Milen Dyankov
@milendyankov

Developer advocate
@Liferay

# What's **not new** in **modular Java!**

Featuring
JDK 9 Early Access
with Project Jigsaw

Think of

not new

as in

not new

concept

and not as in

not new

car

## Why is it about time to start thinking about modularity in Java?

2005

JSR 277

JSR 294

2014

JSR 376

JEP 200

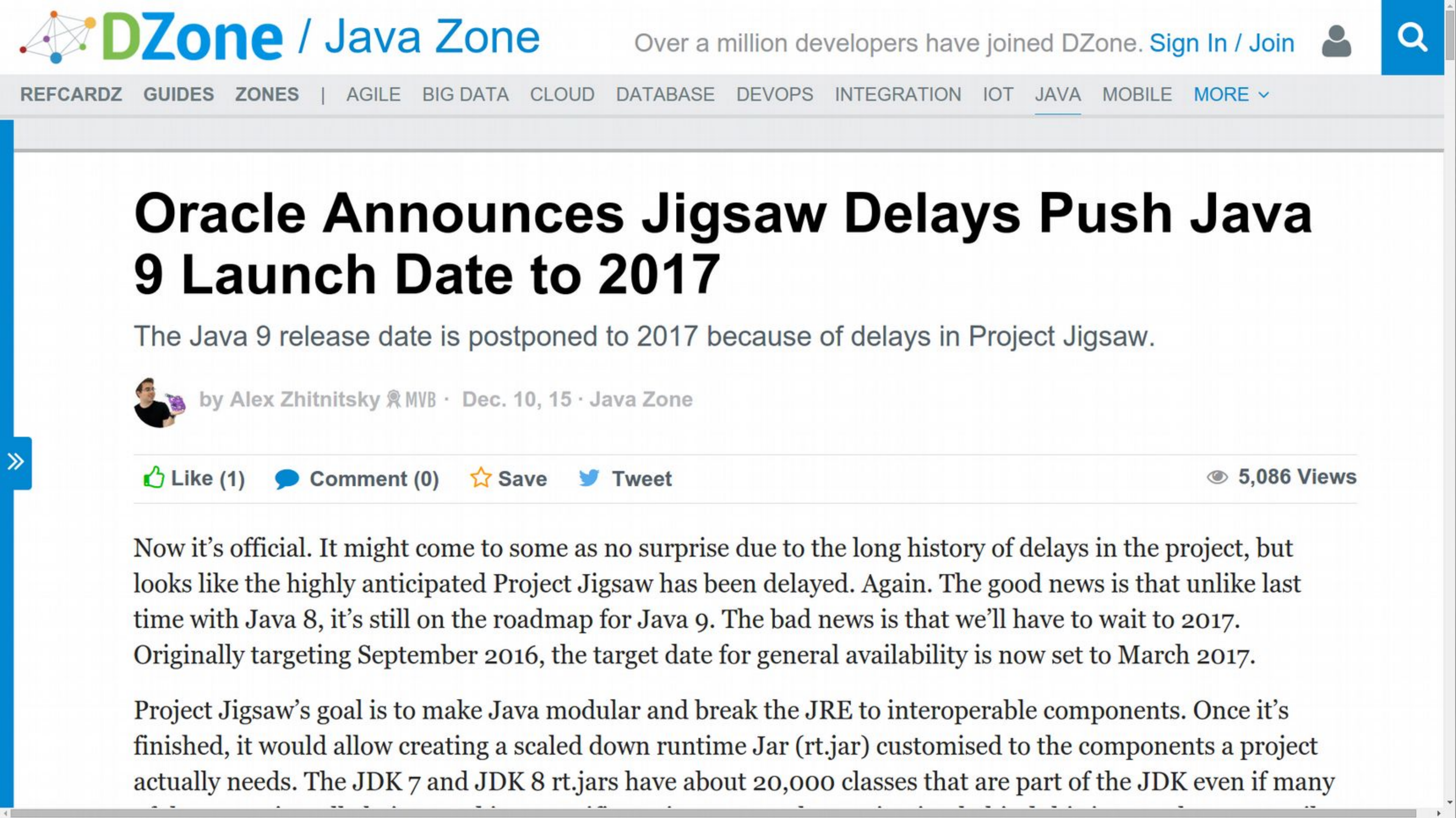
JEP 201

JEP 220

JEP 260

JEP 261

...



# Oracle Announces Jigsaw Delays Push Java 9 Launch Date to 2017

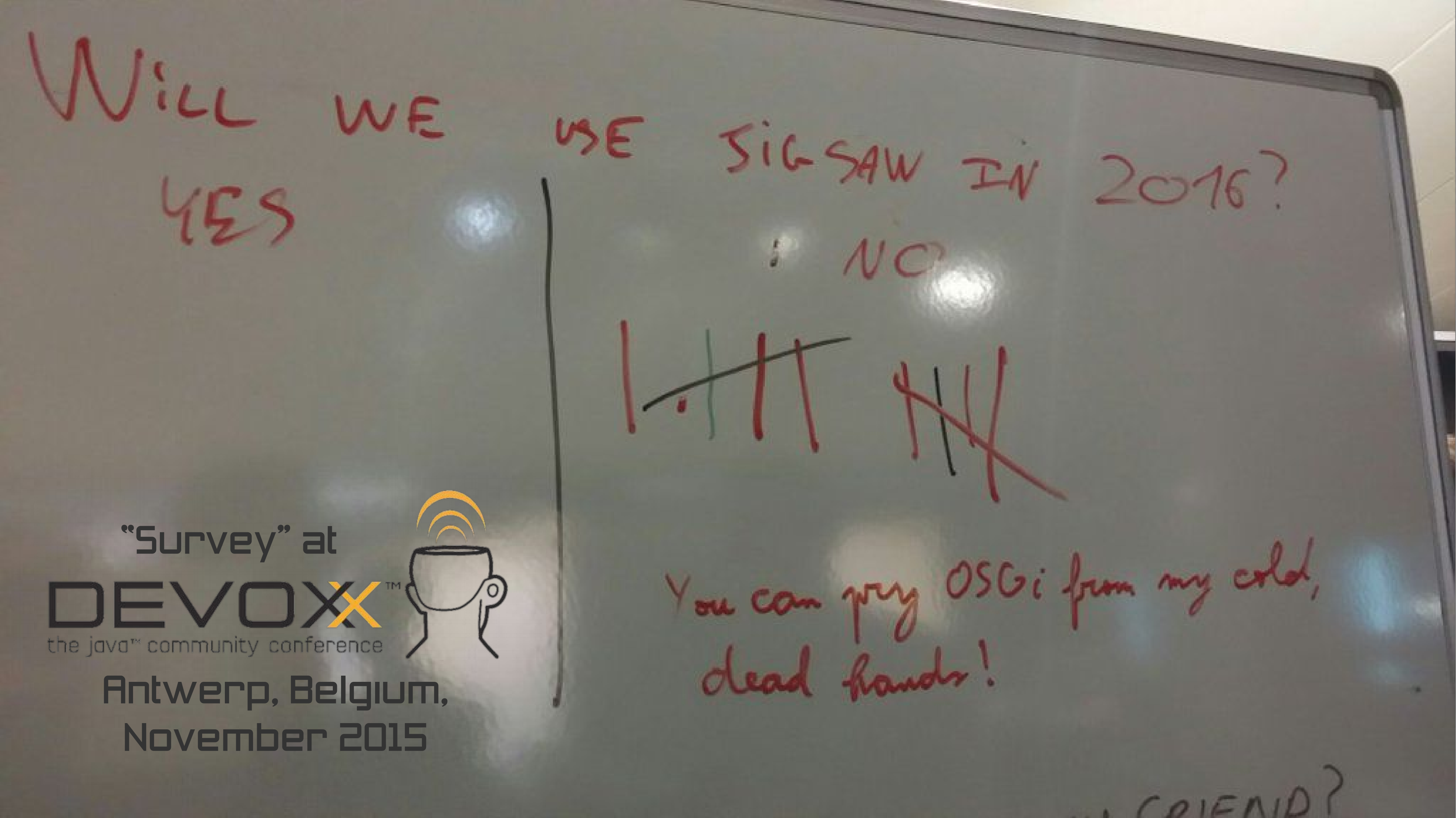
The Java 9 release date is postponed to 2017 because of delays in Project Jigsaw.

by Alex Zhitnitsky MVB · Dec. 10, 15 · Java Zone

Like (1) Comment (0) Save Tweet 5,086 Views

Now it's official. It might come to some as no surprise due to the long history of delays in the project, but looks like the highly anticipated Project Jigsaw has been delayed. Again. The good news is that unlike last time with Java 8, it's still on the roadmap for Java 9. The bad news is that we'll have to wait to 2017. Originally targeting September 2016, the target date for general availability is now set to March 2017.

Project Jigsaw's goal is to make Java modular and break the JRE to interoperable components. Once it's finished, it would allow creating a scaled down runtime Jar (rt.jar) customised to the components a project actually needs. The JDK 7 and JDK 8 rt.jars have about 20,000 classes that are part of the JDK even if many



WILL WE USE JIGSAW IN 2016?

| YES | NO |
|---|---|
| | ~~||||~~ ~~||||~~ |

You can pry OSGi from my cold, dead hands!

FRIEND?

“Survey” at

DEVOXX™
the java™ community conference

Antwerp, Belgium, November 2015

# What is modularity /particularly in Java/ anyway?

"When I use a word," Humpty Dumpty said, in rather a scornful tone, "it means just what I choose it to mean – neither more nor less."

