JUDCon

JBoss Users & Developers Conference

London:2011

Weld-OSGi

Injecting easiness in OSGi

Mathieu ANCELIN

- Software engineer @SERLI
- Java & OSS guy
  - JOnAS, GlassFish, Weld, etc ...
  - Poitou-Charentes JUG crew member
- CDI 1.1 (JSR-346) expert group member
- What else?
- @TrevorReznik

A few words about SERLI

- Software engineering company based in France
- 65 people
- 80% of the business is Java-related
- Small company working for big ones
- OSS contribution : 10% of workforce
- www.serli.com @SerliFr

Before we start

#judcon

#weld-osgi

Agenda

- CDI, the best part of Java EE 6
- OSGi, deep dive into modularity and dynamism
- Meet Weld-OSGi
  - How does it work?
  - Features and programming model
  - Pros and cons
- Back to the future
- Demo
- Conclusion

CDI

CDI

@ApplicationScoped

@Inject

@SessionScoped

@Model

@Observes

@Any

@Default

@Dispose

@Named

@RequestScoped

@Produces

@Typed

@Singleton

@Stereotype

@Qualifier

@Scope

@New

CDI

@Any

@ApplicationScoped

@Model

@Default

@Inject

@Dispose

@Observes

@SessionScoped

@Named

@RequestScoped

@Produces

@Typed

@Singleton

@Stereotype

@Qualifier

@Scope

@New

CDI

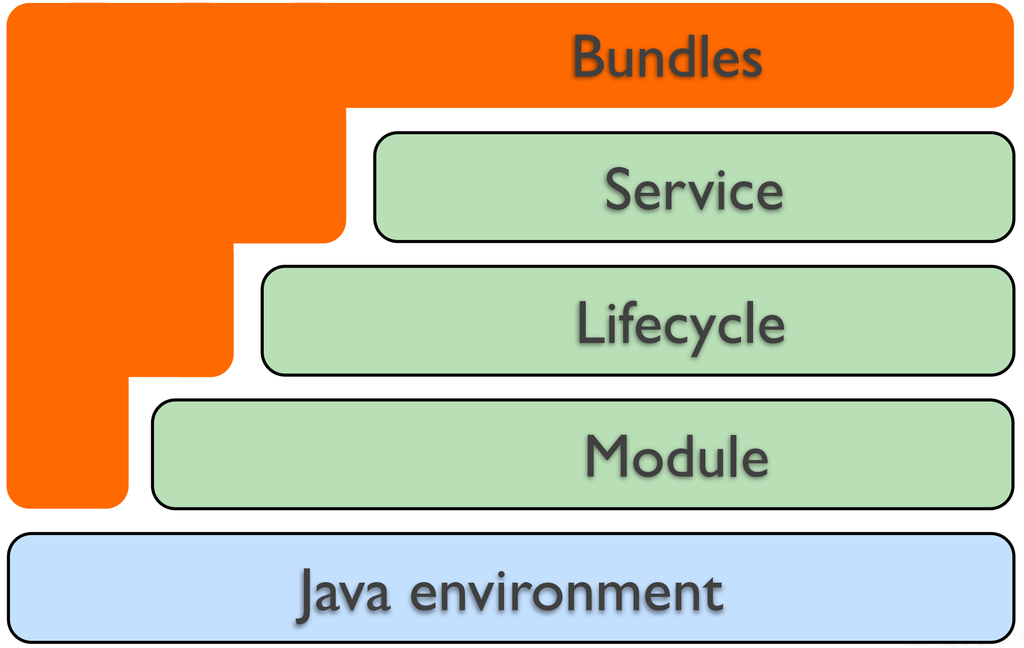
- The best part of Java EE 6 (coolest)
  - #annotationsEverywhere
- Basically there is no limite of what you can do
  - if you can think about it, you can do it
  - standard extensions :-)
- JBoss Weld is the reference implementation
  - pretty mature, good community
- Limited to Java EE 6?
  - well, not necessarily ...

Environnements for CDI/Weld

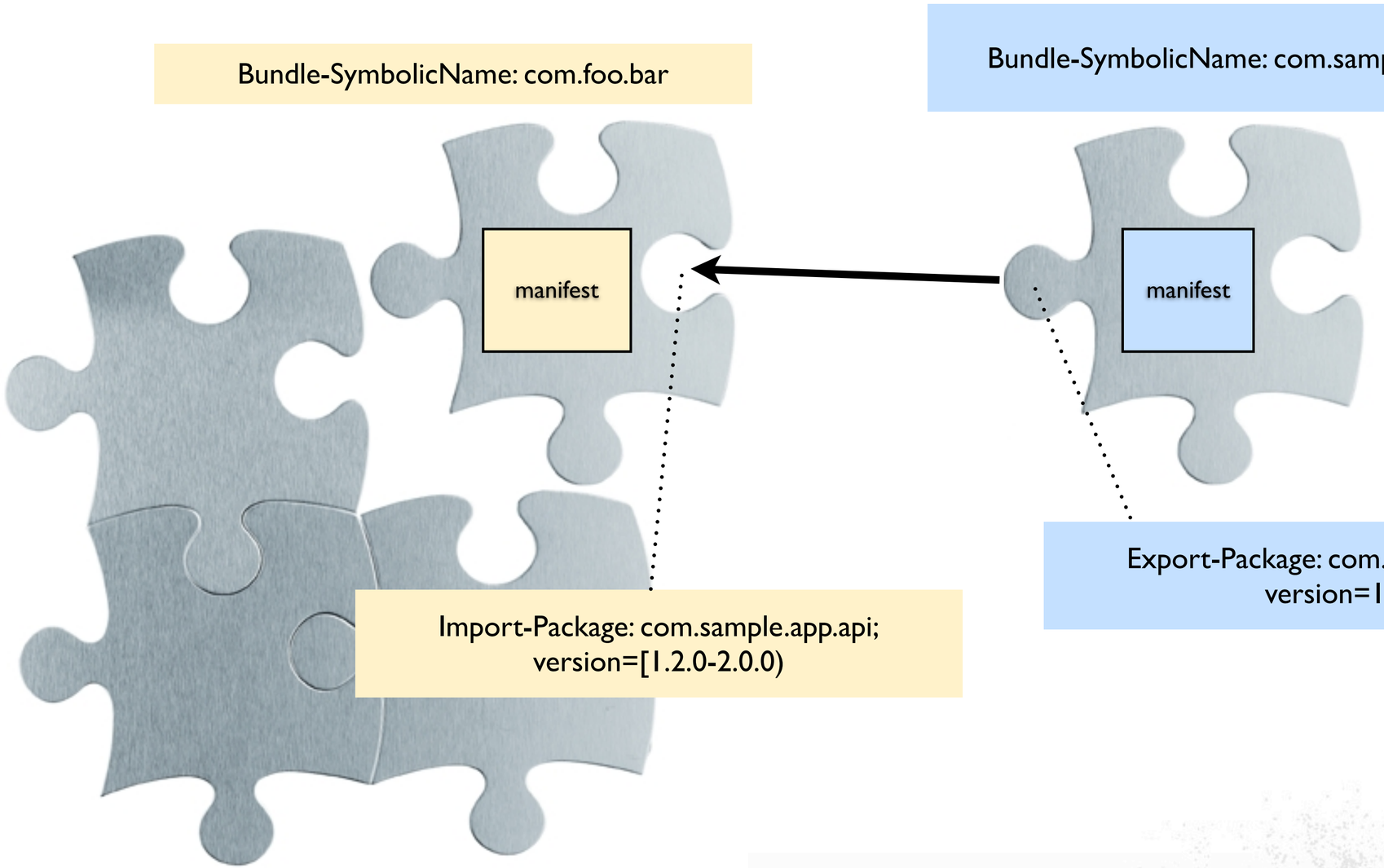
- You can boostrap Weld very easily outside Java EE environment
  - You can bootstrap it anywhere :-)
- For instance
  - Weld-Servlet
    - Jetty
    - Tomcat 6/7
  - Weld-SE
    - Good old Desktop Java apps.
  - You name it ?

