



# CDI\* for Seam 2 developers

Brief migration notes  
or what does CDI mean for Seam 2 developer

Martin Kouba  
Red Hat 2012

\* Contexts and Dependency Injection for the Java EE platform



# Source available at github:

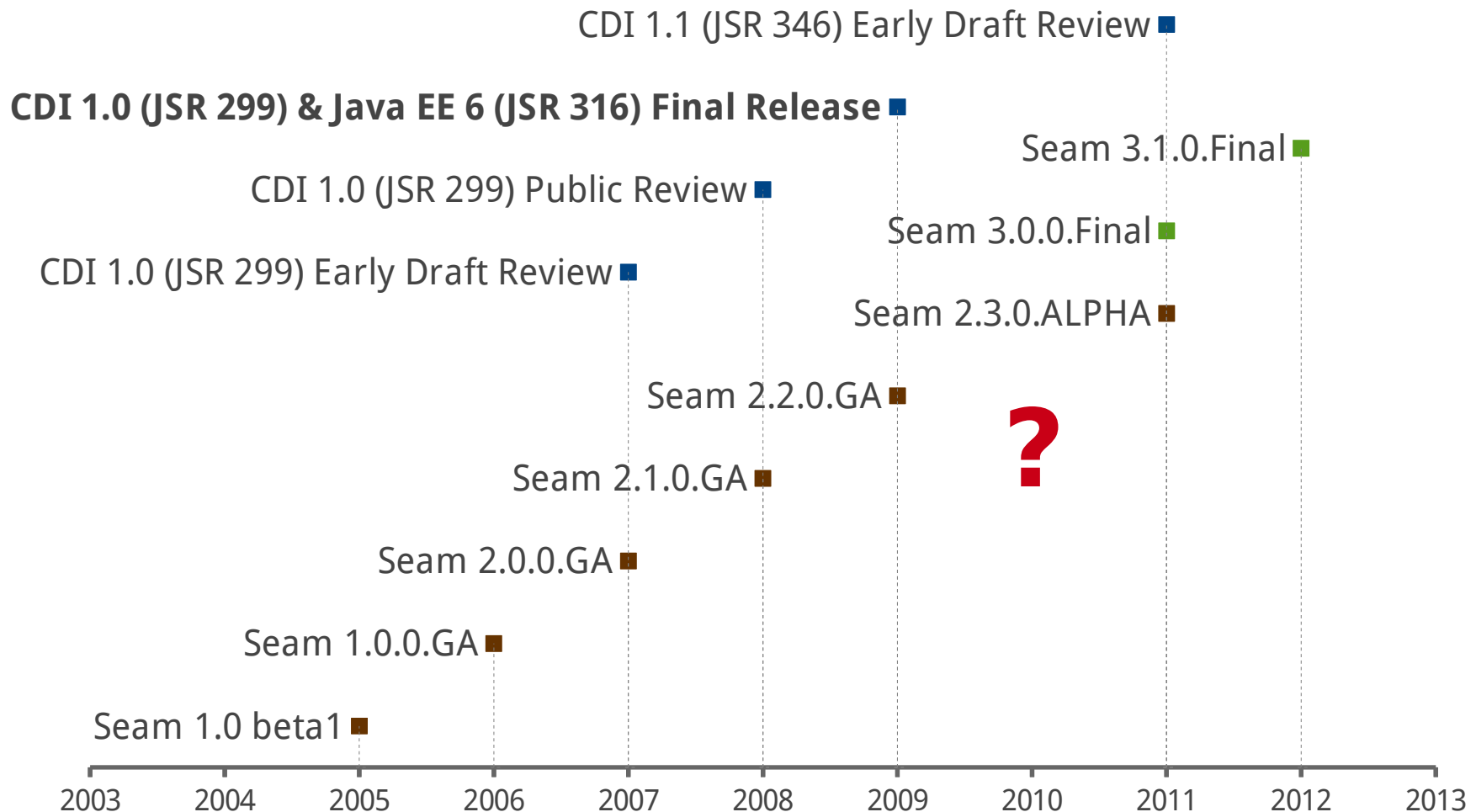
<https://github.com/mkouba/cdi4seam2dev>

- presentation in PDF format
- example source code

# Agenda

1. In relation to... a little bit of history
2. Seam 2 vs CDI → the big picture
3. Component models
4. Bijection vs dependency injection (live demo :-)
5. Factory methods vs producers
6. Events
7. Interceptors (and decorators)
8. Questions