# Modularity Maturity Model

proposed by Dr Graham Charters
at the OSGi Community Event 2011

| | | |
|---|---|---|
| Level 1 | Ad Hoc | nothing |
| Level 2 | Modules | decoupled from artifact |
| Level 3 | Modularity | decoupled from identity |
| Level 4 | Loose-Coupling | decoupled from implementation |
| Level 5 | Devolution | decoupled from ownership |
| Level 6 | Dynamism | decoupled from time |

# Modularity Maturity Model

proposed by Dr Graham Charters
at the OSGi Community Event 2011

| | | |
|---|---|---|
| Level 1 | Ad Hoc | nothing |
| Level 2 | Modules | decoupled from artifact |
| Level 3 | Modularity | decoupled from identity |
| Level 4 | Loose-Coupling | decoupled from implementation |
| Level 5 | Devolution | decoupled from ownership |
| Level 6 | Dynamism | decoupled from time |
| Level 7 | Peter Kriens | only available to people who are Peter Kriens |

# Modularity Maturity Model

proposed by Peter Kriens
in foreword to "Java Application Architecture"

| | | |
|---|---|---|
| Level 1 | Ad Hoc | Unmanaged / chaos |
| Level 2 | Modules | Managing dependencies |
| Level 3 | Modularity | Proper isolation |
| Level 4 | Loose-Coupling | Minimize coupling |
| Level 5 | Devolution | Service-oriented architecture |
| Level 6 | Dynamism | |

## Buzzword compliant

## Modularity Maturity Model

| | |
|---|---|
| Level 1 | Monolith |
| Level 2 | Composite |
| Level 3 | Containers |
| Level 4 | Discovery |
| Level 5 | µServices |

## Buzzword compliant
## Modularity Maturity Model

| | | |
|---|---|---|
| Level 1 | Monolith | Unaware of own dependencies |
| Level 2 | Composite | Aware of infrastructural dependencies |
| Level 3 | Containers | Aware of functional dependencies |
| Level 4 | Discovery | Aware of functional requirements |
| Level 5 | µServices | Adapts to changing requirements |

# Buzzword compliant
# Modularity Maturity Model

|  |  |
|---|---|
| Level 1 | Monolith |
| Level 2 | Composite |
| Level 3 | Containers |
| Level 4 | Discovery |
| Level 5 | µServices |

# Buzzword compliant
## Modularity Maturity Model

|  |  |  |
|---|---|---|
| Level 1 | Monolith | Java |
| Level 2 | Composite | Maven |
| Level 3 | Containers | |
| Level 4 | Discovery | |
| Level 5 | µServices | |

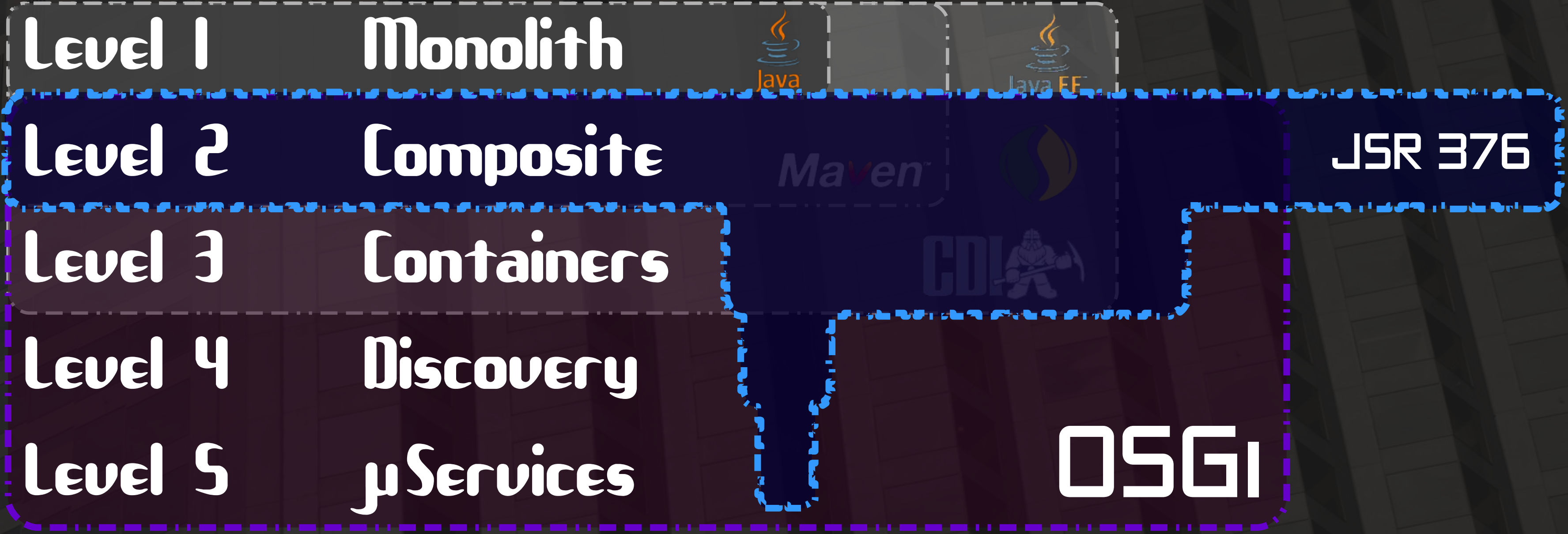
## Buzzword compliant
## Modularity Maturity Model

| | |
|---|---|
| Level 1 | Monolith |
| Level 2 | Composite |
| Level 3 | Containers |
| Level 4 | Discovery |
| Level 5 | µServices |

# Buzzword compliant
# Modularity Maturity Model

| | | |
|---|---|---|
| Level 1 | Monolith | Java |
| Level 2 | Composite | Maven |
| Level 3 | Containers | Java EE |
| Level 4 | Discovery | |
| Level 5 | µServices | OSGi |

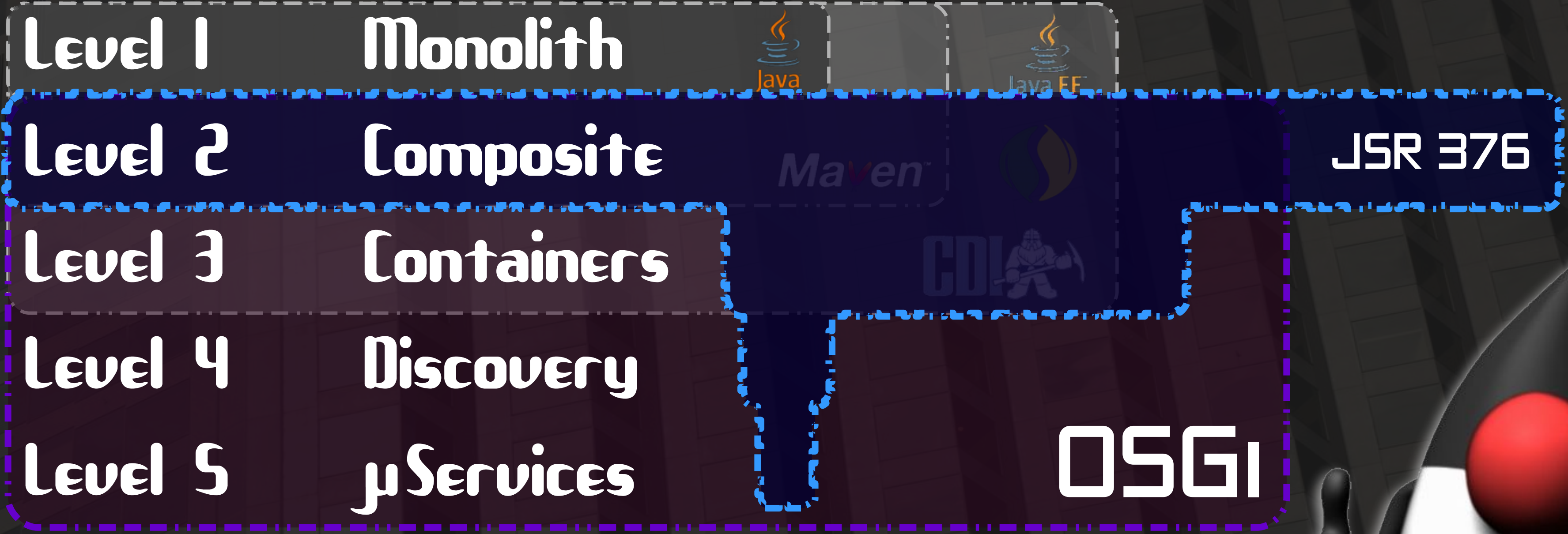
# Buzzword compliant
# Modularity Maturity Model



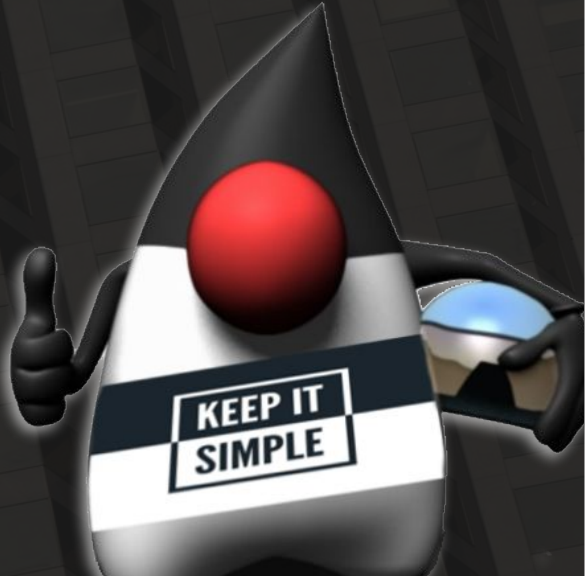
| | |
|---|---|
| Level 1 | Monolith |
| Level 2 | Composite |
| Level 3 | Containers |
| Level 4 | Discovery |
| Level 5 | µServices |

## Buzzword compliant

# Modularity Maturity Model



| | |
|---|---|
| Level 1 | Monolith |
| Level 2 | Composite |
| Level 3 | Containers |
| Level 4 | Discovery |
| Level 5 | µServices |



## What is modularity from application perspective ?

Java application

Java application

Libraries

Java Platform

## OSGi

# OSGi

There is nothing we can do about it!