OSGi

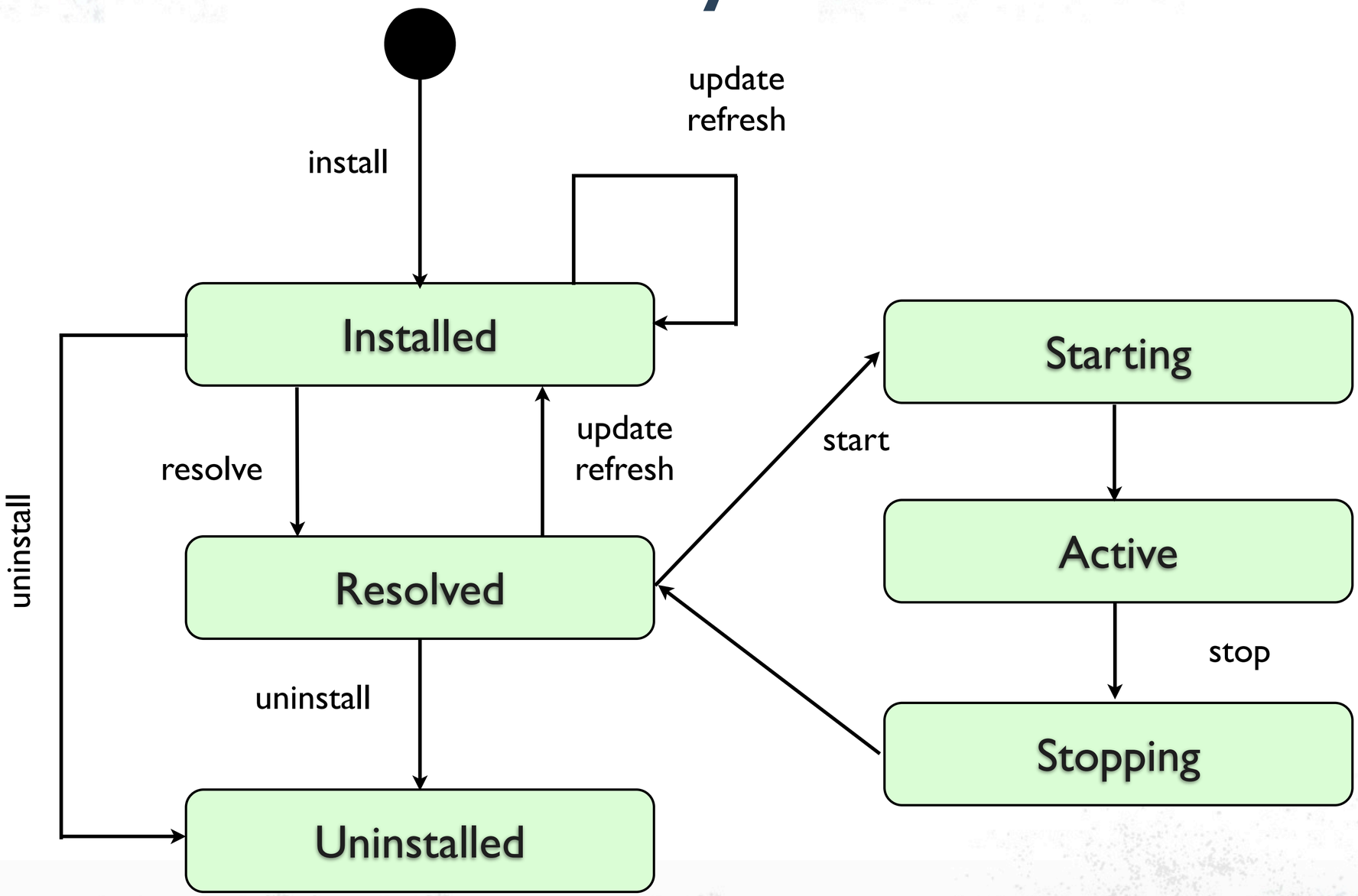
- An amazing modular and dynamic platform for Java
- Very stable and powerful but old APIs



