Hacking Java EE

Pete Muir (@plmuir)

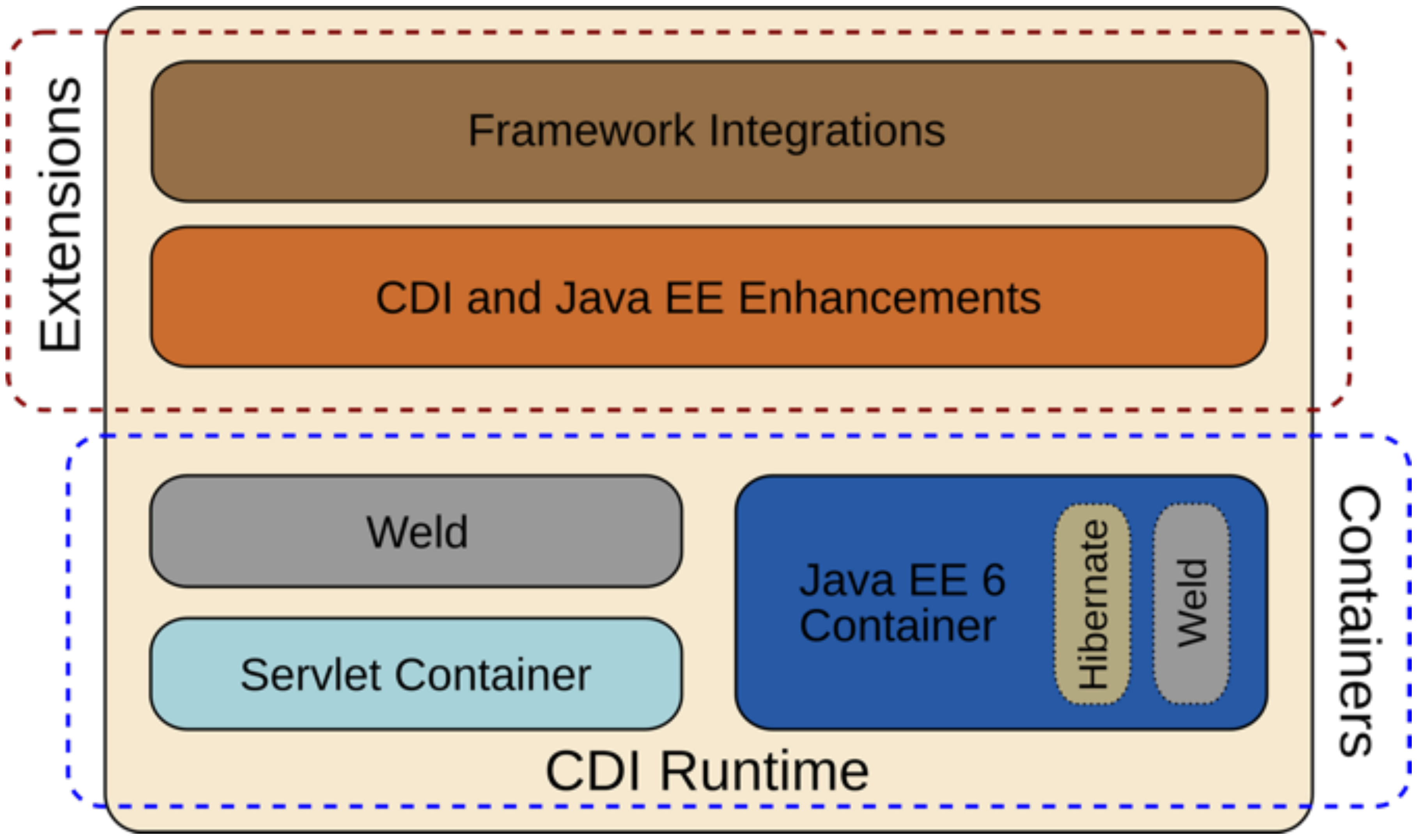
Principal Software Engineer
Red Hat, Inc.

June 2011

Building on Common Ground

Pete Muir

Building on Common Ground

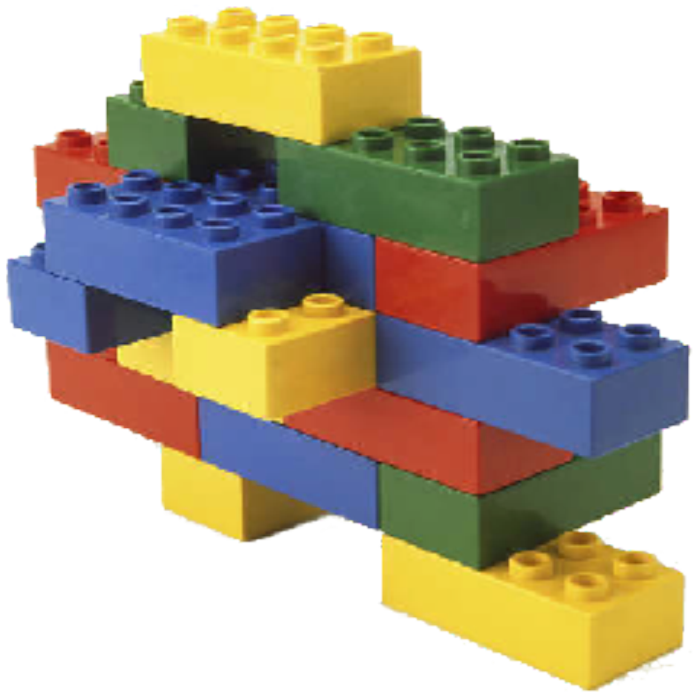


Pete Muir

Portable Extensions
extend the
programming model

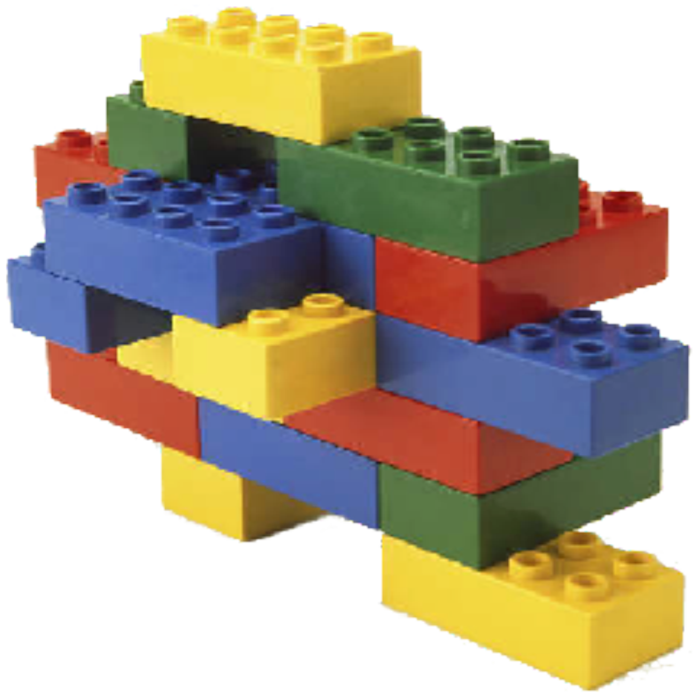
SPI for hacking Java EE:

- register additional beans
- satisfy injection points
- introduce custom scopes
- augment or override bean metadata



SPI for hacking Java EE:

- ✓ register additional beans
- ✓ satisfy injection points
- ✓ introduce custom scopes
- ✓ augment or override bean metadata



How does it work?

Based on Service Provider Interface (SPI) pattern

How does it work?

Based on Service Provider Interface (SPI) pattern

1. Implement empty Extension interface

How does it work?