# In relation to ... a little bit of history



# Seam 2 vs CDI → the big picture

## ■ Seam 2

- is an **application framework**
- built to “fix holes/fill gaps” in specification (Java EE 5)
- the idea of “**Reinvesting in Java EE**” → fixes should find way back into the next revision of the standards

## ■ CDI

- is a **JCP specification**
- originally Web Beans
- version 1.0 (JSR 299) is a part of Java EE 6 (JSR 316)
- implementations include:
  - Weld (RI)
  - Apache OpenWebBeans
  - CanDI
- Seam 3 is a set of modules which extend CDI

# Seam 2 vs CDI → the big picture

## Seam 2 functionalities

### Core

- components
- scopes and contexts
- bijection
- events
- interceptors...

### Tools

- seam-gen

### Integration stuff

- Java EE (JSF, EJB, JAX-WS, ...)
- JBoss projects (RESTEasy, jBPM, ...)
- Third party projects (iText, Quartz Scheduler, ...)

### Out of the box solutions

- security
- i18n
- e-mail
- ...

# Seam 2 vs CDI → the big picture

## CDI covers

### Core

- components
- scopes and contexts
- bijection
- events
- interceptors...

### Integration stuff

- Java EE (JSF, EJB, JAX-WS, ...)
- JBoss projects (RESTEasy, jBPM, ...)
- Third party projects (iText, Quartz Scheduler, ...)

### Tools

- seam-gen

### Out of the box solutions

- security
- i18n
- e-mail

# Seam 2 vs CDI → the big picture

- Summary:
  - CDI covers most of Seam 2 core functionalities in a **standardized**, **typesafe** and **extensible** way



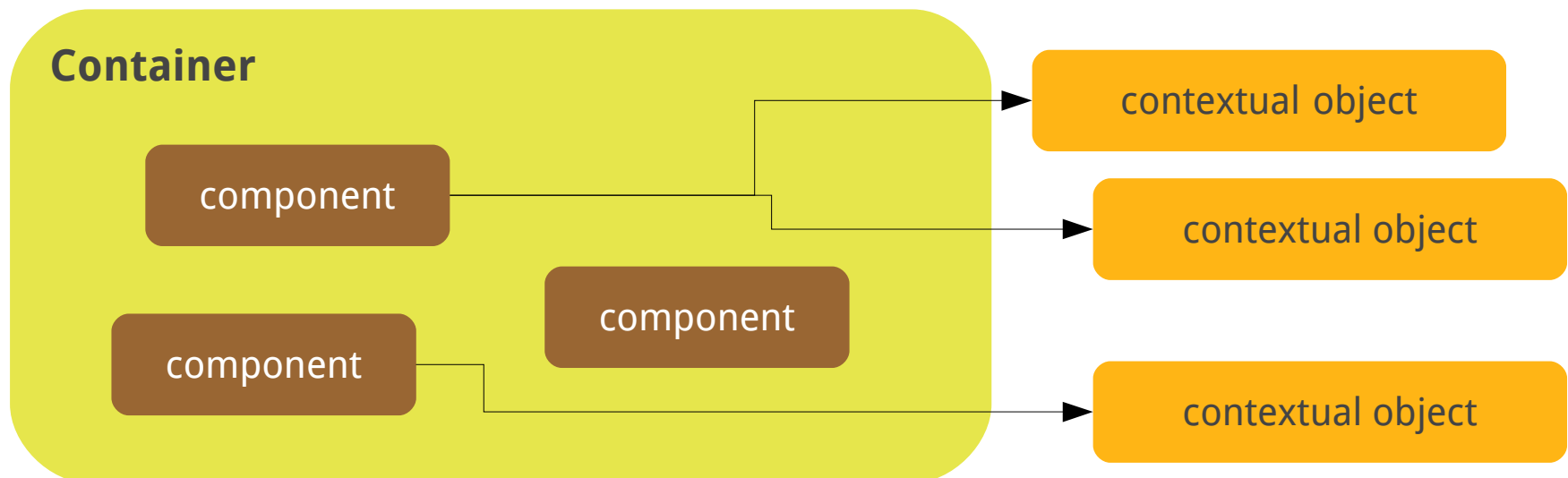


And now for something  
completely different...