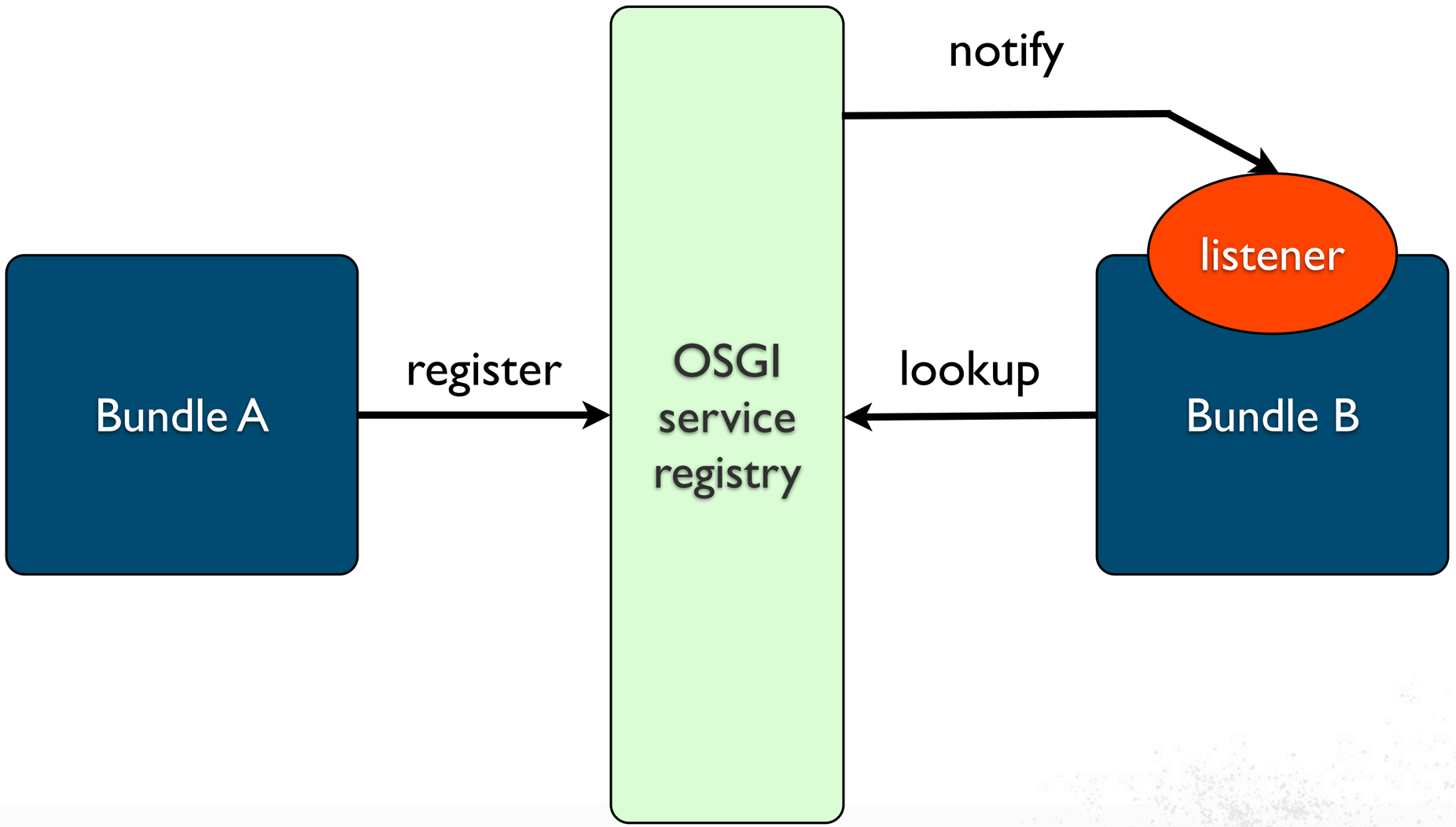
Modules / Bundles



Lifecycle


install
update
refresh
Installed
resolve
update
refresh
uninstall
Resolved
uninstall
Uninstalled
Starting
start
Active
stop
Stopping