Based on Service Provider Interface (SPI) pattern

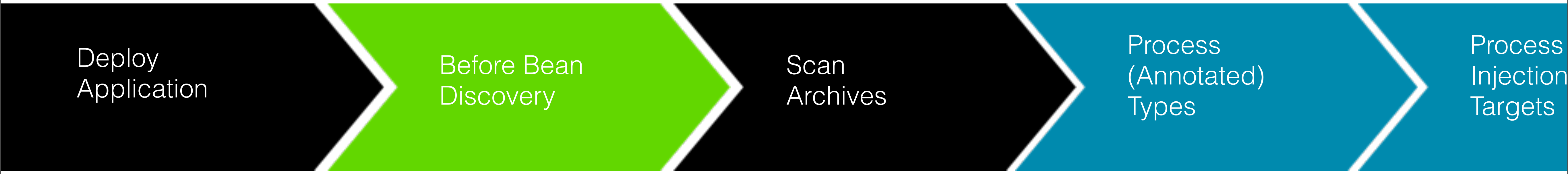
1. Implement empty Extension interface
2. Observe container events to alter deployment

How does it work?

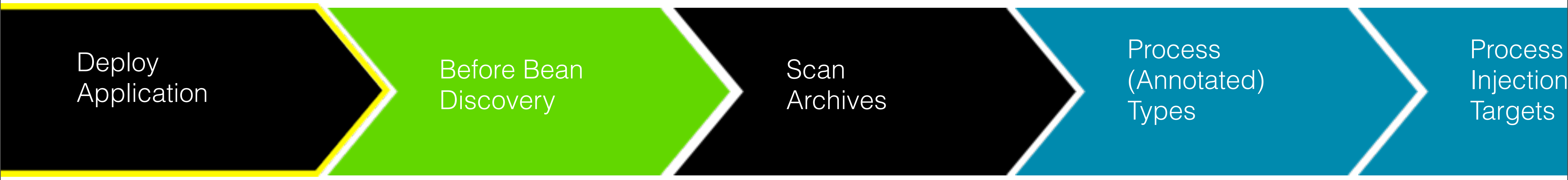
Based on Service Provider Interface (SPI) pattern

1. Implement empty Extension interface
2. Observe container events to alter deployment
3. Register extension

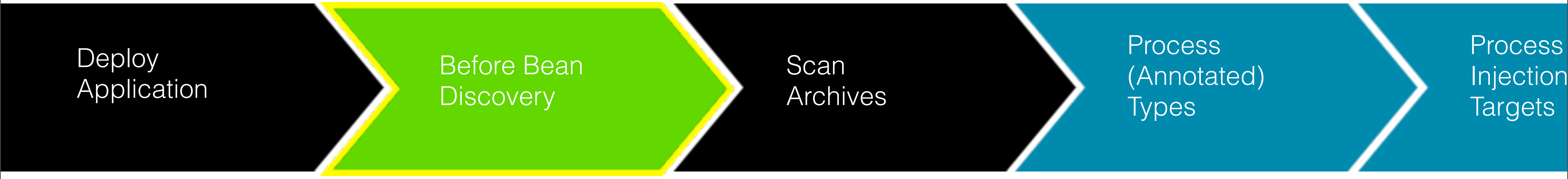
CDI deployment lifecycle with possible activities

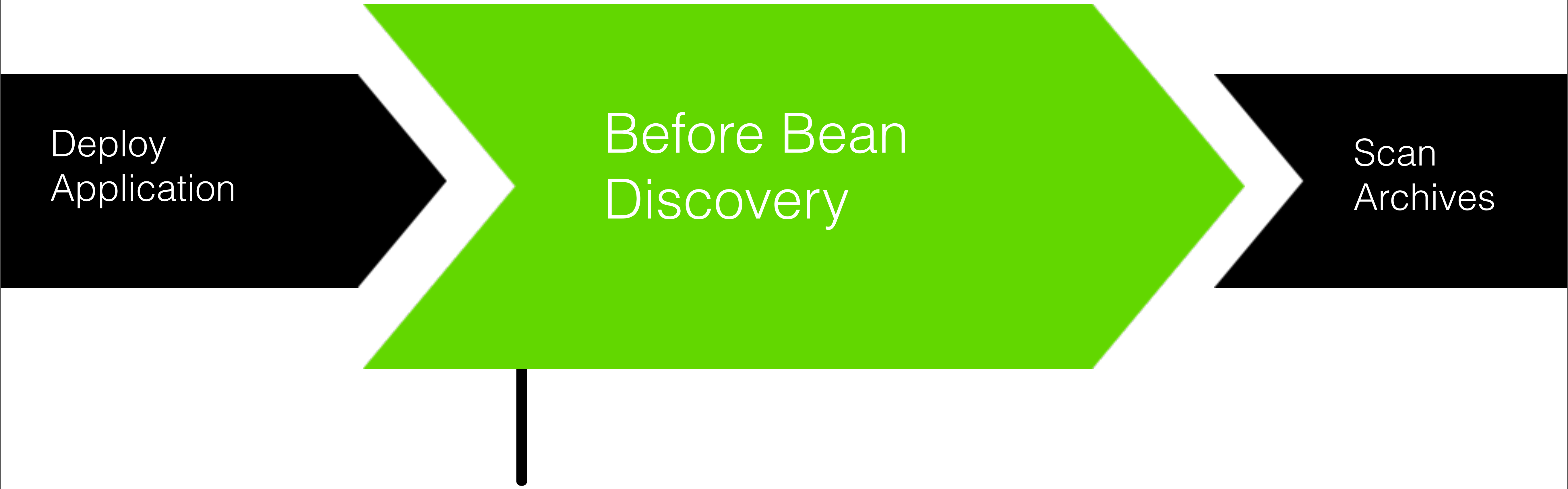


CDI deployment lifecycle with possible activities

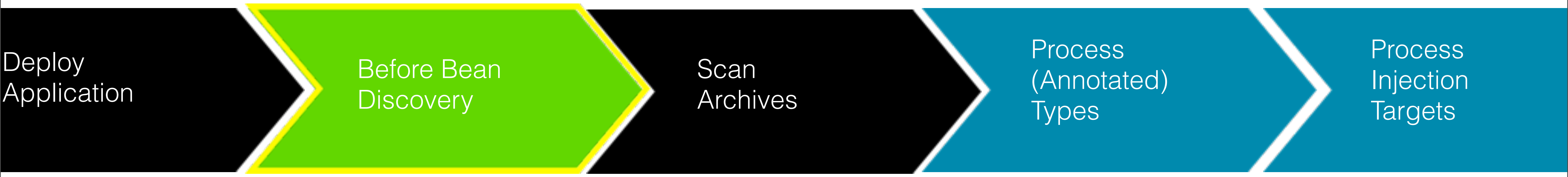


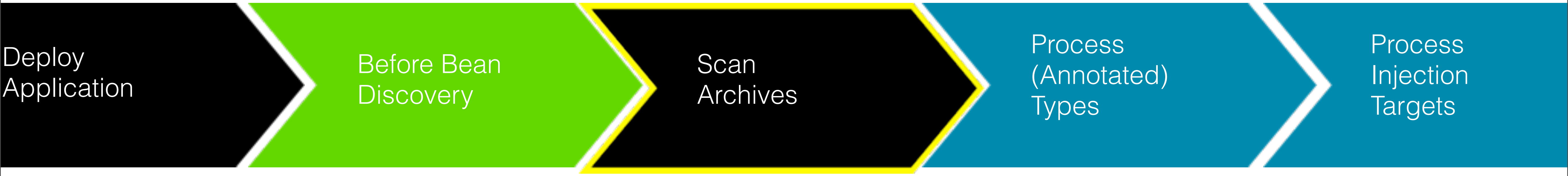
CDI deployment lifecycle with possible activities

