Oracle Coherence and WebLogic 12c
Delivering Real Time Push at Scale

Steve Millidge

About Me

- Founder of C2B2
  - Leading Independent Middleware Experts
  - Non-functional Experts
- Vendor Neutral
  - Red Hat (JBoss), Oracle (Fusion), VMWare (vFabric), Open Source (Apache)
- 20 Years Middleware Expertise
- 15 years Field Consultancy

C2B2

JBoss by Red Hat READY BUSINESS PARTNER

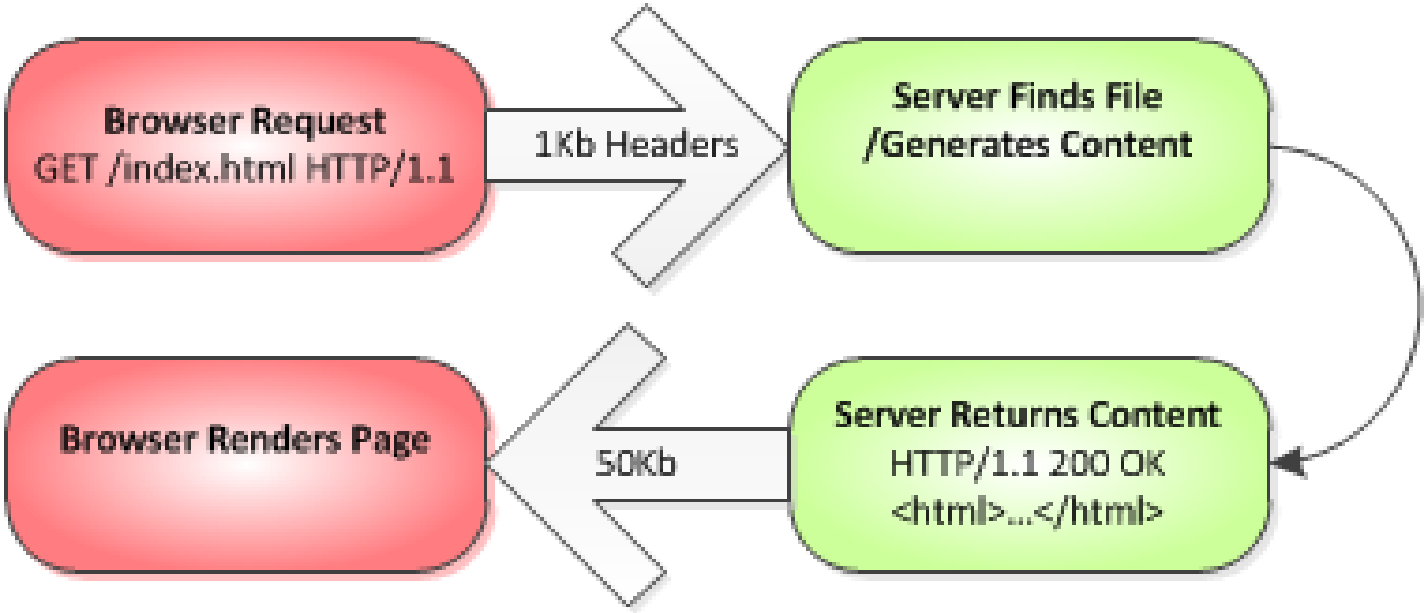
ORACLE Gold Partner Specialized Application Grid

vmware PARTNER PROFESSIONAL SOLUTION PROVIDER

Agenda

- Introduction to Web Sockets and Push
- WebSockets in WebLogic 12c
- Introduction to Coherence
- Test Demo
- Code walkthrough - Hooking up to Coherence
- Summary

