JAX-RS.next

Michal Gajdos
michal.gajdos@oracle.com

January, 2015

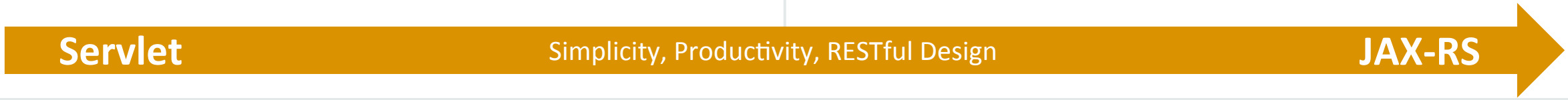
Safe Harbor Statement

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.

HTTP in Java EE: From Servlet to JAX-RS

- Generic
  - Protocol-independent
  - In retrospect, most important Java EE technology
    - But brought HTTP to Java EE
- Old-school Programming Model
  - Static, Interface-driven
  - Singleton-Scoped
  - Lot's of distractions / boilerplate

- Laser-focused on REST Services
  - HTTP Centric
    - method matching, content negotiation, …
  - Payload format independence
    - decoupled from business logic
- Modern Programming Model
  - Dynamic, POJO-based, Annotation-driven
  - Request-Scoped
  - No distractions / boilerplate



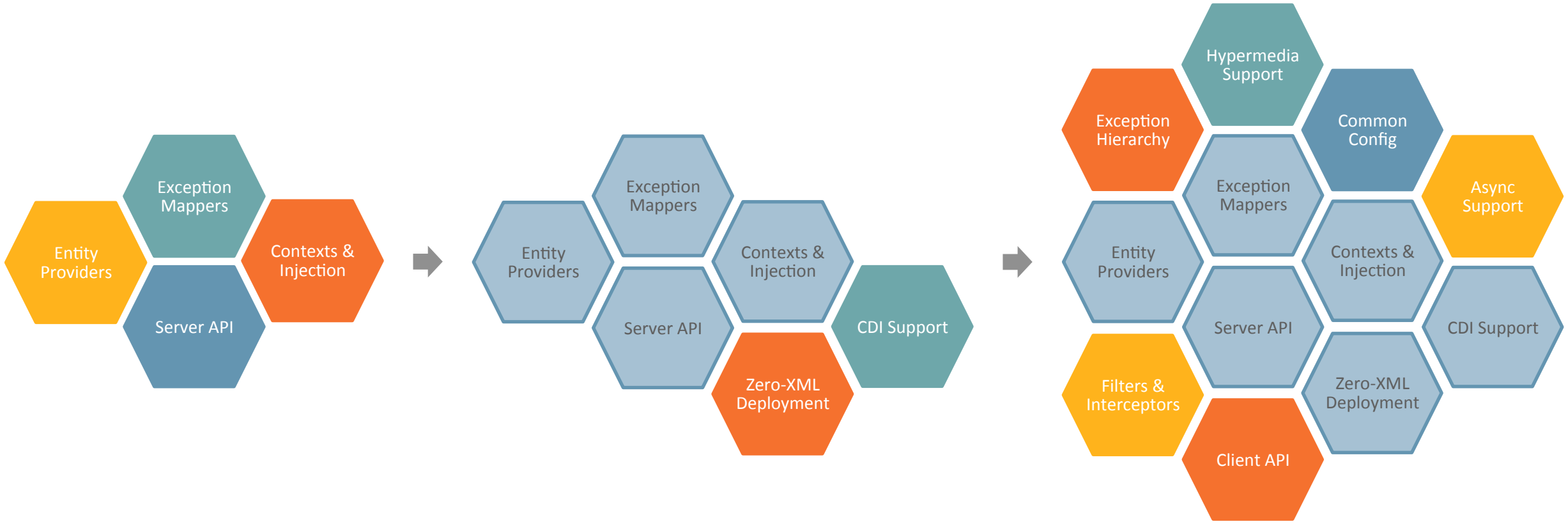
HTTP in Java EE: From Servlet to JAX-RS

```
public class MyServlet extends HttpServlet {
  protected void doGet(HttpServletRequest req, HttpServletResponse res) {
        if (!acceptsTextPlain(req)) {
             res.sendError(406); // Not Acceptable
             return;
        }
        if (!isGreetingPath(req)) {
            res.sendError(404); // Not Found
            return;
        }
        String name = req.getParameter("name");
        if (name == null) { name = "Joe"; }
        PrintWriter pw = res.getWriter();
        pw.print("Hi " + name + "!");
        pw.flush();
    }
    … // TODO: acceptsTextPlain(…) & isGreetingPath(…) implementations
}
```

```
@Path("greeting")
public class MyResource {
    @GET @Produces("text/plain")
    public String greet(@QueryParam("name") @DefaultValue("Joe") String name) {
        return "Hi " + name + "!";
    }
}
```

