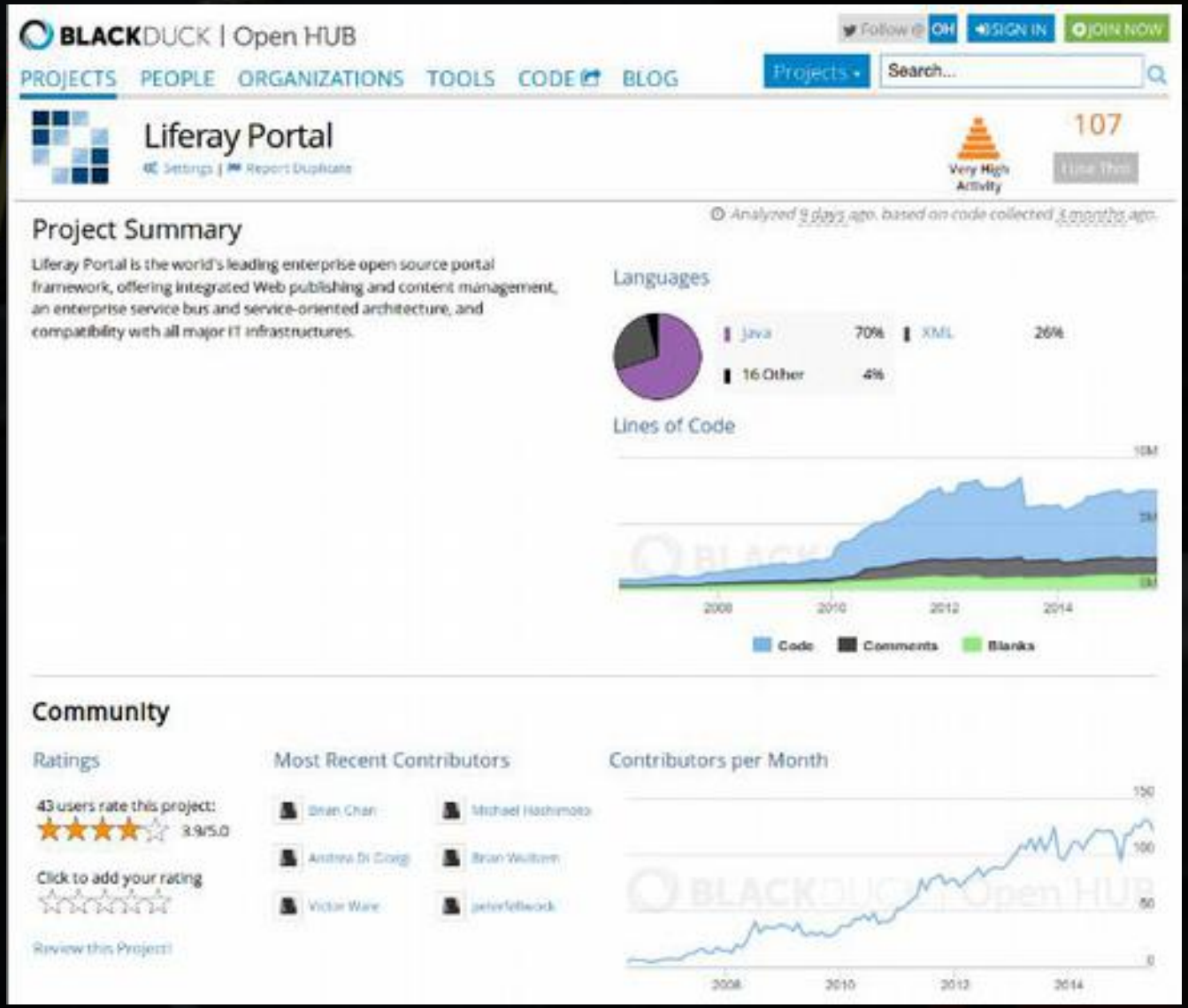
Milen Dyankov
@milendyankov

Developer Advocate
@Liferay

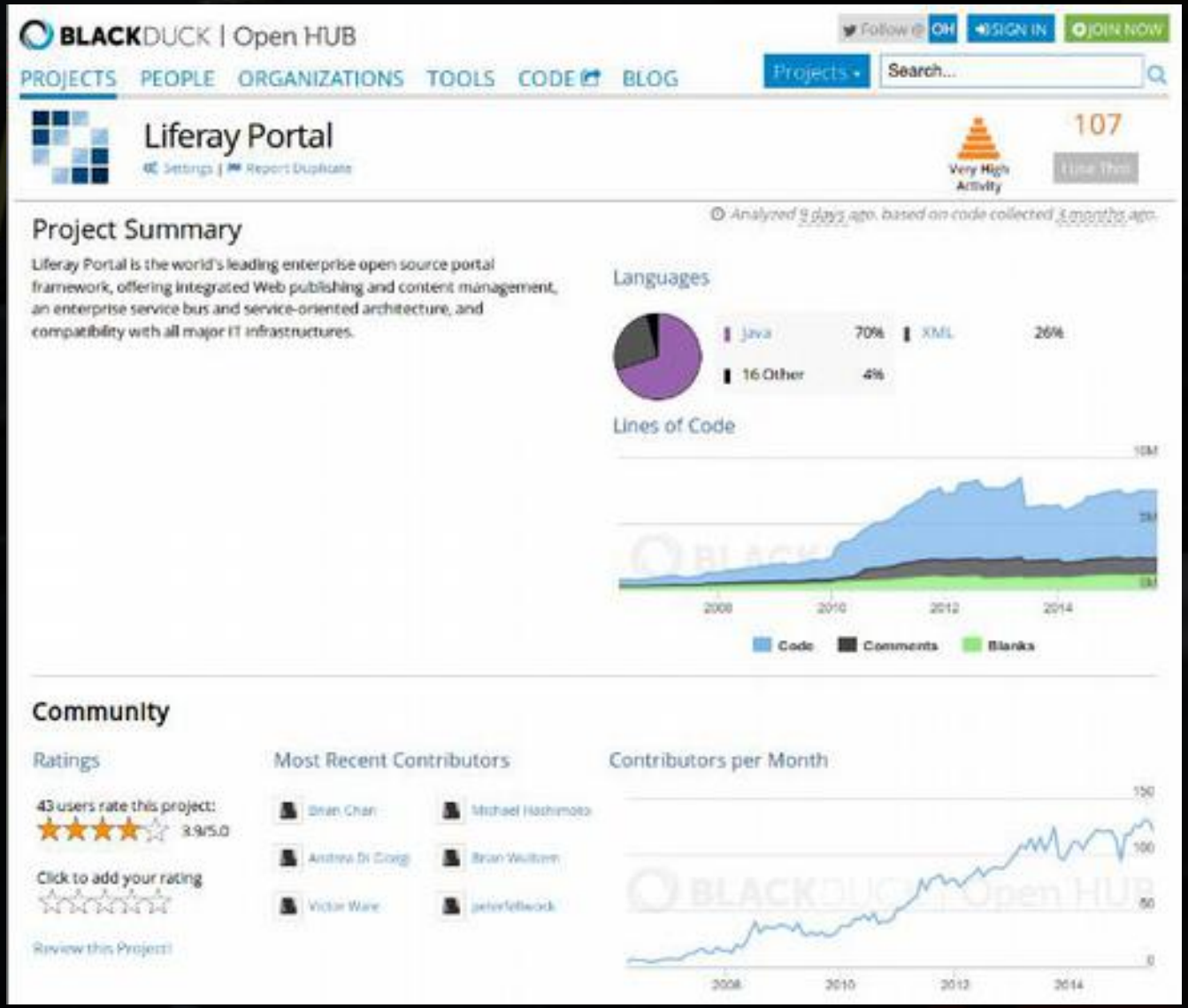
# What's not new in modular Java!