Standard HTTP model

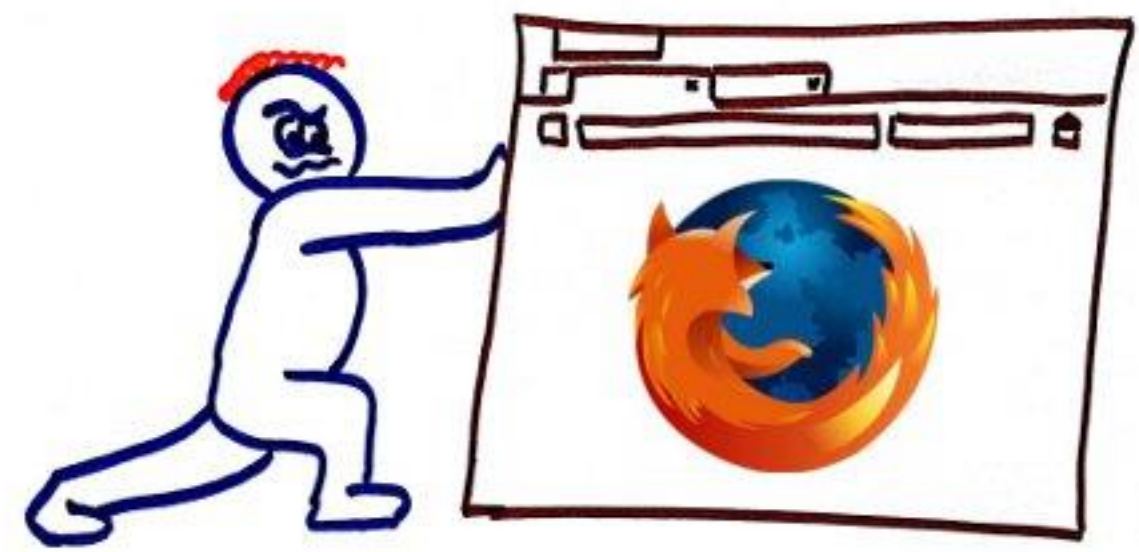


- Client requests data – server responds

The problems with HTTP

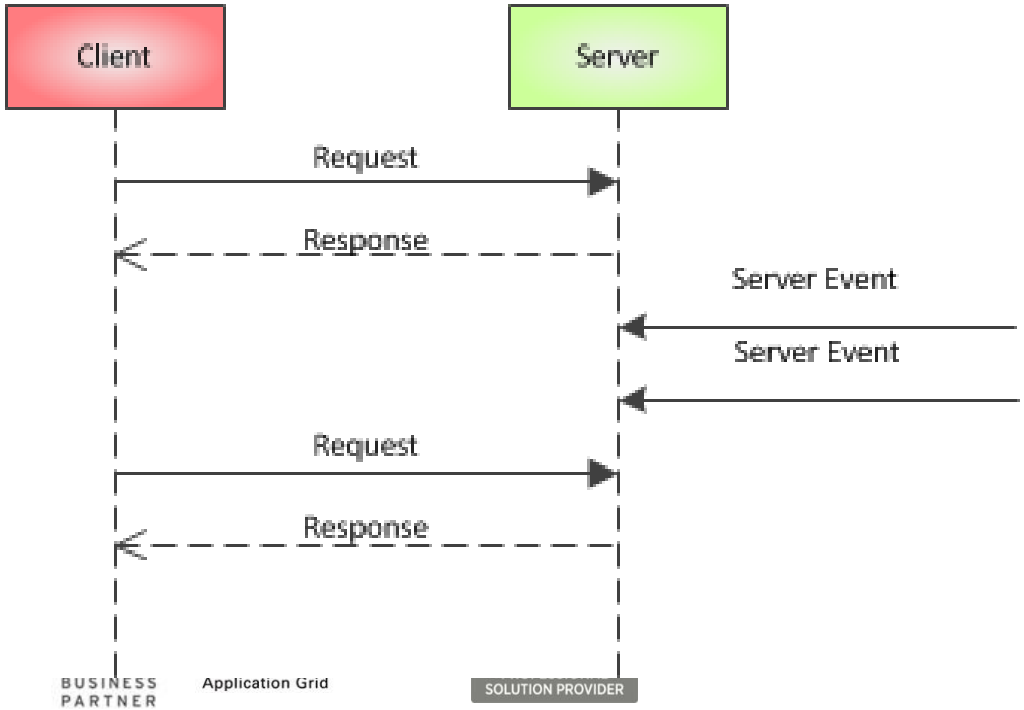
- HTTP is a request-response protocol
- HTTP is half-duplex – one way traffic
- HTTP is stateless – lots of redundant data
- New connection required for each transaction

Push to browser



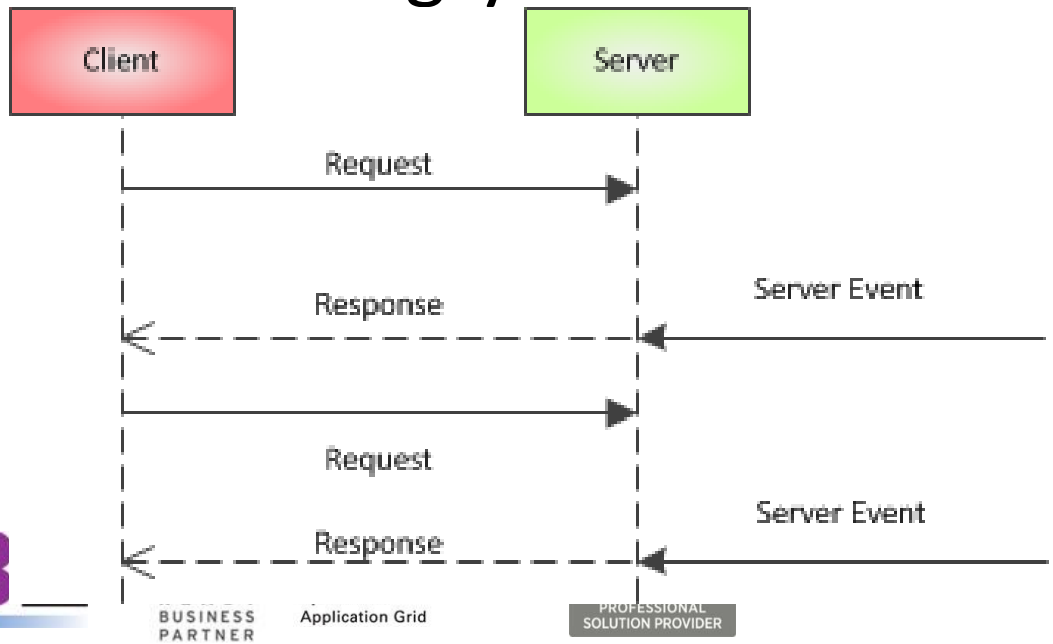
Simulated Push – HTTP Polling

- Regular requests at a set interval
- Near real-time
- Server events may occur between requests



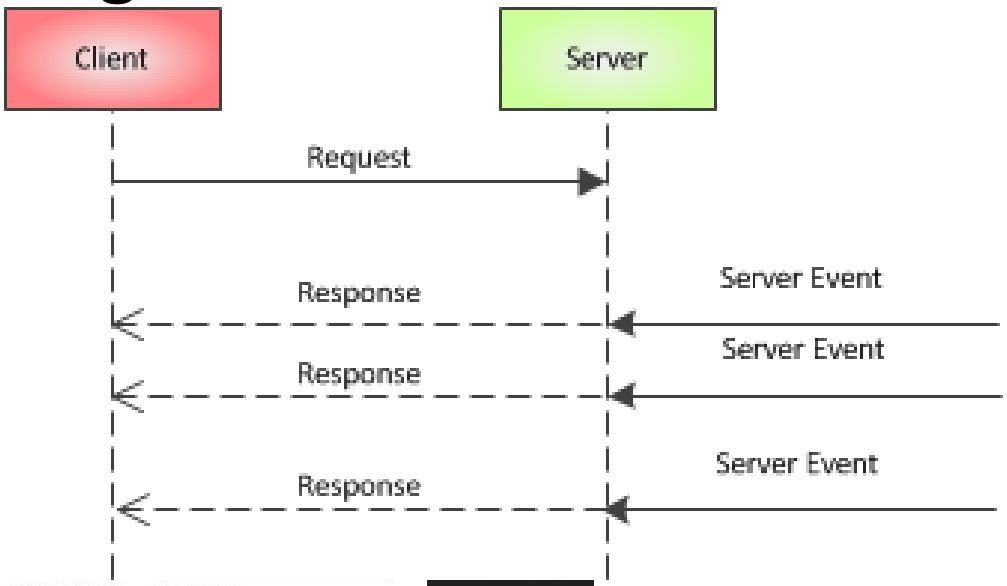
Simulated Push – Long polling

- Connection is kept open
- Response is blocked until an event occurs or a timeout occurs
- Resource hungry on the server