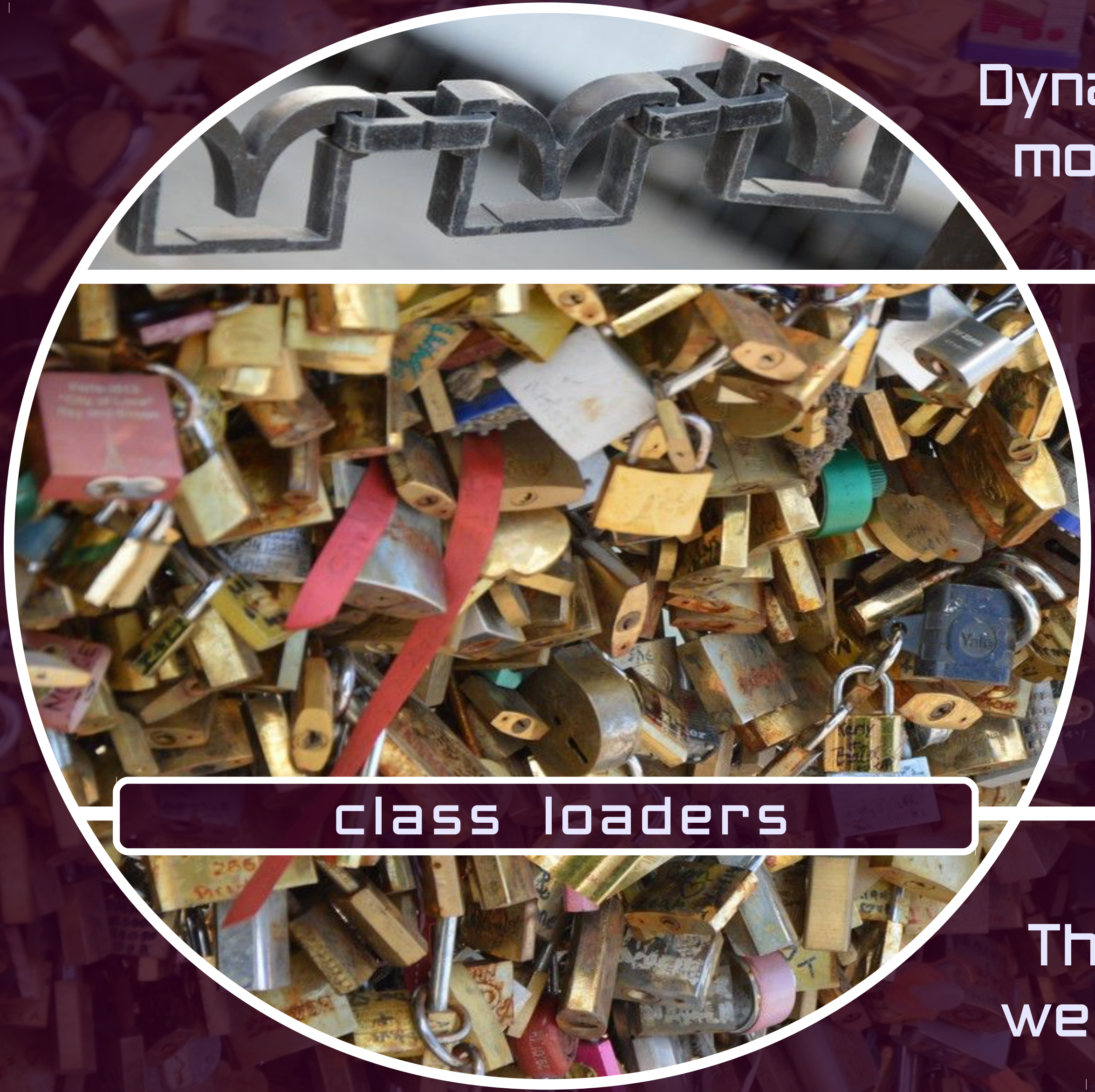
class loaders

There is nothing
we can do about it!

OSGi

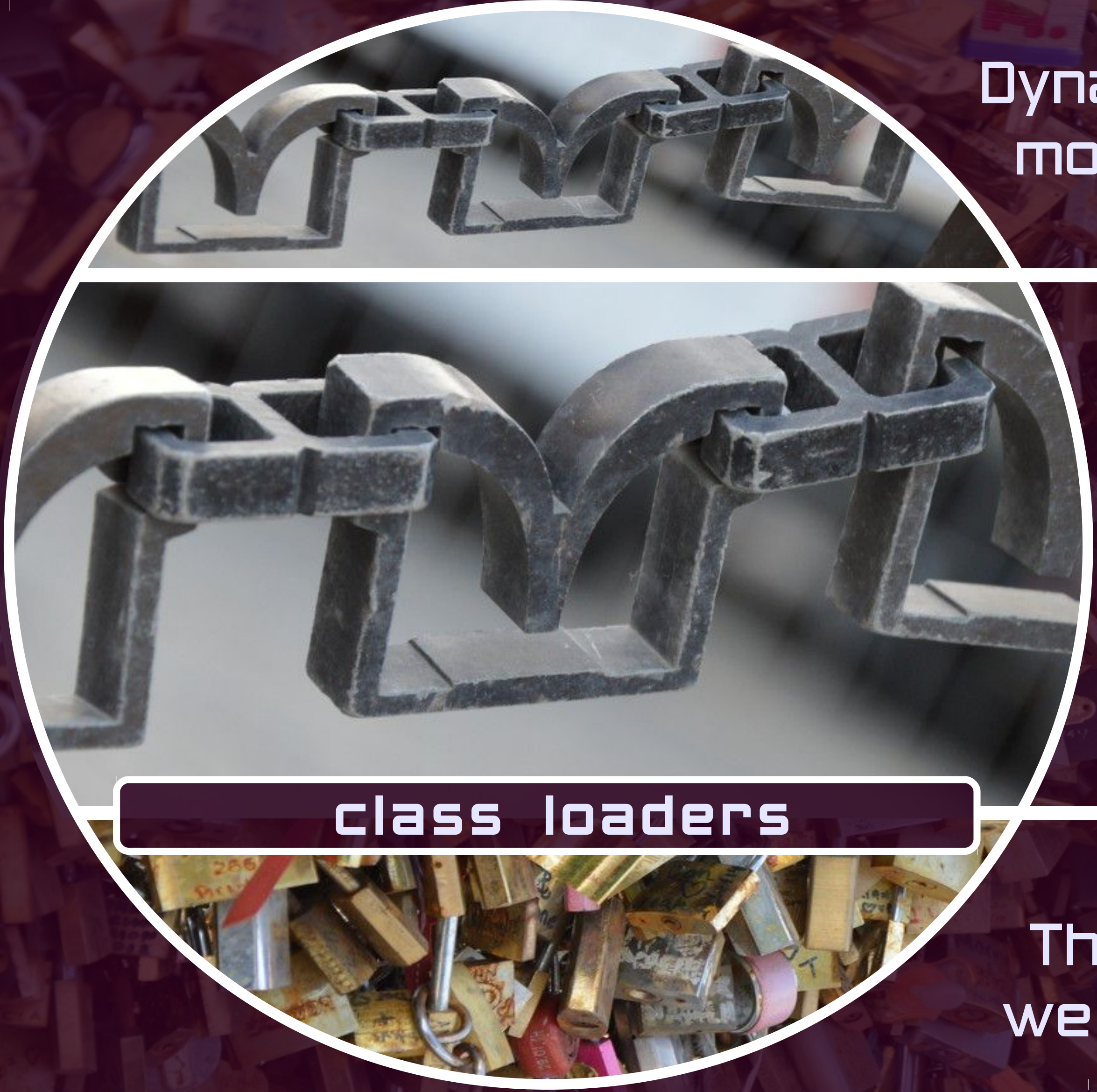
## OSGi

Dynamic multi-layer modular runtime!



There is nothing we can do about it!

# OSGi



class loaders

Dynamic multi-layer modular runtime!

It's so easy, everyone should release bundles (modules)!

There is nothing we can do about it!

"Many
people claim
OSGi is hard without
acknowledging that modularizing
applications is the hard part.
. . .
JSR 376 will demonstrate that OSGi
was just the messenger and actually not the cause."

Peter Kriens

## JSR 376

## JSR 376

Modules are
first class citizens!

Nothing to do about it,
must use modules!

Modules are
first class citizens!

JSR 376

## JSR 376

Nothing to do about it, must use modules!

It's so easy, everyone must release modules!

Modules are first class citizens!

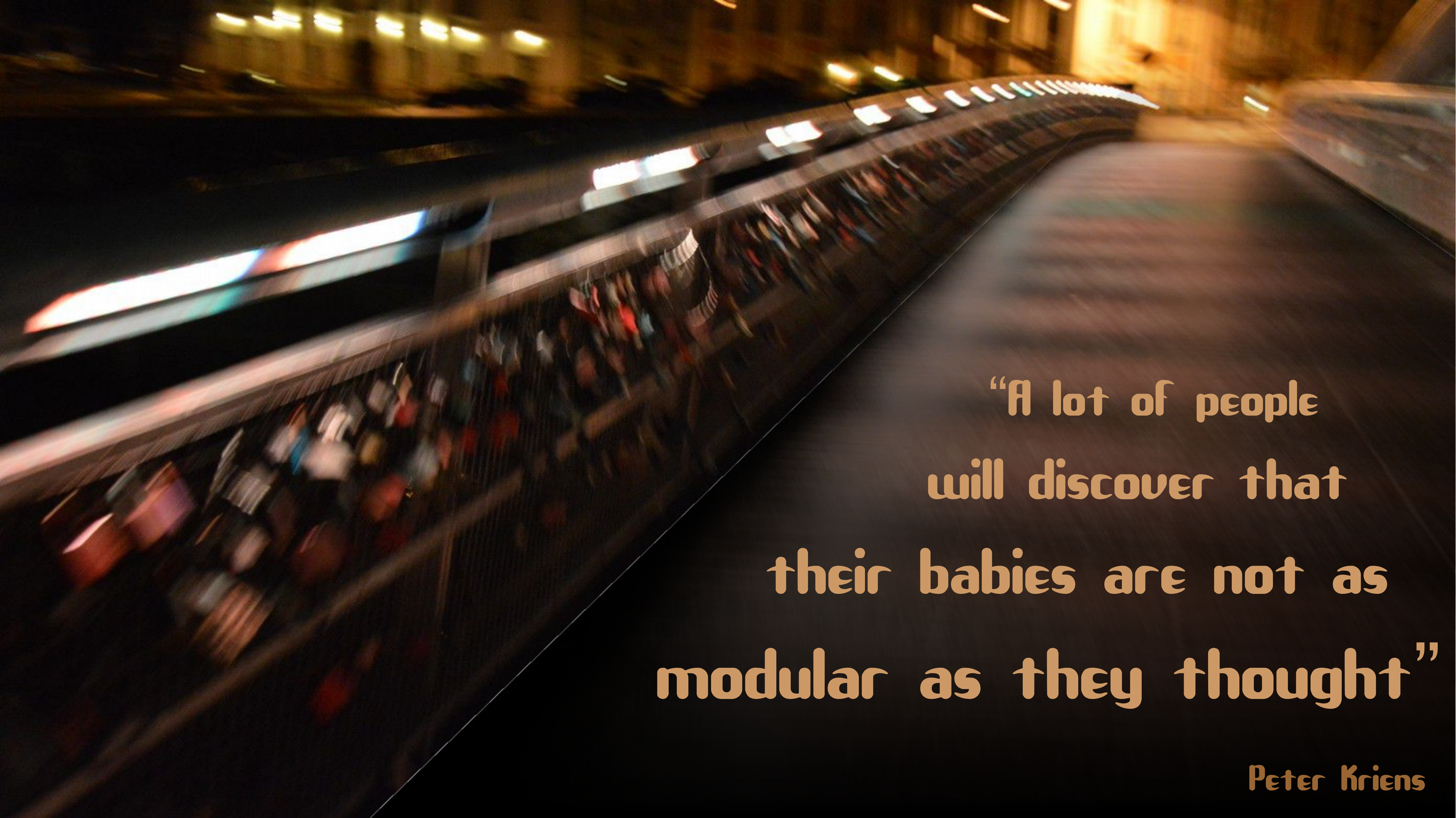
not new

except now you kinda have to

new

Modular Java SE Applications!