Servlet Simplicity, Productivity, RESTful Design **JAX-RS**

Evolution of JAX-RS API



JAX-RS 1.0 **JAX-RS 1.1** **JAX-RS 2.0**

JSR-370 – JAX-RS.next

- Performance
  - Reactive Programming Model
  - Java SE 8 Streams
  - Non-blocking I/O
- Java EE Alignment
  - CDI Alignment
  - Declarative Security Model
  - MVC 1.0 & JSONB 1.0 Integration
- Filling the Gaps
  - Server-Sent Events
  - Improved Hypermedia Support

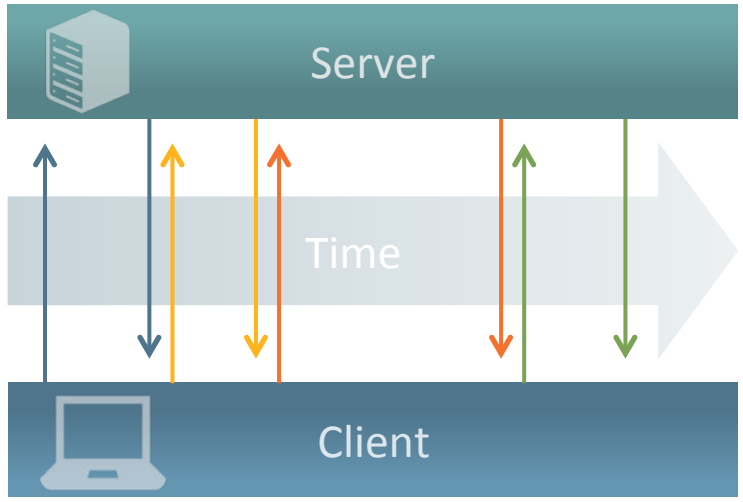
- Continued Evolution
  - All simple problems have already been solved…
- Targeted to Ship with Java EE 8

Program Agenda

1. Server-Sent Events
2. Non-Blocking I/O
3. Declarative Security
4. MVC Integration
5. CDI Alignment, JSON-B, …

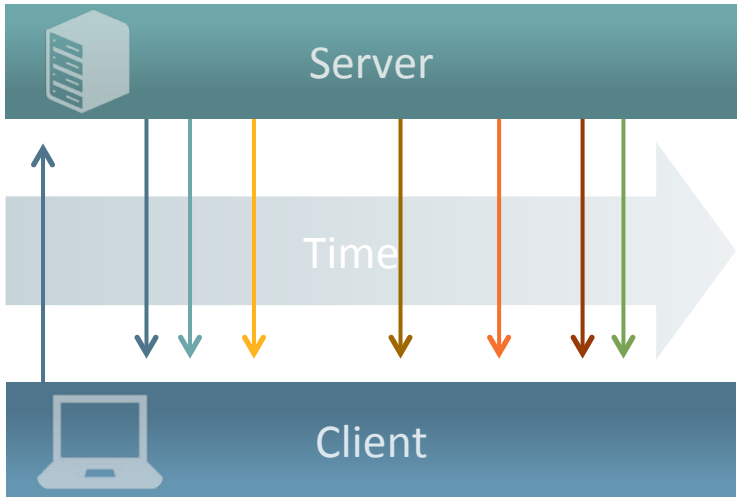
Ajax Long-Polling vs. Server-Sent Events vs. WebSocket

Ajax Long-Polling



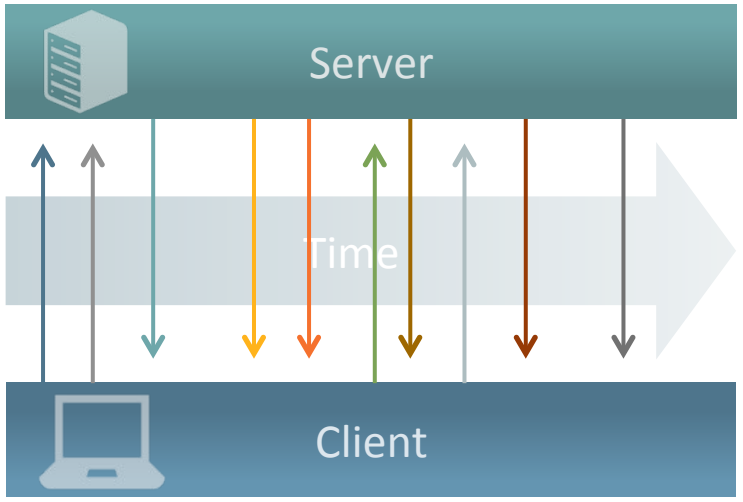
- Plain old HTTP
- Manual reconnect
- Undefined format

Server-Sent Events



- Plain old HTTP
- Seamless reconnect & redelivery
- HTML 5 standard protocol

WebSocket



- New Protocol (via HTTP upgrade)
- Full-duplex
- HTML 5 standard protocol

Why Server-Sent Events?