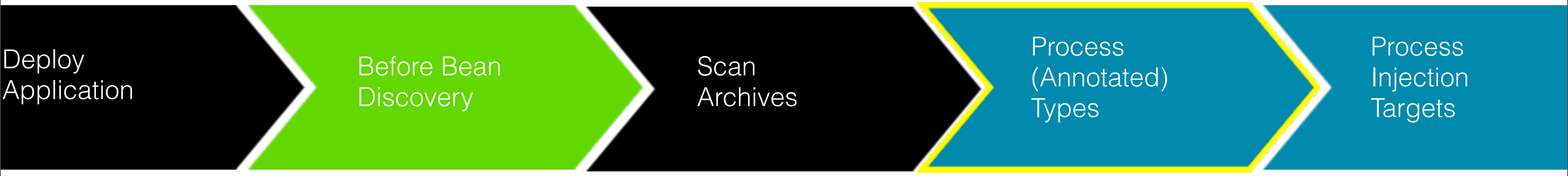


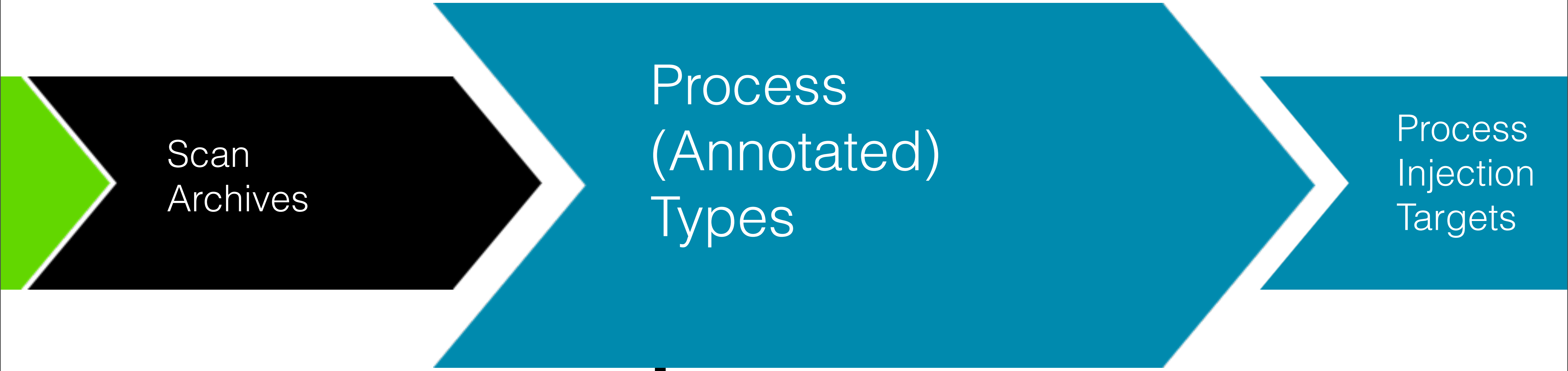


- add scope
- add annotated type
- add qualifier
- add interceptor binding
- add stereotype

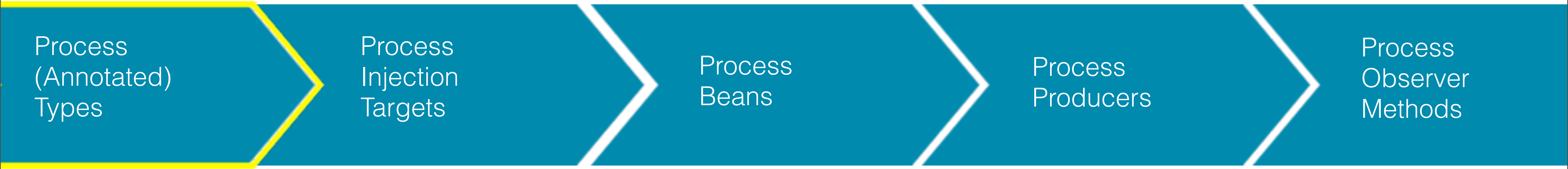

Deploy Application
Before Bean Discovery
Scan Archives
Process (Annotated) Types
Process Injection Targets

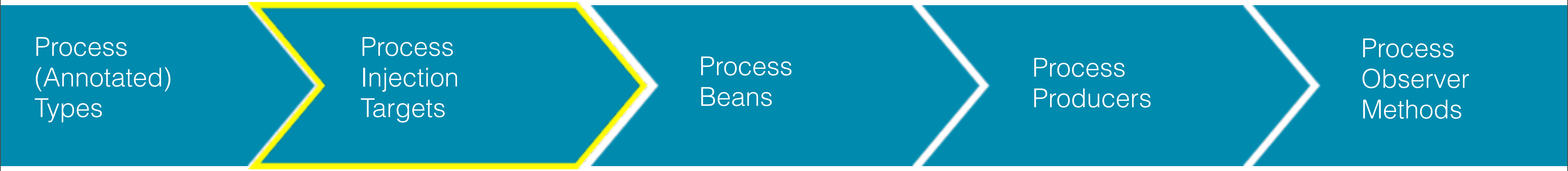

Deploy Application
Before Bean Discovery
Scan Archives
Process (Annotated) Types
Process Injection Targets

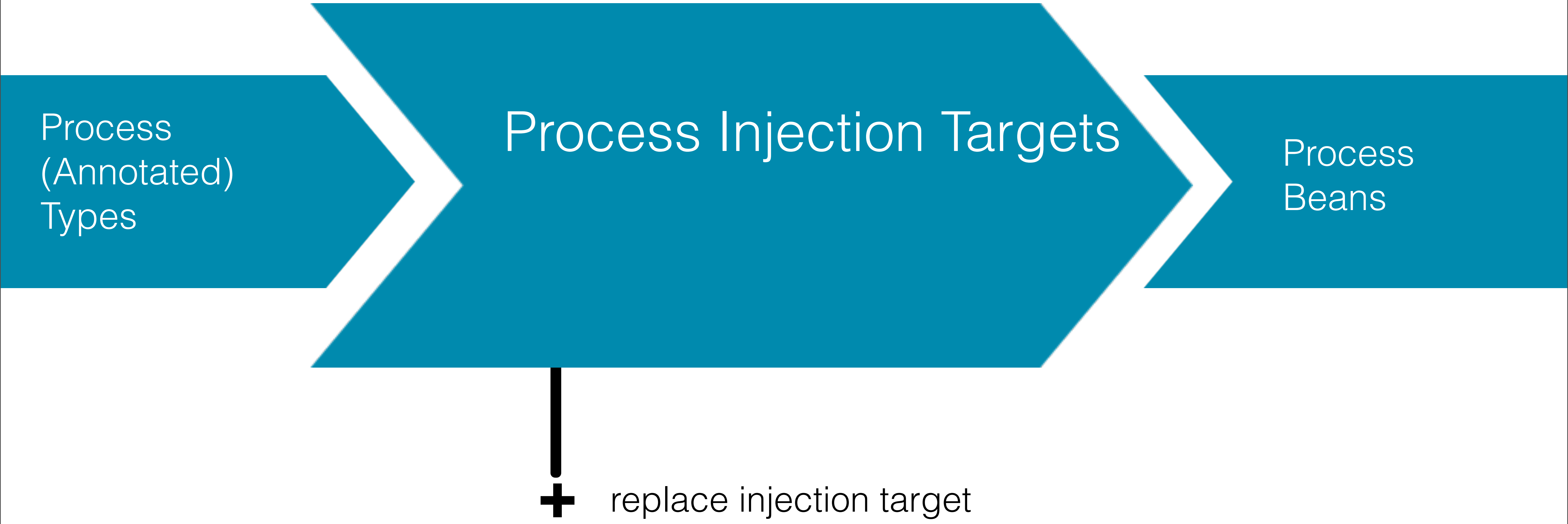

Deploy Application
Before Bean Discovery
Scan Archives
Process (Annotated) Types
Process Injection Targets