Modular Java SE Platform!



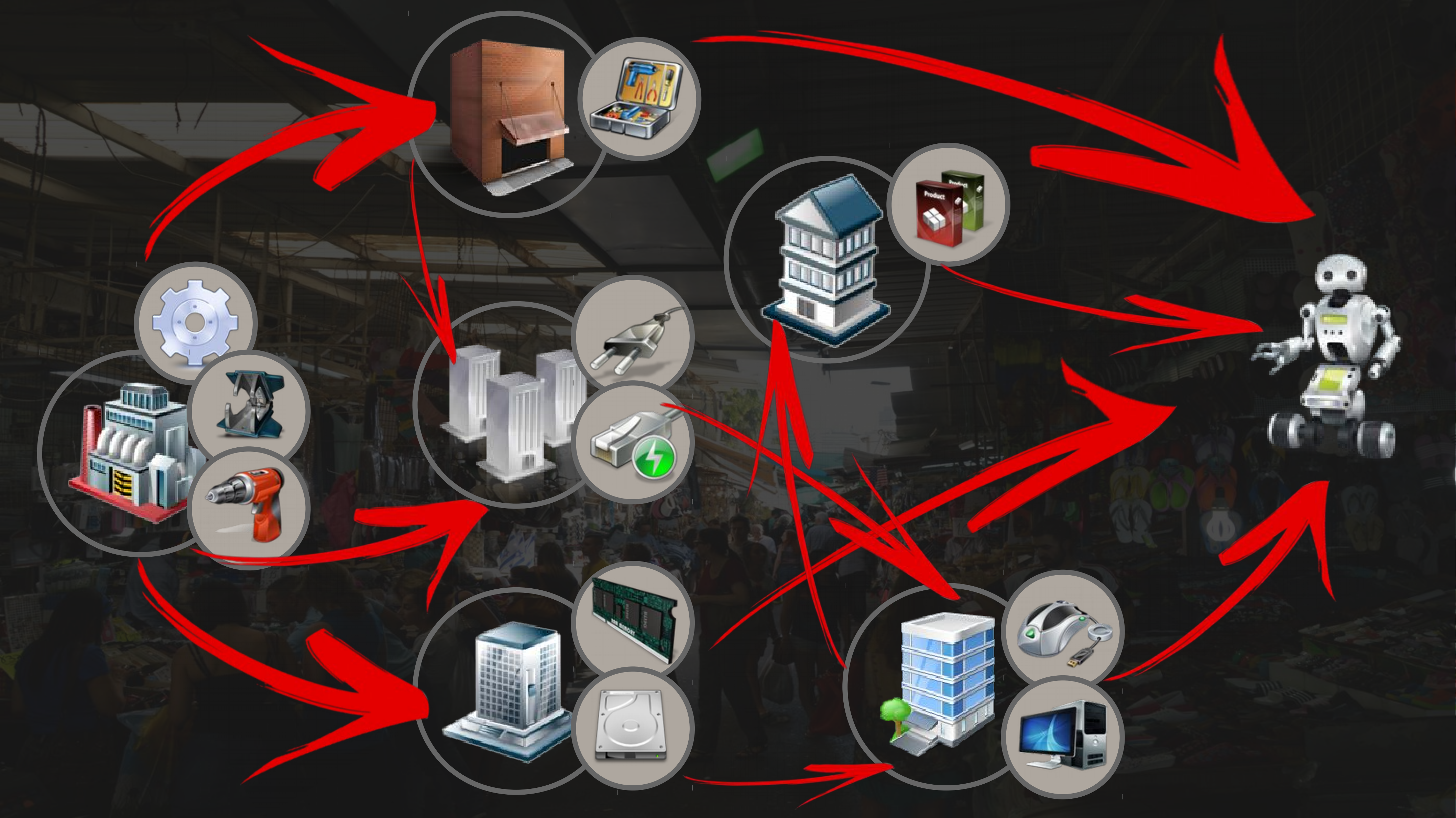
“A lot of people will discover that their babies are not as modular as they thought”

