JAX-RS 2.1 New Features

What's in the queue for REST in Java EE 8?

Markus KARG (Head Crashing Informatics, JSR 339, JSR 370)

Java Forum Stuttgart, 2015-07-09

Legal Disclaimer

This presentation expresses solely my personal opinion and is not necessarily aligned with the official statement of any of my customers or employers, the JSR 339 and 370 Expert Group, Oracle Corp., or any other named company.

All trademarks belong to their particular owners, even if not declared explicitly.

The Cheeky™ comic character is used by courtesy of *inviticon℠*.

How To Become An EG Member

Markus

- Born 1973
- ZX Spectrum (~1985)
- State-Qualified Information Scientist (1997)
- Java Addict (1997)
- WebDAV Support for JAX-RS (2008)
- Jersey Contributor (Jersey 0.8)
- JAX-RS EG Member (JSR 339, 370)

https://headcrashing.wordpress.com
markus@headcrashing.eu

Today's Agenda

- JAX-RS As The Heart Of Java EE
- **Proposed Changes**
- Anticipated Schedule
- Status Quo
- Q & A

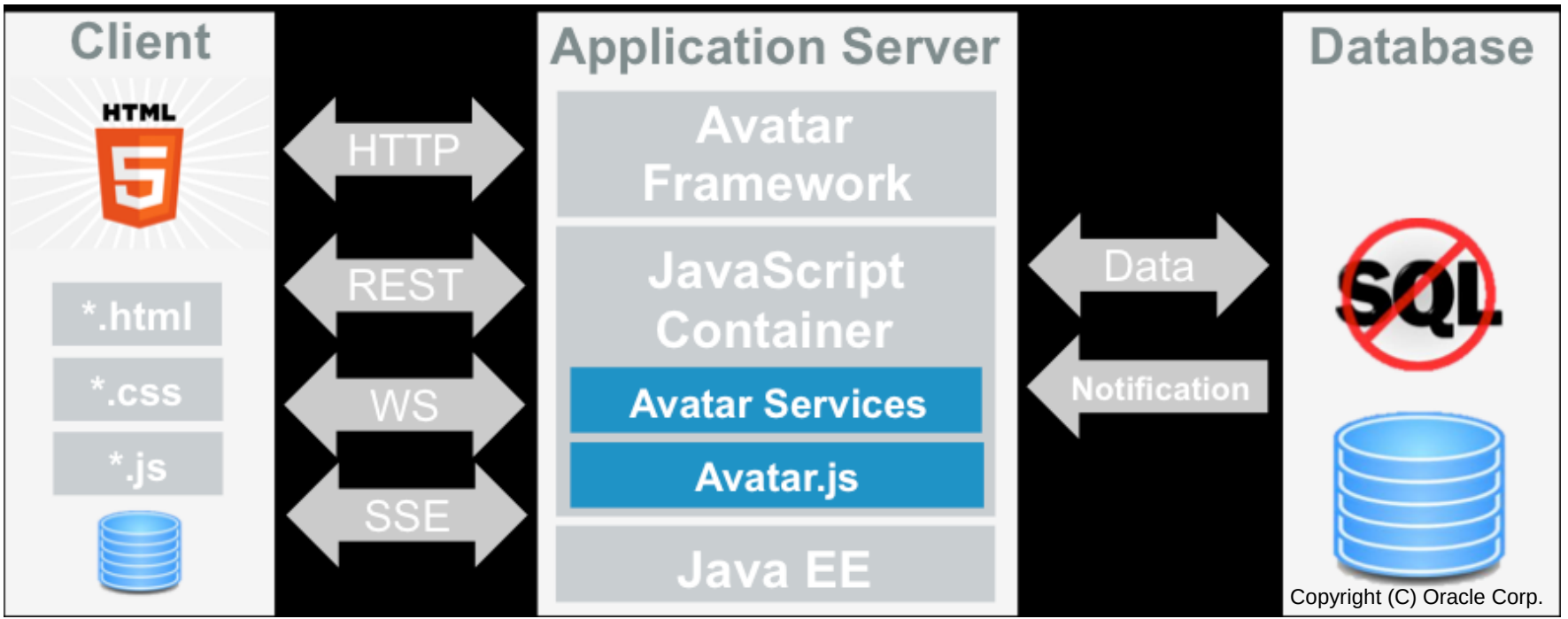
„JAX-RS is one of Java EE's most important APIs.“

(Ed. Burns, Oracle, paraphrased)

Java EE Through The Ages

Stone Age	*Middle Ages*	*Past*	*Present*	*Future*
Propr. RPC Era	**CORBA Era**	**SOAP Era**	**REST Era**	**Cloud Era**
PC	PC	PC	PC	Mobile Things
Binary-RPC	Binary-RPC	XML-RPC	XML-Document	JSON-Document
EJB	EJB	EJB	EJB	CDI
BMP	CMP	JPA	JPA	JPA++
RMI/JRMP	RMI/IIOP	JAX-WS/HTTP1.0	JAX-RS/HTTP1.1	JAX-RS NG/HTTP2.0
JRE/Server	JRE/Server	JRE/Cluster	JRE/VM	JRE/Container

JAX-RS – Powering The Post-Enterprise Era



WARNING

All code shown is *non-functional* and serves solely *illustrative* purposes.

Planned Changes

- Java 8: Lambdas, Streams & CompletableFuture
- SSE: Pushing Events To The Client
- Improved CDI Integration
- NIO in Providers / Filters / Interceptors
- Declarative Security
- WARNING: JAXB becomes conditional
- JSON-B becomes mandatory
- Improved HATEOAS
- Reactive Client API: Simplifying asynchronous chains
- Support for MVC (JSR 370)