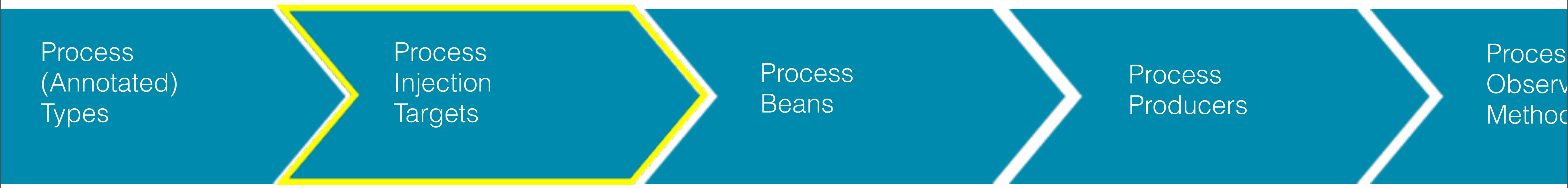


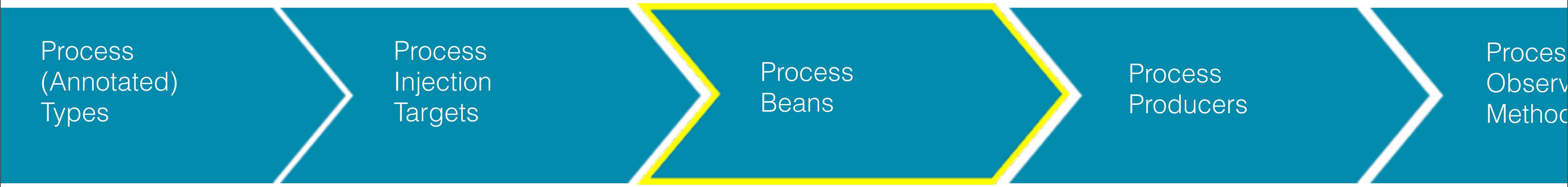
- veto types and prevent further processing
- replace annotated type

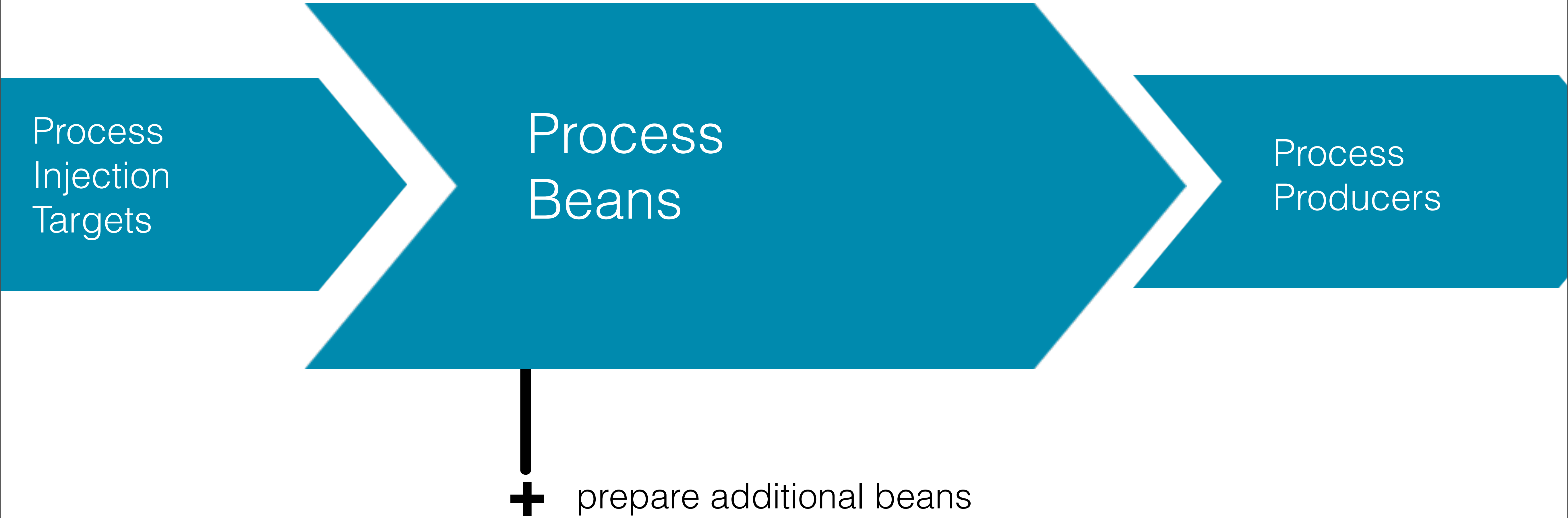

Process (Annotated) Types
Process Injection Targets
Process Beans
Process Producers
Process Observer Methods

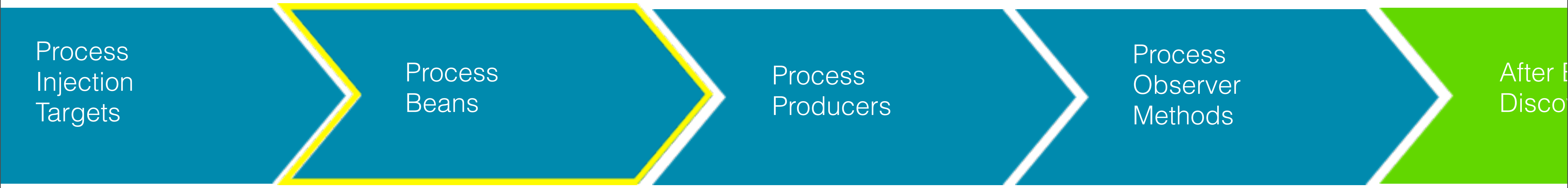

Process (Annotated) Types
Process Injection Targets
Process Beans
Process Producers
Process Observer Methods

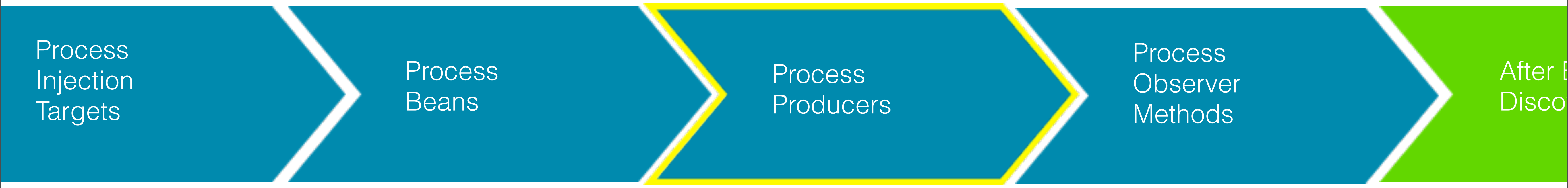

Process (Annotated) Types
Process Injection Targets
Process Beans
replace injection target

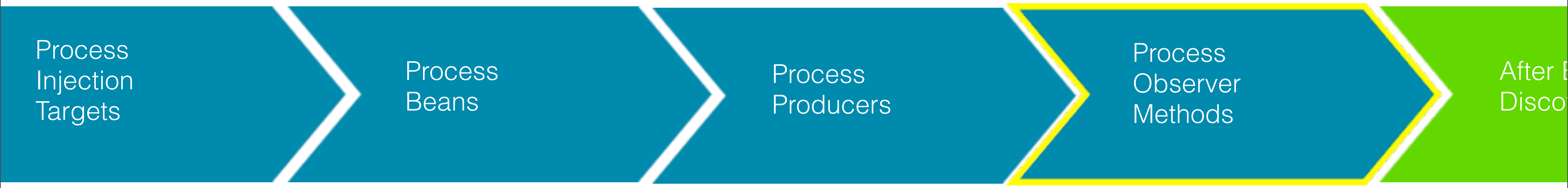

Process (Annotated) Types
Process Injection Targets
Process Beans
Process Producers
Proces Observ Methoc