- Streaming asynchronous events from Server to Client

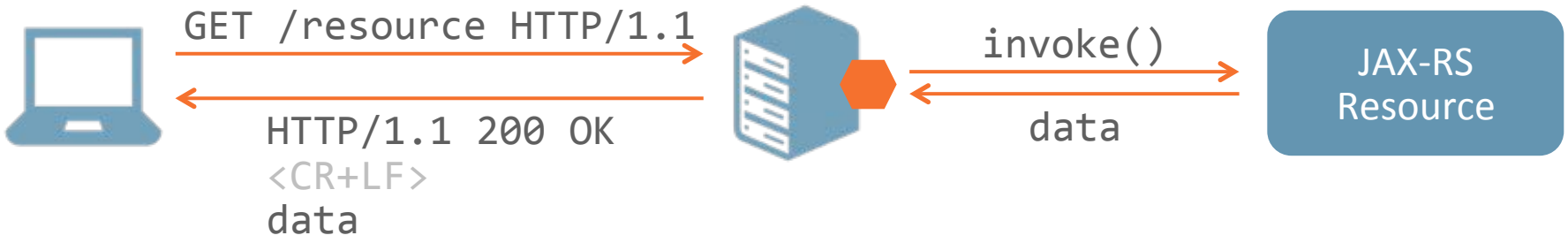
- Fits well Ajax Long-Polling Use Cases
  - Progress for long-running tasks
  - Stream of real-time data updates (e.g. Stock Ticker, Monitoring)
  - Instant server state change notifications (e.g. SysAdmin Message Push)
  - Task processing distribution
  - …

- Part of HTML5 Standard by W3C
  - Built-in support in all modern browsers
  - Standard event message format

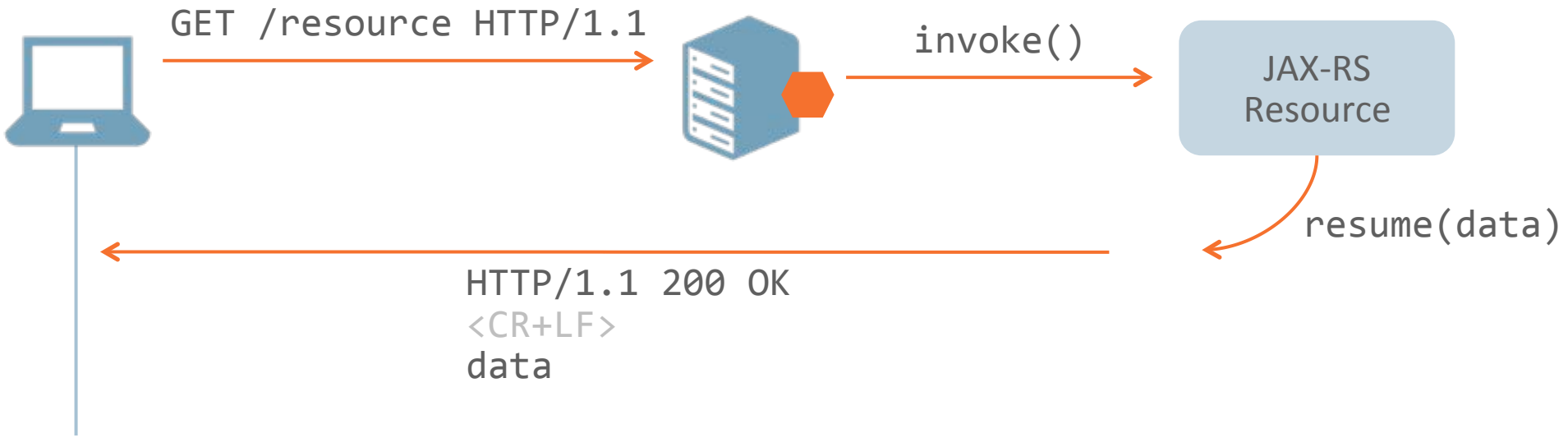
An SSE Event Message

```
: Comment lines start with a ‘:’ prefix
: Events can have numeric IDs…
id: 1234
: …can contain reconnect delay instructions…
retry: 5000
: …can be named…
event: text-message
: …typically contain one or more lines of event payload data.
data: Hello, this is a
data: multi-line message.
: Events are separated from each other by a blank line.
<blank line>
```