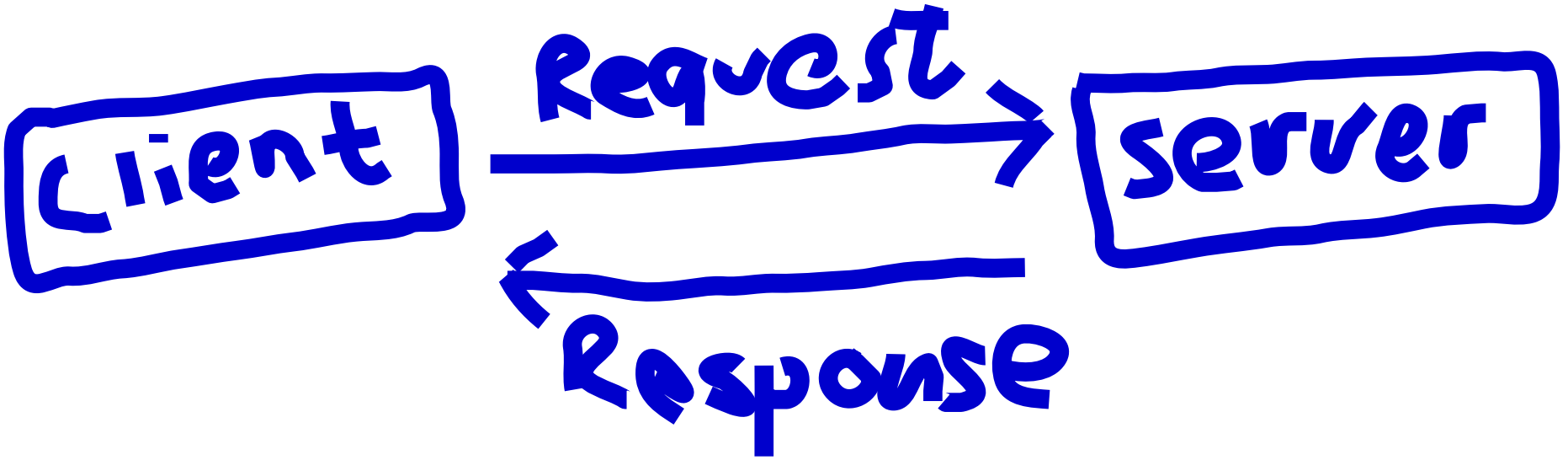
https://jcp.org/en/jsr/detail?id=370#2


Java 8

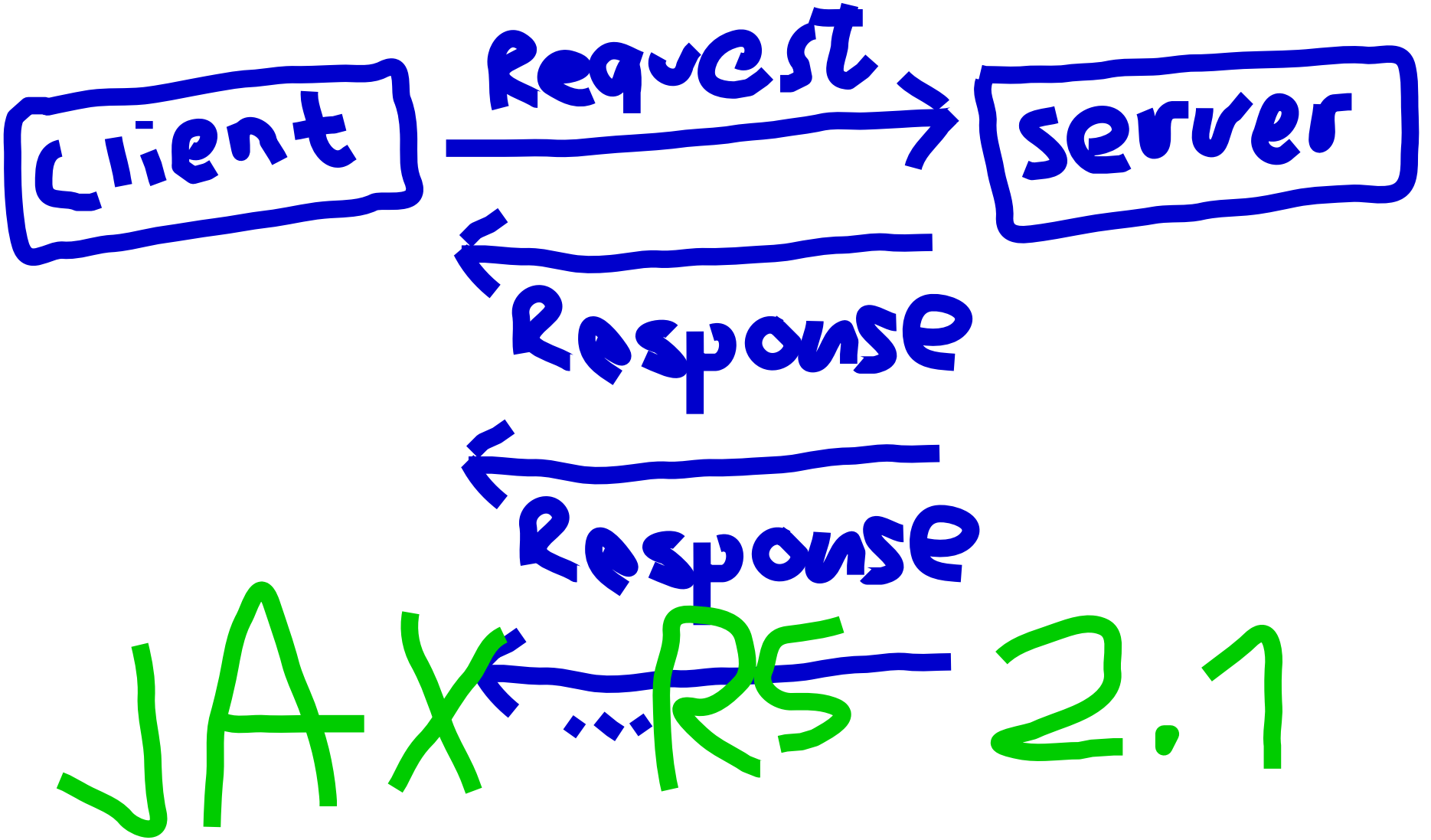
Java 8: Lambdas, Streams & CompletableFuture

- Code runs faster and is more concise
  - Concise code with Lambda Expressions
  - *Map-reduce* solutions with `Stream`
    - Example: `MessageBodyReader` could parse entity with *parallel threads*
  - *Reactive programming* with `CompletableFuture`
  - Possible API simplifications not discussed yet :-(
    - Example: `Stream<T>` or `CompletableFuture<T>` as a result type
    - Example: `Optional<T>` as header types


SSE



JAX-RS 2.0


Client
Request
Server
Response
Response
JAX-RS 2.1

SSE (Server Sent Events)

- **SSE** here is literally that particular technology, but *not* a paradigm!
  - **WebSockets** are *not* planned to be supported!
- **RESTful SSE** is **RE**presentational **S**tate-*Changes* **T**ransfer
- JAX-RS originally was about REST
- REST *typically* is interpreted as Request-Response
- REST *does not mandate* Request-Respone

Imagine the combination of SSE with an reactive API!

„Whenever event of type X is received, process it just like a request or response.“

(See Wikipedia on SSE)

```
@Path(„jerseyDemo“) @Produces(APPLICATION_JSON)
public class JerseyDemo {
  private static SseBroadcaster broadcaster = new SseBroadcaster();

  @GET @Path(„events“) @Produces(SseFeature.SERVER_SENT_EVENTS)
  public EventOutput connect() {
    EventOutput eventOutput = new EventOutput();
    broadcaster.add(eventOutput);
    return eventOutput; // must use EventBuilder to push Event instance into broadcaster
  }
}
```

Bad: Exposes technology, **missing SoC**.
Bad: Mixes up pub/sub with SSE. What if my source is JMS, hardware, etc.? What if we add WebSockets?

```
@Path(„counterProposal“)
public class CounterProposal {
  @Inject MyService myService; // we don't care where the event actually comes from

  @SSE
  public Supplier<MyEvent> connect() {
    return myService.eventSource(); // will invoke EntityProvider for each MyEvent
  }
}
```


