchoEndpoint {

    @OnOpen
    public void onOpen(Session session) throws IOException {
        session.getBasicRemote().sendText("onOpen");
    }

    @OnMessage
    public void echo(Session session, String message) throws IOException {
        session.getBasicRemote().sendText(message + " (from your server)");
        session.close();
    }

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

Java API for WebSocket – Programmatic Endpoint

```java
public class EchoProgrammaticEndpoint extends Endpoint {
    @Override
    public void onOpen(final Session session, EndpointConfig config) {

        session.addMessageHandler(String.class, new MessageHandler.Whole<String>() {
            @Override
            public void onMessage(String message) {
                try {
                    session.getBasicRemote().sendText(message + " (from your server)");
                    session.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }

            }
        });
    }

    @Override
    public void onError(Session session, Throwable thr) {
        thr.printStackTrace();
    }
}
```

WebSocket vs Java 9 / JDK 1.9

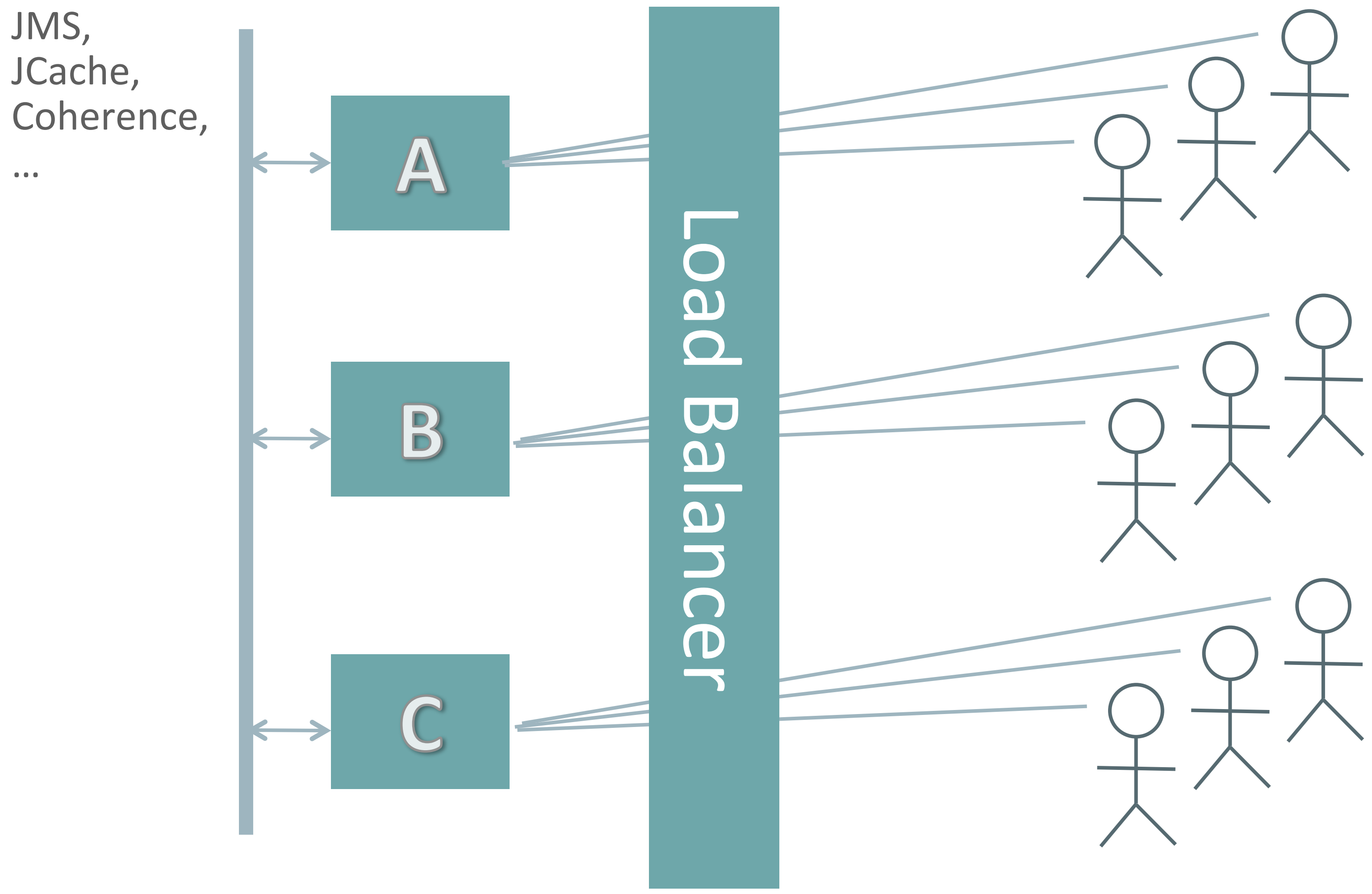
Lambdas, WebSocket client

- Some classes in the WebSocket API could be replaced / disassembled to separate Java 8 Consumer<T>s
  - MessageHandlers already can be written as lambda functions
- Default Methods, CompletionStage<T>, Method Parameter Reflection, Streams, …
- JDK 9 – JEP 110: HTTP/2 Client: http://openjdk.java.net/jeps/110
  - *Define a new HTTP client API that implements HTTP/2 and WebSocket, …*