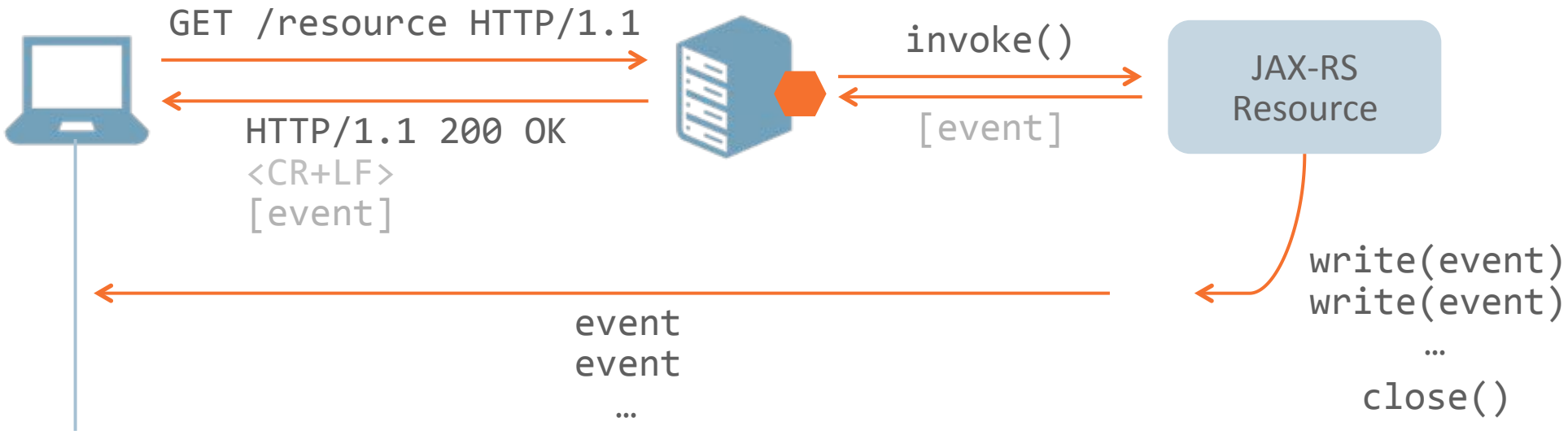
Typical HTTP Request / Response Flow



Asynchronous HTTP Request / Response Flow



Server-Sent Events Flow



Jersey Server-Sent Events API

Producer API – Server side

`OutboundEvent`

- single event; outgoing

`EventOutput`

- single client connection; outbound

`SseBroadcaster`

- connection aggregation
- `BroadcasterListener`
  - broadcast notification

Consumer API – Client side

`InboundEvent`

- single event; incoming

`EventInput`

- streaming connection; inbound

`EventSource`

- IoC connection; inbound
- `EventListener`
  - asynchronous event processing

Server-side – Connecting to an Event Source

```
@Path("messages")
@Produces(APPLICATION_JSON)
public class MessageBoardResource {

    private static SseBroadcaster broadcaster = new SseBroadcaster();

    @GET @Path("stream")
    @Produces(SseFeature.SERVER_SENT_EVENTS)
    public EventOutput connect() {

        EventOutput eventOutput = new EventOutput();
        broadcaster.add(eventOutput);
        return eventOutput;

    }

    …
```