Component models

# What is a component?

- component is a source of contextual objects
- contextual objects define application state and/or logic
- components are usually configured with metadata (annotations, XML)



# Component models

## Diff #1 - terminology

- Seam → components
- CDI → beans

# Component models

## Diff #2 – metadata definition

### ■ Seam

- define metadata via annotations and XML

### ■ CDI

- define metadata via annotations and programmatically in portable extension (during app initialization)
- XML configuration is not covered by spec → use JBoss Solder<sup>1</sup>

<sup>1</sup> <http://seamframework.org/Seam3/Solder>

# Component models

## Diff #3 – component types

### ■ Seam

- Session bean
- JavaBean
- Factory method
- restricted:
  - Message-driven bean
    - may not be bound to a Seam context
  - Entity bean
    - do not support bijection or context demarcation

### ■ CDI

- Session bean
- Managed bean
- Producer method/field
- Resource
  - represents a reference to a Java EE resource
- a portable extension may provide other kinds of beans

# Component models

## Diff #4 – component names

### ■ Seam

- each component **must have the name defined explicitly** via `@Name` or XML descriptor,
- name is string-based and unique across the application,
- name is **involved in bi-jection lookup** mechanism,
- component is automatically available in Unified EL (JSP, JSF)

### ■ CDI

- beans **have no name by default** (typesafe resolution),
- though may have name defined via `@Named` (EL name resolution – suitable only for UI),
- and if so, they are available in EL

# Component models

## Diff #5 – registration process

### ■ Seam

- scans archives which contain `seam.properties` or `components.xml` at specified location
- each component has to be marked **explicitly** in order to be recognized by the container (`@Name` or XML descriptor)

### ■ CDI

- scans archives and folders on the classpath which contain `beans.xml` at specified location
- every Java class in the bean archive that meets certain conditions is **implicitly** recognized as a bean - no special declaration is required<sup>1</sup>

<sup>1</sup> CDI 1.0 doesn't solve explicit exclusion (either use some extension like JBoss Solder or wait for CDI 1.1 :-)

# Component models

## Diff #6 – scopes and contexts

### ■ Seam

- **fixed set of contexts**<sup>1</sup>,
- the concept of contextual variables
- @Scope annotation with values of the ScopeType enumeration,
- contexts are accessible for clients directly (rw)

### ■ CDI

- **set of built-in contexts**<sup>1</sup>,
- this set may be extended
- each scope has its own annotation
- no built-in business process, page, method and stateless scope
- dependent pseudo-scope
- CDI contexts cannot be modified by clients

<sup>1</sup> <http://seamframework.org/Seam3/Seam2ToSeam3MigrationNotes>



# Component models

## Diff #7 – basic metadata

### ■ Seam

- name → @Name
- scope → @Scope
- roles → @Roles
  - single Java class to act as a base for multiple components (comprises name and a scope)
- conditional installation → @Install

### ■ CDI

- name (optional) → @Named
- scope → @RequestScoped, ...
- set of bean types
- set of qualifiers
  - used to distinguish between multiple components sharing the same bean type
- conditional installation → @Alternative, @Specializes, @Veto<sup>1</sup>, @Requires<sup>1</sup>

<sup>1</sup> CDI 1.1 (JSR 346)

# Component models

## Diff #8 – asynchronicity

### ■ Seam

- supports asynchronous method invocation via `Dispatcher` component
  - EJB `TimerService`,
  - or Quartz Scheduler implementation

### ■ CDI

- does not specify asynchronous method invocation
  - try using EJB `@Asynchronous` observer methods