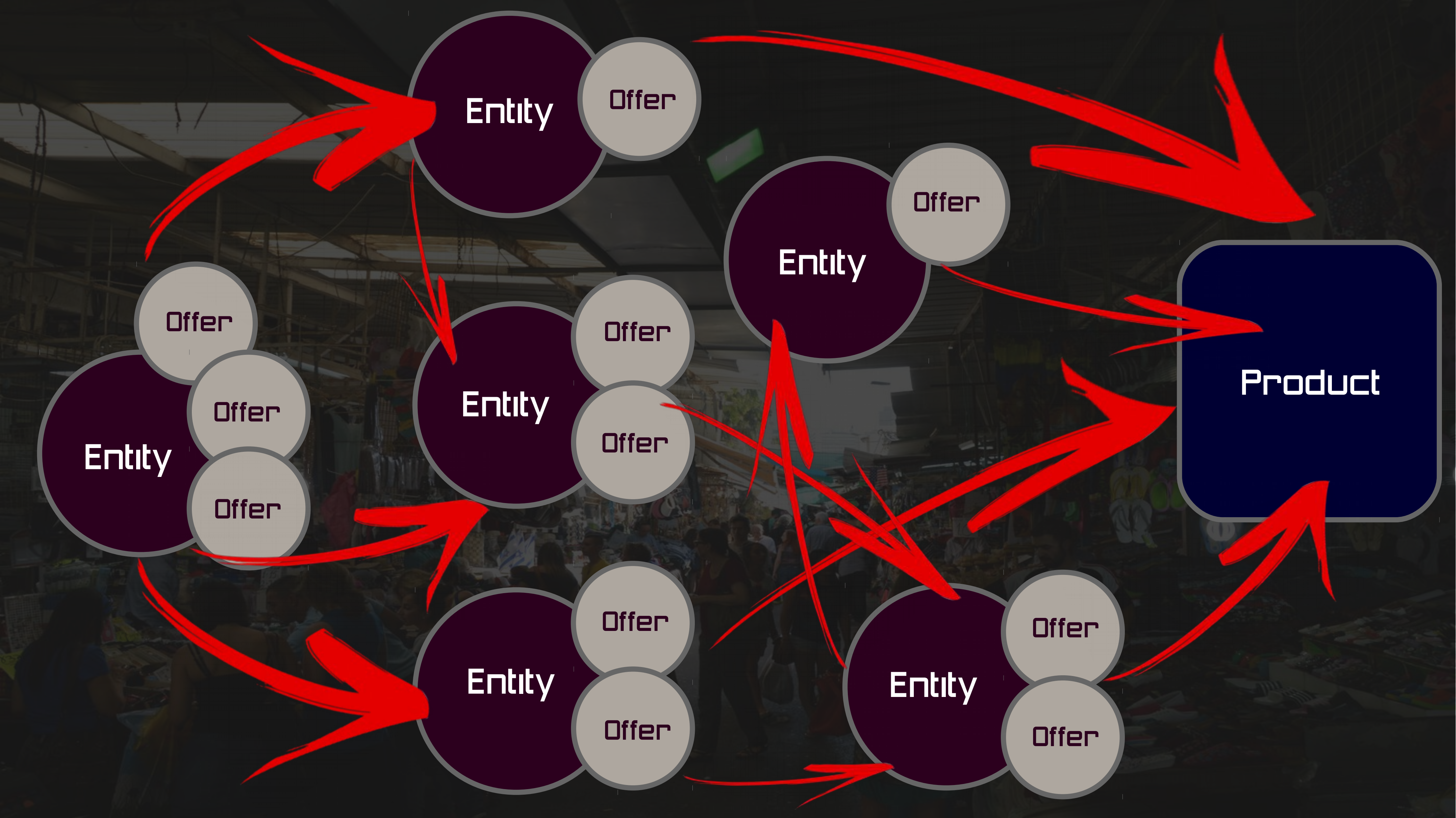
Peter Kriens

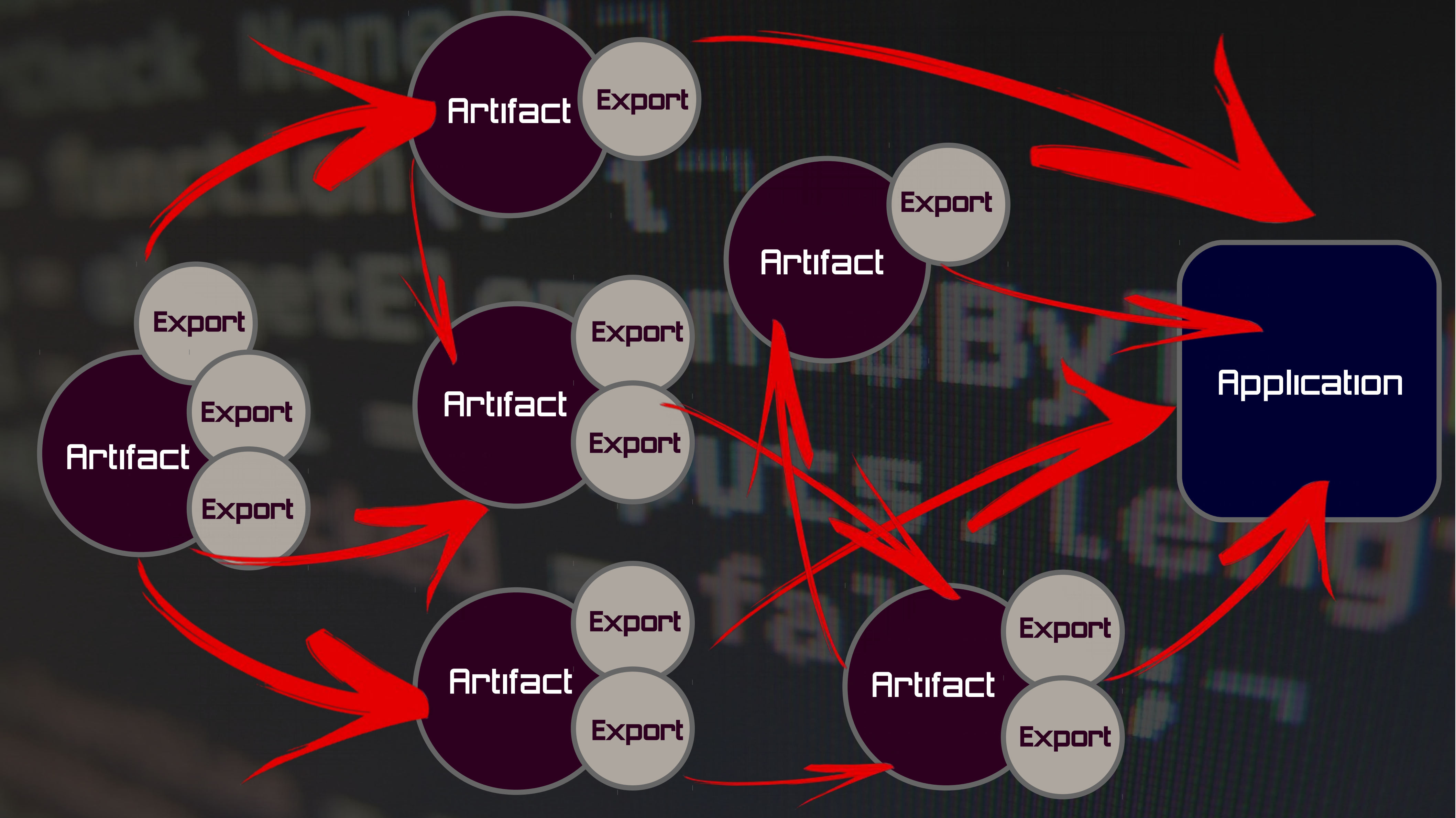
## When is “Keep it simple!” not enough?

product
intermediate
intermediate
material

Product
Product


Product
Product


Entity
Offer
Entity
Offer
Offer
Offer
Entity
Offer
Offer
Offer
Entity
Offer
Offer
Entity
Offer
Offer
Entity
Offer
Offer
Product


Artifact
Export
Artifact
Export
Artifact
Export
Export
Export
Artifact
Export
Export
Application
Artifact
Export
Export
Artifact
Export
Export

## Level 2 decoupled from artifact

Artifact

## OSGi

MANIFEST.MF

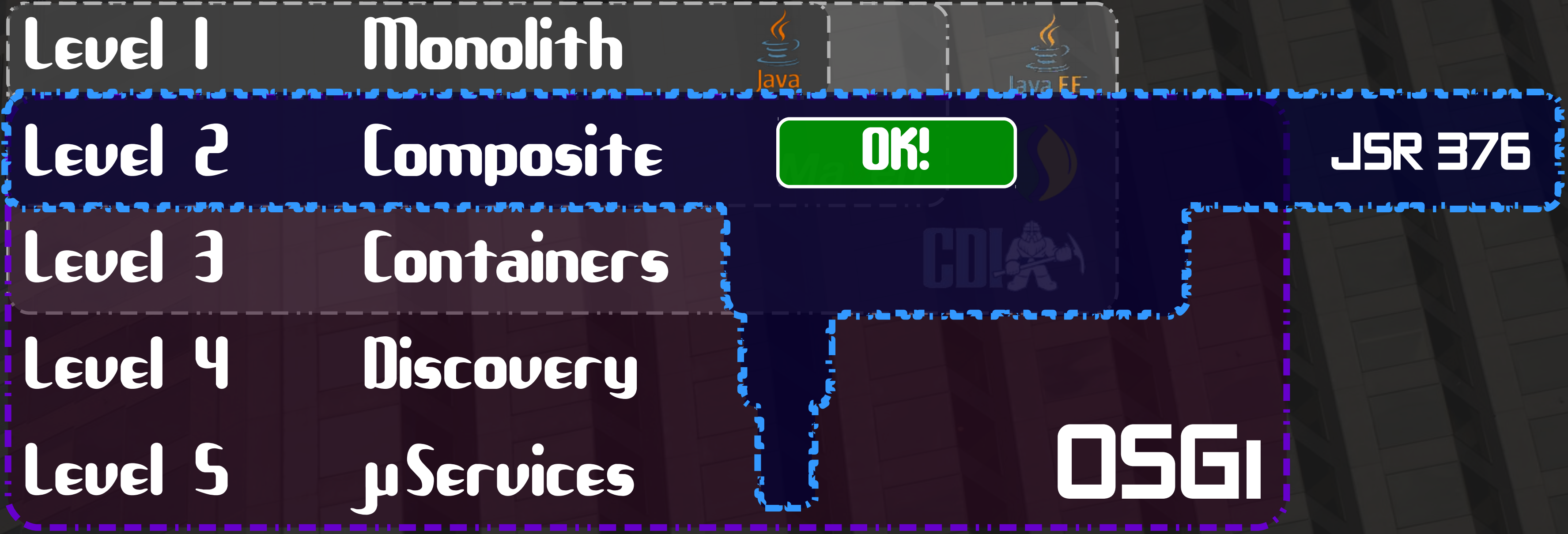
```
Manifest-Version: 1.0
Bundle-SymbolicName: \
     com.mycompany.mymodule

...
```

## JSR 376

module-info.java

```
module com.mycompany.mymodule {

...

}
```

# Buzzword compliant
# Modularity Maturity Model



| | |
|---|---|
| Level 1 | Monolith |
| Level 2 | Composite |
| Level 3 | Containers |
| Level 4 | Discovery |
| Level 5 | µServices |

## Level 3 decoupled from identity

Export
Artifact

## OSGi

MANIFEST.MF

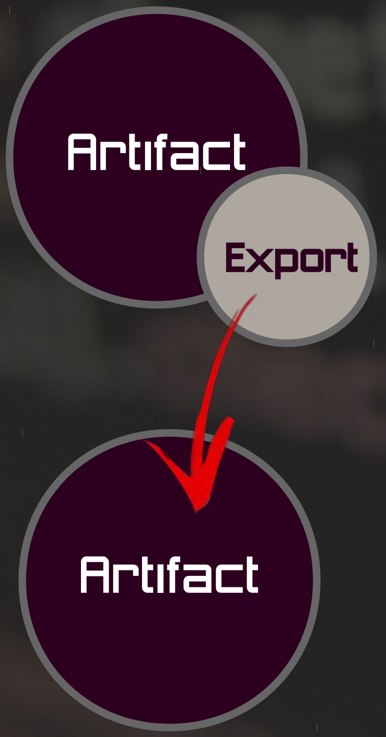
```
Manifest-Version: 1.0
Bundle-SymbolicName: \
     com.mycompany.mymodule

Export-Package: \
     com.mycompany.mypackage

...
```

## JSR 376

module-info.java

```
module com.mycompany.mymodule {

  exports com.mycompany.mypackage;

...

}
```

## Level 3 decoupled from identity



## OSGi

**MANIFEST.MF**

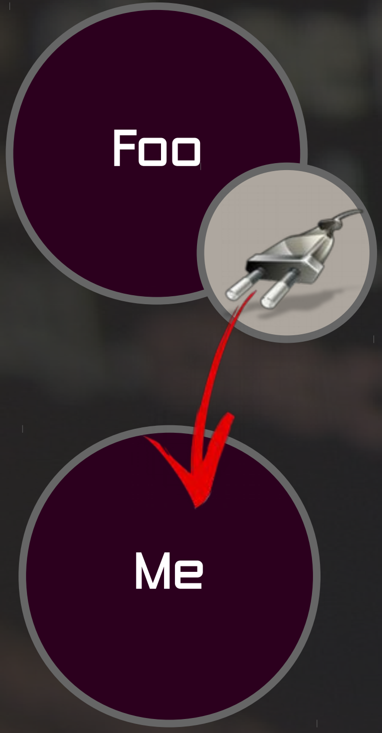
```
Manifest-Version: 1.0
Bundle-SymbolicName: \
     com.mycompany.mymodule

~~Require-Bundle: \~~
~~     other.module~~

Import-Package: \
     com.some.package;
     version="[2,3)",...

...
```

## JSR 376

**module-info.java**

```
module com.mycompany.mymodule {

  requires other.module;

...

}
```

## Level 3 decoupled from identity



## OSGi

**MANIFEST.MF**

```
Manifest-Version: 1.0
Bundle-SymbolicName: \
     com.mycompany.mymodule

~~Require-Bundle: \
     com.foo~~

Import-Package: \
     com.generic.powerplug;
     version="[2,3)",...

...
```

**I need power plug!**

## JSR 376

**module-info.java**

```
module com.mycompany.mymodule {

  requires com.foo;

...

}
```

**I need Foo because I know it offers power plugs and I know only Foo offers power plugs!**

## Level 3 decoupled from identity

## OSGi

MANIFEST.MF

```
Manifest-Version: 1.0
Bundle-SymbolicName: \
      com.mycompany.mymodule
```

~~Require-Bundle: \
      com.foo~~

```
Import-Package: \
      com.generic.powerplug;
      version="[2,3)",...

...
```

I'm compatible with all 2.x.x versions!

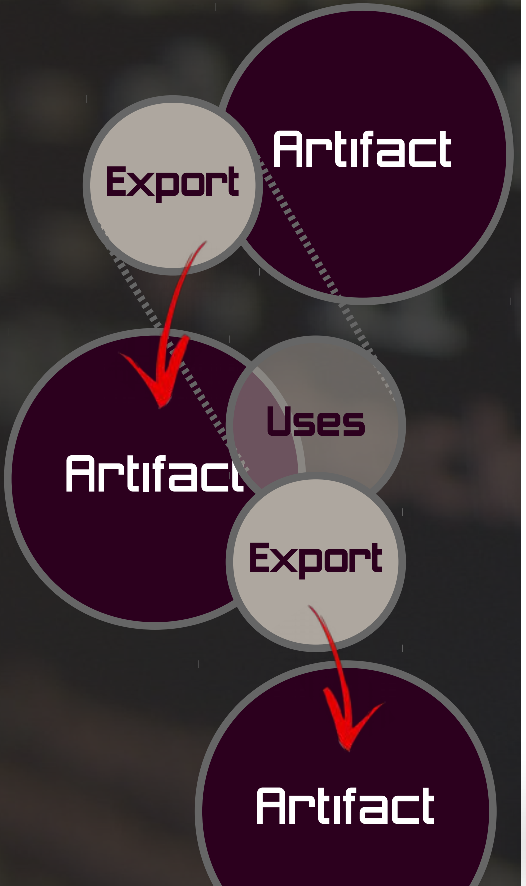
## JSR 376

module-info.java

```
module com.mycompany.mymodule {

  requires com.foo;

...

}
```

I expect developer/user to know which version will work and provide it on module path!