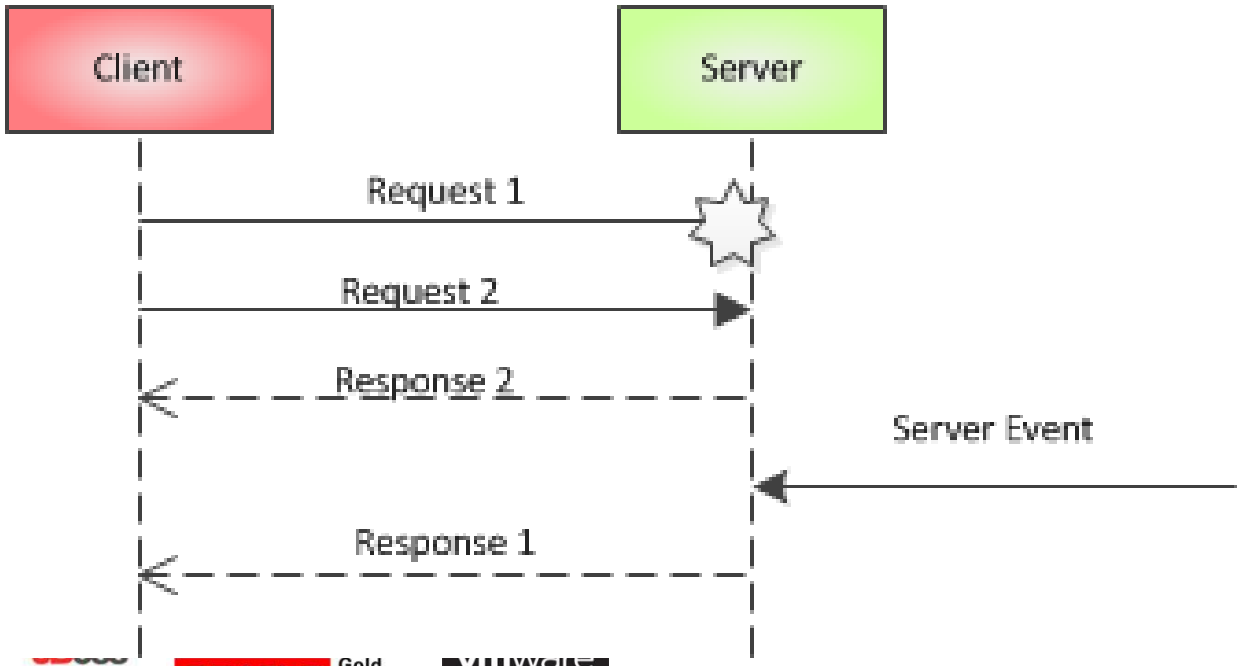
HTTP Streaming

- Long lived HTTP connection
- Or XMLHttpRequest connection
- Browser needs to close and reconnect the streaming channel to release memory



Reverse AJAX/Comet

- Utilises long polling or HTTP Streaming techniques
- Complex development
- Poor scalability



The Future - Web Sockets

- New to HTML 5
- Enables Full Duplex communication between a browser and a Server
- Allows Web Servers to push updates to browsers
- Better than “long polling”
- Establishes a Dedicated Socket to the Backend Web Socket Server

Web Socket standards

- **This is all in flux**
- Rfc 6455 defines the protocol
- W3C SSE http://dev.w3.org/html5/eventsource/
- W3C WebSockets http://dev.w3.org/html5/websockets/

Browser Support

- Chrome 4+
- Internet Explorer 10+
- Firefox 4+
- Opera 10.7+
- Safari 5+

Benefits over old techniques

- Reduced latency
- Reduced network traffic
- Reduced CPU/memory usage on the server
- Scalable
- Simplified development

Web Socket Protocol

- **Client**

```
•GET /chat HTTP/1.1
•Host: server.example.com
Upgrade: websocket
•Connection: Upgrade
•Sec-WebSocket-Key:
dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
•Sec-WebSocket-Protocol:
chat, superchat
•Sec-WebSocket-Version: 13
```

- **Server**

```
•HTTP/1.1 101 Switching
•Protocols Upgrade:
websocket Connection:
Upgrade Sec-WebSocket-
Accept:
s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
•Sec-WebSocket-Protocol:
chat
```

From the RFC

•Conceptually, WebSocket is really just a layer on top of TCP that does the following:

- adds a web origin-based security model for browsers
- adds an addressing and protocol naming mechanism to support multiple services on one port and multiple host names on one IP address
- layers a framing mechanism on top of TCP to get back to the IP packet mechanism that TCP is built on, but without length limits
- includes an additional closing handshake in-band that is designed to work in the presence of proxies and other intermediaries

JavaScript API

- **Web Socket**

- WebSocket(location,protocol)

- Function onmessage
- Function onopen
- Function onclose
- Function onerror
- close()
- send(data)

- **Server Sent Events**

- EventSource(location)

- Function onmessage
- Function onopen
- Function onerror

W3C Definition

```
[Constructor(in DOMString url, in optional DOMString protocol)]
interface WebSocket {
  readonly attribute DOMString URL;

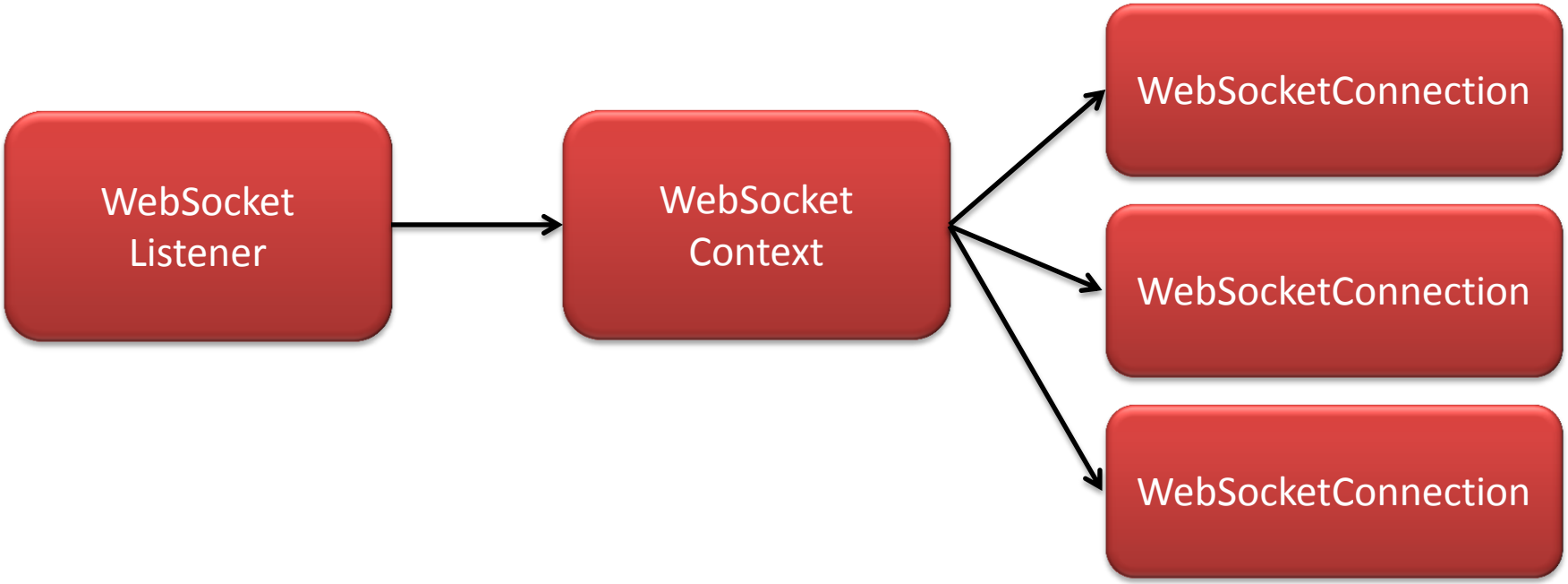
  // ready state
  const unsigned short CONNECTING = 0;
  const unsigned short OPEN = 1;
  const unsigned short CLOSED = 2;
  readonly attribute unsigned short readyState;
  readonly attribute unsigned long bufferedAmount;

  // networking
           attribute Function onopen;
           attribute Function onmessage;
           attribute Function onclose;
  boolean send(in DOMString data);
  void close();
};
WebSocket implements EventTarget;
```