# LIFERAY

## BEFORE VERSION 7.0

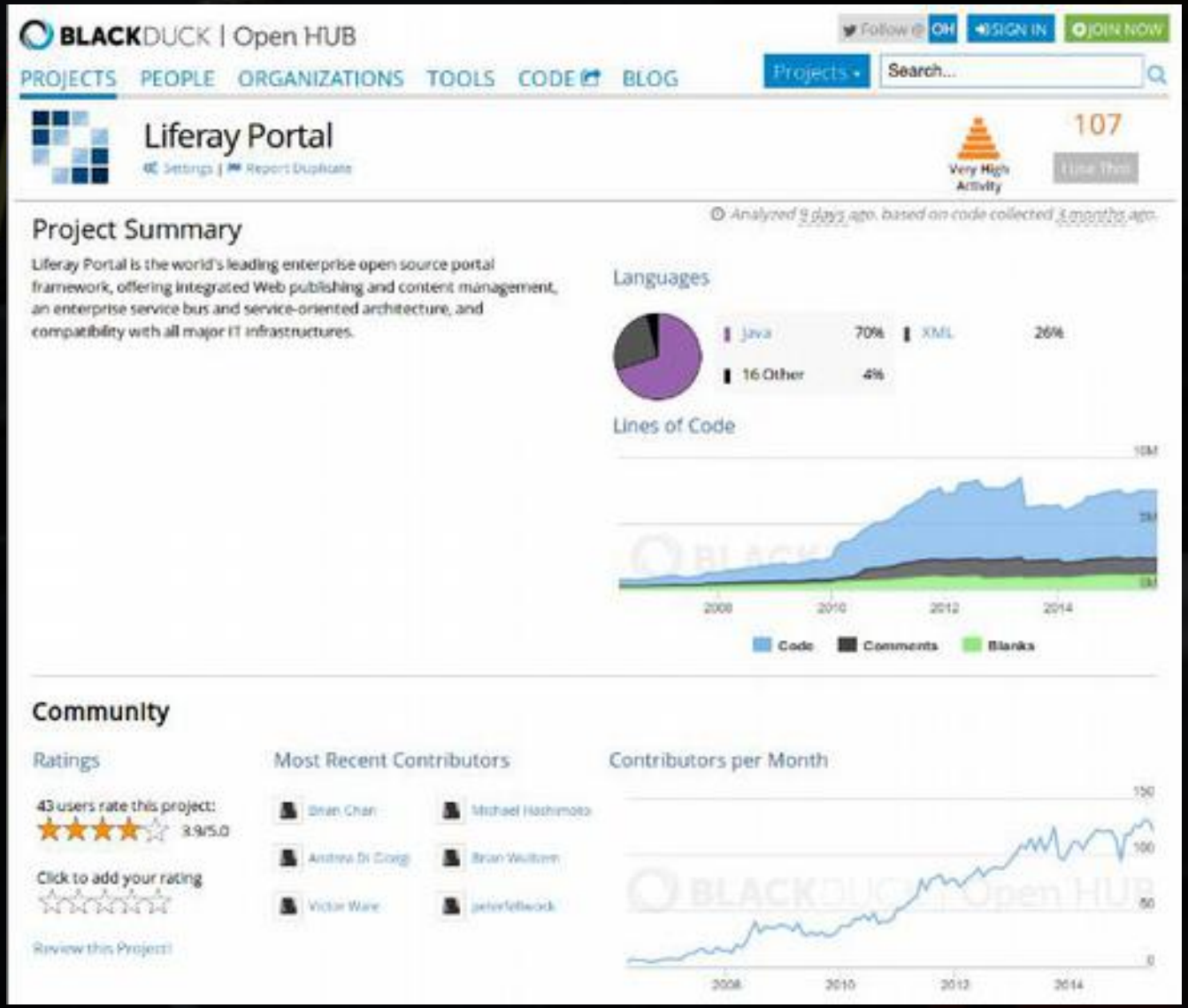
- ~400mb WAR file
- Custom extensions as WARs
- Classloader proxies
- Hacks to bend JEE rules
- Custom code per AppServer
- No enforced encapsulation




# LIFERAY

FROM 7.0 ONWARDS

- ~150mb WAR file
- Standard OSGi runtime
- Extensions are OSGi modules
- Enforced encapsulation
- Mostly AppServer agnostic
- Independently deployable μServices in the same JVM




LIFERAY

TODAY

1 platform

over 100 apps

over 600 modules

over 2500 µServices

# Why choose OSGi when JPMS brings modularity in JDK?


BE AWARE
OF
INVISIBILITY
Please Follow
ONE-WAY ARROWS
McD
Mahalo!


2005

JSR 277

JSR 294

2014

JSR 376

JEP 200

JEP 201

JEP 220

JEP 260

JEP 261

...

JSRs

Search JSRs

» JSRs by Platform
» JSRs by Technology
» JSRs by Stage
» JSRs by Committee
» List of All JSRs

My JCP

Sign-in

Register for Site



JCP Info

» About JCP
» Get Involved
» Community Resources

# JSR #376
# Java™ Platform Module System
# Public Review Ballot

**Ballot duration: 2017-04-25 to: 2017-05-08**

These are the final results of the Public Review Ballot for JSR #376. The EC has not approved this ballot.

**Votes**

EC

| | | | |
|---|---|---|---|
| Azul Systems, Inc. ✓ | Credit Suisse ✗ | Eclipse Foundation, Inc ✗ | Fujitsu Limited ✓ |
| Gemalto M2M GmbH ✓ | Goldman Sachs & Co. ✓ | Grimstad, Ivar ✗ | Hazelcast ✗ |
| Hewlett Packard Enterprise ✗ | IBM ✗ | Intel Corp. ✓ | Keil, Werner ✗ |
| London Java Community ✗ | MicroDoc ✓ | NXP Semiconductors ✓ | Oracle ✓ |
| Red Hat ✗ | SAP SE ✗ | Software AG ✗ | SouJava ✓ |
| Tomitribe ✗ | Twitter, Inc. ✗ | V2COM ✓ | |

**Icon Legend**

- Yes ✓
- No ✗
- Abstain
- Not voted ☐

JAVA 9 | September 22, 2017

# Java 9 Release Now Available!

By: Yolande Poirier

The Java 9 release introduces more than 150 new features including the module system, which enables developers to scale down the Java SE platform for small devices, improves performance and security, and makes it easier to construct and maintain libraries and large applications.