## Level 3
## decoupled from identity



## OSGi

### MANIFEST.MF

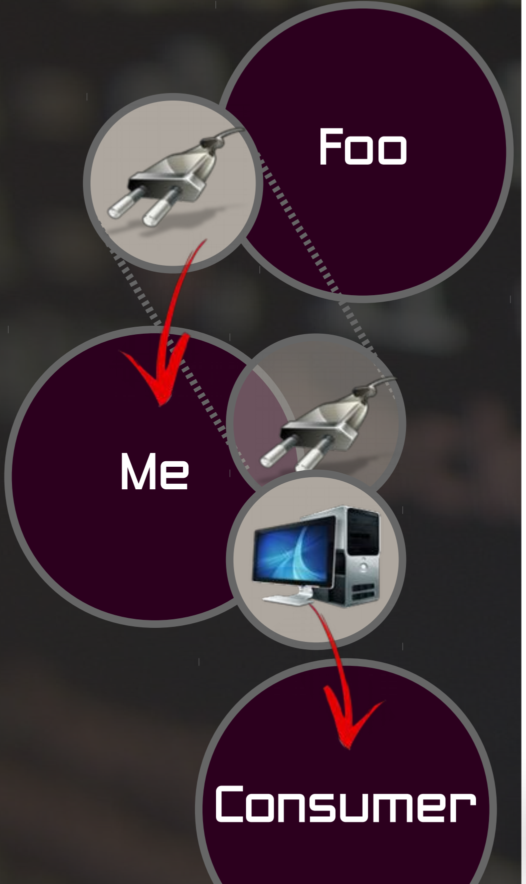
```
Manifest-Version: 1.0
Bundle-SymbolicName: \
     com.mycompany.mymodule

Export-Package: \
com.mycompany.mypackage;\
     uses:="com.some.package”

...
```

## JSR 376

### module-info.java

```
module com.mycompany.mymodule {

  exports com.mycompany.mypackage;

  requires public other.module;

  ...

}
```

## Level 3 decoupled from identity



## OSGi

MANIFEST.MF

```
Manifest-Version: 1.0
Bundle-SymbolicName: \
     com.mycompany.devices

Export-Package: \
com.mycompany.pc; \
     uses:="foo.tools.powerplug”

...
```

I used a power plug from Foo! You may need it!

## JSR 376

module-info.java

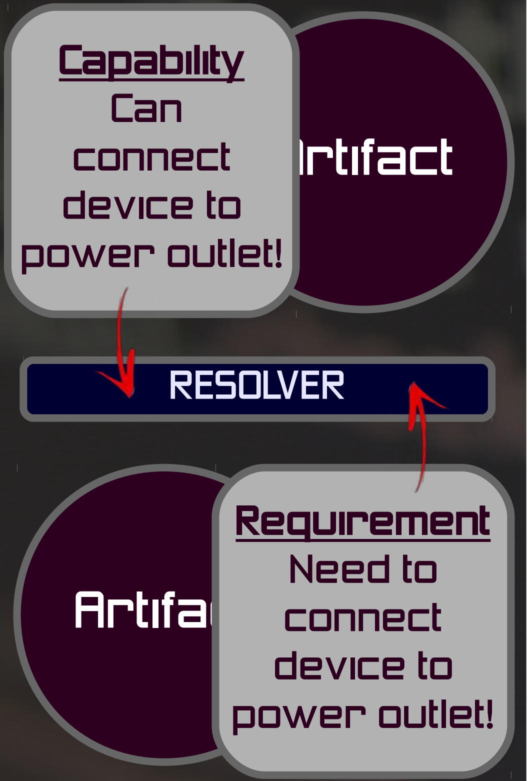
```
module com.mycompany.devices {

  exports com.mycompany.pc;

  requires public foo.tools;

  ...

}
```

I used something from Foo tools, so you now depend on Foo tools as well!

# Buzzword compliant
# Modularity Maturity Model

| | | | |
|---|---|---|---|
| Level 1 | Monolith | Java | Java EE |
| Level 2 | Composite | OK! | JSR 376 |
| Level 3 | Containers | Not fully decoupled from identity! | |
| Level 4 | Discovery | | |
| Level 5 | µServices | | OSGi |

## Level 4 decoupled from implementation


Capability
Can connect device to power outlet!
Artifact
RESOLVER
Artifact
Requirement
Need to connect device to power outlet!

## OSGi

- Bundles with custom metadata
- Requirements and Capabilities with LDAP like filters
- Bundle lifecycle events and listeners
- Extender pattern

## JSR 376

- Nothing OOTB. Use OSGi :)
- Probably doable via external resolver dynamic modules and layers
- JEE or 3rd party solutions on top of JSR 376 may provide solutions

## Buzzword compliant

# Modularity Maturity Model

| | |
|---|---|
| Level 1 | Monolith |
| Level 2 | Composite |
| Level 3 | Containers |
| Level 4 | Discovery |
| Level 5 | µServices |

Java

Java EE

OK!

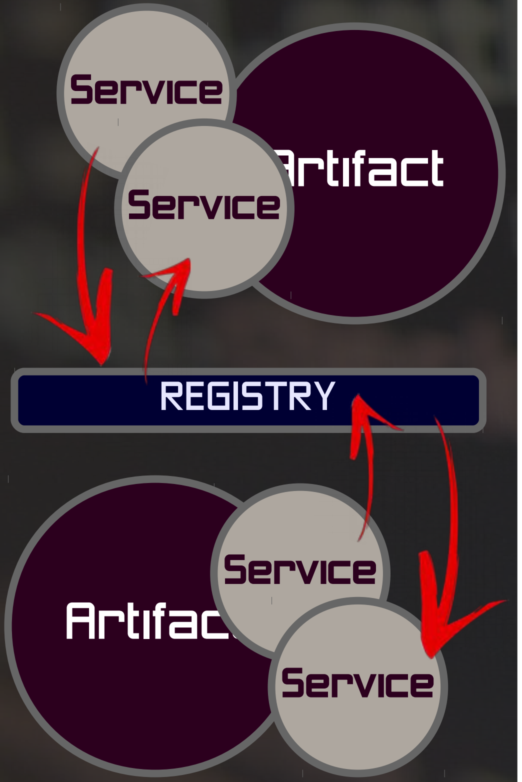
JSR 376

Not fully decoupled from identity!

Some very basic APIs only!

OSGi

## Level 5 µServices decoupled from ownership & time



## OSGi

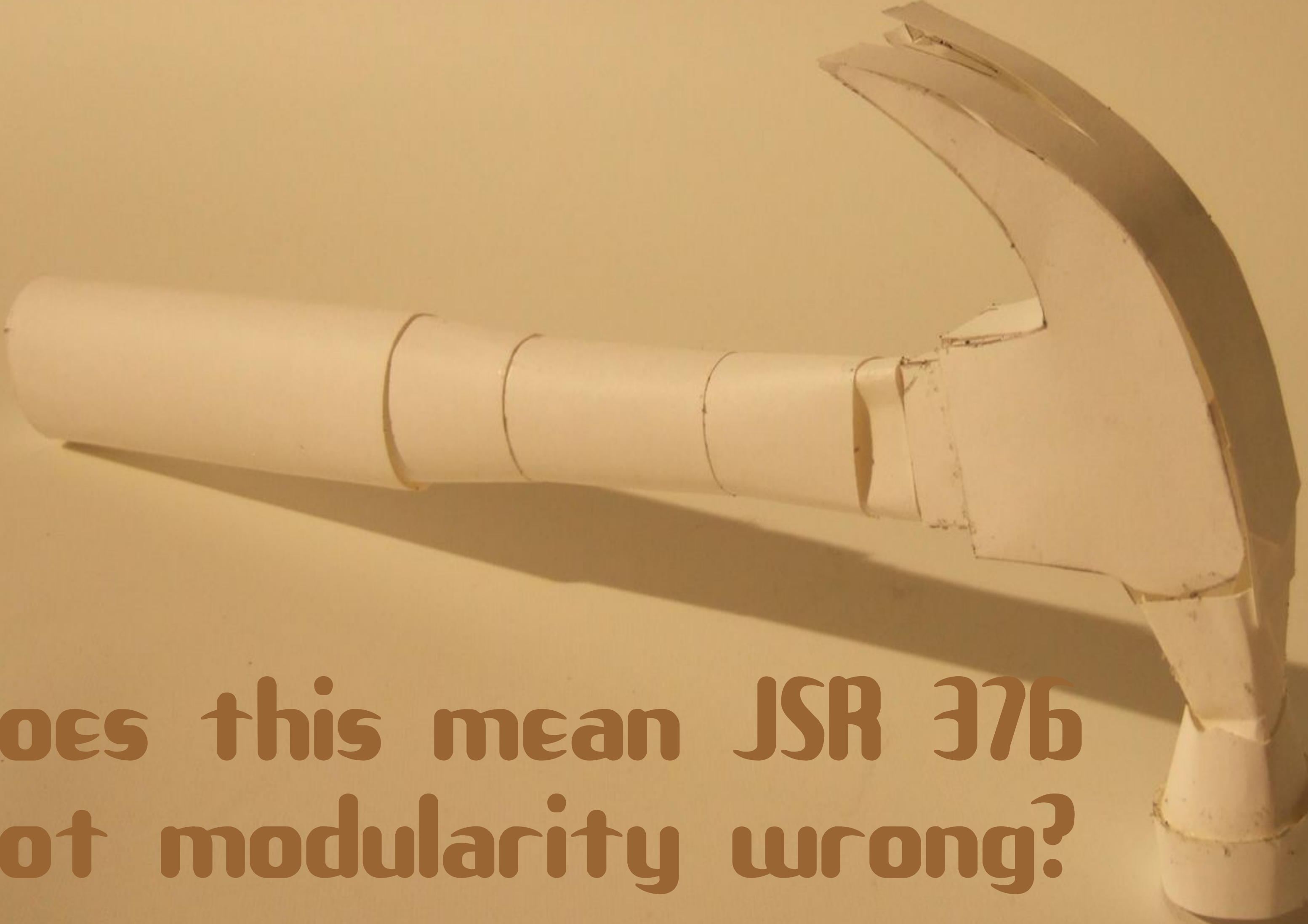
- Service registry with metadata
- Finding services via LDAP like filters
- Service lifecycle, events and listeners
- Multiple component frameworks
- Whiteboard pattern

## JSR 376

- Traditional Java ServiceLoader (not dynamic) moved to module descriptor
- Alternative: minimal standalone Java applications with external service discovery

# Buzzword compliant
# Modularity Maturity Model

| | | | |
|---|---|---|---|
| Level 1 | Monolith | | |
| Level 2 | Composite | OK! | JSR 376 |
| Level 3 | Containers | Not fully decoupled from identity! | |
| Level 4 | Discovery | Some very basic APIs only! | |
| Level 5 | μServices | Very limited service layer! DIY dynamism! | |



Does this mean JSR 376 got modularity wrong?