CDI

JAX-RS 2.0

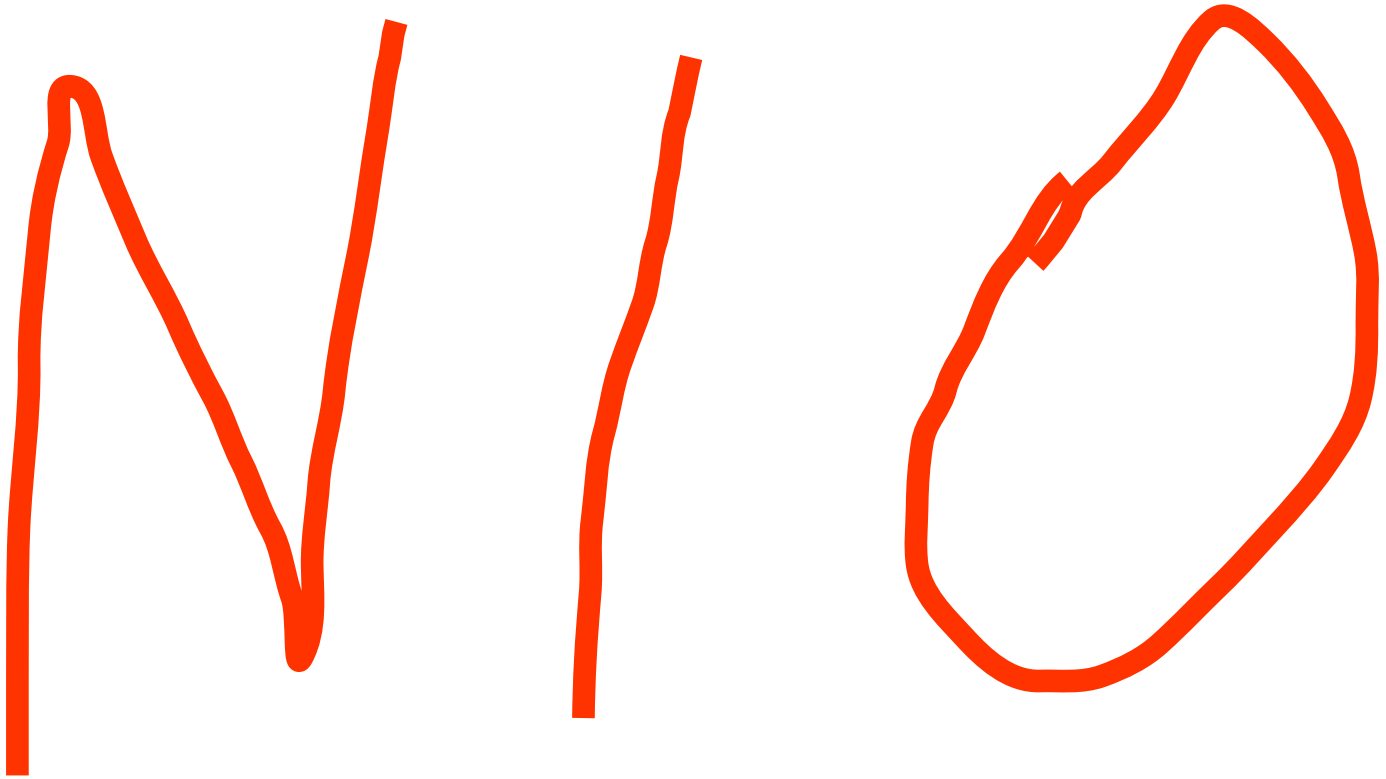

JAX-RS
CDI
JAX-RS 2.1

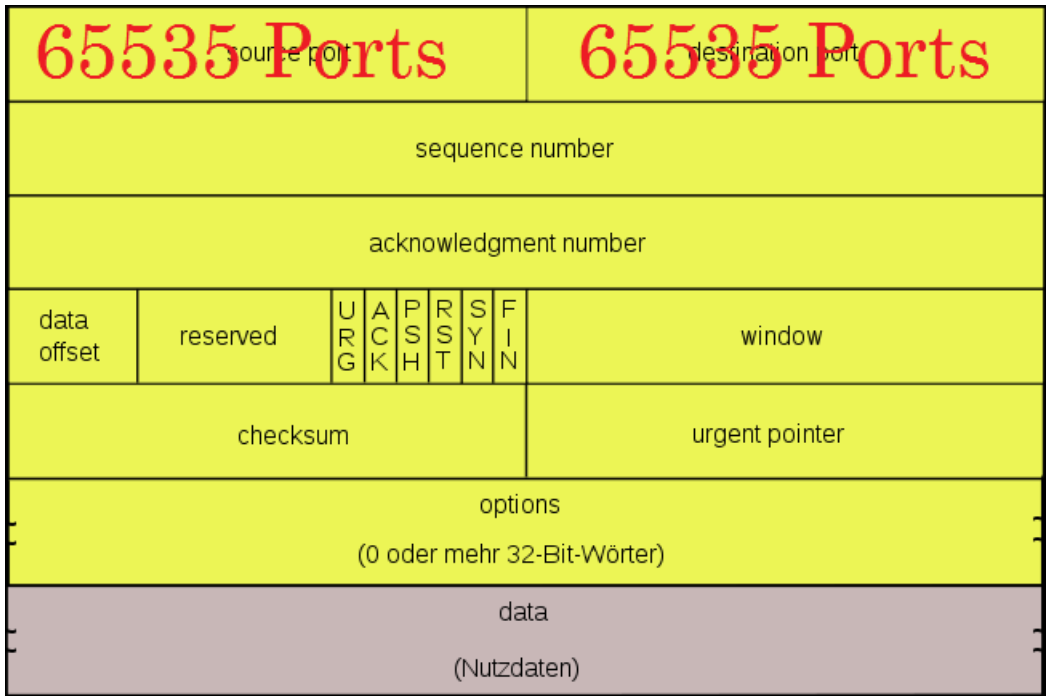
Improved CDI Integration

- Historically two incompatible solutions for the same set of problems
  - Lifecycle, Scopes, Factories, Extensions, Injection, Annotations etc.
  - Example: JAX-RS Container manages lifecycle of resource instance, CDI needs to do that instead.
- JAX-RS is older, but CDI is much more flexible and extensible
- JAX-RS can run on Java SE, CDI 1.x could not, but CDI 2.0 will
- MVC (JSR 371) *enforces* CDI

Vision: *Replace JAX-RS-Injection-Technology by CDI*

http://hnusfialovej.cz/2015/02/25/jersey-further-improves-cdi-integration/
https://blogs.oracle.com/japod/entry/container_agnostic_cdi_support_in
http://blog.dejavu.sk/2015/03/11/jersey-cdi-integration-few-notes-and-ear-support/?utm_source=oracle&utm_medium=blog&utm_campaign=mgajdos


NIO



JAX RS 2.0

According to TCP we can serve 64K sessions.

Can we really?

Actually the *thread count* is the limiting factor:

- Thread creation time
- Thread context switch time
- Thread memory overhead
- Thread handles

So we must *reuse* threads –

but we cannot as long a thread is **blocked**!

JAX-RS 2.0 wants it to be blocked! :-(

javax.ws.rs.ext

Interface WriterInterceptorContext

All Superinterfaces:
InterceptorContext

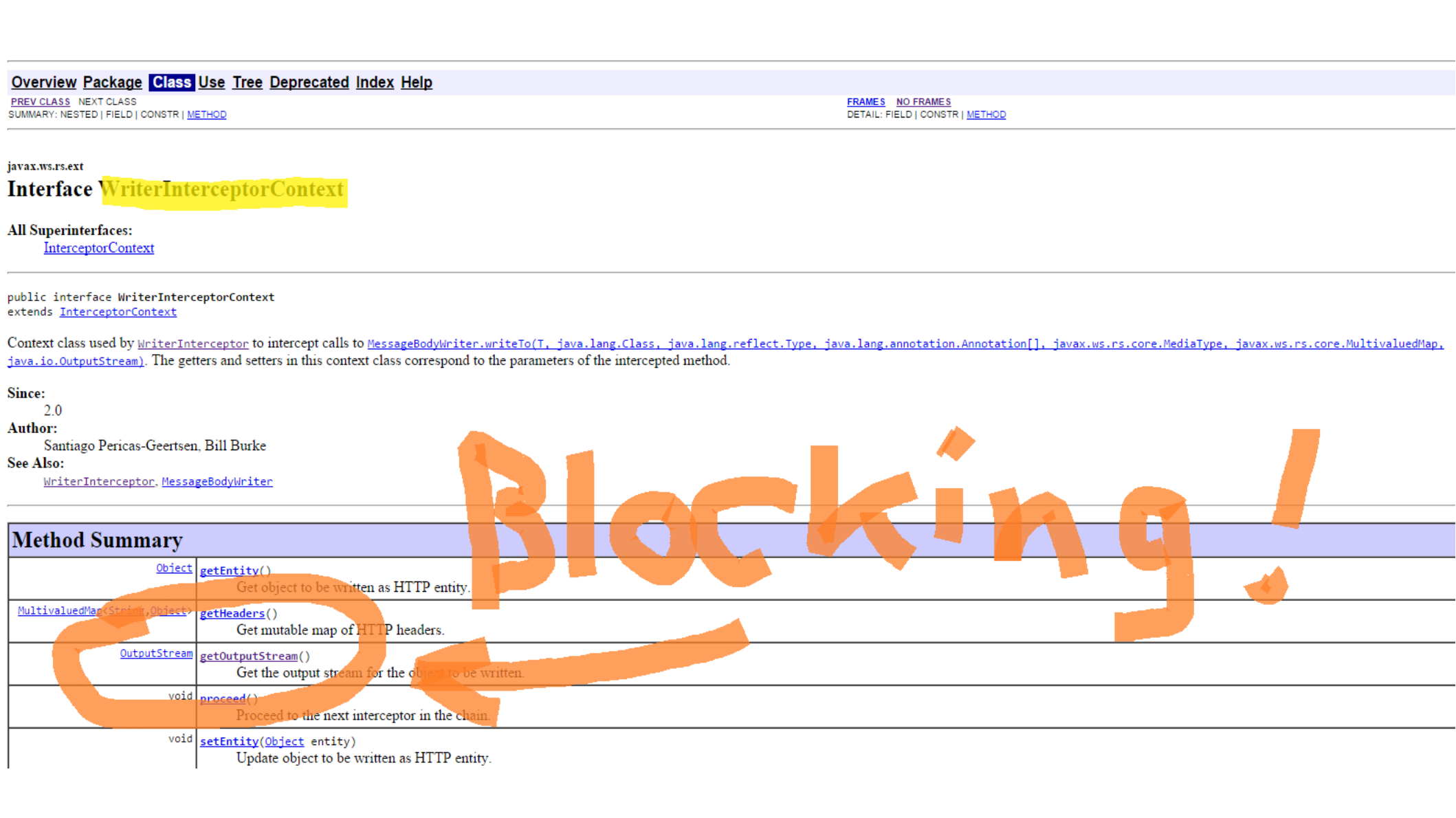
public interface WriterInterceptorContext
extends InterceptorContext