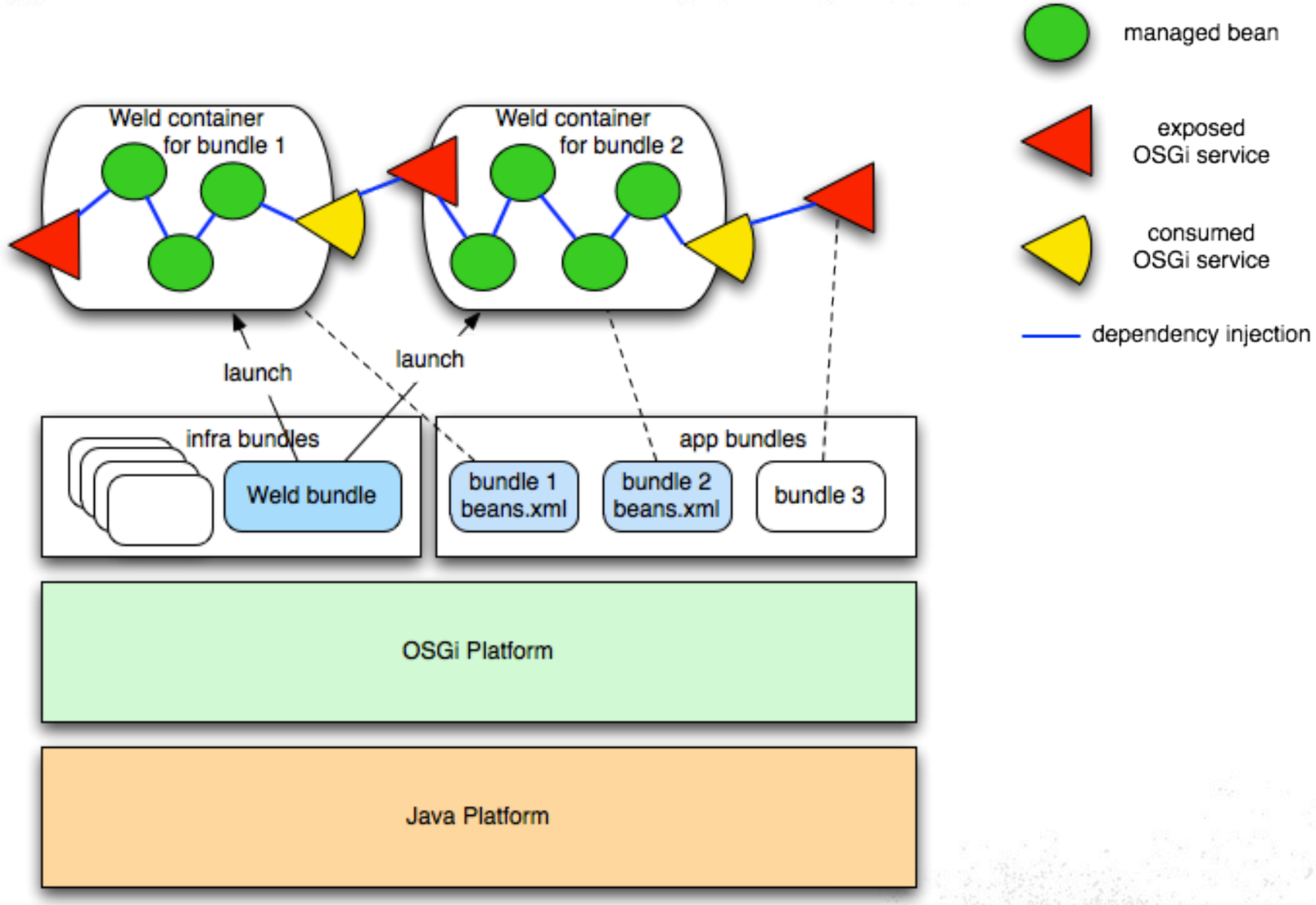
Services


notify
listener
Bundle A
register
OSGI
service
registry
lookup
Bundle B

Weld-OSGi

- (try to be) the best of both worlds
  - dynamic, typesafe, annotations, etc ...
- CDI extension to use CDI programming model inside OSGi
- A JBoss Weld project
  - need to bootstrap Weld in an OSGi environment
- Developed by SERLI R&D team
  - Mathieu & Matthieu
- You don't need to know OSGi
  - make the OSGi programming model disappear in favor of standard CDI
  - but still compatible

How does it work ?



Features

- Use Weld/CDI inside OSGi environment
- OSGi injection utilities
- Dynamic services publication
- Dynamic services injection
- Integration with OSGi events API
- Inter-bundles events

Using Weld / CDI in OSGi

- Install a bundle with META-INF/beans.xml file
- If you don't need automatic startup
  - Specify that Weld-OSGi must not handle the bundle
    - entry in the bundle manifest : Embedded-CDIContainer: true
  - Specification of an embedded mode in CDI 1.1
  - Special Weld-OSGi events

```
public void start(@Observes BundleContainerInitialized event) {}

public void stop(@Observes BundleContainerShutdown event) {}
```

Embedding CDI

```
EmbeddedCDIContainer cdi =
    new EmbeddedContainer(bundleContext).initialize();
MyService service =
    cdi.instance().select(MyService.class).get();
service.doSomething();
cdi.stop();
```

or

```
WeldContainer weld =
    new WeldContainer(bundleContext).initialize();
MyService service =
    weld.instance().select(MyService.class).get();
service.doSomething();
weld.stop();
```

OSGi injection utilities

- Easier access to OSGi APIs (if needed)
  - Injection of the current bundle
  - Injection of the current bundleContext
  - Injection of the current metadata
  - Injection bundle files (inside OSGi container)
- Other utilities are added while moving forward

OSGi injection utilities

- Easier access to OSGi APIs (if needed)
  - Injection of the current bundle
  - Injection of the current bundleContext
  - Injection of the current metadata
  - Injection bundle files (inside OSGi container)
- Other utilities are added while moving forward

```
@Inject Bundle bundle;
@Inject BundleContext context;
@Inject @BundleHeaders Map<String, String> headers;
@Inject @BundleHeader("Bundle-SymbolicName") String symbolicName;
@Inject @BundleDataFile("text.txt") File text;
```

Services publication

Services publication

- Declarative publication

```
@Publish
@ApplicationScoped
@Lang(EN)
public class MyServiceImpl implements MyService {
    ...
}
```

Services publication

- Declarative publication

```
@Publish
@ApplicationScoped
@Lang(EN)
public class MyServiceImpl implements MyService {
    ...
}
```

- Dynamic publication

```
@Inject Instance<MyService> instance;
@Inject ServiceRegistry registry;
MyService service = instance.get();
Registration<MyService> reg = registry.register(service);
...
reg.unregister();
```

Services injection

Services injection

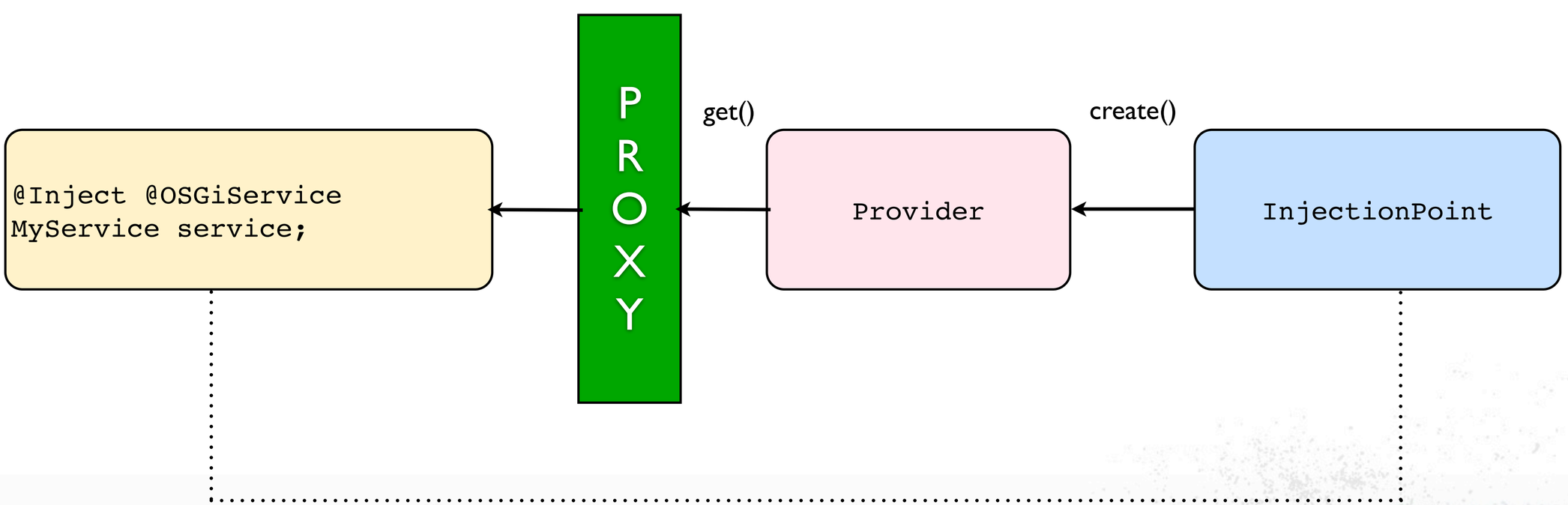
- Dynamic injection

```
@Inject @OSGiService MyService service;
service.doSomething(); // fail if no services available
```