When they say

**modular Java**

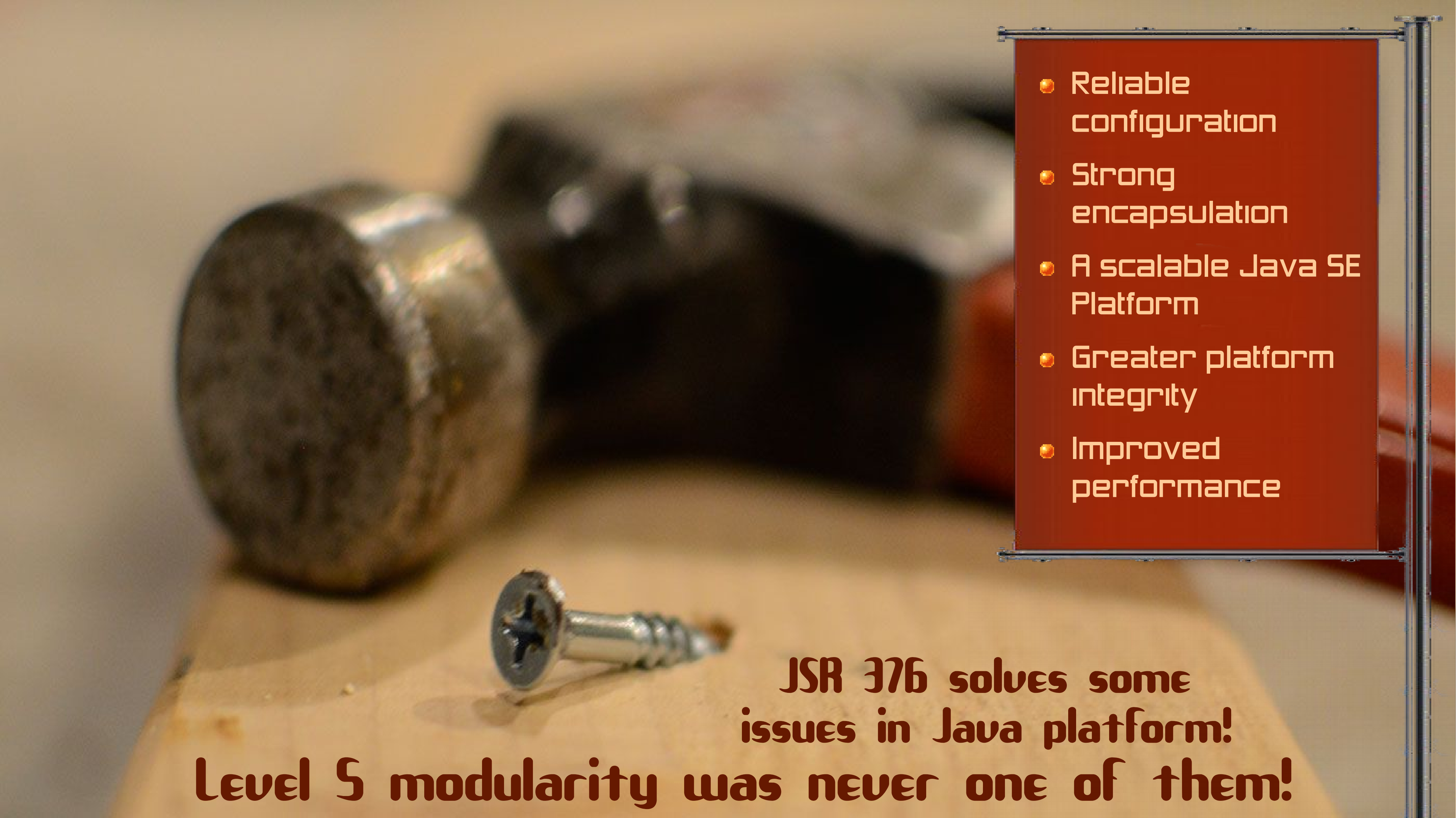
it **means**

**just what they**

**choose it to mean** –

neither

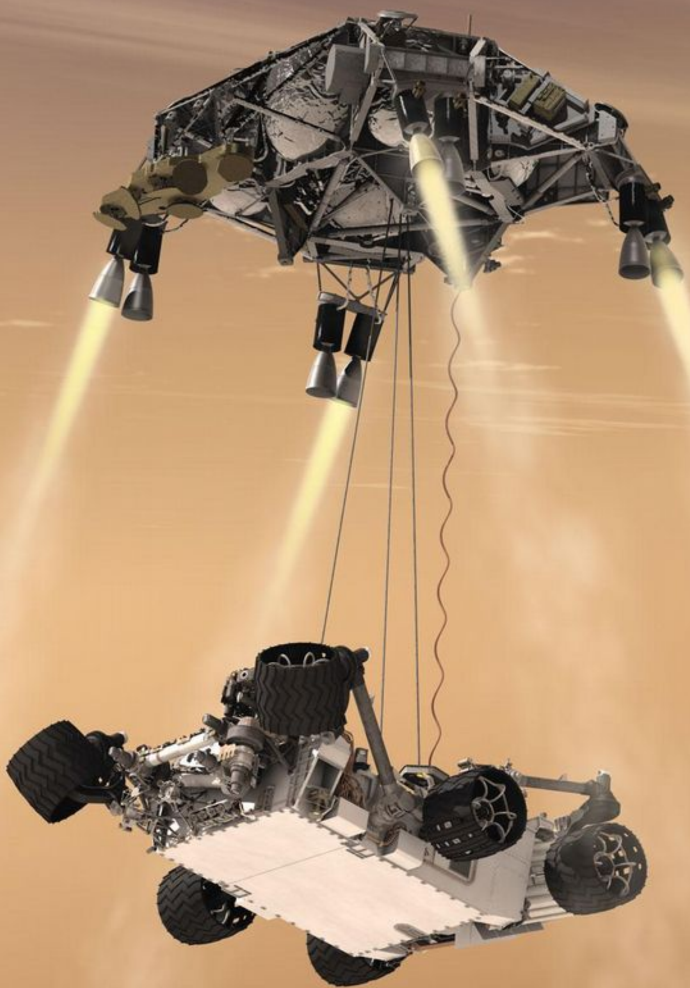
more nor less!



- Reliable configuration
- Strong encapsulation
- A scalable Java SE Platform
- Greater platform integrity
- Improved performance

JSR 376 solves some issues in Java platform!

Level 5 modularity was never one of them!



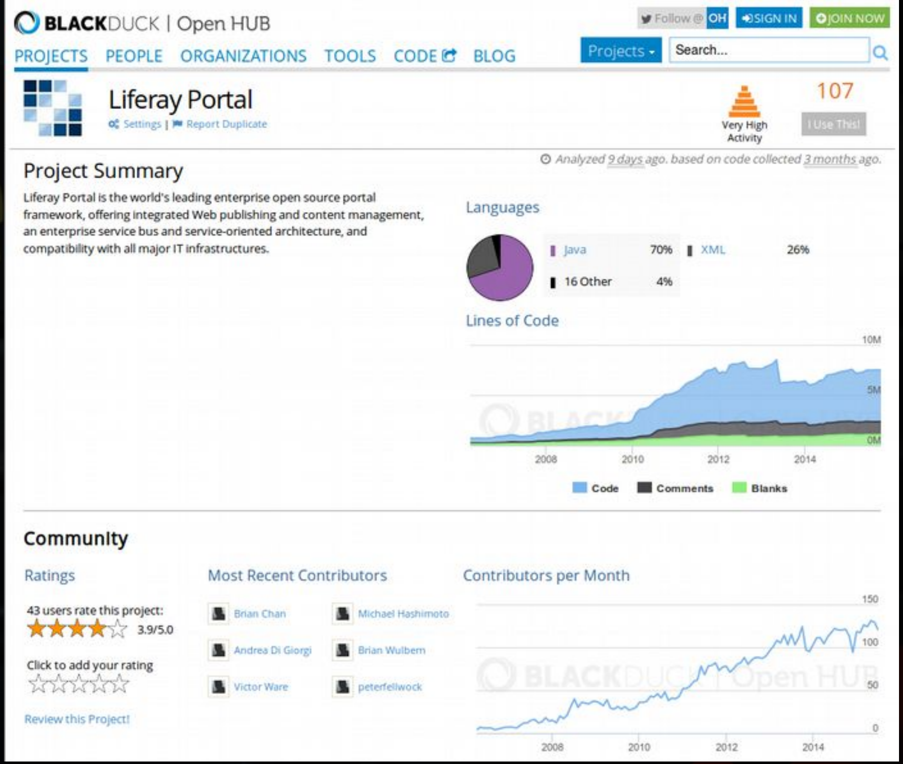
"... once modularization becomes part of the Java core tool set, developers will begin to embrace it en-masse, and as they do so, they will seek more robust and more mature solutions. Enter OSGi!"

Victor Grazi

## When
an application
needs
**modularity**
at
higher level ?

The essence of modularity is

**Not Knowing**



BLACKDUCK | Open HUB

Follow @ OH · SIGN IN · JOIN NOW

PROJECTS · PEOPLE · ORGANIZATIONS · TOOLS · CODE · BLOG · Projects · Search...

Liferay Portal

Settings | Report Duplicate

Very High Activity

107

I Use This!