WebSockets in WebLogic

WebLogic Key Classes

- Note this is only in WebLogic 12c



WebLogic Annotations

```
@WebSocket (
        pathPatterns = {"/chat/*"},
        dispatchPolicy = "ChatWorkManager",
         timeout = 30,
        maxConnections = 1000 )
public class MyListener implements
                                WebSocketListener { ... }
```

JSR 356

- ```
  @ServerEndpoint(path="/echo")
  public class EchoBean {

  @OnMessage
  public String echo(String message) {
  return message + " (from your server)";
    }
  }
  ```
  ```

Oracle Coherence
Real Time Push @ Scale?

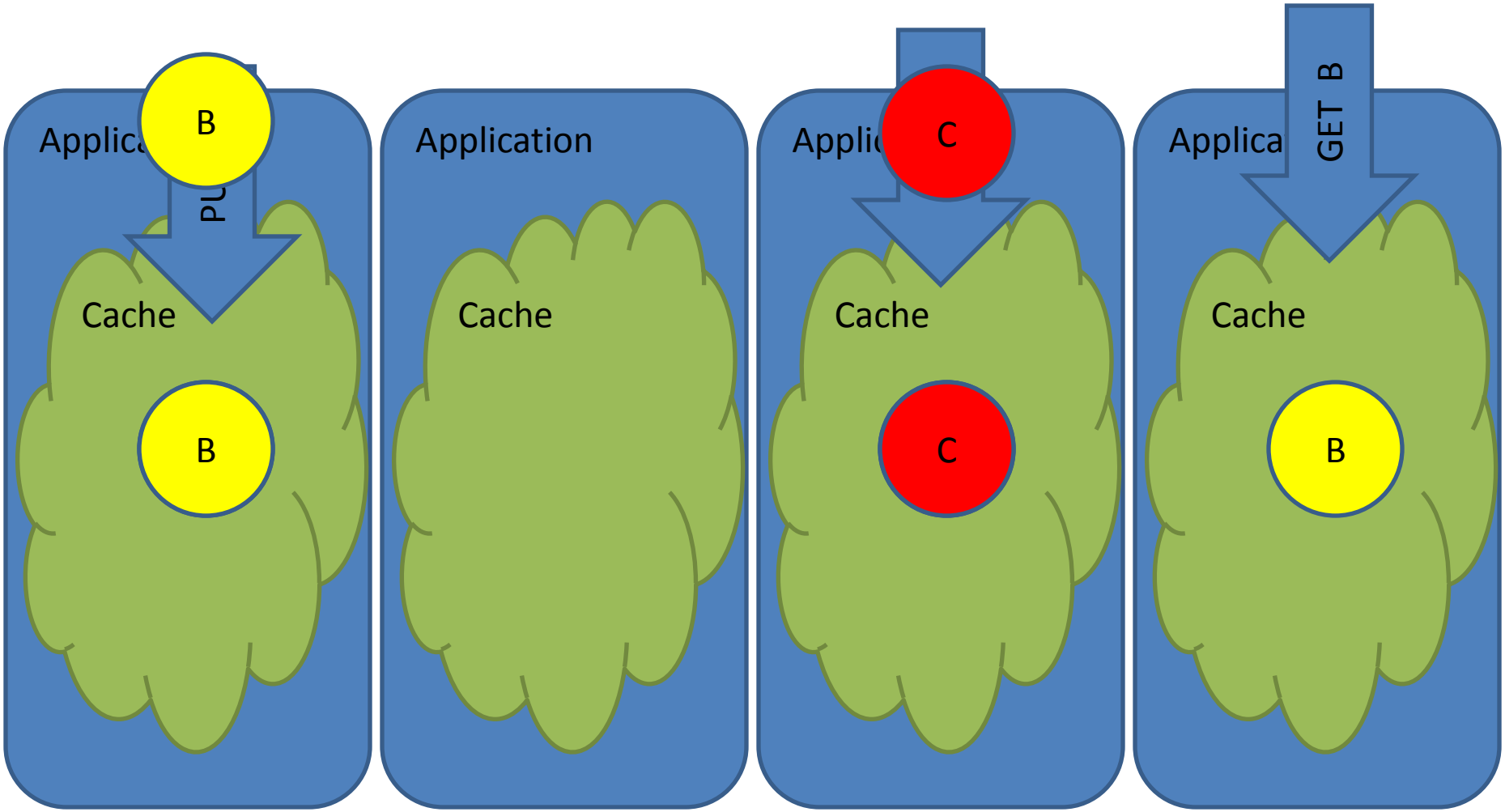
JCache

- JCache standard api for caches