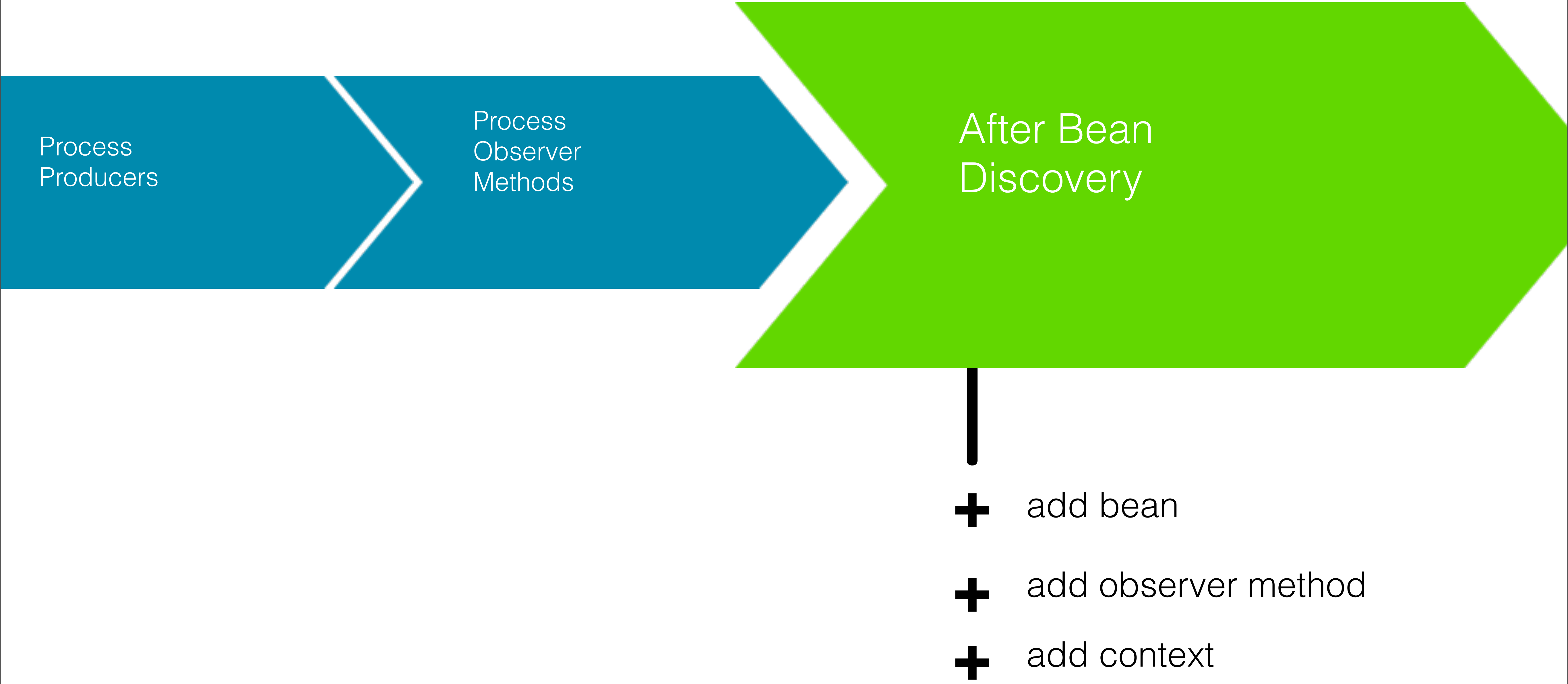

Process (Annotated) Types
Process Injection Targets
Process Beans
Process Producers
Proces Observ Methoc

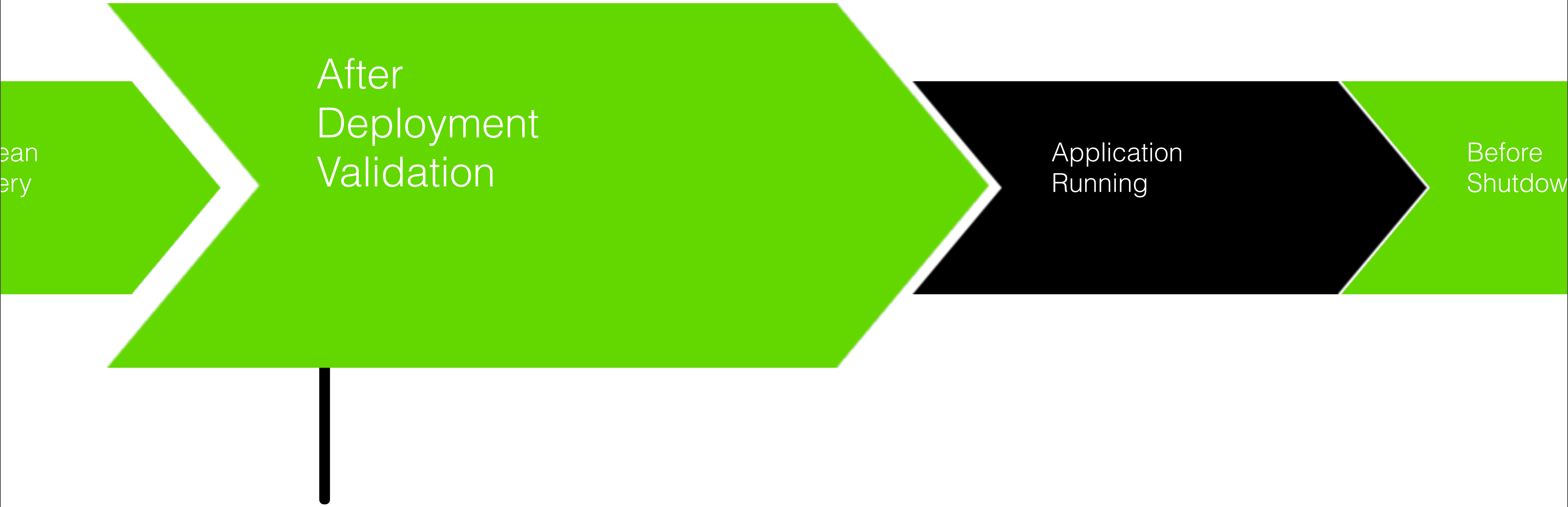

Process
Injection
Targets
Process
Beans
Process
Producers
+ prepare additional beans


Process Injection Targets
Process Beans
Process Producers
Process Observer Methods
After E Disco

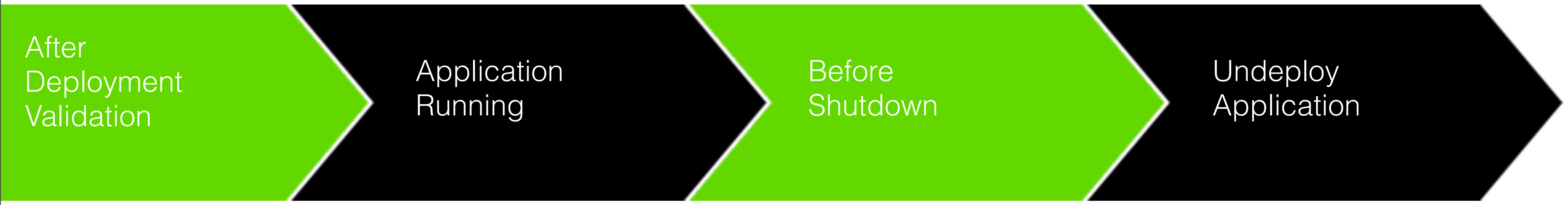

Process Injection Targets
Process Beans
Process Producers
Process Observer Methods
After E
Disco


Process
Injection
Targets
Process
Beans
Process
Producers
Process
Observer
Methods
After E
Disco