Context class used by WriterInterceptor to intercept calls to MessageBodyWriter.writeTo(T, java.lang.Class, java.lang.reflect.Type, java.lang.annotation.Annotation[], javax.ws.rs.core.MediaType, javax.ws.rs.core.MultivaluedMap, java.io.OutputStream). The getters and setters in this context class correspond to the parameters of the intercepted method.

Since:
2.0

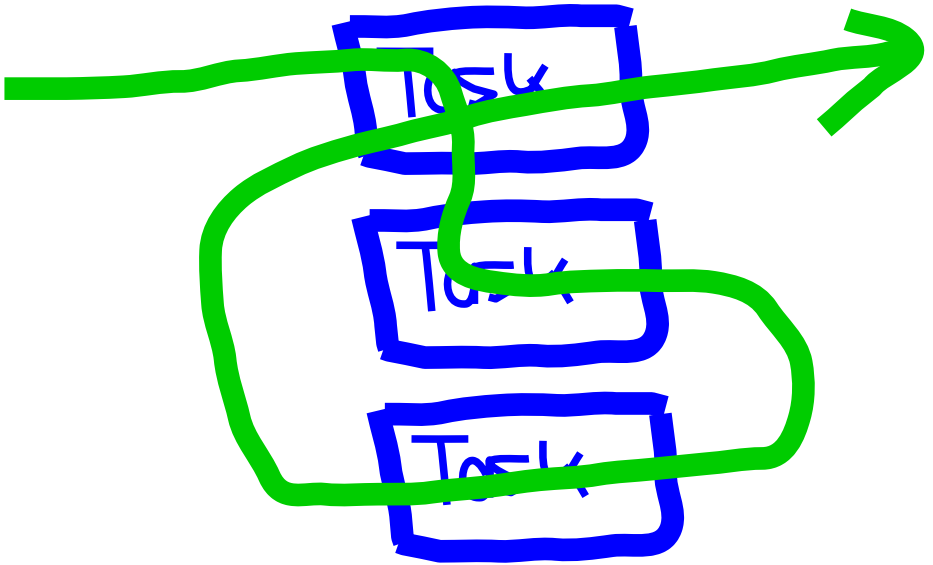
Author:
Santiago Pericas-Geertsen, Bill Burke

See Also:
WriterInterceptor, MessageBodyWriter



Method Summary

Object	getEntity() Get object to be written as HTTP entity.
MultivaluedMap<String,Object>	getHeaders() Get mutable map of HTTP headers.
OutputStream	getOutputStream() Get the output stream for the object to be written.
void	proceed() Proceed to the next interceptor in the chain.
void	setEntity(Object entity) Update object to be written as HTTP entity.



JAX-RS 2.1

JAX-RS 2.1 allows *non-blocking* interceptors and filters thanks to NIO API (i. e. Non-blocking).

Thread does not block anymore, but simply puts „open work“ (`Future`) aside for some time.

Threads can be reused more easily => Less threads needed.

Many more clients per host possible. :-)

Limiting factor now is *RAM* to hold `Future`s which is plentiful these days.

Vision: No blocking APIs used => 65K sessions powered by just N+1 threads.

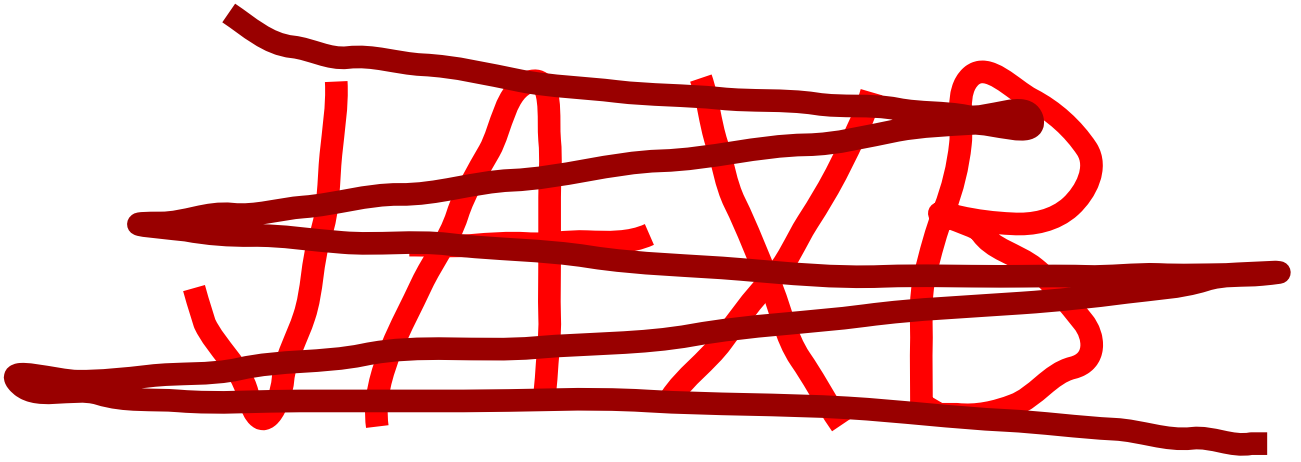
Core idea: Don't use `InputStream` / `OutputStream`, but `Channel`s and `Buffer`s.