- JSR 107 (Coming in JEE7)
- Provides Map Semantics for Cache
  - put(Key,Object)
  - Object get(Key)
- Most Caches support this api
- Coherence supports the api

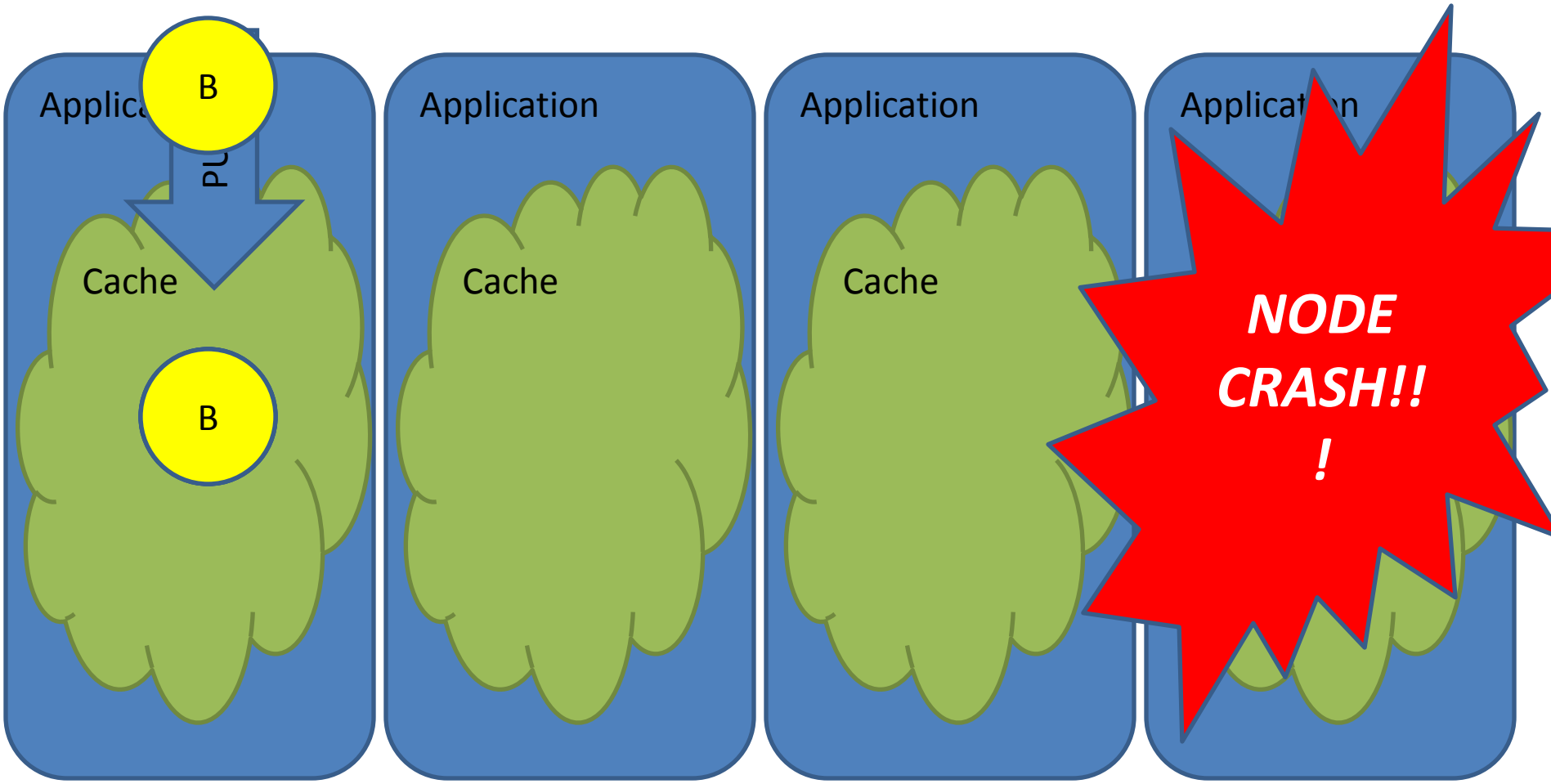
Data Cache

- Data Caches aren't New
  - Hibernate Session Cache
  - Entity Bean Cache
  - JPA Cache
  - Custom Caches
  - Open Source Caches
- Typically Cache Database Data
- Data Grids are Cache on Steroids!

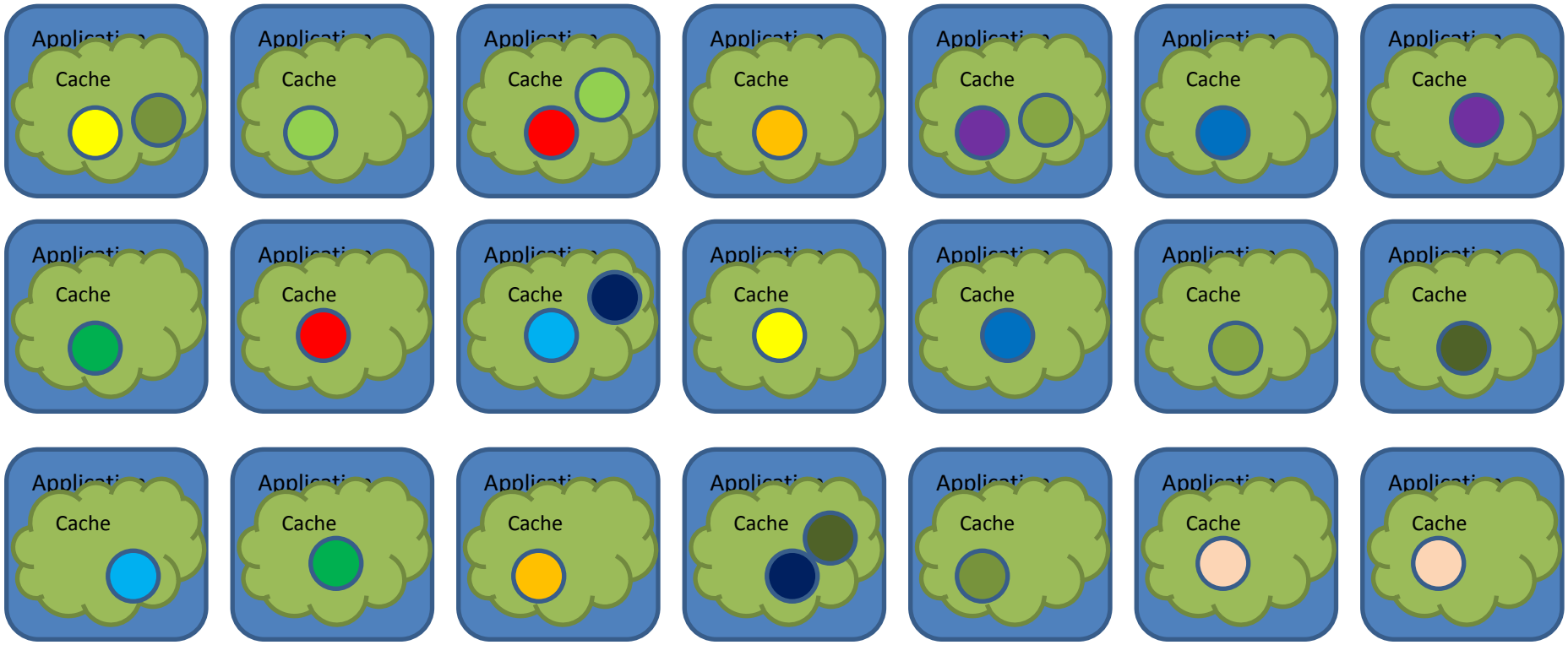
Cache Partitioning



HA Cache Partitioning



Typical Coherence Grid



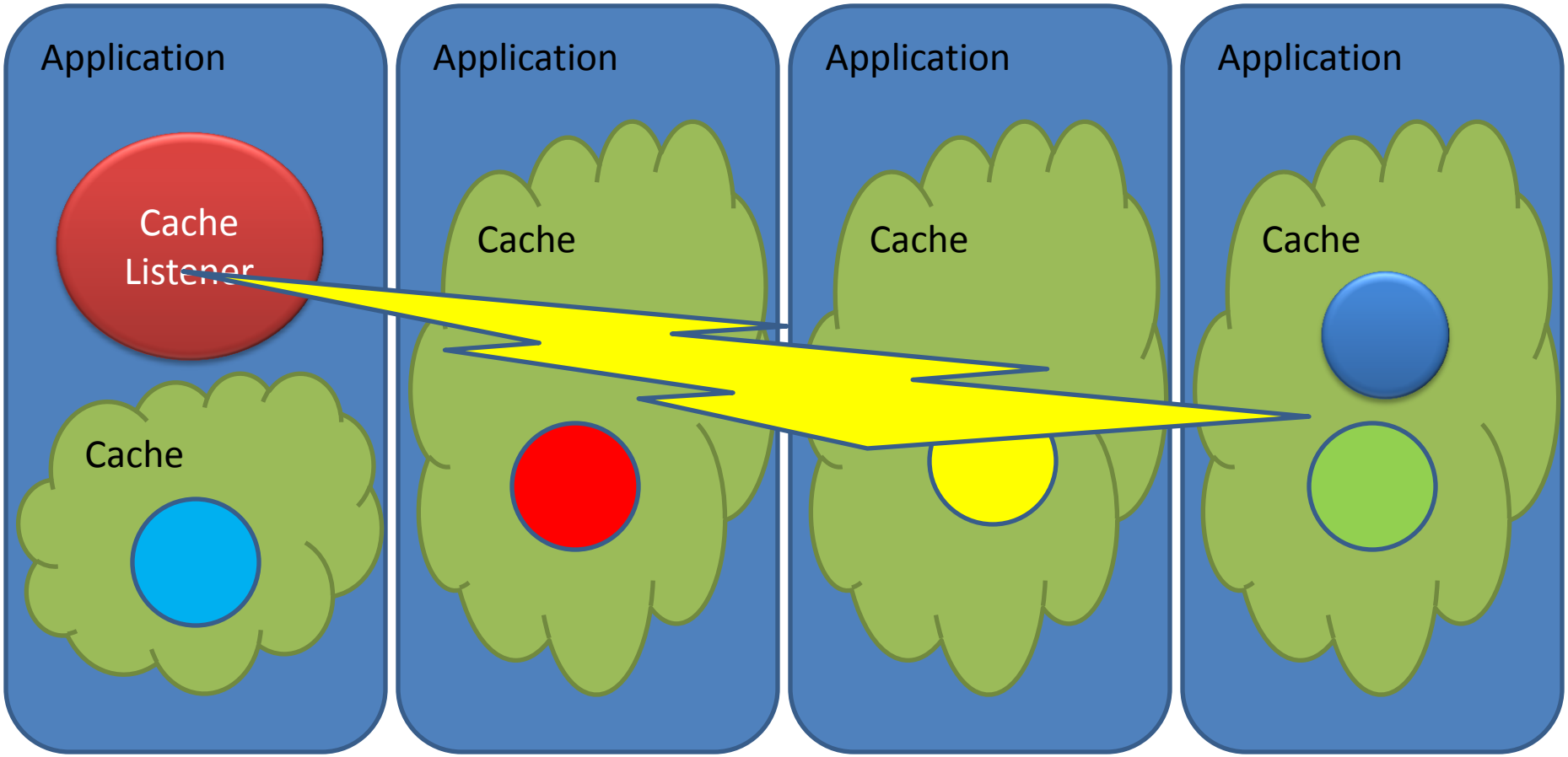
Coherence Partitioned Cache

- Linear Scalability
  - 2 hops for Read (Worst Case)
  - 2 hops for Write (Worst Case)