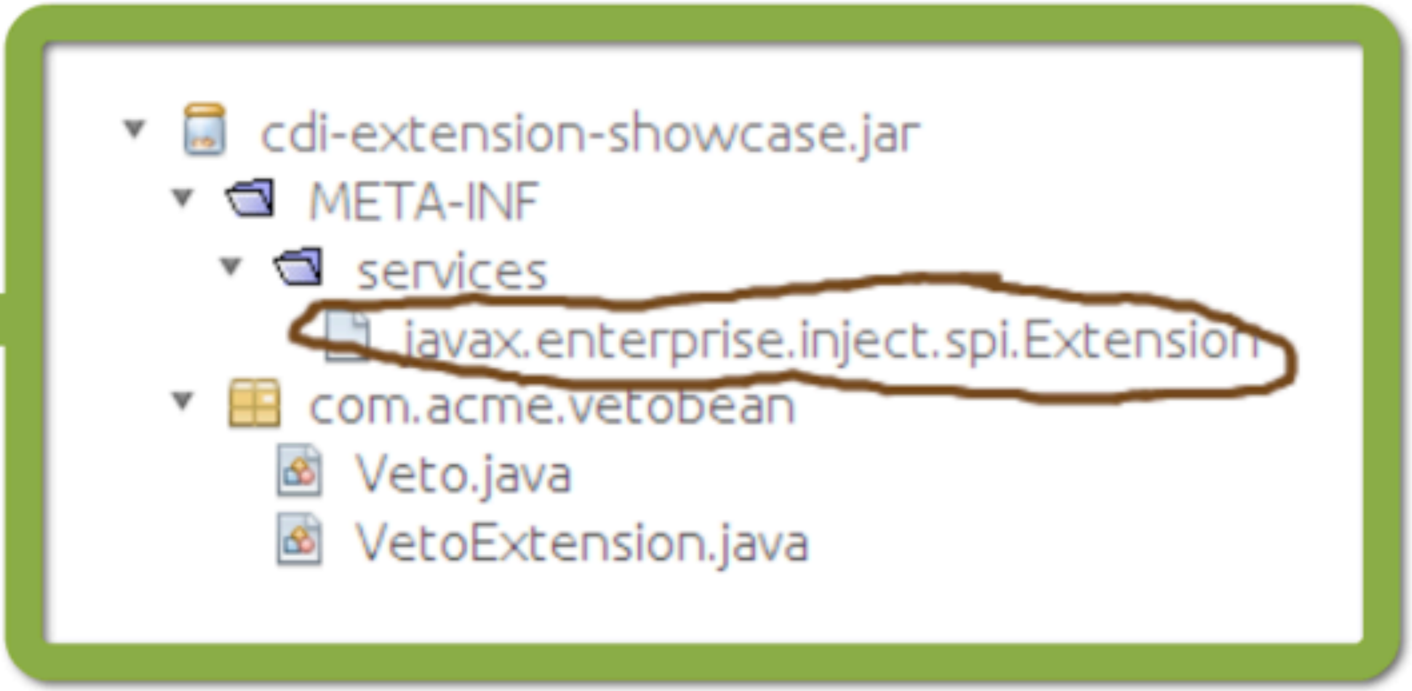

Process Producers
Process Observer Methods
After Bean Discovery
add bean
add observer method
add context



- assessment
- cleanup


After Deployment Validation
Application Running
Before Shutdown
Undeploy Application

When is an extension recognized?



```
cdi-extension-showcase.jar
  META-INF
    services
      javax.enterprise.inject.spi.Extension
  com.acme.vetobean
    Veto.java
    VetoExtension.java
```