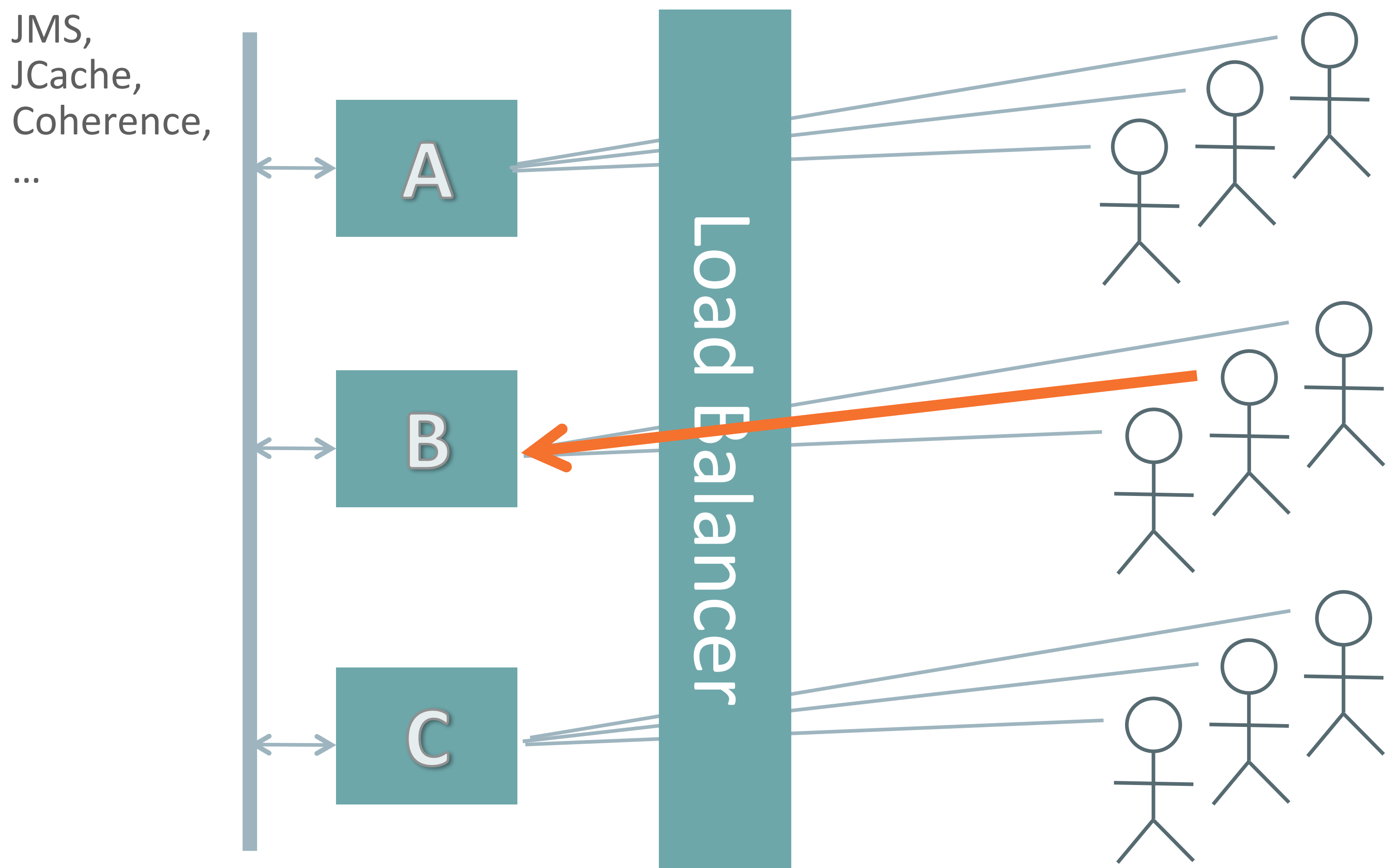
WebSocket and Clustering

- Different to “classic” cluster
  - We’d like to talk to other sessions (clients) directly
- Clustered environment present different challenges
  - “Finding a WebSocket Session” might not be as trivial as it seem
- Broadcast (mass-notification) is common usecase
- Clustering can help
  - Broadcasting to “all sessions” can be faster in clustered environment