# Inversion of Control

## Seam bijection vs CDI dependency injection

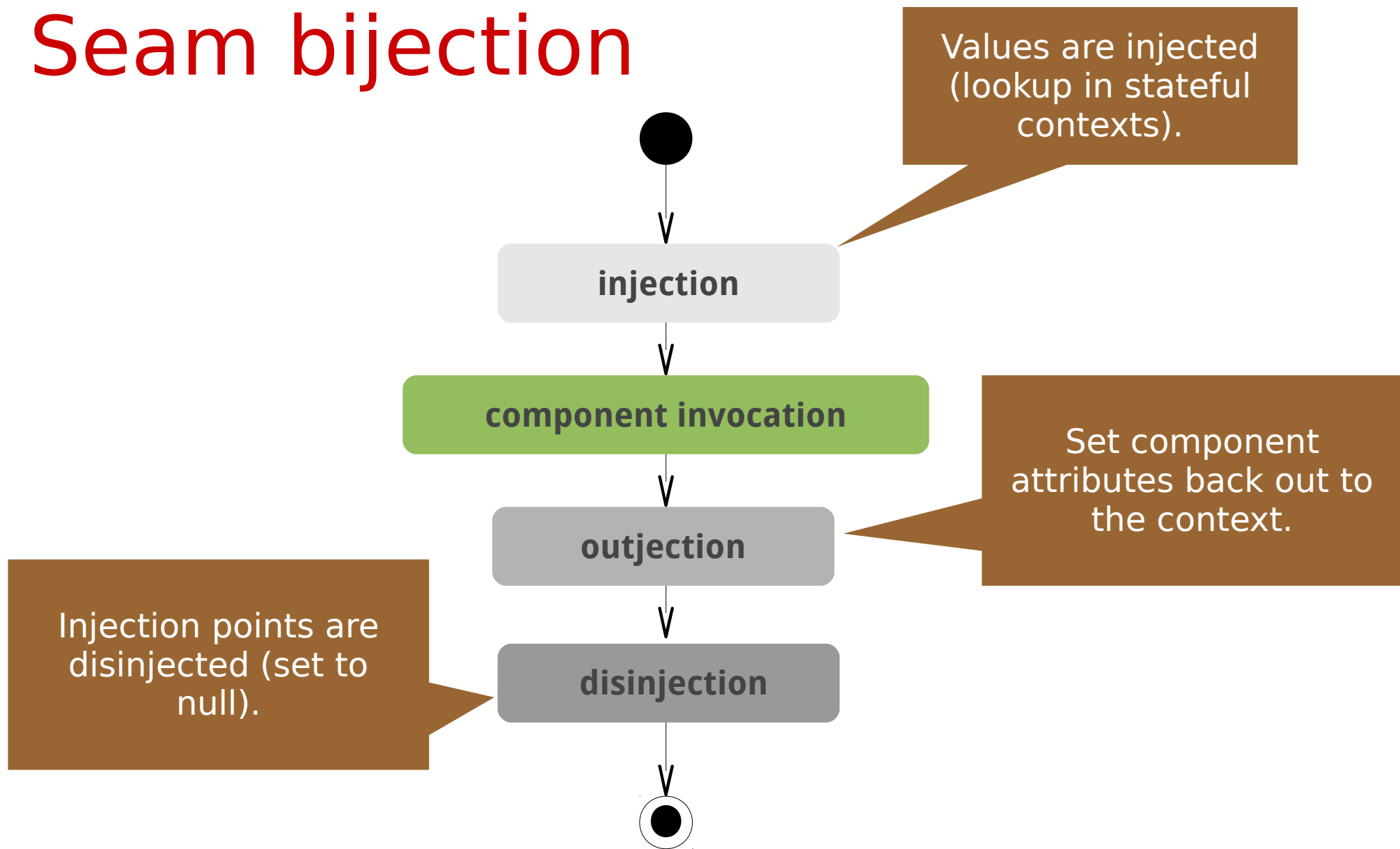
# IoC

## Seam bijection

- bijection is performed dynamically via an interceptor **every time a component method is invoked**
  - bidirectional → injection and outjection
  - injection points: setter method and instance variable
  - **component name is always involved in lookup (!)**
  - `null` may be a result of Seam bijection (!)
  - components are not initialized automatically
    - `@In(create=true)`, `@AutoCreate`
- Seam uses also static injection configuring components via property settings

# IoC

## Seam bijection



# IoC

## CDI approach

- static injection - performed only **once per component lifecycle**
  - when creating contextual objects
    - for normal scopes **client proxy** is injected
  - injection points: constructor, field, initializer method
  - typesafe resolution - the process of matching a bean to an injection point
    - bean is assignable to a given injection point if it **has a bean type that matches the required type has all the required qualifiers**
  - ambiguous and unsatisfied dependency is an error
  - no outjection and disinjection
  - beans are initialized automatically

# IoC

## Programmatic lookup

### ■ Seam 2

- static method  
`Component.getInstance()`  
is often used
  - for optimization →  
`@BypassInterceptors` is  
not suitable everywhere
  - in integration code

### ■ CDI

- is possible via built-in  
bean `Instance`<sup>1</sup> (requires  
injection though)
- or `BeanManager`<sup>2</sup>
- should not be needed in  
application code  
anyway :-)

<sup>1</sup> `javax.enterprise.inject.Instance`

<sup>2</sup> `javax.enterprise.inject.spi.BeanManager`

IoC

# Seam bijection vs CDI injection

- time for a very simple live demo!



# IoC

## Java EE integration

### ■ Seam

- only Seam components support bijection

### ■ CDI

- all Java EE 6 components supporting injection<sup>1</sup> may inject beans via the dependency injection service,
- however their lifecycle is not managed by CDI;
- components supporting injection include: servlets, servlet filters and listeners, JSP tag handlers, JAX-WS endpoints, ...