- High Availability
  - Configurable Duplicates
- Location Independent Access
  - Grid knows where data is
- More Nodes = More Data in Memory

Coherence Key Features

- Elasticity
- Grow and Shrink the Cluster
- Grid Execution
- Push code around the cluster
- Grid Queries
- Near Cache
- Continuous Query Cache
- Many Caching Strategies
-

Grid Events Subsystem



Coherence Event Listeners

- Coherence classes MapEvent, MapListener
- Each time a record is changed, Coherence raises a MapEvent
- Override entryUpdated, entryInserted and entryDeleted to process these events
- MapListeners can have filters

Events

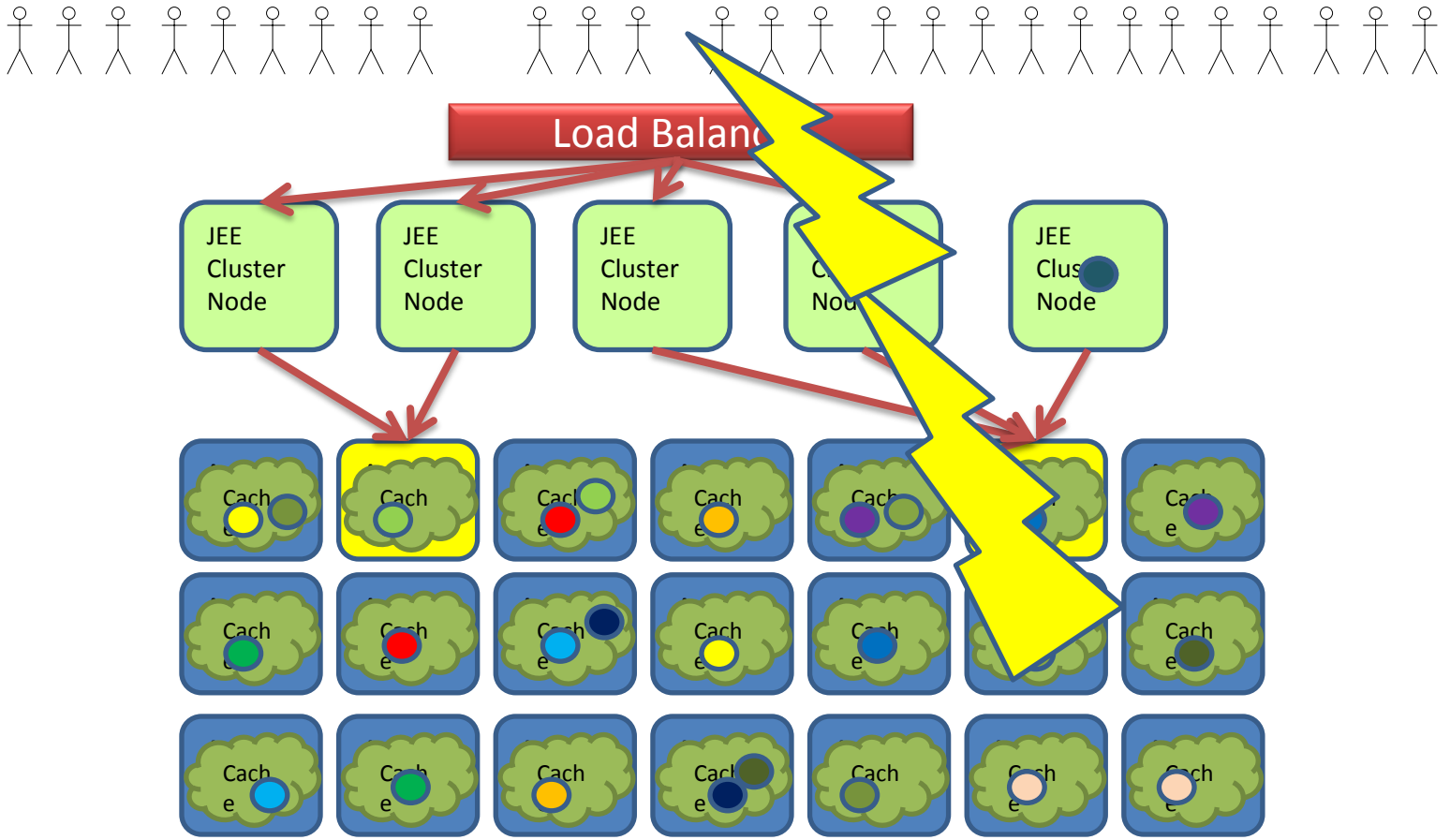
- E_ALL