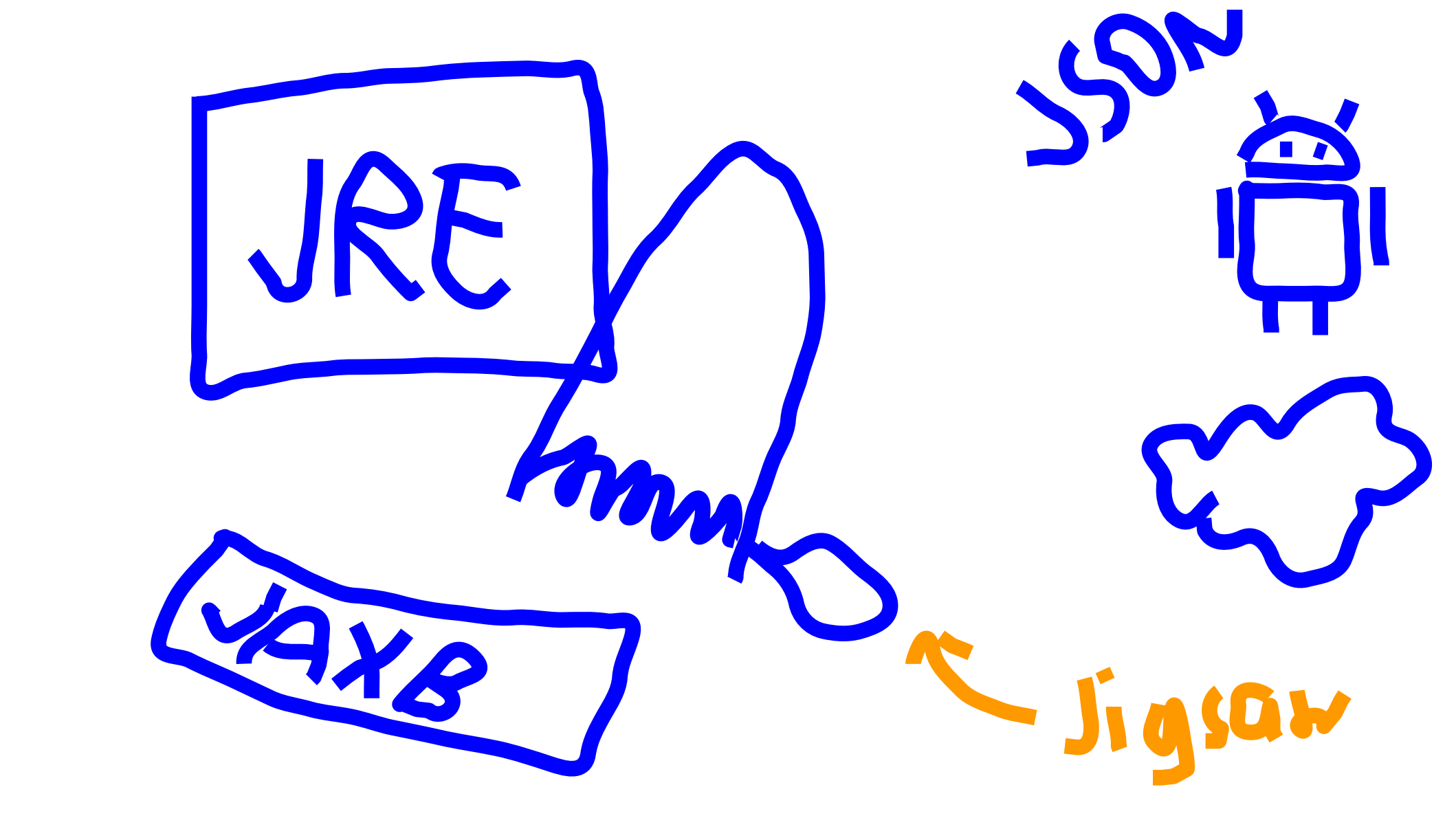
http://docs.oracle.com/javase/8/docs/api/java/nio/package-summary.html

Declarative Security

Declarative Security

- Several existing APIs for serveral platforms, e. g. Java EE
- Alignment with upcoming Java EE standard
- Can be used in Java SE
- Support for Oauth
- Jersey covers several aspects, but we need an industrial standard



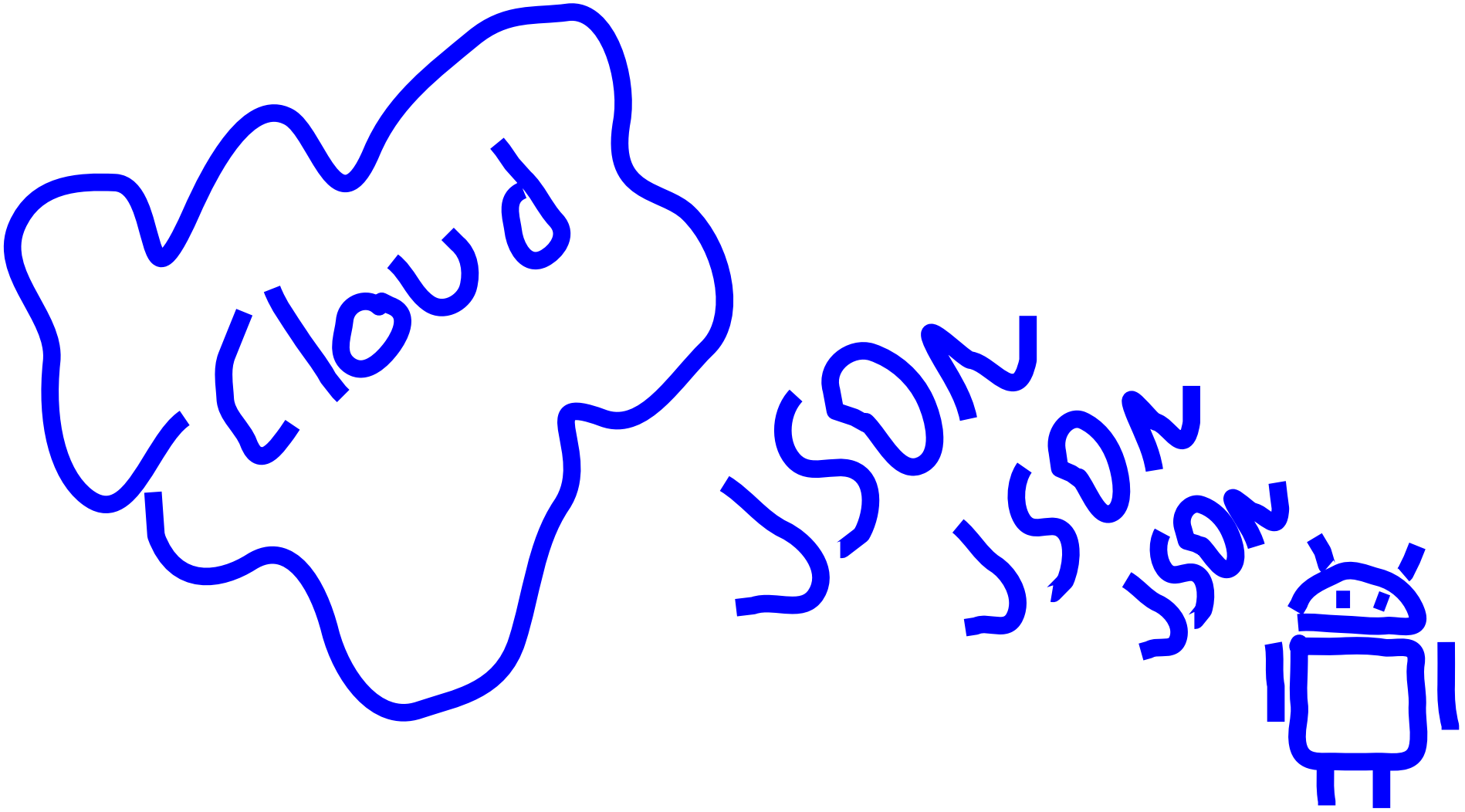

JRE
JAXB
JSON
Jigsaw

Good Bye, XML!

- JAXB provides binding between XML and Java, was part of Java SE 8
- JAXB was *mandatory* in JAX-RS 2.0
- JAXB is *likely* to get stripped from Java SE 9 due to project Jigsaw
- JAXB is *not supported* on Android
- **JAXB becomes *conditional* with JAX-RS 2.1**
  - If the platform provides JAXB, JAX-RS 2.1 MUST support it.
  - If the platform doesn't provide JAXB, JAX-RS 2.1 CAN support it.
- **WORA won't work anymore, must bundle JAXB with application!**