META-INF/services/javax.enterprise.inject.spi.Extension

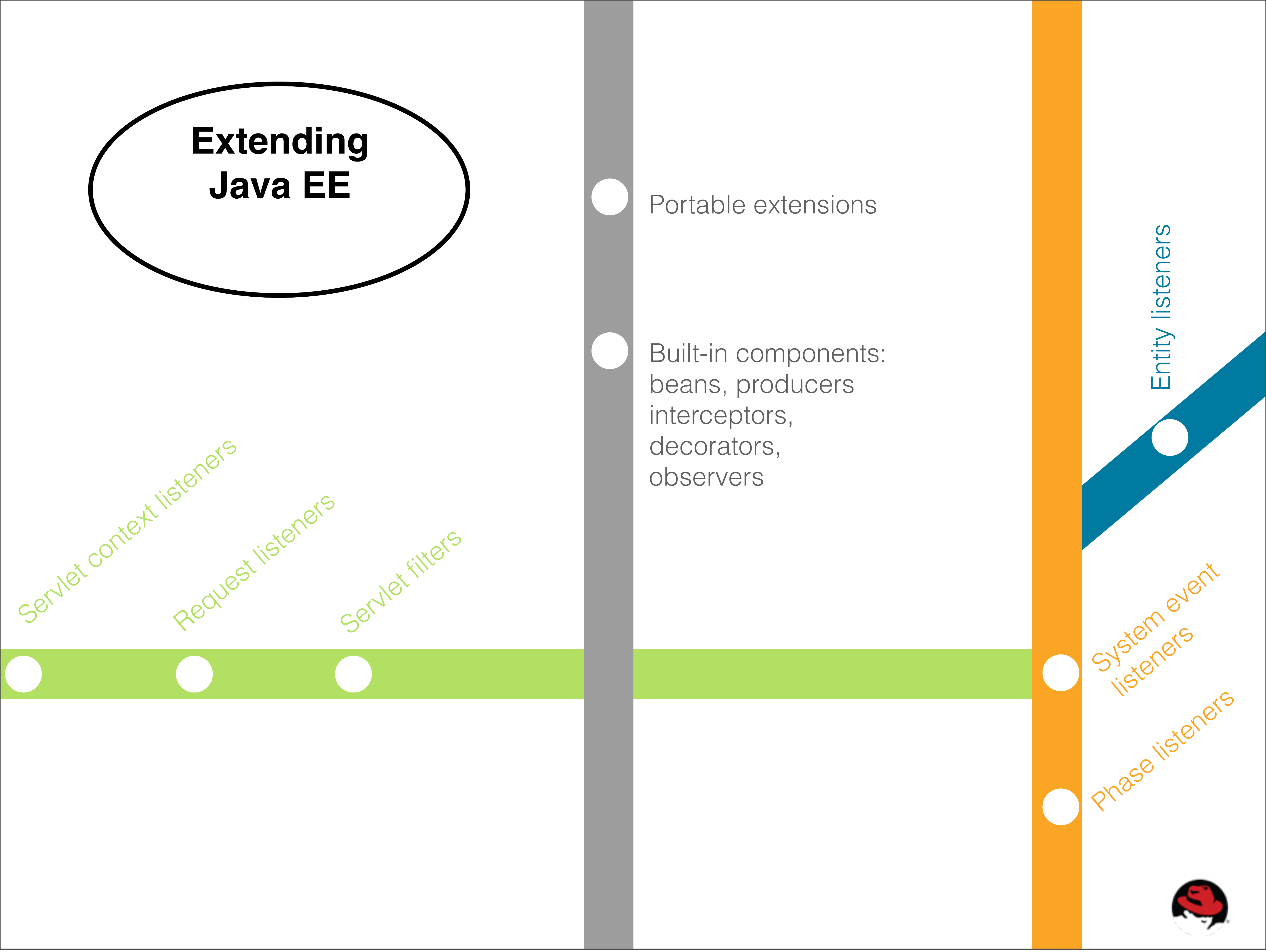
```
com.acme.vetobean.VetoExtension
```

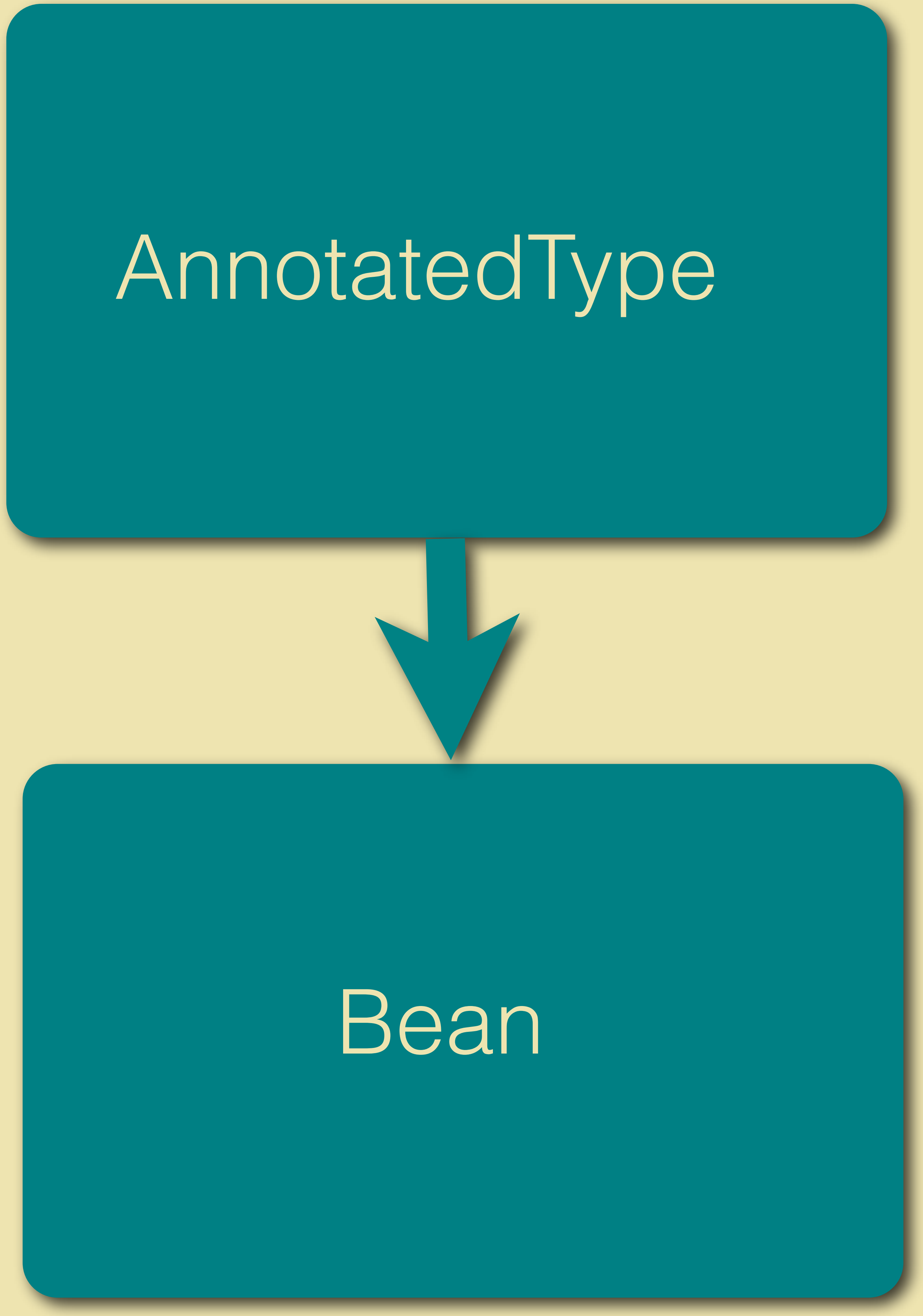
Pete Muir

Extensions

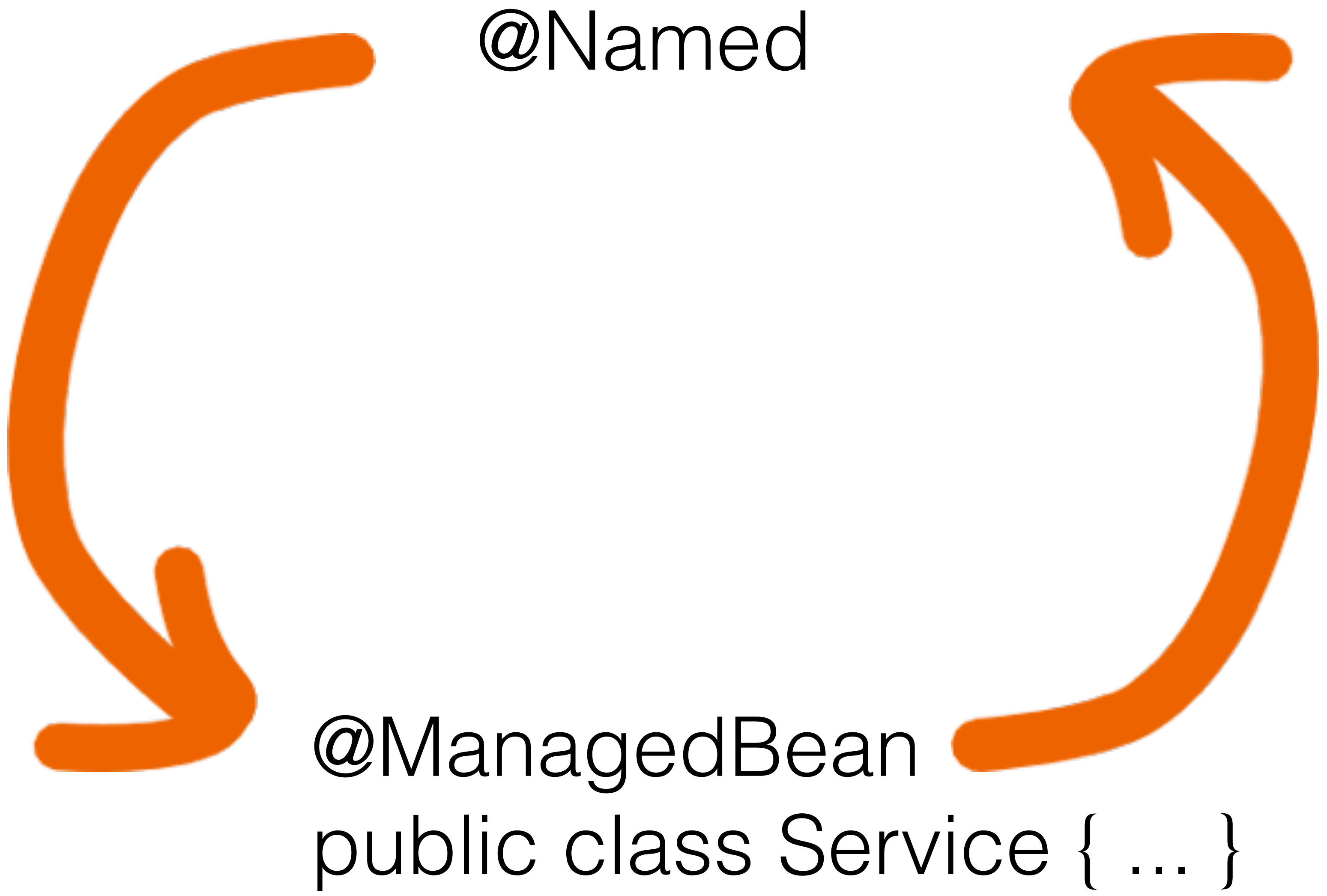
"All we have to decide is what to do with the beans that are given to us."