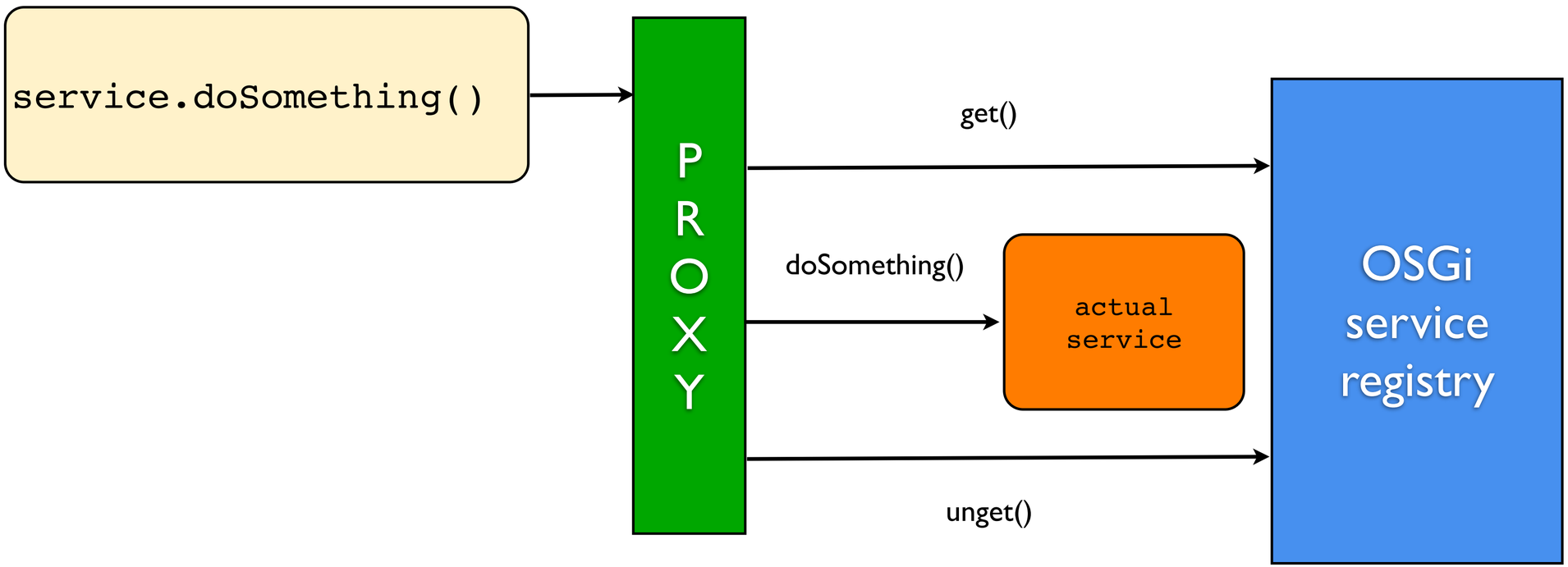
Services injection

- Dynamic injection

```
@Inject @OSGiService MyService service;
service.doSomething(); // fail if no services available
```



Services injection



Services injection

- Programmatic injection – whiteboard pattern (like Instance<T>)

```
@Inject Service<MyService> service;

for (MyService actualService : service.first()) {
    actualService.doSomething(); // called on 0-1 service
}
for (MyService actualService : service) {
    actualService.doSomething(); // called on 0-n service(s)
}
service.get().doSomething(); // can fail, not dynamic
service.size();
service.isUnsatisfied();
service.isAmbiguous();
```

Services injection – filters

```
@Publish
@Lang(EN) @Country(US)
public class MyServiceImpl implements MyService {
    ...
}
```

Services injection – filters

```
@Publish
@Lang(EN) @Country(US)
public class MyServiceImpl implements MyService {
    ...
}


@Inject @OSGiService
        @Filter("(&(lang=*)(country=US))") MyService service;
```

Services injection – filters

```
@Publish
@Lang(EN) @Country(US)
public class MyServiceImpl implements MyService {
    ...
}
```

```
@Inject @Filter("(&(lang=*)(country=US))")
                    Service<MyService> service;
```

Services injection – filters

```
@Publish
@Lang(EN) @Country(US)
public class MyServiceImpl implements MyService {
    ...
}
```

```
@Inject @Filter("(&(lang=*)(country=US))")
                    Service<MyService> service;
```

```
@Inject @OSGiService @Lang(EN) @Country(US) MyService service;
```

Services injection – filters

```
@Publish
@Lang(EN) @Country(US)
public class MyServiceImpl implements MyService {
    ...
}
```