https://blogs.oracle.com/japod/entry/jersey_2_x_client_on1


JSONB


cloud
JSON
JSON
JSON

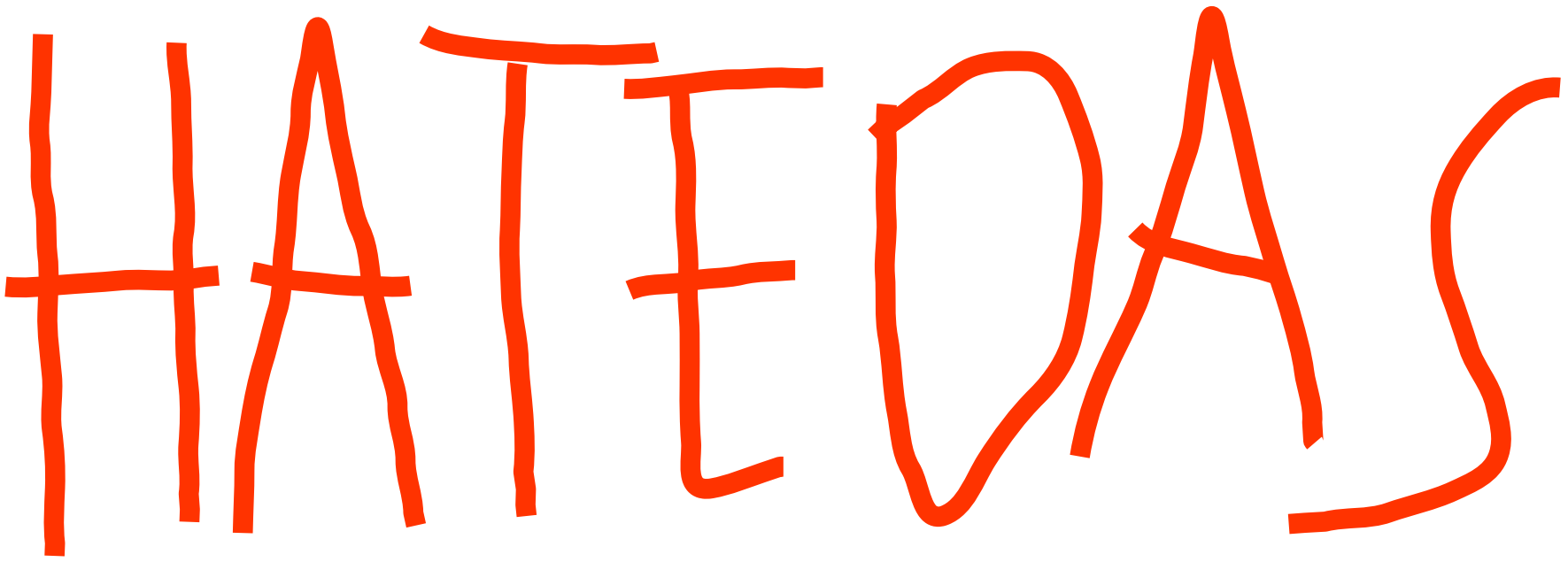
Java API for JSON Binding (JSON-B)

- JSON is the de-facto standard for RESTful web services.
- JSON-B API is to JSON what JAXB is to XML.
- It's simply straightforward to declare JSON-B support as MANDATORY.

Best Practice

Implement a Gateway Service providing JSON *and* XML using two Entity Providers.

- Never use `@Produces` *at methods* but only *at Message Body Writers*.
- JAX-RS will select the right Entity Provider with respect to `Accept:` header.
- All kinds of clients will work, it is extensible, and provides good SoC.


HATEDAS

REST Maturity Model (Richardson)

0	• *SOAP* or RPC/XML • *Single* URI for whole Service • *Single* HTTP verb for alle actions
1	**+ Native HTTP** **+ Different URIs for separate ressources**
2	**+ Different HTTP verbs (DELETE, OPTIONS, HEAD...)**
3	**+ HATEOAS (Hypermedia Navigation)...** • **„Basics“ since JAX-RS 2.0** • „`@InjectLink`“ in Jersey, but not part of JAX-RS so far

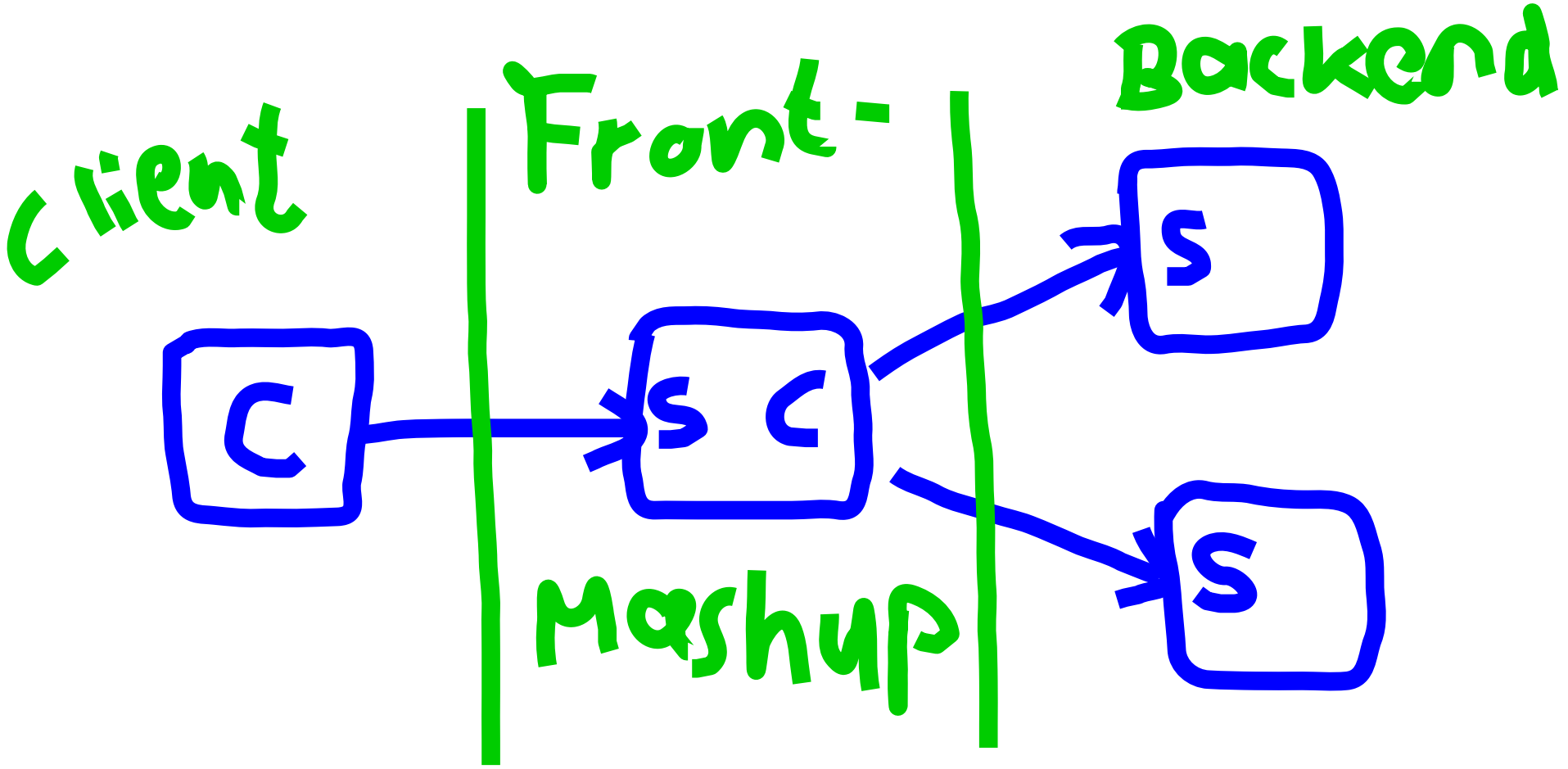
http://martinfowler.com/articles/richardsonMaturityModel.html

```
HTTP/1.1 200 OK
    Content-Type: application/json
    Content-Length:

    {
  "BankAccount": {
    "iban": "DE1234567891234",
    "balance": {
      "currency": "EUR",
      "balance": "123.45"
    },
    "links": [
      {
        "rel": "statement",
        "href": "account/DE1234567891234"
      }
    ]
  }
}
```

```
@Path("account/{iban}")
public class BankAccount {
    @PathParam("iban")
    IBAN iban;

    @GET
    public AccountStatement statement() {
        return new AccountStatement(iban);
    }
}

public class AccountStatement {
    @InjectLink(resource=BankAccount.class, method="statement")
    URI u;
}
```

- Only `URI` or `String`, only in entity, doesn't support Link class!
- **Counter Proposal: Let entity provider inject, and provide it an SPI to resolve URIs**
  - **Keeps entity clean, preserves existing separation of concerns as of JAX-RS 2.0**