  - All Events
- E_INSERTED
  - Inserts
- E_DELETED
  - Deletes
- E_UPDATED
  - Updated
- E_UPDATED_ENTERED
  - Updated and now matched
- E_UPDATED_LEFT
  - Updated and now not matched
- E_UPDATED_WITHIN
  - Updated still matched
- E_KEYSET
  - All updated which change the matching set

JSR 107 API

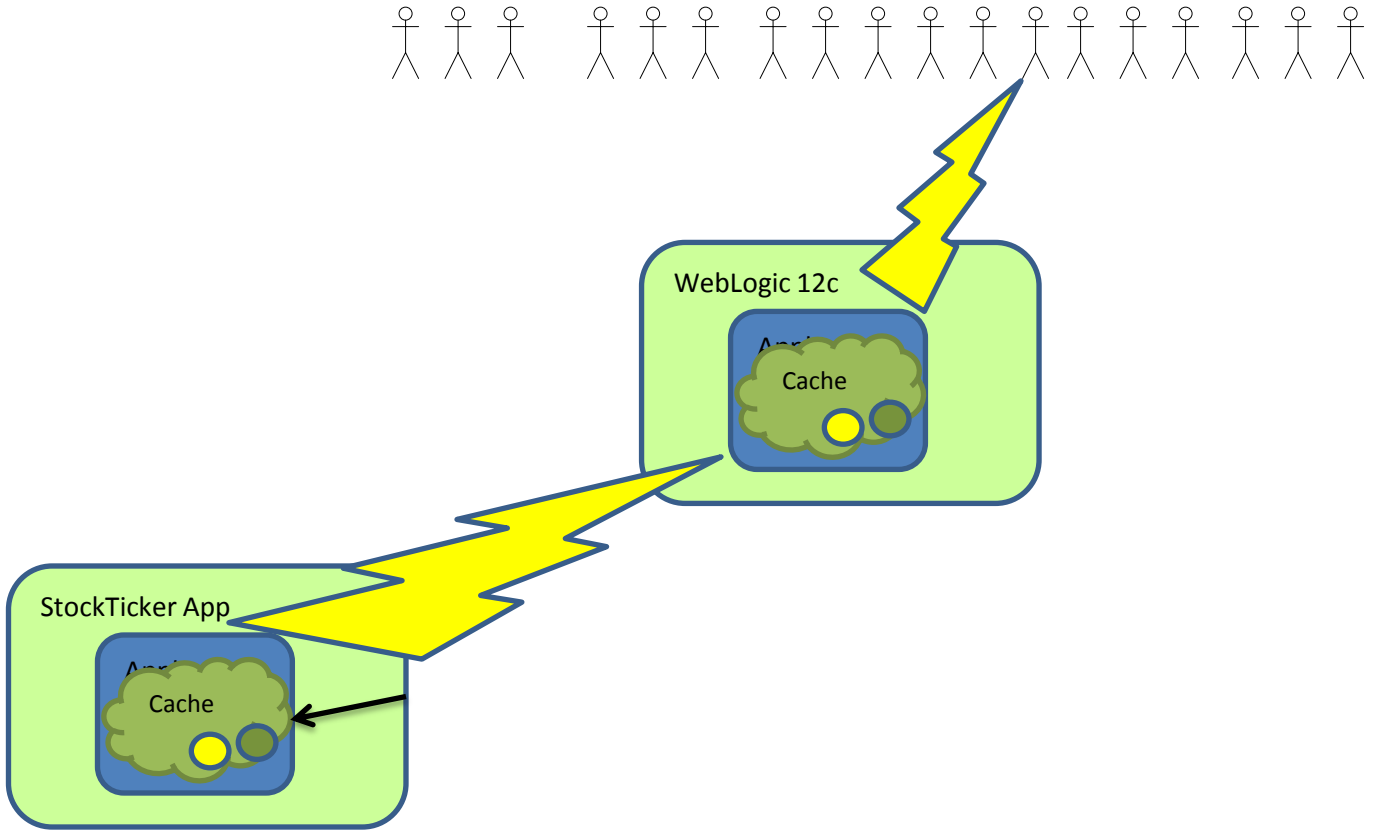
```
•boolean registerCacheEntryListener(
•   CacheEntryListener cacheEntryListener);
```

- Fired on
  - Expired
  - Removed
  - Updated
  - Read

Best Practice Architecture



Stock Ticker Architecture

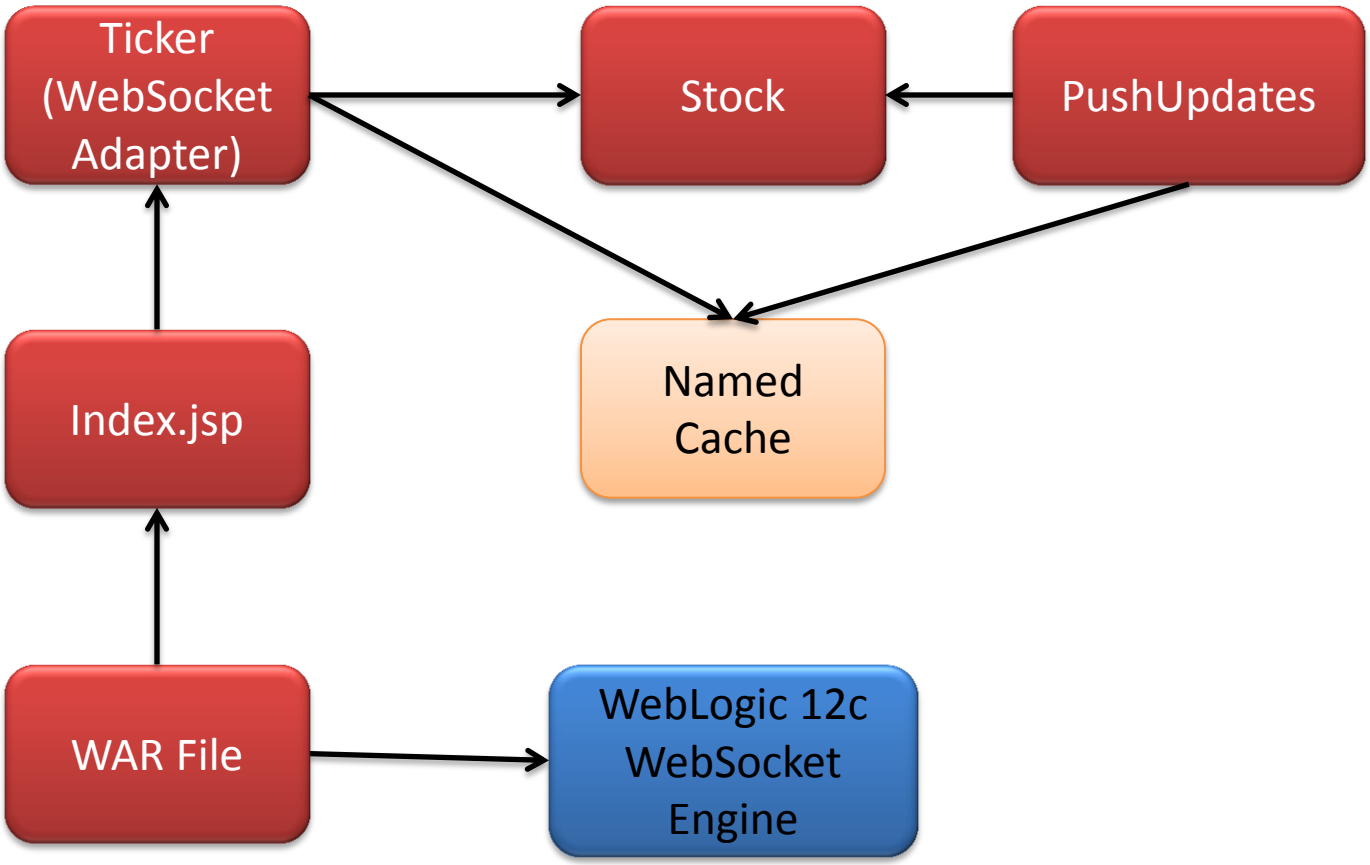


Demo

http://demo.c2b2.co.uk:7080

Follow the Link

Stock Ticker Classes



PushUpdates