Extending Java EE




AnnotatedType
Bean

Substituting bean metadata



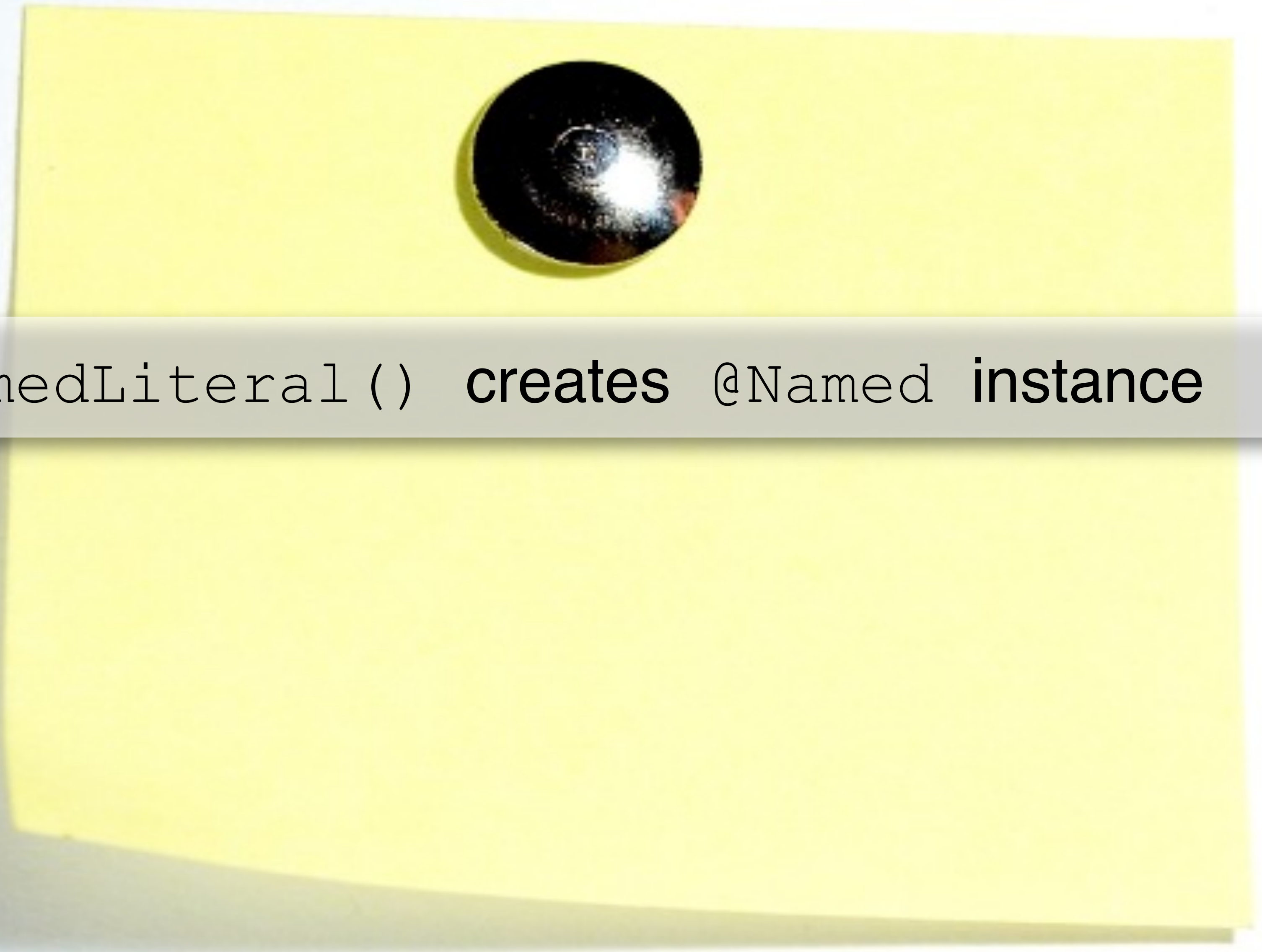
```
@Named
```

```
@ManagedBean
public class Service { ... }
```

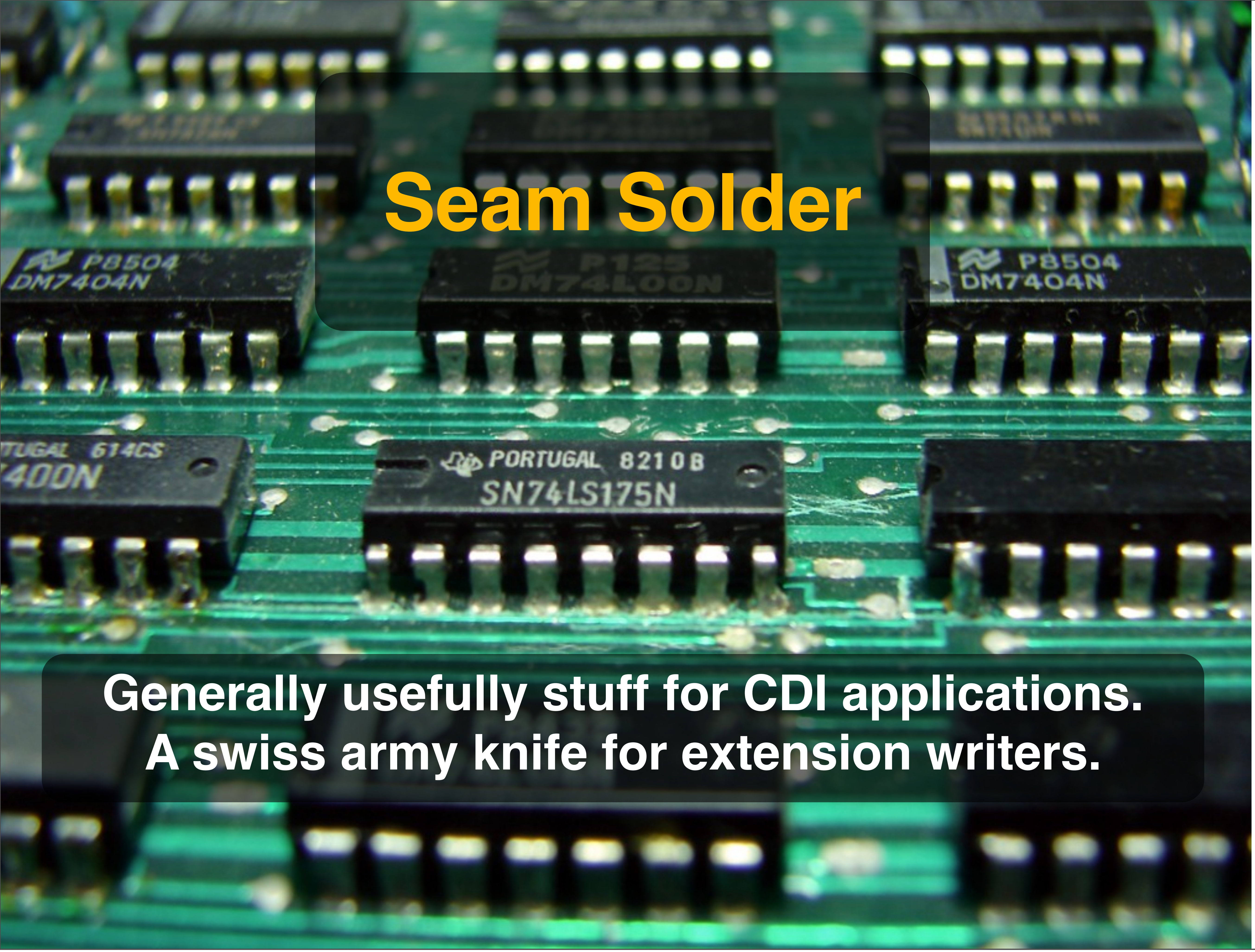
I think I need to create an annotation...

`new AnnotationLiteral<X>() {}` creates `annotation X` instance

`new AnnotationLiteral<Named>() {}` creates `@Named` instance



`new NamedLiteral()` creates `@Named` instance



Seam Solder

Generally usefully stuff for CDI applications.
A swiss army knife for extension writers.

AnnotatedType builder

Bean builders

Annotation inspectors

Reflection utilities

BeanManager locator

Annotation literals
(for standard annotations)

Resource loading

Method injector

Pete Muir

Ah, but can we test it?

arquillian

Pete Muir

```java
@RunWith(Arquillian.class)
public class GreeterTestCase {
   @Deployment
   public static JavaArchive createDeployment() {
   return ShrinkWrap.create(JavaArchive.class)
      .addClass(Greeter.class)
      .addAsManifestResource(EmptyAsset.INSTANCE, "beans.xml");
   }

   @Inject Greeter greeter;

   @Test
   public void shouldBeAbleToInvokeBean() throws Exception {
      Assert.assertEquals("Hello, Earthlings", greeter.greet("Earthlings"));
   }
}
```

Extension Showcase

- Alias annotations
- Veto beans
- Register beans from third-party library
- Introduce behaviors to built-in annotations
- Override injection point types
- Bridge Servlet events to CDI
- Expose standard component as beans
- Emulate EJB services for managed beans
- Set conversation boundaries declaratively
- Initialize beans on application startup

Who's writing extensions?

Seam 3 project - http://seamframework.org/Seam3

MyFaces CODI - http://www.irian.at/myfaces_codi

SoftwareMill - https://github.com/softwaremill/
softwaremill-common

List your extension @
http://tinyurl.com/cdi-extensions

CDI combines loose coupling with strong typing.

CDI makes Java EE flexible, portable and extensible.

CDI extensions make Java EE competitive and awesome!

Q & A

email: pete.muir@jboss.org

twitter: @plmuir

blog: http://in.relation.to/Bloggers/Pete