Think of

not new

as in

not new

concept

and not as in

not new

car

# 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 |
| 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 |


# Buzzword compliant

# Modularity Maturity Model

# Buzzword compliant

# Modularity Maturity Model

# Buzzword compliant
# Modularity Maturity Model

Level 1 Monolith

Level 2 Composite

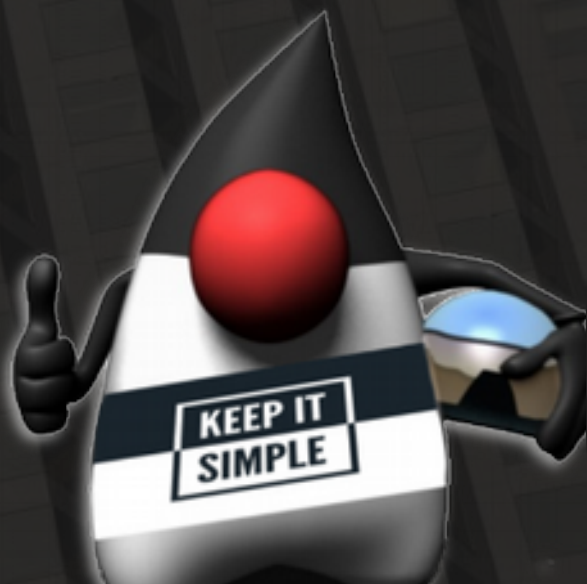
Level 3 Containers

Level 4 Discovery

Level 5 µServices

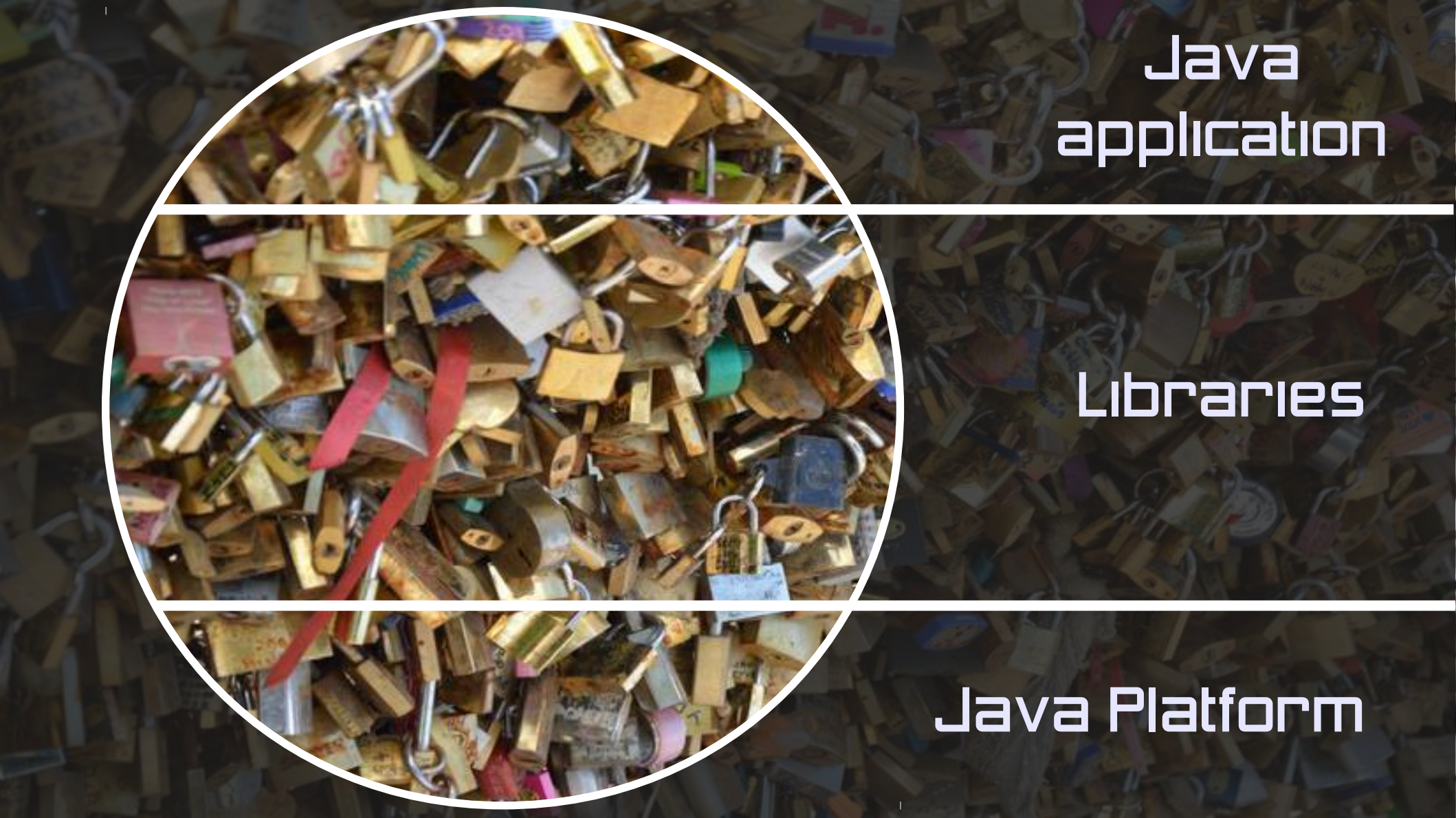
JPMS

OSGi



# What is modularity from application perspective ?


Java
application


Java
application
Libraries
Java Platform


OSGi

There is nothing
we can do about it!

OSGi

class loaders

There is nothing
we can do about it!

OSGi

Dynamic multi-layer
modular runtime!

class loaders

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!

OSGi


"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

JPMS

Modules are
first class citizens!

JPMS

Nothing to do about it,
just use modules!

Modules are
first class citizens!

JPMS

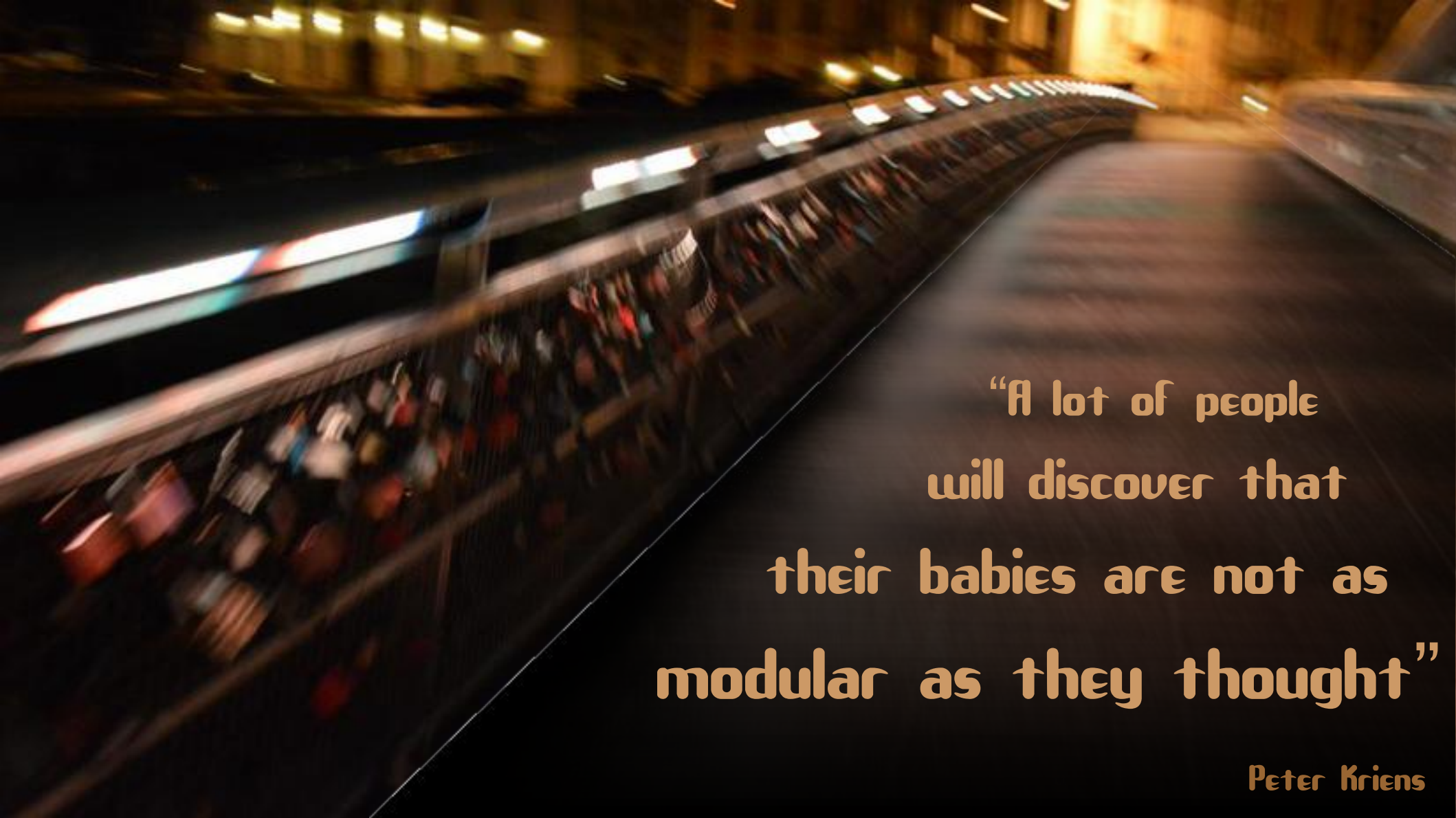
Nothing to do about it,
just use modules!