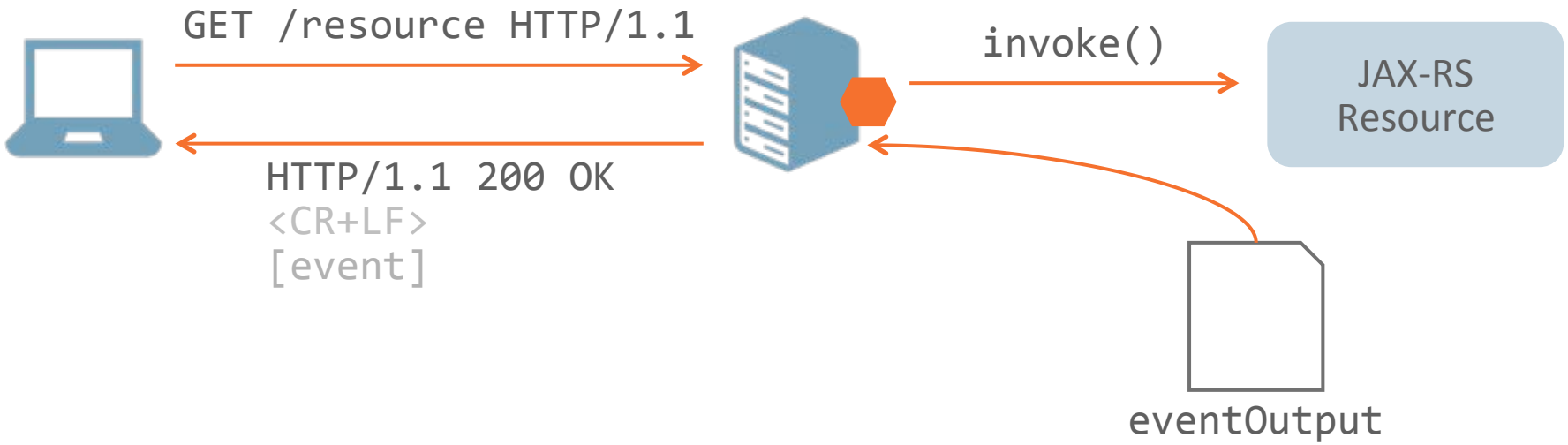
Create Broadcaster

Create Outbound Connection

Store Outbound Connection

Return Outbound Connection

Server-Sent Events Flow



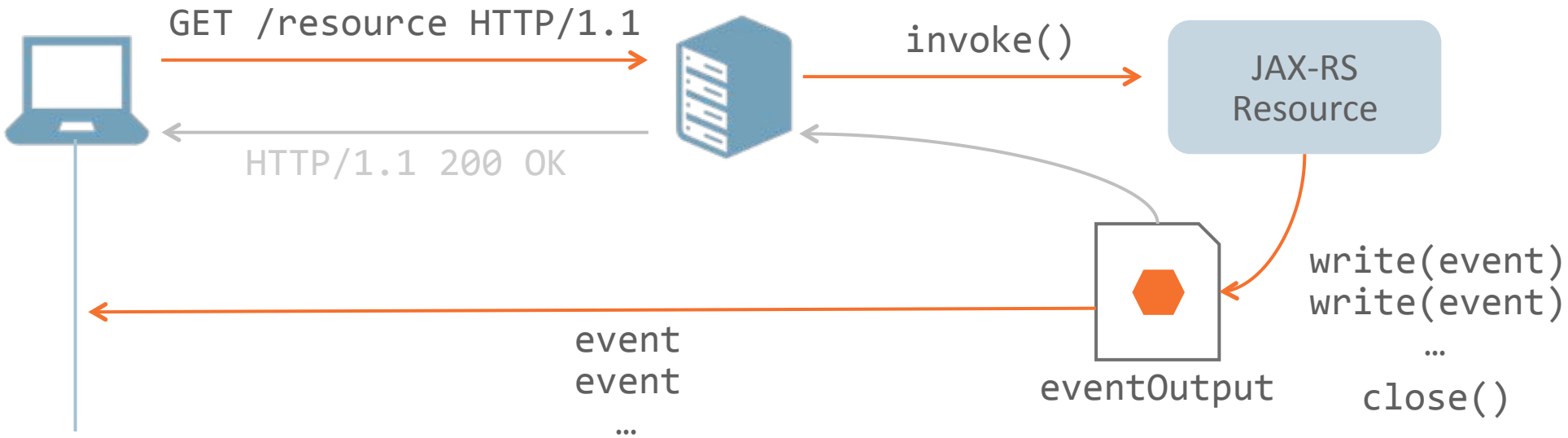
Server-side – Dispatching Events

```
    …

    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    public String postMessage(Message message) {

        OutboundEvent event = new OutboundEvent.Builder()
                .id(getNextId())
                .mediaType(MediaType.APPLICATION_JSON_TYPE)
                .data(Message.class, message)
                .build();


        broadcaster.broadcast(event); // invokes eventOutput.write(event);

        return "Message posted!"
    }
}
```

Create Outbound Event

Send Outbound Event

Server-Sent Events Flow



Client-side Event Processing – IoC Model

```
EventSource eventSource =
    new EventSource(target.path("messages/stream")) {

        @Override
        public void onEvent(InboundEvent event) {

            String name = event.getName();
            Message message = event.readData(Message.class);

            display(name, message);
        }
    };

…

eventSource.close();
```

Create Event Source

Implement Event Callback

Process Inbound Event

Close Event Source

Client-side Event Processing – Pull Model

```
EventInput events = target.path("messages/stream")
                          .request().get(EventInput.class);

while (!stop) {

    InboundEvent event = events.read();
    String name = event.getName();
    Message message = event.readData(Message.class);

    display(name, message);
}

events.close();
```

Retrieve Event Stream

Implement Event Loop

Process Inbound Event

Close Event Source

Program Agenda

1. Server-Sent Events
2. Non-Blocking I/O
3. Declarative Security
4. MVC Integration
5. CDI Alignment, JSON-B, …

Servlet 3.1 Non-Blocking I/O API

- Receiving data
  - `ServletInputStream`
    - `isReady()`, `isFinished()`, `setReadListener(ReadListener)`
  - `ReadListener`
    - `onDataAvailable()`, `onError(Throwable)`, `onAllDataRead()`

- Sending data
  - `ServletOutputStream`
    - `isReady()`, `setWriteListener(WriteListener)`
  - `WriteListener`
    - `onWritePossible()`, `onError(Throwable)`

Non-Blocking I/O API in JAX-RS

- Main Constraints and Goals
  - Backward-compatibility
  - Alignment with Servlet 3.1 Non-Blocking I/O API
    - At least conceptually
  - Support for standalone JAX-RS deployments
    - no hard dependency on Servlet API

- Problems to solve
  - Unclear impact on request/response processing chain
  - Multiple extension points to cover (in a backward-compatible way)
    - Filters, Interceptors, Entity Providers, Exception Mappers, Parameter Converters
    - Potential interoperability issues in mixed models
      - e.g. blocking and non-blocking request filter in the same processing chain

Program Agenda

1. Server-Sent Events
2. Non-Blocking I/O
3. Declarative Security
4. MVC Integration
5. CDI Alignment, JSON-B, …

Securing Resources in JAX-RS

- New Security JSR planned for Java EE 8
  - Focused on unification, ease of use and portability improvements
  - JAX-RS EG will monitor progress of this JSR
- OAuth 2.0 gaining a lot of adoption from many big players
  - Google, Facebook, Twitter, …
- Vision for REST Services Security Model in Java EE
  - Easy and Straightforward
  - Declarative whenever possible
    - Java EE Security Annotations fit JAX-RS very well
    - Jersey already provides annotation-based authorization

Existing Jersey Security Features

- Leveraging standard JAX-RS extension points
- Off-the-shelf modules & components ready to use