Reactive
Programming

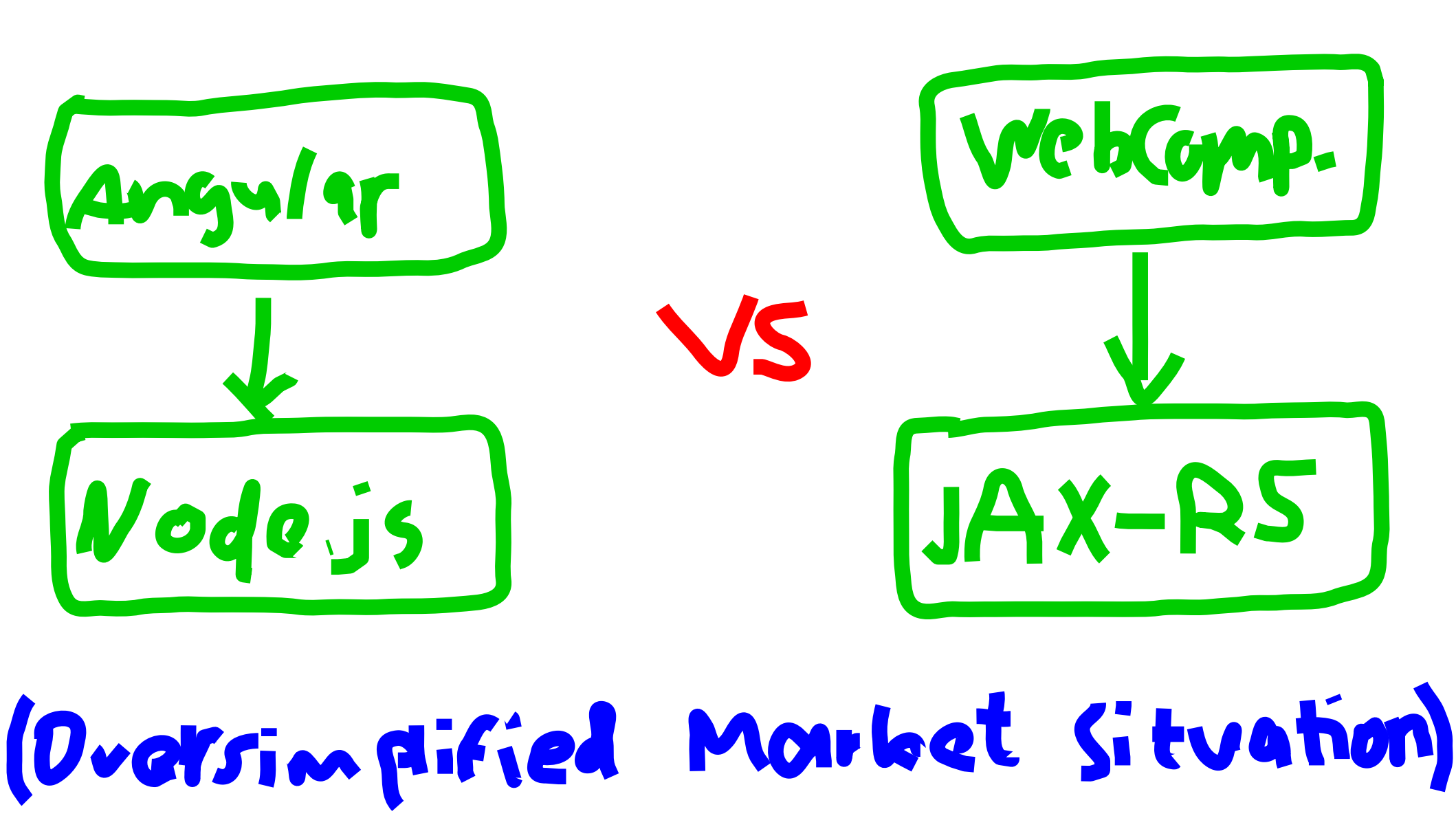

Client
Front-
Mashup
Backend
C
S
C
S
S

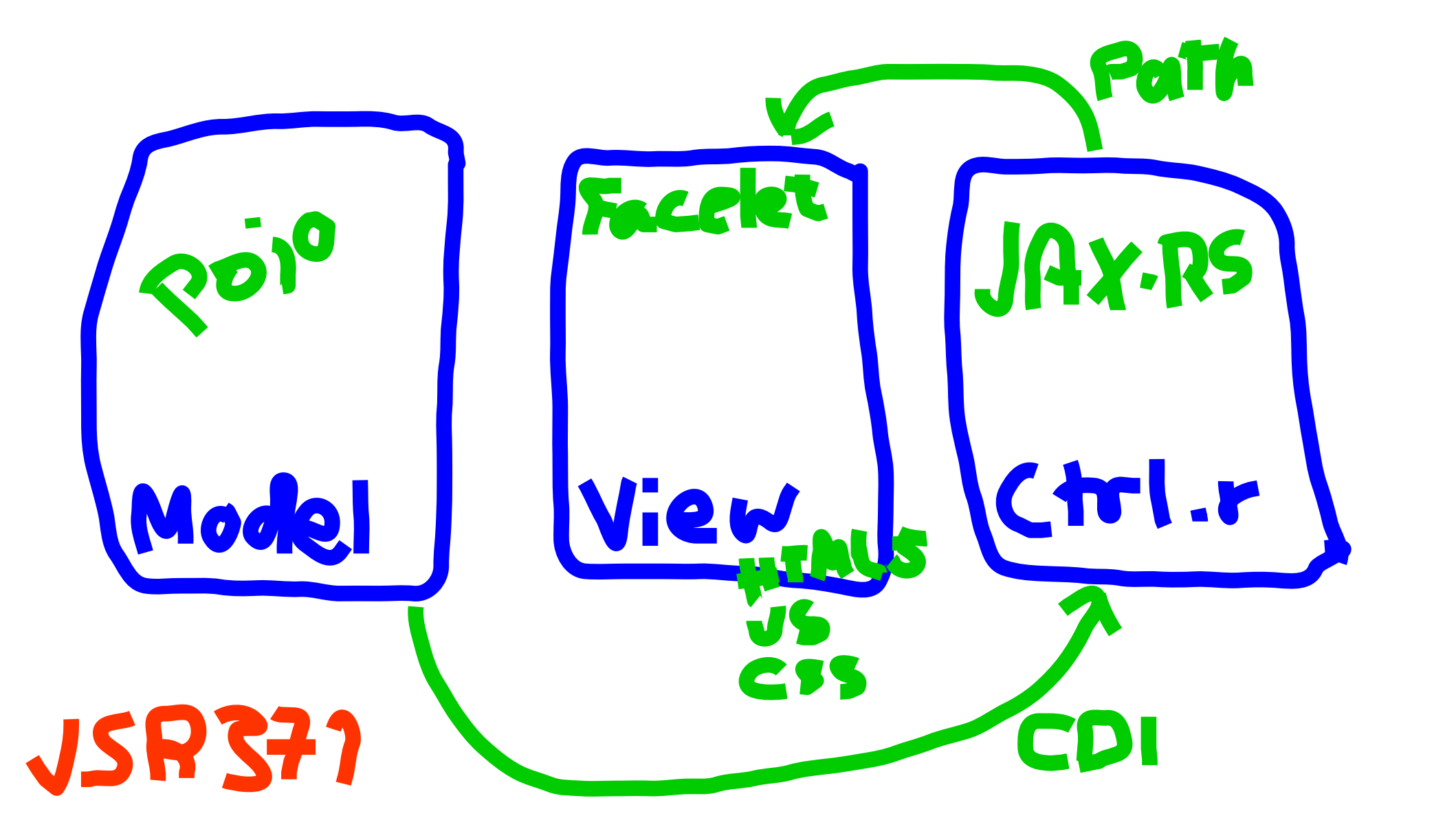
```java
@Path("/") public class ReactiveDemo {
	@Resource ManagedExecutorService CONTAINER;

	/*
	 * Result is void, but actually a String entity is the outcome!
	 * Must always invoke thenApply(asyncResume::resume) explicitly.
	 */
	@GET public void serverSideDemo_CurrentVersion(@Suspended final AsyncResponse asyncResponse) {
		CompletableFuture.runAsync(ReactiveDemo::veryExpensiveOperation, CONTAINER).thenApply(asyncResponse::resume);
	}

	@GET public CompletableFuture<String> serverSideDemo_CounterProposal() {
		return CompletableFuture.supplyAsync(ReactiveDemo::veryExpensiveOperation, CONTAINER);
	}

	/*
	 * Rather complex boiler plate to get Java 8 CompletableFuture.
	 * Strange rx(CONTAINER) method with each call.
	 */
	@GET public String clientSideDemo_JerseyProposal() {
		final RxClient<RxCompletableFutureInvoker> client = Rx.newClient(RxCompletionStageInvoker.class);
		final CompletableFuture<String> getA = client.target("some uri A").request().rx(CONTAINER).get();
		final CompletableFuture<String> getB = client.target("some uri B").request().rx(CONTAINER).get();
		return getA.thenCombine(getB, (a,b) -> a + ":" + b).join();
	}

	@GET public String clientSideDemo_CounterProposal() {
		final Client client = ClientBuilder.newClient(CONTAINER);
		final CompletableFuture<String> getA = client.target("some uri A").request().get();
		final CompletableFuture<String> getB = client.target("some uri B").request().get();
		return getA.thenCombine(getB, (a,b) -> a + ":" + b).join();
	}
}
```