It's so easy,
everyone
will
release
modules!

Modules are
first class citizens!

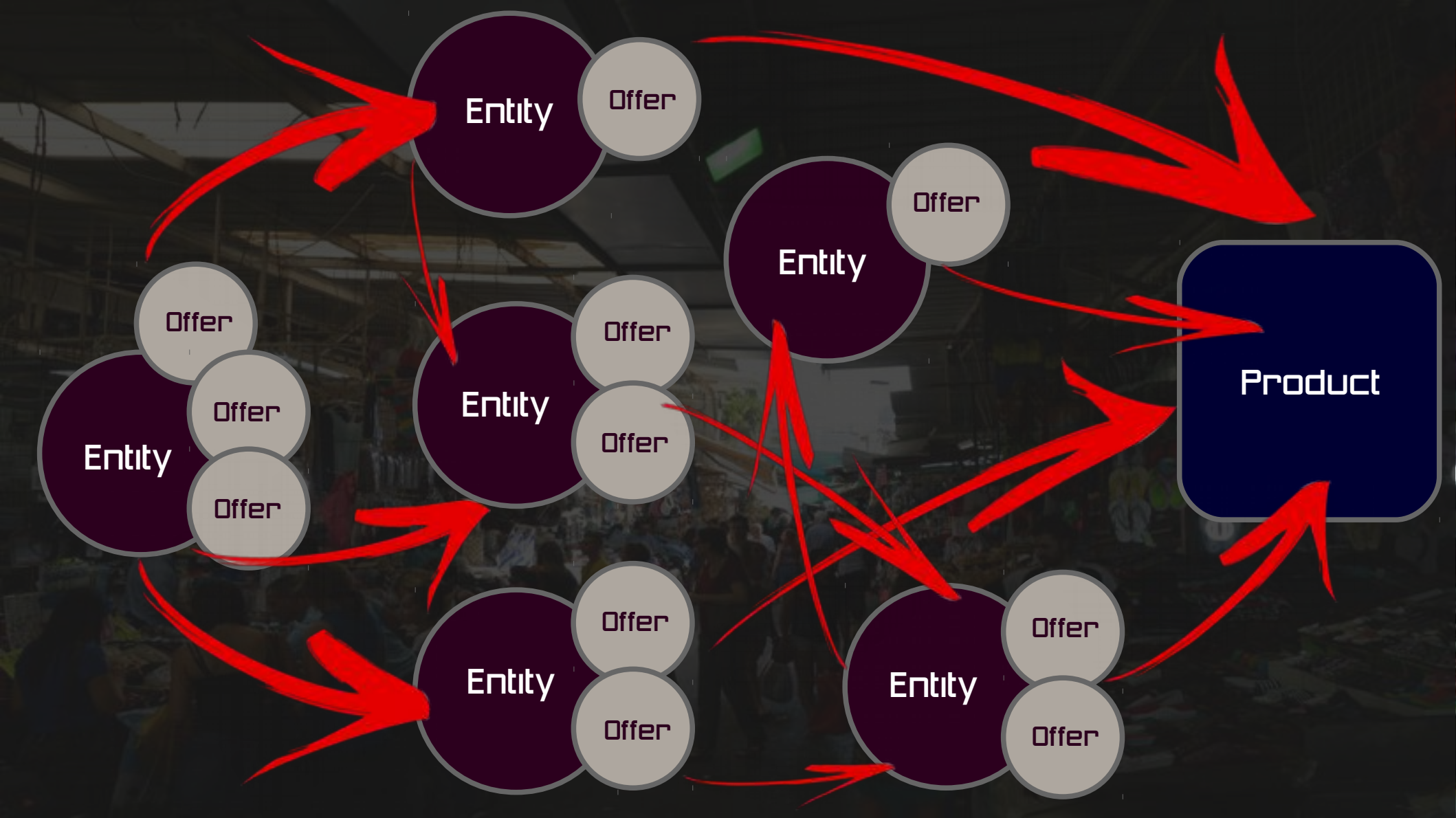
JPMS

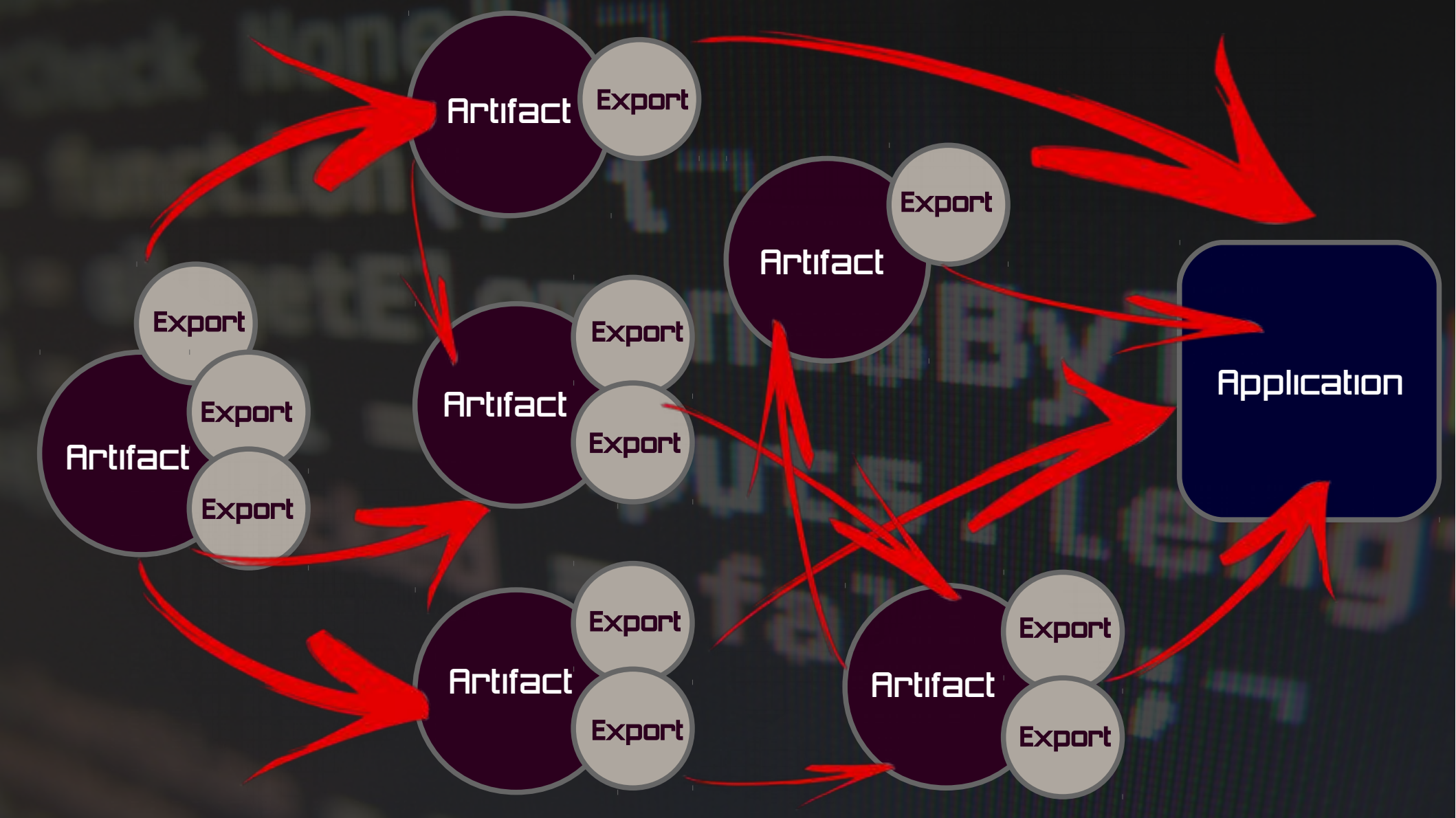

not new
except now
you kinda
have to
new
Modular
Java SE
Applications!
Modular
Java SE
Platform!


"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


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


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

# Level 2
## decoupled from artifact

Artifact

## OSGi


**MANIFEST.MF**

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