- Client-side
  - HTTP Basic & Digest Authentication Support
- Server-side
  - Declarative Role-base Authorization (`javax.annotation.security`)
    - `@RolesAllowed`, `@PermitAll`, `@DenyAll`

- OAuth 1.0, 2.0 Support
  - Client and Server Side

Authorization in Jersey

```
@Path("messages")
@Produces(APPLICATION_JSON)
@DenyAll
public class MessageBoardResource {
    …

    @GET @Path("stream")
    @Produces(SseFeature.SERVER_SENT_EVENTS)
    @PermitAll
    public EventOutput connect() { … }

    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    @RolesAllowed("publisher")
    public String postMessage(Message message) { … }
}
```

No access by default

Public access

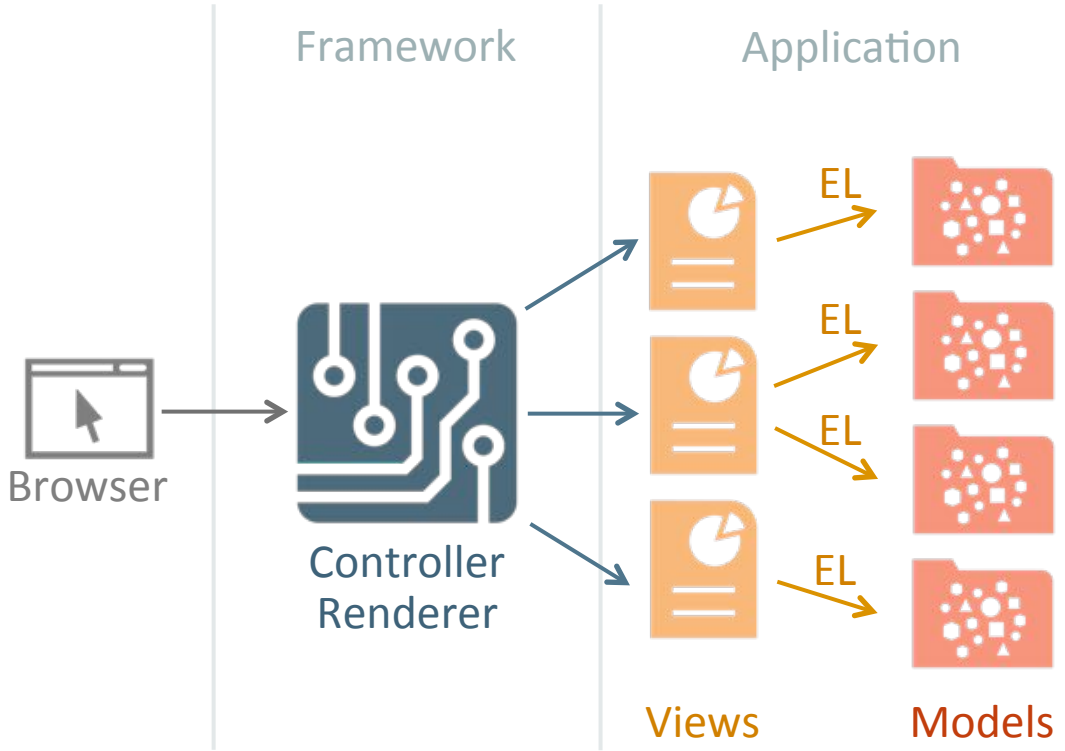
Only "publisher" role

Program Agenda

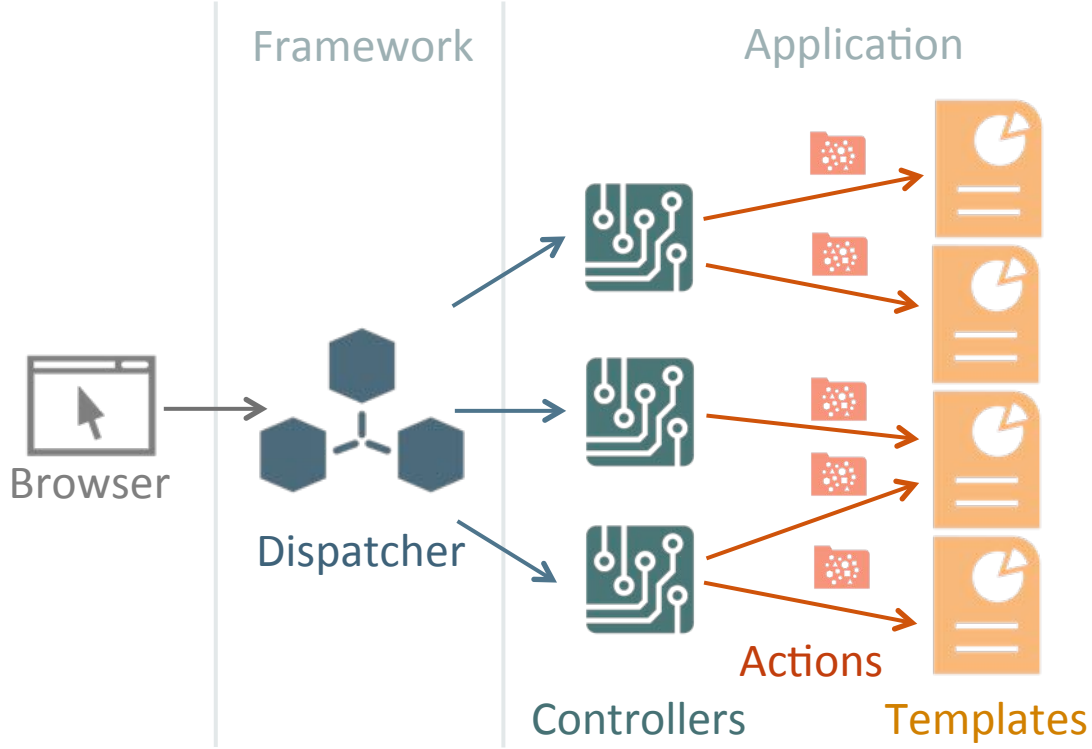
1. Server-Sent Events
2. Non-Blocking I/O
3. Declarative Security
4. MVC Integration
5. CDI Alignment, JSON-B, …