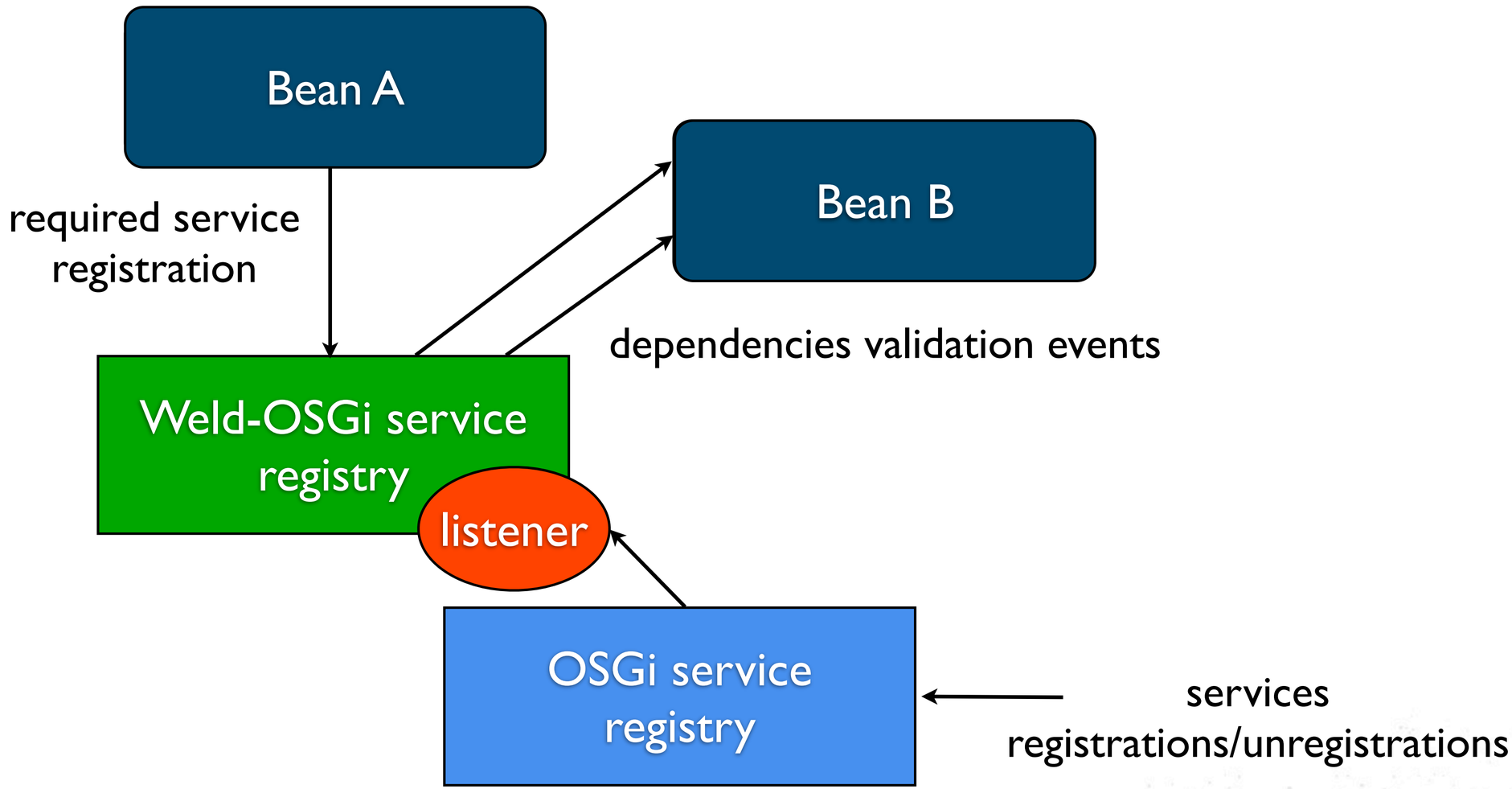
```
@Inject @Filter("(&(lang=*)(country=US))")
                Service<MyService> service;
```

```
@Inject @Lang(EN) @Country(US) Service<MyService> service;
```

Required services

- When you absolutely need specific service(s) at runtime
- Weld-OSGi tell you when required services are available
  - can work in an atomic fashion for the whole bundle
  - can target specific types of services

Required services



Required services

```
@Inject @OSGiService @Required MyService service;
@Inject @OSGiService @Required MyBean bean;

public void start(@Observes Valid evt) {
    System.out.println("services are available");
    service.doSomething();
}

public void stop(@Observes Invalid evt) {
    System.out.println("services are unavailable");
}
```

Required services

```
@Inject @Required Service<MyService> service;
@Inject @Required Service<MyBean> bean;

public void start(@Observes Valid evt) {
    System.out.println("services are available");
    service.get().doSomething();
}

public void stop(@Observes Invalid evt) {
    System.out.println("services are unavailable");
}
```

Required services

```
@Inject @Required Service<MyService> service;
@Inject @Required Service<MyBean> bean;

public void start(@Observes Valid evt) {
    System.out.println("services are available");
    service.get().doSomething();
}

public void stop(@Observes Invalid evt) {
    System.out.println("services are unavailable");
}
```

Required services

```
@Inject @OSGiService @Required MyService service;

public void start(@Observes @Specification(MyService.class)
                                        ServiceAvailable evt) {
    System.out.println("service is available");
    service.doSomething();
}

public void stop(@Observes @Specification(MyService.class)
                                      ServiceUnavailable evt) {
    System.out.println("service is unavailable");
}
```

Required services

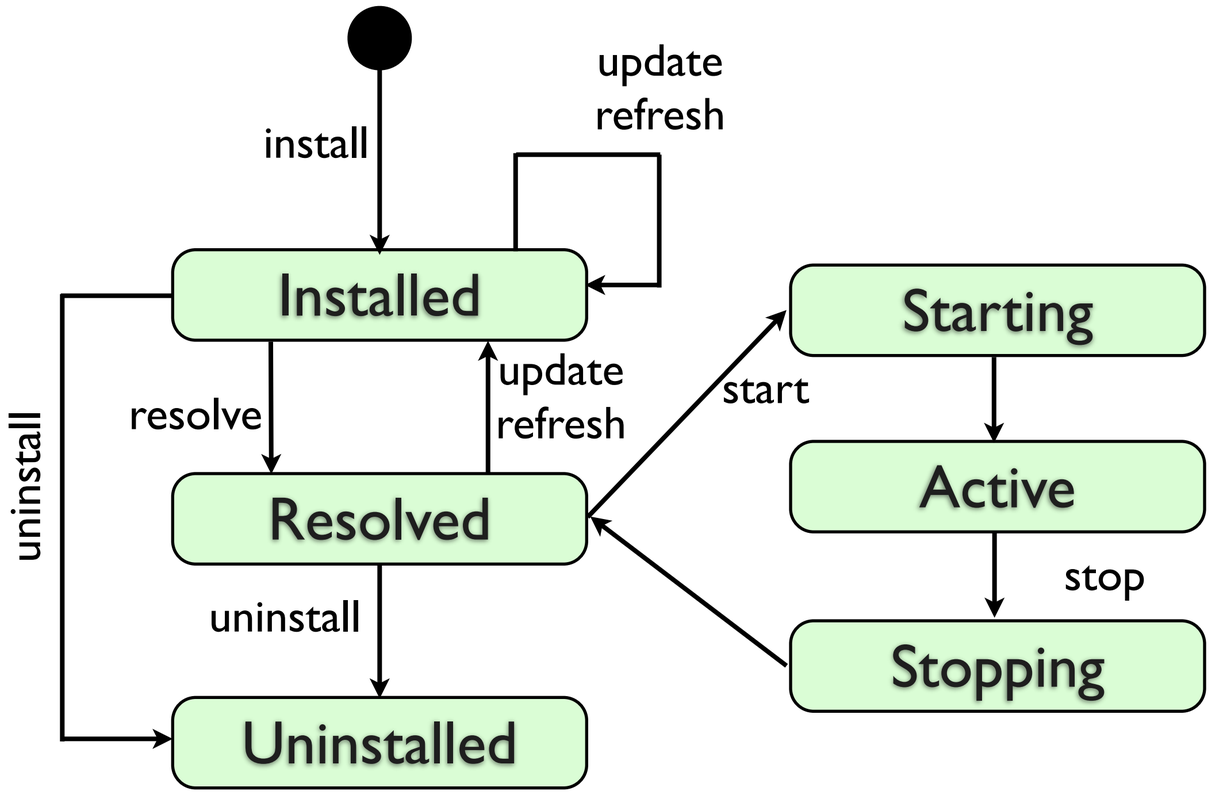
```
@Inject @Required Service<MyService> service;

public void start(@Observes @Specification(MyService.class)
                                        ServiceAvailable evt) {
    System.out.println("service is available");
    service.get().doSomething();
}

public void stop(@Observes @Specification(MyService.class)
                                      ServiceUnavailable evt) {
    System.out.println("service is unavailable");
}
```