...
```

## JPMS

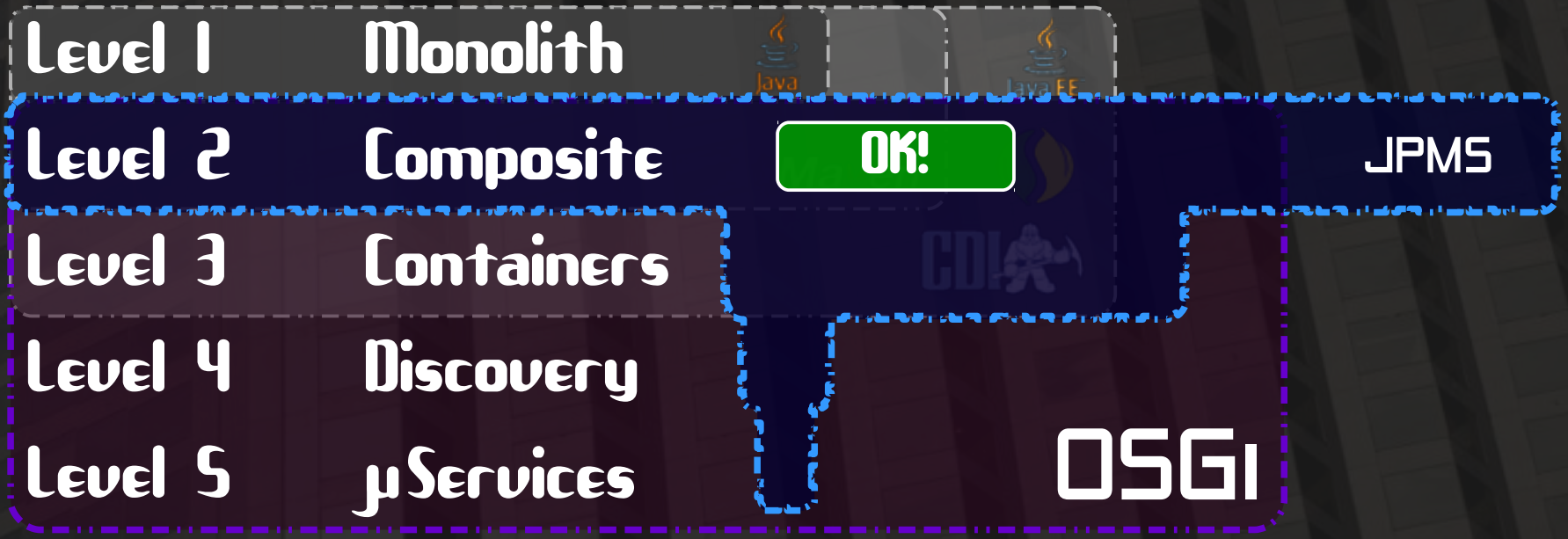


**module-info.java**

```
module com.mycompany.mymodule {

...

}
```

# Buzzword compliant

# Modularity Maturity Model



# Level 3
## decoupled from identity


## OSGi

**MANIFEST.MF**

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

Export-Package: \
     com.mycompany.mypackage

...
```

## JPMS

**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)",...

...
```

## JPMS

**module-info.java**

```
module com.mycompany.mymodule {

  requires other.module;

...