Analyzed 9 days ago. based on code collected 3 months ago.

Project Summary

Liferay Portal is the world's leading enterprise open source portal framework, offering integrated Web publishing and content management, an enterprise service bus and service-oriented architecture, and compatibility with all major IT infrastructures.

Languages

| | | | |
|---|---|---|---|
| Java | 70% | XML | 26% |
| 16 Other | 4% | | |

Lines of Code

Community

Ratings

43 users rate this project:
★★★★☆ 3.9/5.0

Click to add your rating
☆☆☆☆☆

Review this Project!

Most Recent Contributors

| | |
|---|---|
| Brian Chan | Michael Hashimoto |
| Andrea Di Giorgi | Brian Wulbern |
| Victor Ware | peterfellwock |

Contributors per Month

SOME EXAMPLES OF HOW LIFERAY® DEALS WITH NOT KNOWING

- 1 platform
- over 100 apps
- over 600 modules
- over 2500 µServices

The essence of modularity is **Not Knowing**

## Some examples of how LIFERAY deals with not knowing

```
Require-Capability: \
    osgi.contract; \
    filter:="(&(osgi.contract=JavaJAXRS)(version=2))"
```

```
Provide-Capability: \
    osgi.contract; \
        osgi.contract=JavaJAXRS; \
        uses:=      "javax.ws.rs, \
                     javax.ws.rs.core, \
                     javax.ws.rs.client, \
                     javax.ws.rs.container, \
                     javax.ws.rs.ext"; \
        version:Version=2
```

The essence of modularity is **Not Knowing**

SOME EXAMPLES OF HOW LIFERAY DEALS WITH NOT KNOWING

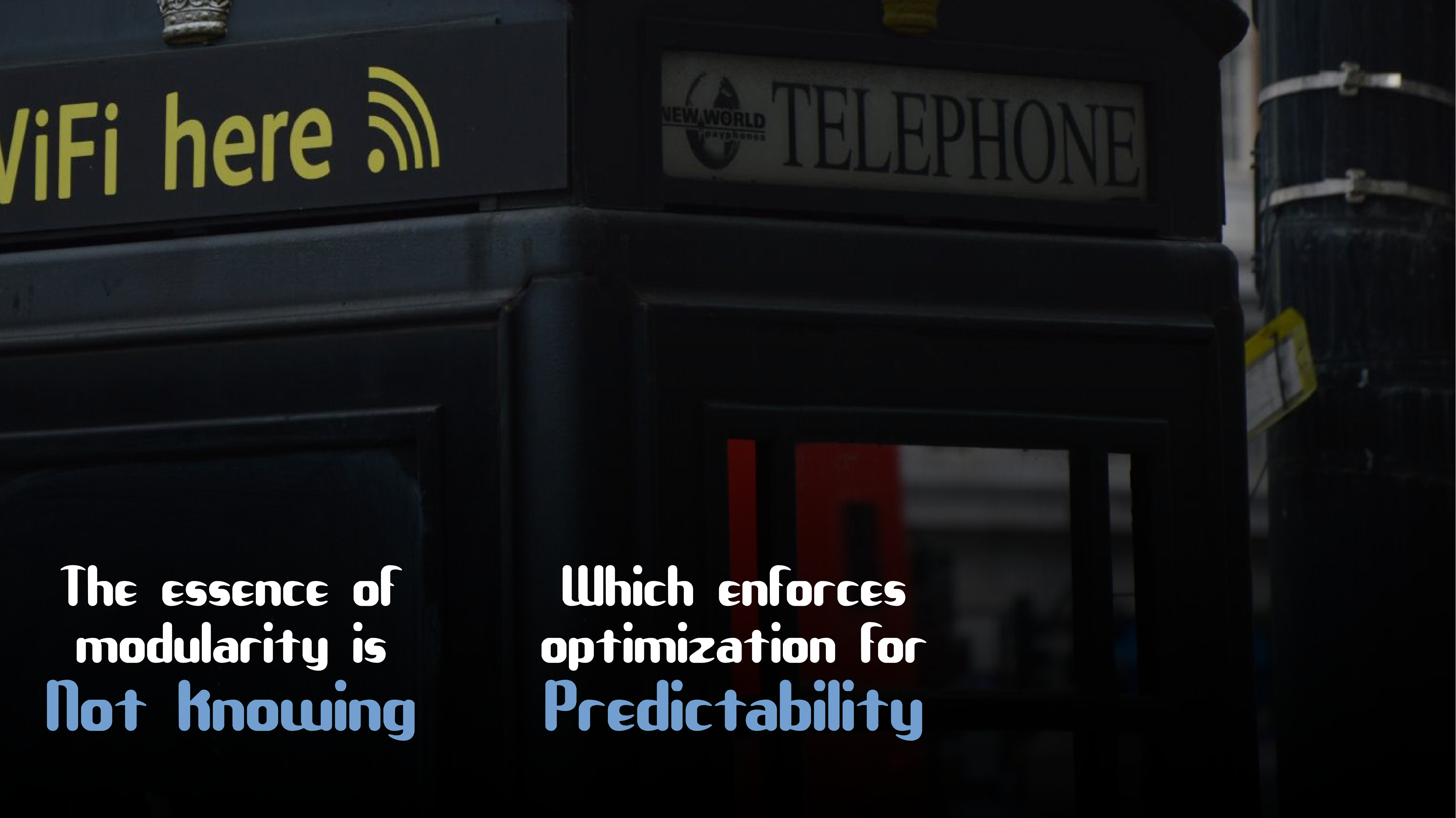
```
@Component(
 immediate = true,
 property = {"javax.portlet.name=other_Portlet"},
 service = PortletFilter.class
)
public class MyFilter implements RenderFilter {

    ...
```

The essence of modularity is **Not Knowing**

Some examples of how LIFERAY deals with not knowing

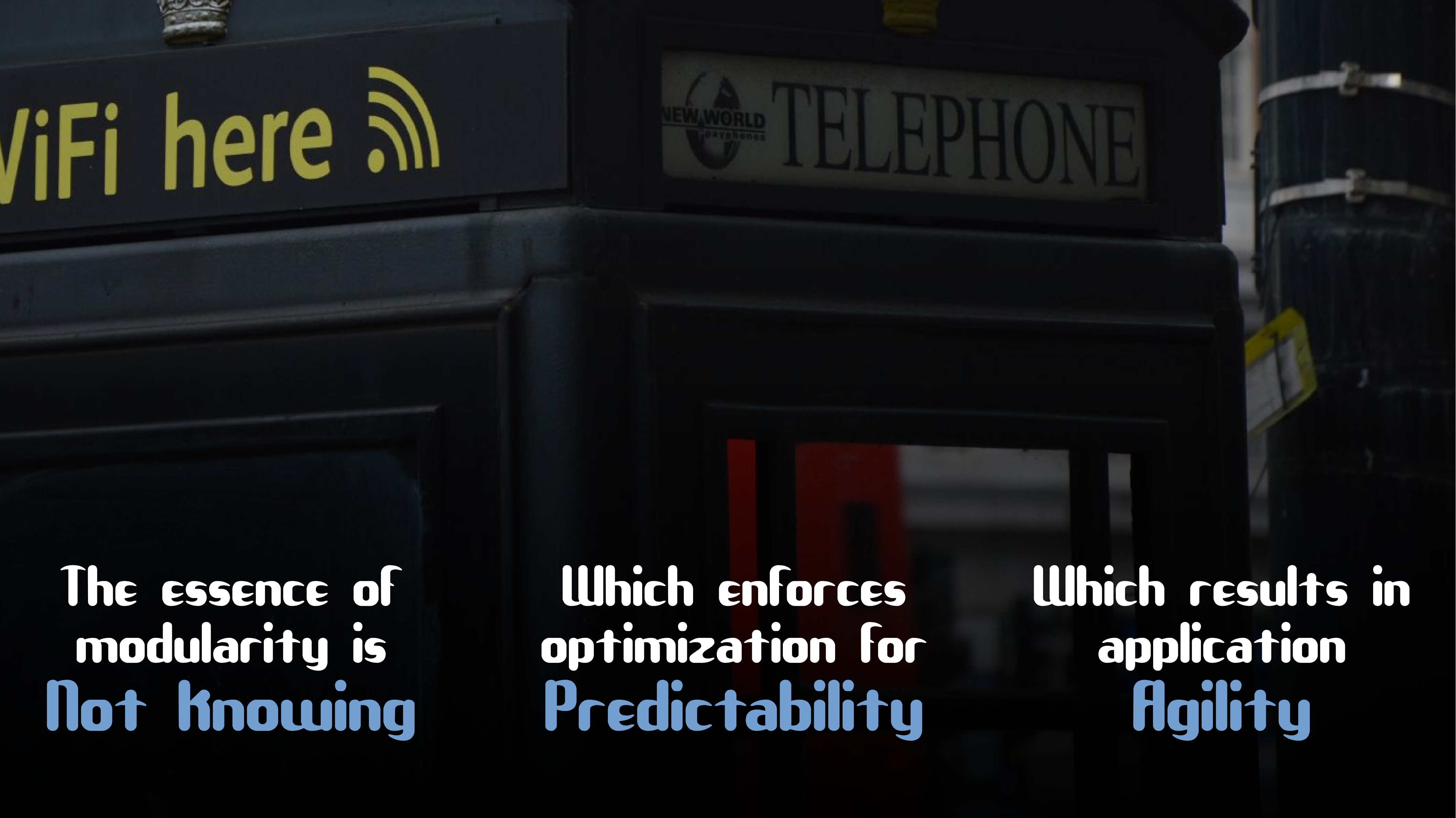
```
@Component(
 immediate = true,
 property = {"destination.name=" + MONITORING},
 service = {MessageListener.class}
)
public class MonitoringMessageListener ...

   @Reference(
    cardinality = ReferenceCardinality.MULTIPLE,
    policy = ReferencePolicy.DYNAMIC,
    policyOption = ReferencePolicyOption.GREEDY
   )
   protected synchronized void registerProcessor(
       ...
```

The essence of modularity is **Not Knowing**



The essence of modularity is
**Not Knowing**

Which enforces optimization for
**Predictability**



The essence of modularity is **Not Knowing**

Which enforces optimization for **Predictability**

Which results in application **Agility**

MOD CONF

MODULARITY CONFERENCE

NOV.15 | DARMSTADT, GERMANY

milen.dyankov@liferay.com
@MilenDyankov

http://www.liferay.com
@Liferay