Reactive Programming

- Java 8 provides core technology: `CompletableFuture`
- No comprehensive standard for reactive Java so far
- Several frameworks available, e. g. RxJava, JavaFX Bindings, etc.
- Jersey already integrates with some of them
- Discussion in JAX-RS completely open

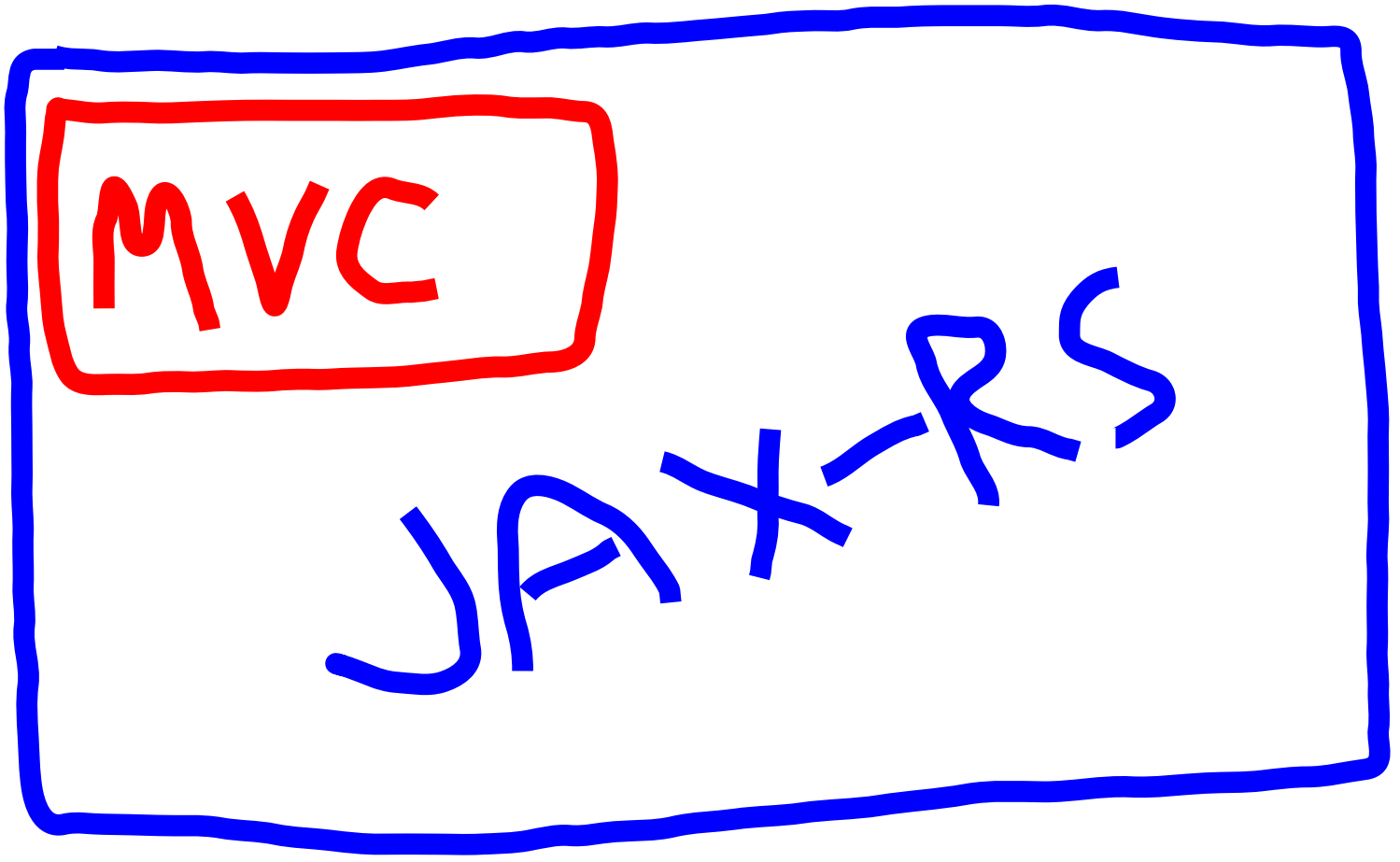

MVC


Angular
Node.js
VS
WebComp.
JAX-RS
(Oversimplified Market Situation)


Path
Facelet
JAX-RS
Poio
Model
View
HTML
JS
CSS
Ctrl.r
CDI
JSR371

```
@Path("books/{isbn}") public class BookController {

  @Inject Book book;

  @GET @Controller public String view(@PathParam("isbn") ISBN isbn)
    return "BookView.jsp"; // Route
  }
}

@Produces @Named getBook() {
     return jpaEntityManager.find(isbn);
}
```

Open question: How to pass ISBN into producer method?


MVC
JAX-RS


MVC
REST
JAX-RS
Framework
ex.

MVC (JSR 371)

- New web standards and frameworks exert pressure upon Java EE
- Client-side components (vs Server-side components in JSF)
- Controller: JAX-RS Resource (must use CDI)
- Model: POJO
- View: CDI-plugable engine, e. g. Facelet (HTML5 + CSS3 + JavaScript)
- Offloading compontent resolution to browser (e. g. WebComponents)
- JAX-RS must learn to deal with Facelet
- JSR 371 published Early Draft on May 25th (25 pages ontop JAX-RS)

https://jcp.org/aboutJava/communityprocess/edr/jsr371/index.html

JAX-RS 2.1 Anticipated Schedule

- Q3/2014 Expert Group Formation
- Q1/2015 Early Draft - **delayed**
- Q3/2015 Public Review
- Q1/2016 Proposed Final Draft
- Q3/2016 Final Release

Status Quo

- The schedule obviously is delayed since months.
- Oracle apparently has added proprietary support to Jersey for SSE, declarative security, HATEOAS, reactive API and more.
- **Oracle has not yet presented the EG an API proposal for any of the intended features.**
- On June 6th, Oracle announce a 6 month delay of Java EE 8, as spec leads *have better things to do* than writing specs.

https://blogs.oracle.com/java/entry/java_ee_8_roadmap_update

https://jersey.java.net/documentation/latest/declarative-linking.html

How The Expert Group Works

- „Benevolent Dictatorship“
- Oracle develops Jersey, i. e. JAX-RS RI.
- When Oracle thinks it's time, they show us new features and ask for our opinion.
- We discuss how it's meant to work.
- We propose changes.
- Oracle decides, we have to live with the result.

Questions?