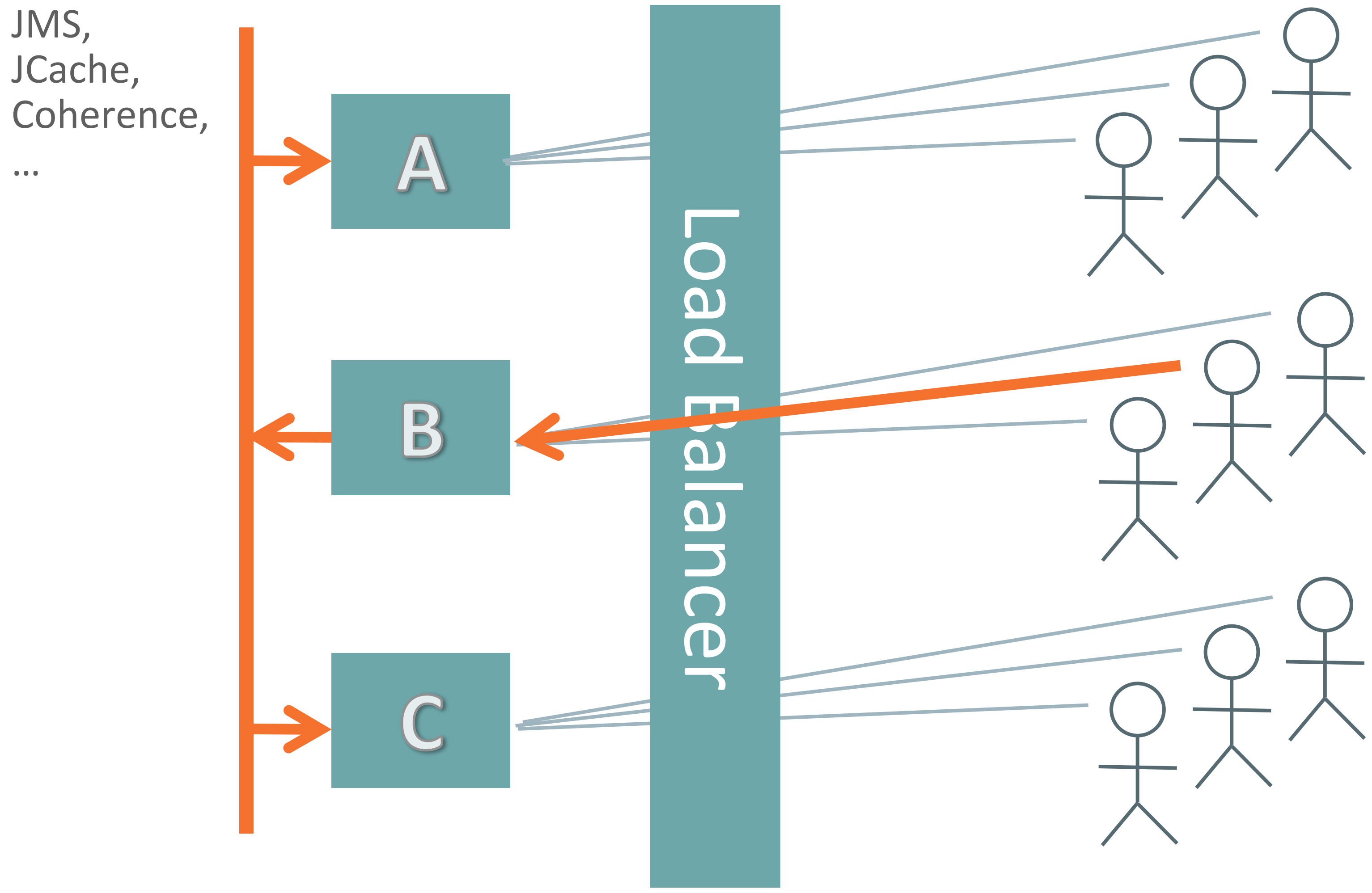
WebSocket and Clustering



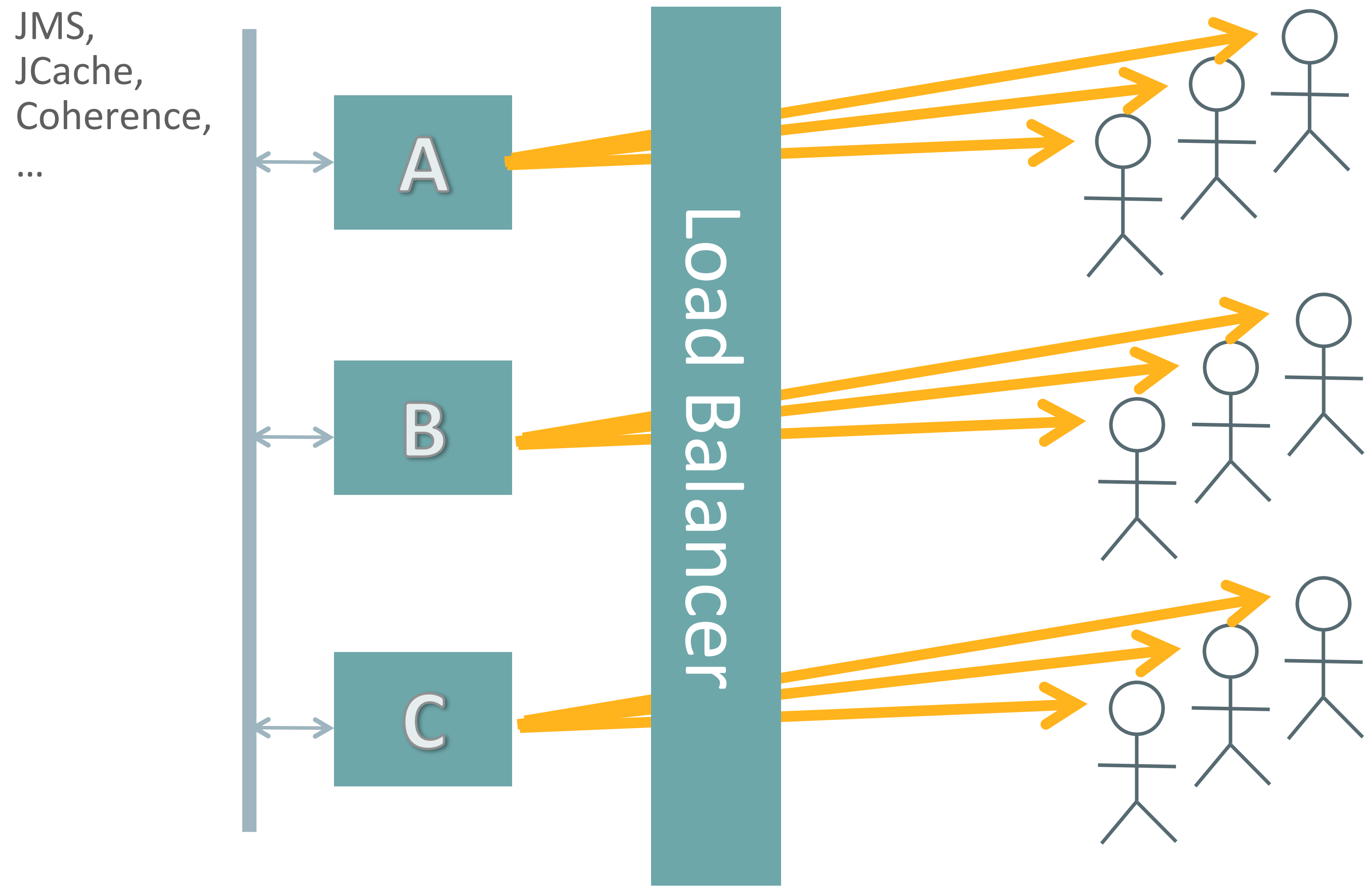
WebSocket and Clustering



WebSocket and Clustering



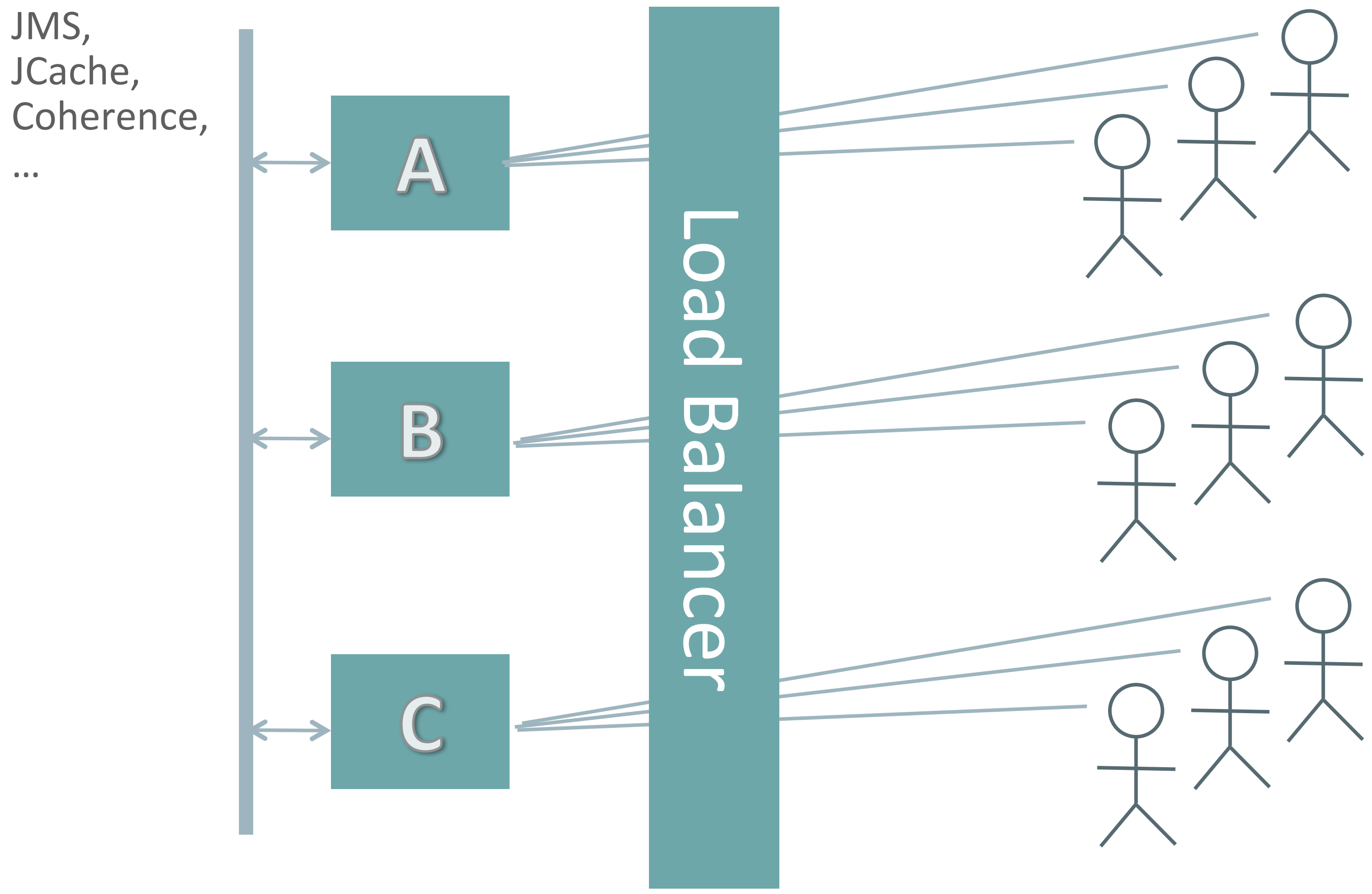
WebSocket and Clustering



WebSocket and Clustering

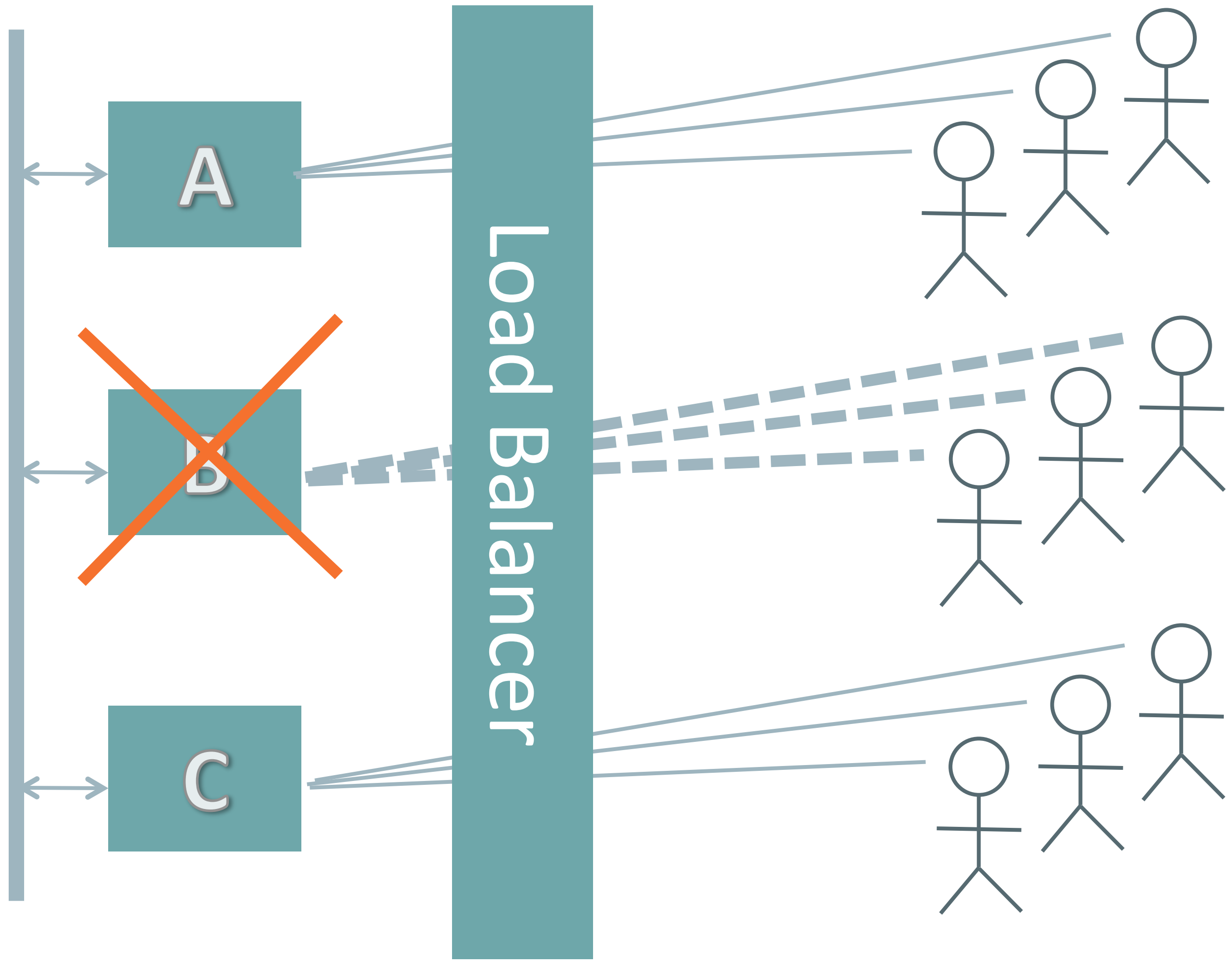
- Different requirements
- Load balancers will need to manage more open connections
- Cannot re-balance when session is already created
  - Node is added – ok, but I cannot easily decrease load on other nodes, I can use the new node only for new connections
- Node going down is not transparent to client
  - Robust clients can hide this, but for now, this requires custom solution
  - WLS 12.2.1 offers a solution for this..

WebSocket and Clustering



WebSocket and Clustering

JMS,
JCache,
Coherence,
…



WebSocket and Clustering

- Might require more application code
  - Proper handling on client side
  - Even more code when you want to have something like auto reconnect + “session recovery”
- Infrastructure requirements
  - Similar to SSL (persistent connection, …)
- Much more effective when compared to the same app implemented using long-polling

CREATE THE FUTURE

ORACLE®