OSGi events API

- OSGi generates a lot of events to let you interact with bundle and service layers
- Easier to handle dynamism with
  - bundle events
  - service events

OSGi events – Bundles

- Available events
  - BundleInstalled
  - BundleResolved
  - BundleStarting
  - BundleStarted
  - BundleStopping
  - BundleStopped
  - BundleUninstalled
  - BundleUpdated
  - BundleUnresolved



OSGi events – Bundles

```
public void bindBundle(@Observes BundleInstalled evt) {}
```

OSGi events – Bundles

```
public void bindBundle(@Observes @BundleVersion("1.2.3")
                                    BundleInstalled evt) {}
```

OSGi events – Bundles

```
public void bindBundle(@Observes @BundleName("com.foo.bar")
                                        BundleInstalled evt) {}
```

OSGi events – Bundles

```
public void bindBundle(@Observes @BundleName("com.foo.bar")
                       @BundleVersion("1.2.3") BundleInstalled evt) {}
```

OSGi events – Services

- Available events
  - ServiceArrival
  - ServiceDeparture
  - ServiceChanged

OSGi events – Services

- Available events
  - ServiceArrival
  - ServiceDeparture
  - ServiceChanged

```
void bindService(@Observes ServiceArrival evt) {}
```

OSGi events – Services

- Available events
  - ServiceArrival
  - ServiceDeparture
  - ServiceChanged

```
void bindService(@Observes @Filter("(lang=US)") ServiceArrival evt) {}
```

OSGi events – Services

- Available events
  - ServiceArrival
  - ServiceDeparture
  - ServiceChanged

```
void bindService(@Observes @Specification(MyService.class)
                                          ServiceArrival evt) {}
```

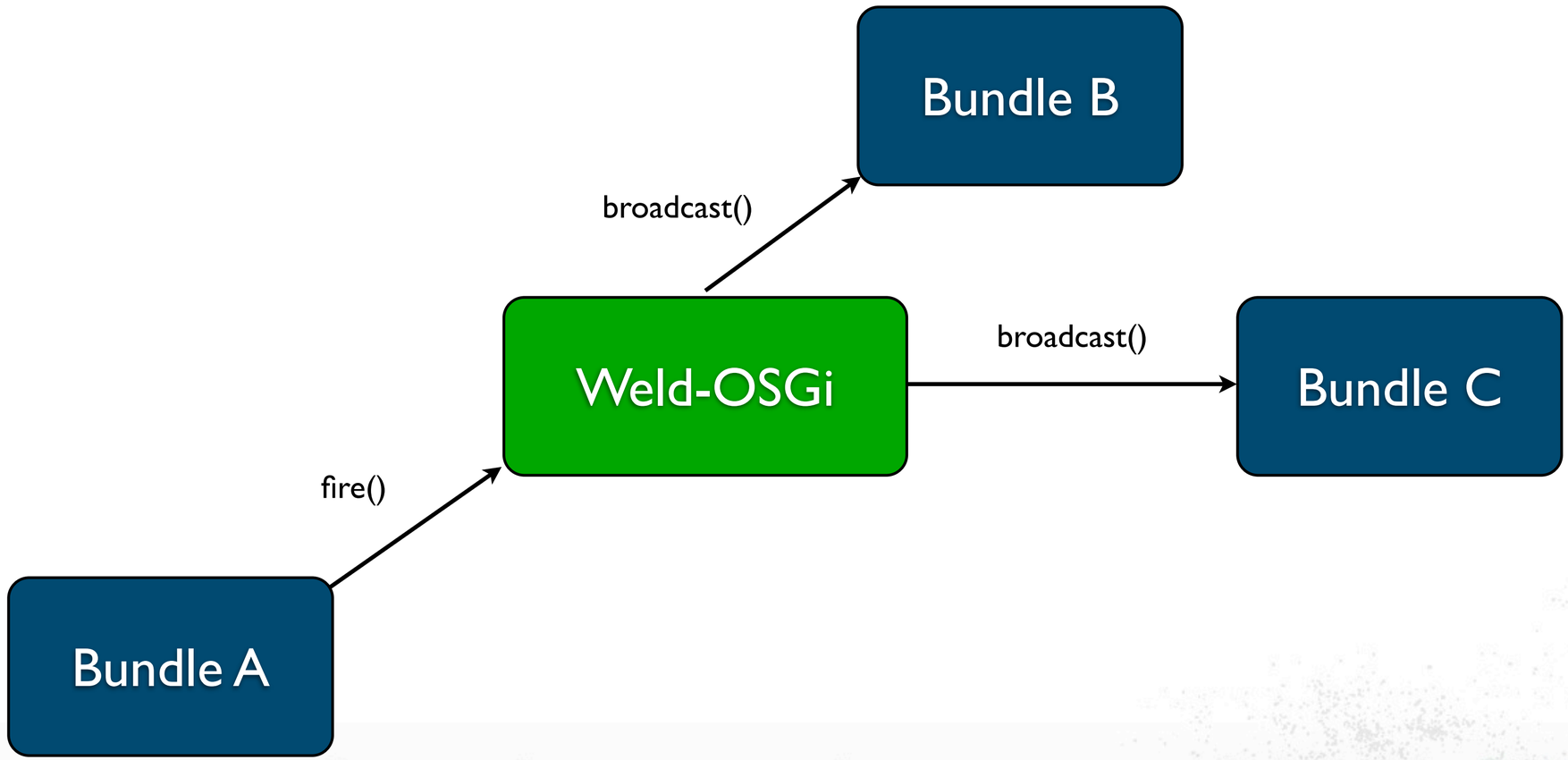
OSGi events – Services

- Available events
  - ServiceArrival
  - ServiceDeparture
  - ServiceChanged

```
void bindService(@Observes @Specification(MyService.class)
                           @Filter("(lang=US)") ServiceArrival evt) {}
```