}
```

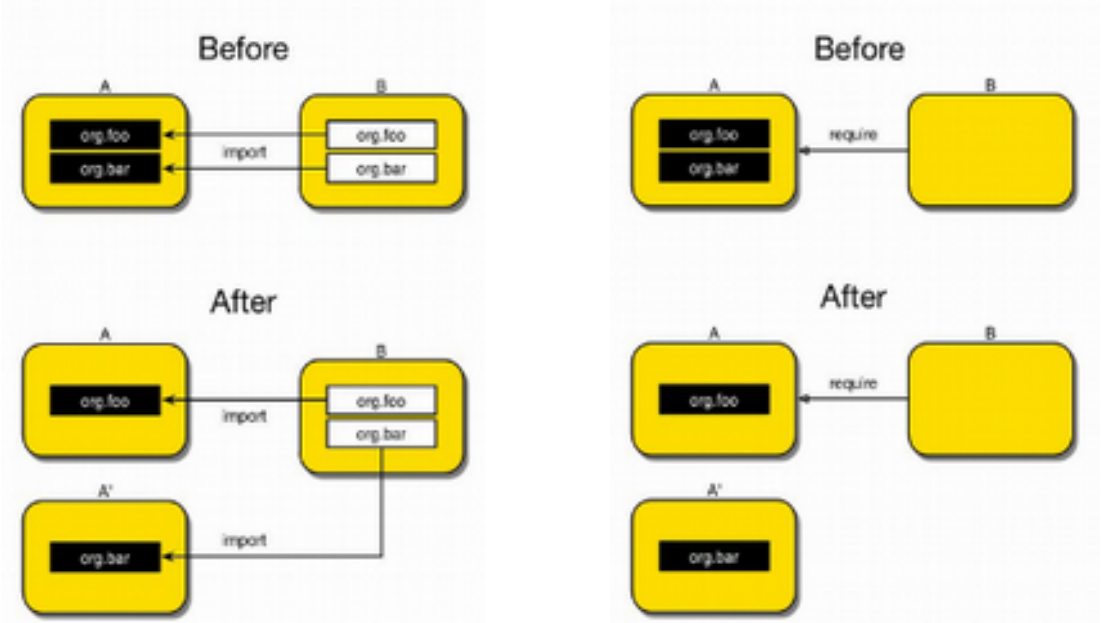
# Level 3
## decoupled from identity


InfoQ
En | 中文 | 日本 | Fr | Br
1,297,957 Oct unique visitors

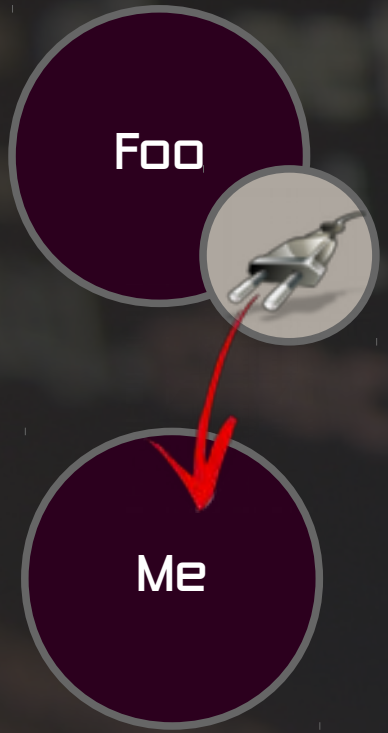
Development | Architecture & Design | Data Science | Culture & Methods | DevOps

### Java 9, OSGi and the Future of Modularity (Part 1)

Posted by: Neil Bartlett and Kai Hackbarth on Sep 22, 2016



# 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!

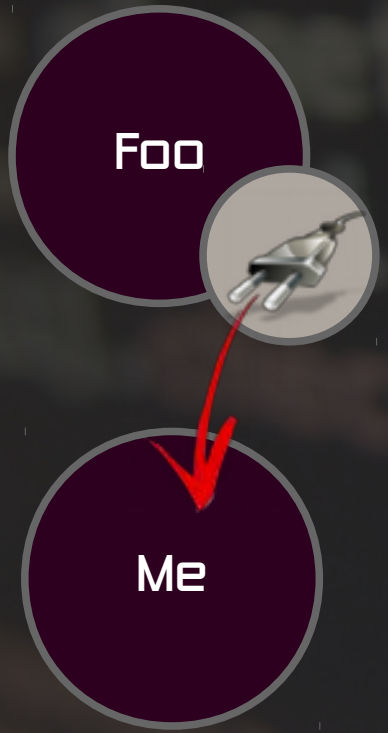
## JPMS

**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!

## JPMS

**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**

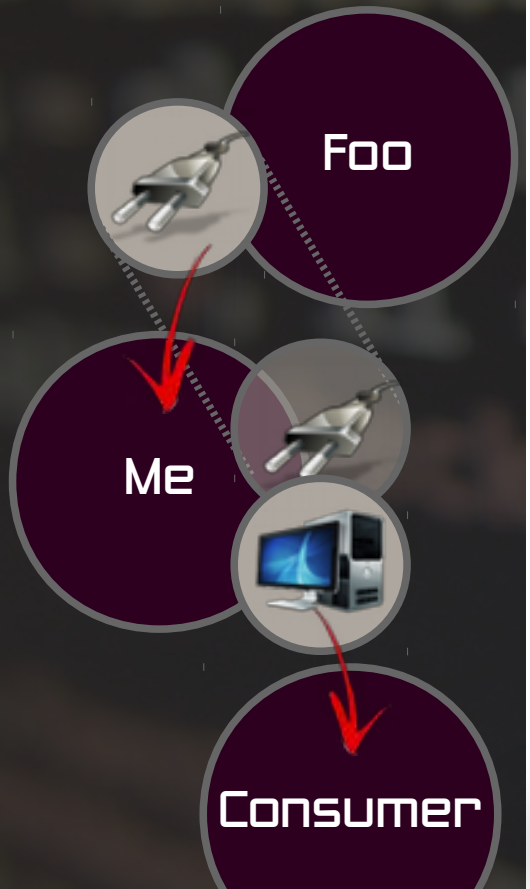
```
Manifest-Version: 1.0
Bundle-SymbolicName: \
     com.mycompany.mymodule

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

...
```

## JPMS

**module-info.java**

```
module com.mycompany.mymodule {

 exports com.mycompany.mypackage;

 requires transitive 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!

## JPMS

**module-info.java**

```
module com.mycompany.devices {

 exports com.mycompany.pc;

 requires transitive foo.tools;

 ...

}
```

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

# Buzzword compliant

# Modularity Maturity Model

# Level 4
## decoupled from implementation


# OSGi

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

# JPMS

- Nothing OOTB. Use OSGi :)
- Probably doable via external resolver dynamic modules and layers
- Jakarta EE or other solutions on top of JPMS may provide solutions