<sup>1</sup> See JSR 316 – EE.5.2.5 Annotations and Injection

# Factory methods vs producer methods/fields

# Factory methods vs producer methods/fields

## Diff #1 - names

### ■ Seam

- component name required
  - use `@Factory.value()` ,
  - if not specified → derived from method name

### ■ CDI

- name not required
  - typesafe resolution :-)
  - may be assigned via `@Named`

# Factory methods vs producer methods/fields

## Diff #2 – parameter injection

- Seam

- not available

- CDI

- producer method → all parameters are injection points

# Factory methods vs producer methods/fields

## Diff #3 - outjection

- Seam

- instead of returning value, factory method may have `void` return type and use outjection to set variables into the context

- CDI

- not available

# Factory methods vs producer methods/fields

## Diff #4 – producer fields

### ■ Seam

- not available

### ■ CDI

- a producer field is a simpler alternative to a producer method
- usefull for Java EE component environment injection



# Events

# Events

## Diff #1 – event type

### ■ Seam

- type is string-based
- parameters are optional

### ■ CDI

- event is an instance of a concrete Java class
  - the event types include all superclasses and interfaces of the runtime class of the event object  
→ observer resolution is typesafe



# Events

## Diff #2 – raising/firing an event

### ■ Seam

- raise via Events component,
- or declaratively
  - use an annotation `@RaiseEvent`
  - navigation rules configuration; `pages.xml`

### ■ CDI

- fire via an instance of the `Event`<sup>1</sup> interface,
- or `BeanManager`
- it's not possible to fire declaratively

<sup>1</sup> `javax.enterprise.event.Event`

<sup>2</sup> `javax.enterprise.inject.spi.BeanManager`

# Events

## Diff #3 – features

### ■ Seam

- asynchronous and timed events via `Dispatcher` component
  - EJB `TimerService`,
  - or Quartz Scheduler impl
- transaction aware events

### ■ CDI

- does not specify asynchronous events
  - try using EJB `@Asynchronous` observer methods
- does not specify timed events
- transaction aware events



# Interceptors

# Interceptors

## Diff #1 – the concept

### ■ Seam

- much of the functionality of Seam is implemented as a set of built-in Seam interceptors<sup>1</sup>
- Seam defines
  - its own API to create custom interceptor for JavaBean components,
  - and EJB 3.0 “adaptation layer”

### ■ CDI

- follows Interceptors 1.1 specification
  - part of EJB 3.1 spec<sup>2</sup>
- defines a typesafe mechanism for associating interceptors to beans using **interceptor bindings**

<sup>1</sup> See `org.jboss.seam.core.Init#DEFAULT_INTERCEPTORS`

<sup>2</sup> JSR 318

# Interceptors

## Diff #2 – binding and enablement

### ■ Seam

- bind to a component **with custom annotation**
- interceptors are registered and enabled automatically
- order is defined via `@Interceptor` annotation
  - around, within attributes

### ■ CDI

- bind to a bean **with custom annotation**
- an interceptor must be **explicitly enabled** by listing its class under the `<interceptors>` element of the `beans.xml` file for each bean archive<sup>1</sup>
- the order of the interceptor declarations determines the interceptor ordering

<sup>1</sup> <https://issues.jboss.org/browse/CDI-18>

# ~~Interceptors~~

## Diff #3 – decorators

### ■ Seam

- no such functionality is supported

### ■ CDI

- similar to interceptors<sup>1</sup>,
- but don't have the generality of an interceptor,
- intercept invocations only for a certain interface,
- and directly implement operations with business semantics

<sup>1</sup> See JSR 299 - Chapter 8. Decorators



# Questions?

# The End

Thanks for listening

## Resources:

- Seam 2 documentation: <http://docs.jboss.org/seam/latest/reference/en-US/html/>
- Seam 2 to Seam 3 Migration Notes: <http://seamframework.org/Seam3/Seam2ToSeam3MigrationNotes>
- CDI Specification (JSR 299): <http://jcp.org/en/jsr/summary?id=299>
- Weld documentation: <http://docs.jboss.org/weld/reference/latest/en-US/html/>
- Java EE 6 Specification (JSR 316): <http://jcp.org/en/jsr/summary?id=316>
- Weld, CDI and Proxies: <https://community.jboss.org/blogs/stuartdouglas/2010/10/12/weld-cdi-and-proxies>