Different Tastes of MVC Frameworks

UI Component-Oriented



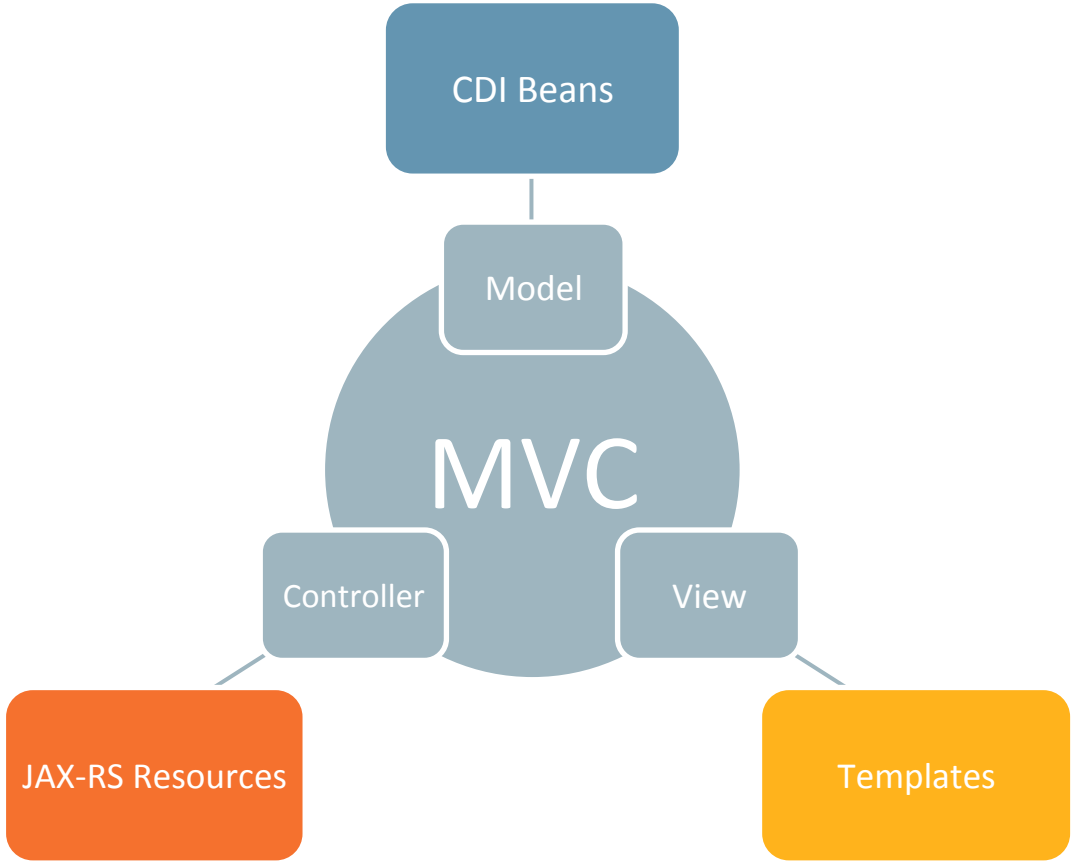
Action-Oriented



MVC and JAX-RS

- New, dedicated JSR 371 – MVC 1.0
  - Focus on action-oriented Framework
  - Leverage existing templating engines (JSP, Facelets)
  - Integrate with JAX-RS

- Model
  - Context, Injection and Bean Validation support
  - CDI Beans
- View
  - Format content based on Model data
  - Templates (JSP, Facelets, FreeMarker, Mustache, …)
- Controller
  - Dispatch model data to View templates
  - JAX-RS Resources
    - Native content-negotiation support