# Buzzword compliant

# Modularity Maturity Model

# 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

# JPMS

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

# Buzzword compliant

# Modularity Maturity Model

# Tooling

# OSGi

Automated discovery and module generation thanks to

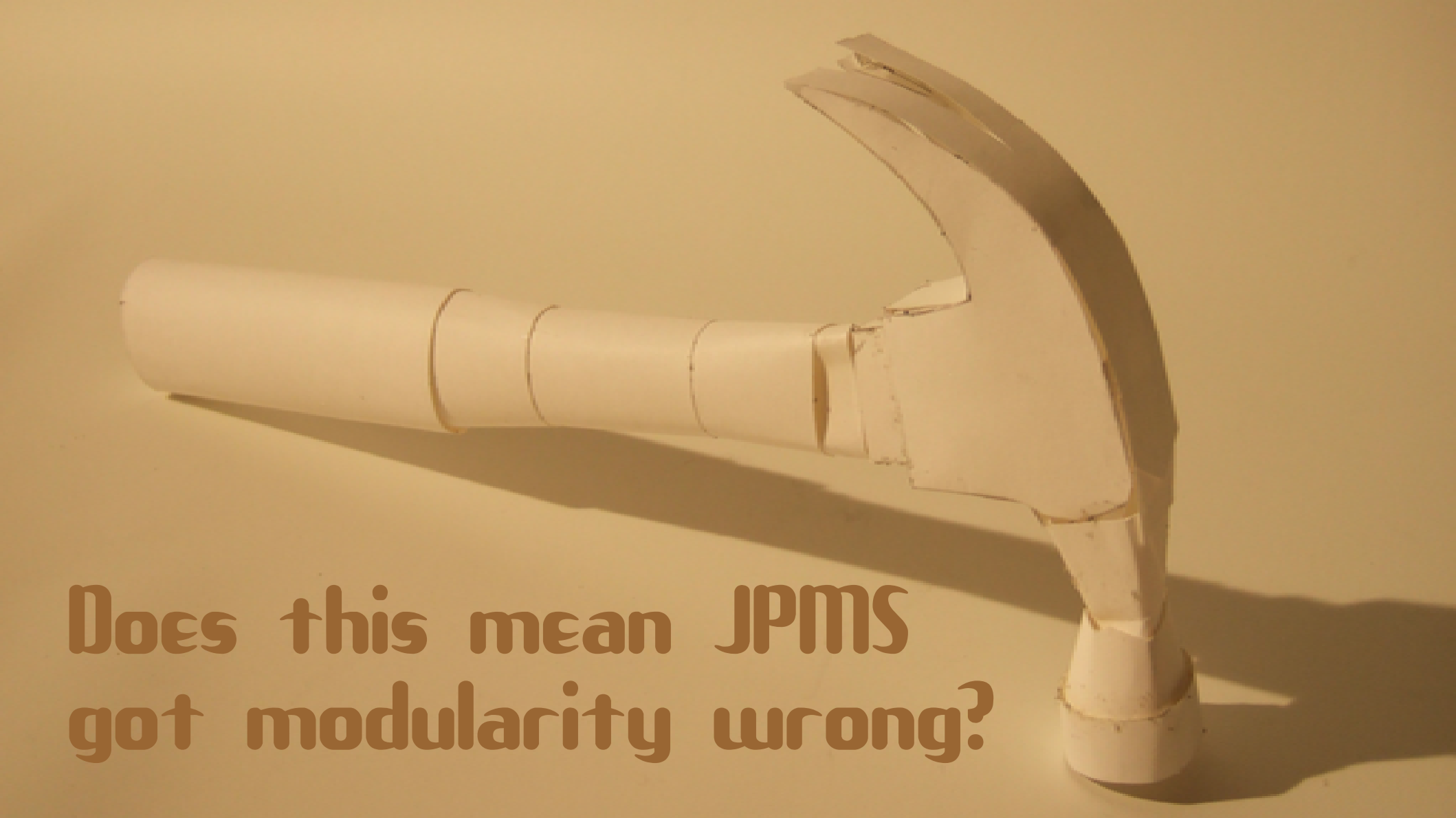

- Maven
- Gradle
- Ant
- Eclipse (bndtools)
- IntelliJ (OSMORC)
- NetBeans

# JPMS

Manual dependency management.

Lots of data duplication for both dependencies and services

- Maven
- Gradle
- Eclipse
- IntelliJ
- NetBeans


Does this mean JPMS
got modularity wrong?

When they say

# modular Java

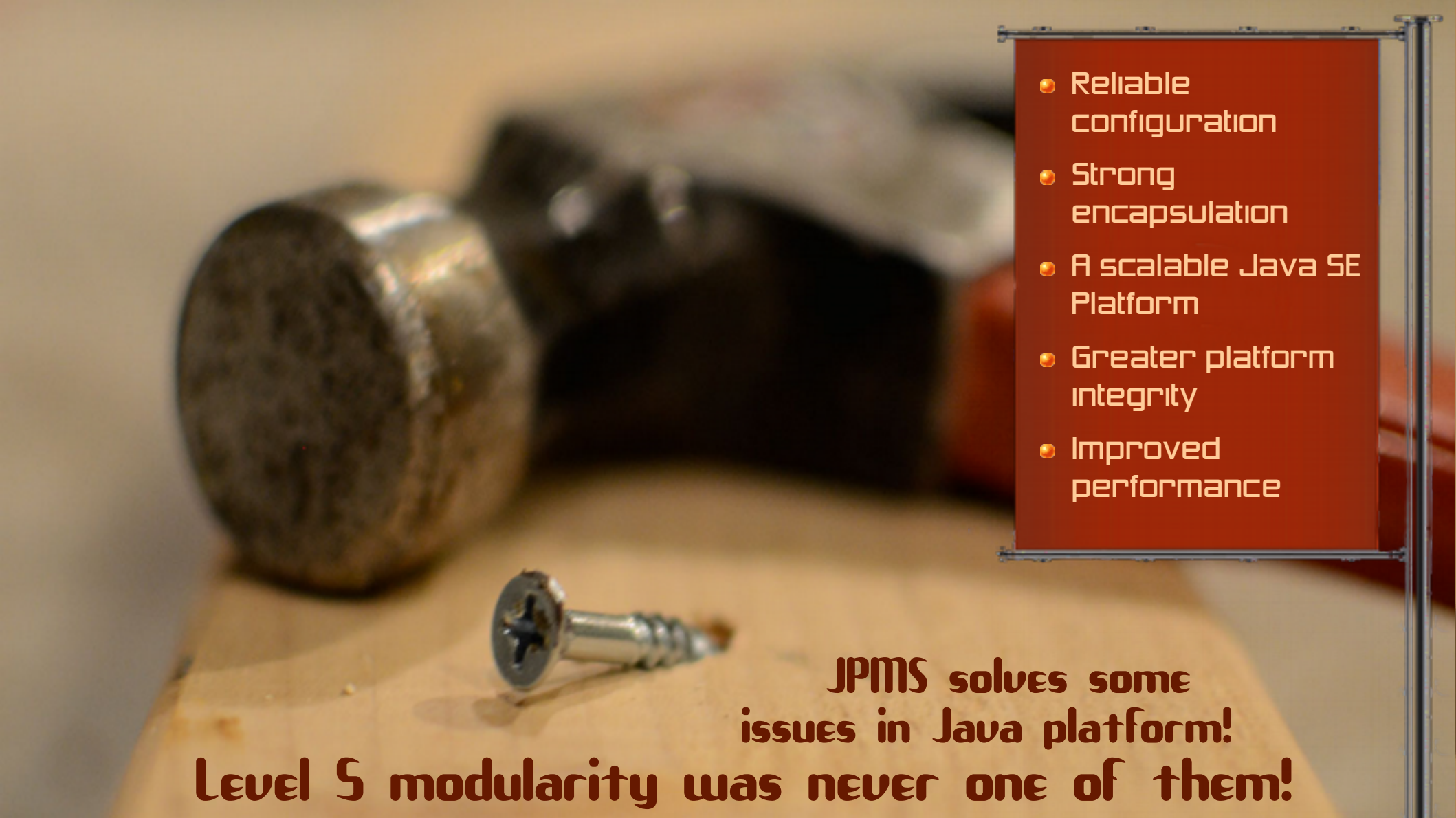
it means

just what they

choose it to mean -

neither

more nor less!