Inter-bundles events

- Communication between bundles (managed by Weld-OSGi) with standard CDI events



Inter-bundles events

```
@Inject Event<InterBundleEvent> event;
event.fire(new InterBundleEvent("Hello bundles"));
```

Inter-bundles events

```
@Inject Event<InterBundleEvent> event;
event.fire(new InterBundleEvent("Hello bundles"));
```

Inter-bundles events

```
@Inject Event<InterBundleEvent> event;
event.fire(new InterBundleEvent("Hello bundles"));
```

```
public void listen(@Observes InterBundleEvent event) {}
```

Inter-bundles events

```
@Inject Event<InterBundleEvent> event;
event.fire(new InterBundleEvent("Hello bundles"));
```

```
public void listen(@Observes @Sent InterBundleEvent event) {}
```

Inter-bundles events

```
@Inject Event<InterBundleEvent> event;
event.fire(new InterBundleEvent("Hello bundles"));
```

```
public void listen(@Observes @Specification(String.class)
                                    InterBundleEvent event) {}
```

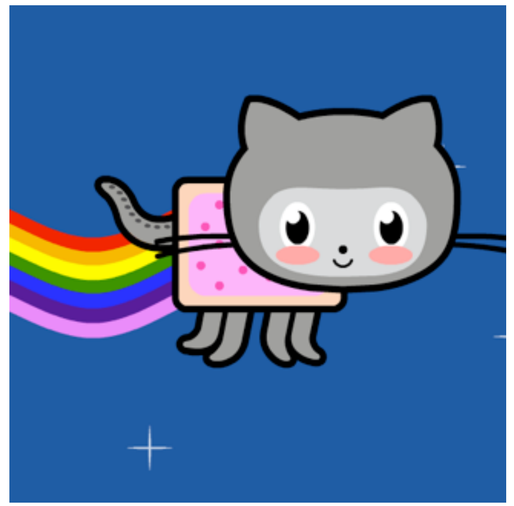
Inter-bundles events

```
@Inject Event<InterBundleEvent> event;
event.fire(new InterBundleEvent("Hello bundles"));
```

```
public void listen(@Observes @Specification(String.class)
                                @Sent InterBundleEvent event) {}
```

Getting started !

- Get Weld-OSGi =>
- Write an OSGi bundle
  - Maven module + maven-bundle-plugin
  - bnd file for manifest customization
- Empty beans.xml file in META-INF



```
<dependency>
    <groupId>org.jboss.weld.osgi</groupId>
    <artifactId>weld-osgi-core-api</artifactId>
    <version>1.1.3-SNAPSHOT</version>
</dependency>
<dependency>
    <groupId>javax.enterprise</groupId>
    <artifactId>cdi-api</artifactId>
    <version>1.0-SP4</version>
</dependency>
```

Pros and cons

- Pros
  - CDI programming model
  - really simplify OSGi (service layer)
    - don't hide it though
    - fully compatible with existing OSGi bundles
      - mixed app (legacy, weld-osgi)
    - one Weld container per bundle
- Cons
  - one Weld container per bundle
  - new API to learn

Back to the future !

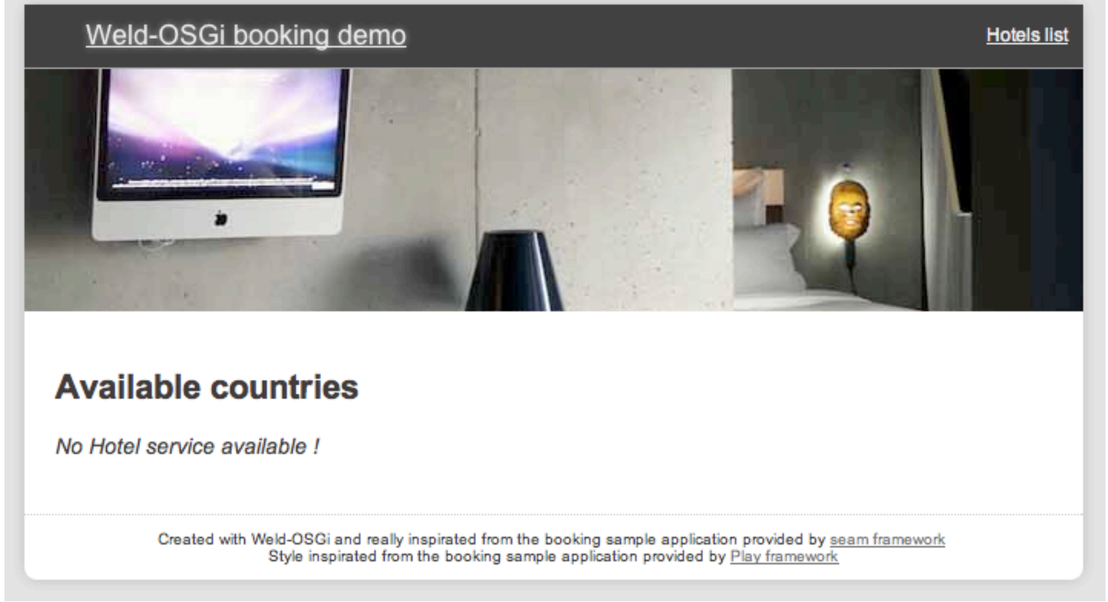
- Integration in Weld core (in progress)
- Forge plugin
  - integration with Weld-OSGi features
    - simplifying OSGi tests (arquillian OSGi)
    - generation of sample OSGi containers
- CDI Extension for hybrid Java EE app servers
  - using Weld-OSGi in Java EE apps
  - work in progress ;-)
- Integration with OSGi enterprise specs

Demo

Murphy's law in action

Demo : the story

- Hotel booking webapp
  - business partners with hotels
- Avoid redeploying the app
  - when new partner is added
  - using OSGi dynamism
- Provide an API to partners
  - Hotel POJO
  - HotelProvider service contract
  - partners will provide an OSGi bundle to deal with their booking system



Conclusion

- Weld-OSGi is cool
  - let's try it :-)
- Can help to change OSGi in people minds
- Enlarge CDI action scope
  - it's not only about Java EE
- Don't hesitate to give us feedback and fill issues

Questions?