```
public static void main(String args[]) throws Exception {

        // attach to Coherence
        CacheFactory.ensureCluster();
        NamedCache cache = CacheFactory.getCache("Stocks");

        // create a random stock and stick it in Coherence
        while(true) {
            Stock stock = new Stock("C2B2","C2B2",Math.random() * 100.0);
            cache.put("C2B2", stock);
             int sleepTime = (int)(500*Math.random() + 500);
            Thread.currentThread().sleep(sleepTime);
        }
    }
```

Ticker

```
@WebSocket (
        pathPatterns = {"graph/*"},
        timeout = 3600
)
public class Ticker extends WebSocketAdapter implements MapListener
{
    public Ticker() {
        try {
            jaxb = JAXBContext.newInstance(Stock.class);
          } catch (JAXBException ex) {
            Logger.getLogger(Ticker.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
```

Ticker init

```
@Override
  public void init(WebSocketContext context) {
     super.init(context);
     CacheFactory.ensureCluster();
     cache = CacheFactory.getCache("Stocks");
     System.out.println("Got Cache " + cache);
     cache.addMapListener(this);
  }
```

Ticker entryUpdated

```
public void entryUpdated(MapEvent me) {
        // marshall to JSON
       Stock stock = (Stock) me.getNewValue();
        Marshaller m = jaxb.createMarshaller();
        m.setProperty("eclipselink.media-type", "application/json");
        StringWriter sw = new StringWriter(30);
        m.marshal(stock, sw);

        // now write to socket
        Set<WebSocketConnection> conns =
                getWebSocketContext().getWebSocketConnections();
        for (WebSocketConnection conn : conns) {
                conn.send(sw.toString());
          }
       }
```

index.jsp

```
var chart;
document.chart = new Highcharts.Chart({
                ….
})
websocket.onmessage = function(event) {
    var object = JSON.parse(event.data);
    var x = (new Date()).getTime();
    var y = object.price;

document.chart.series[0].addPoint([x,y],true,true,false);
  }
```

Summary

- Reduced latency, network traffic and CPU/memory usage on the server
- Highly scalable
- When linked to Coherence, provide enterprise scale, event driven, real time push