Jersey MVC Support

- Jersey extension modules
  - org.glassfish.jersey.ext
- Core Module (jersey-mvc)
  - View Templates
    - Producing Human-Readable Content
    - Convenient Generators for new Media Types
- Integration Modules
  - Templating Engines
    - JSP, FreeMarker, Moustache
  - BeanValidation & Error Reporting
- Minimalistic Public APIs
  - Programmatic
    - `Viewable`
  - Declarative
    - `@Template`
    - `@ErrorTemplate`

Jersey MVC Templates

```
@Path("motorcycle/{name}")
public class MotorcycleResource {

    @Inject @NotNull Catalog<Motorcycle> catalog;

    @GET @Produces(TEXT_HTML)
    @Template(name="motorcycle.jsp")
    public Motorcycle getHtml(@PathParam("name") String name) {
        return catalog.getModel(name);
    }

    @Path("parts")
    public PartsListResource getParts(@PathParam("name") String name) {
        return new PartsListResource(catalog, name);
    }
}
```

Controller

Model

View

Sub-controller

Jersey MVC Templates (contd.)

```
public class PartsListResource {

    private final String name;
    private final Catalog catalog;
    …
    @POST @Consumes(APPLICATION_JSON)
    public String addPartJson(Part part) {
        catalog.getModel(name).addPart(part);
        return “Part added.”;
    }

    @POST @Consumes(APPLICATION_FORM_URLENCODED)
    @Template(name=“confirmation.jsp”)
    public String addPartForm(@BeanParam PartFormBean bean) {
        catalog.getModel(name).addPart(bean.toPart());
        return “Part added.”;
    }
}
```

Sub-Controller

Non-browser Clients

Browser Clients View

Program Agenda

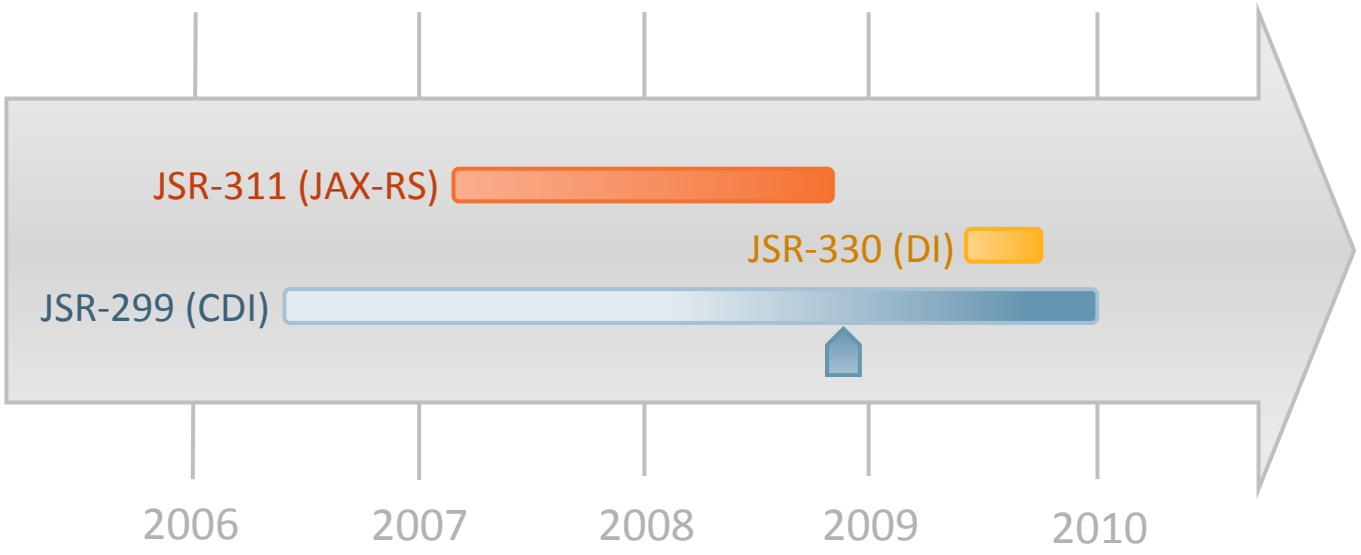
1. Server-Sent Events
2. Non-Blocking I/O
3. Declarative Security
4. MVC Integration
5. CDI Alignment, JSON-B, …

JSR-367 – Java API for JSON Binding

- API with a Feel Similar to JAXB
  - Inspired by existing popular frameworks
- Goals
  - JSON
  - Runtime API
  - Object mapping (Default | Customizable)
- Non-Goals
  - JSON Schema Support
  - Round-trip
  - Tool-time API

JAX-RS / CDI Alignment

A Road to Java EE Context & Dependency Injection



JAX-RS / CDI Alignment

- Hopeful about CDI 2.0 (JSR-365)
  - Define the behavior of CDI outside of a Java EE container
    - Also involves introducing a programmatic bootstrapping API
  - Make CDI more modular to help other Java EE specs to better integrate with it

- Main Pain Points
  - Constructor selection
  - Generic producers
    - `@PathParam("quote") String quote, @PathParam("quote") Quote quote`
  - Bootstrapping & running outside of Java EE
  - Focus on what's important (Contexts and Injection)
    - Need to reduce the footprint dramatically

Q&A