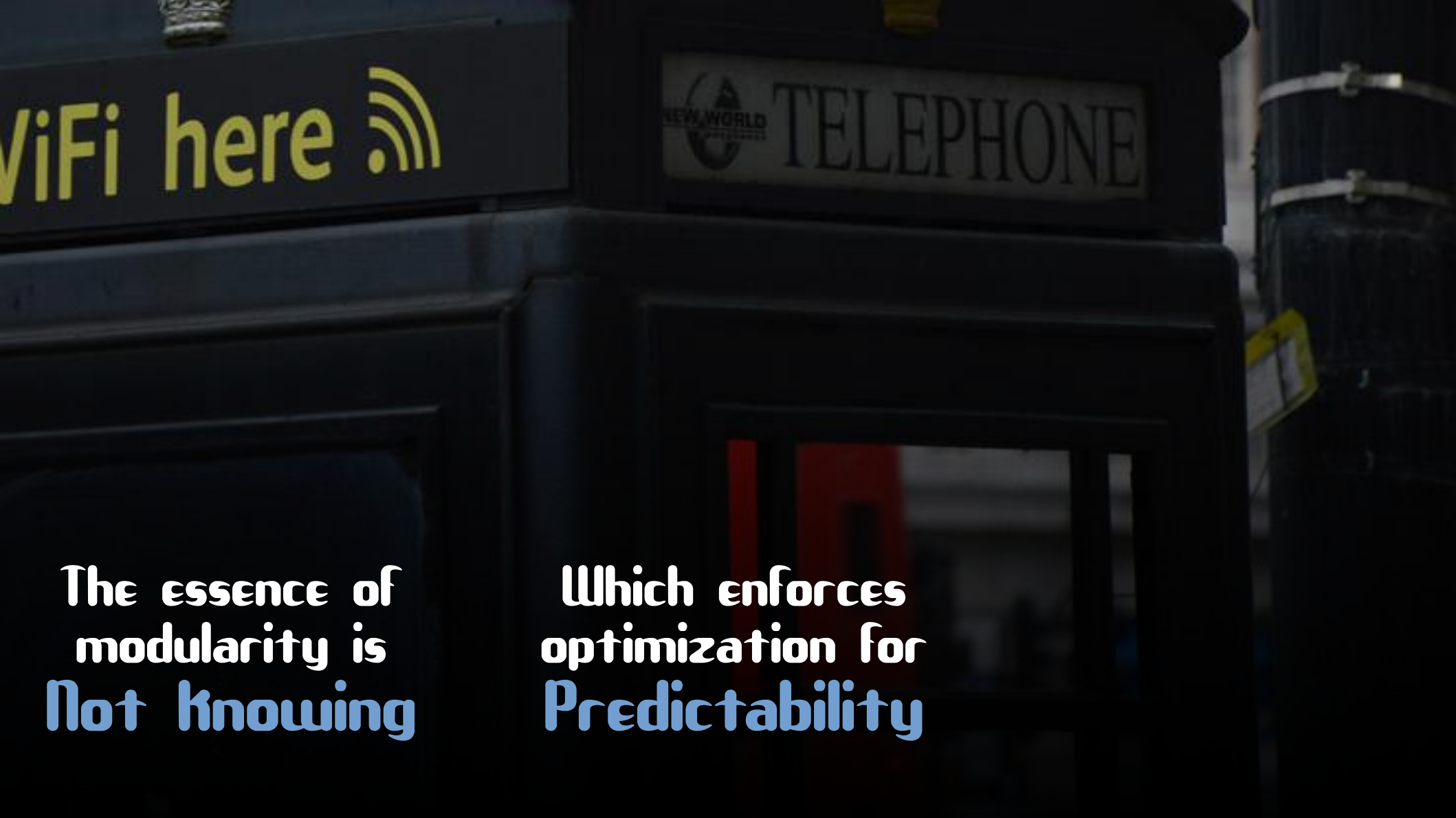

Reliable configuration
Strong encapsulation
A scalable Java SE Platform
Greater platform integrity
Improved performance
JPMS solves some issues in Java platform!
Level 5 modularity was never one of them!

"... 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

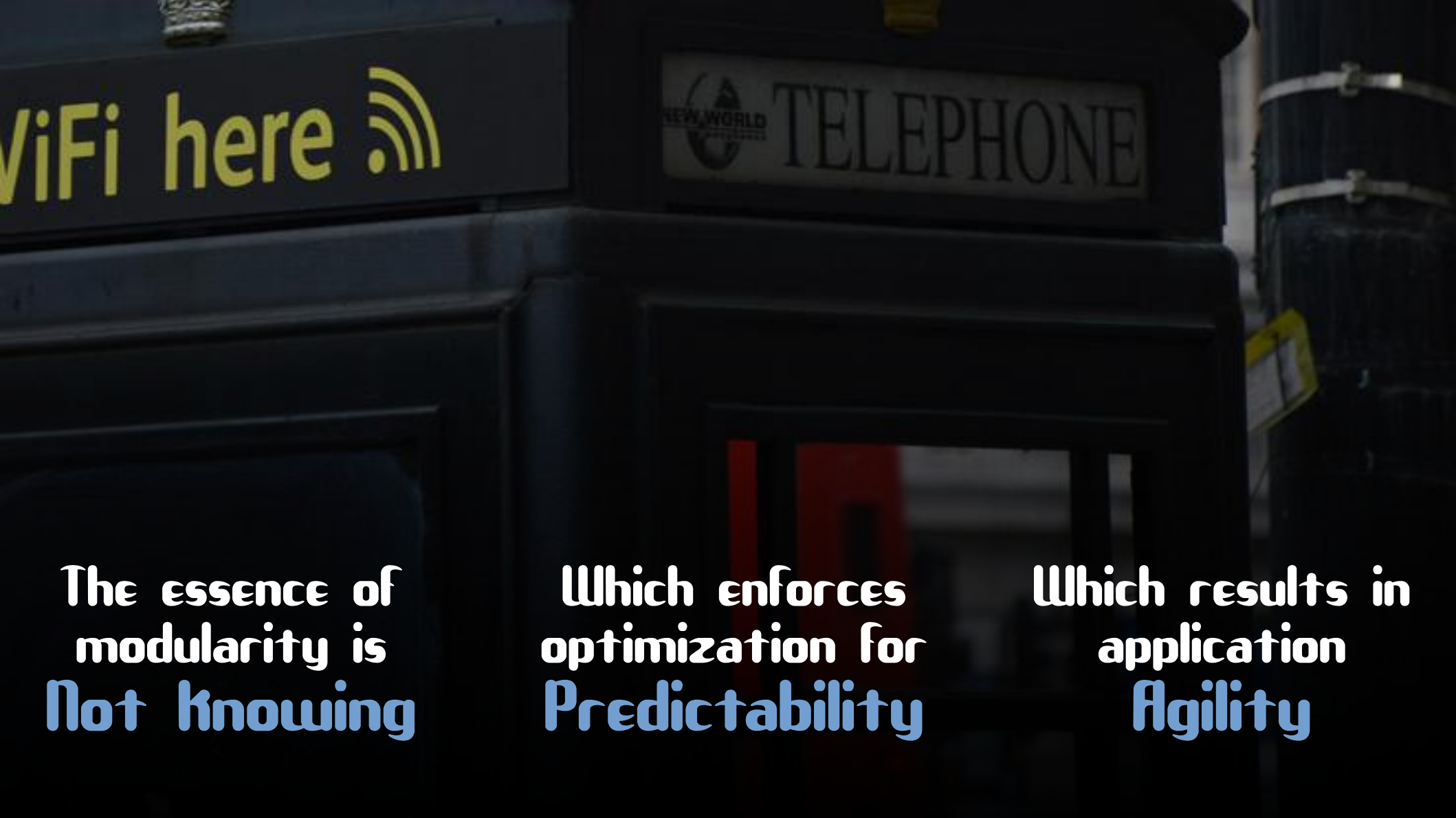



The essence of modularity is

**Not knowing**

Which enforces optimization for

**Predictability**




The essence of modularity is

**Not knowing**

Which enforces optimization for

**Predictability**

Which results in application

**Agility**

MILEN.DYANKOV@LIFERAY.COM
@MILENDYANKOV

HTTP://WWW.LIFERAY.COM
@LIFERAY

# THANK YOU