Milen Dyankov
@milendyankov

# 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**

2005

| | |
|---|---|
| JSR | 277 |
| JSR | 294 |

2014

| | |
|---|---|
| JSR | 376 |
| JEP | 200 |
| JEP | 201 |
| JEP | 220 |
| JEP | 260 |
| JEP | 261 |

...

## Java application

Java application

Libraries

JRE/JDK

## OSGi

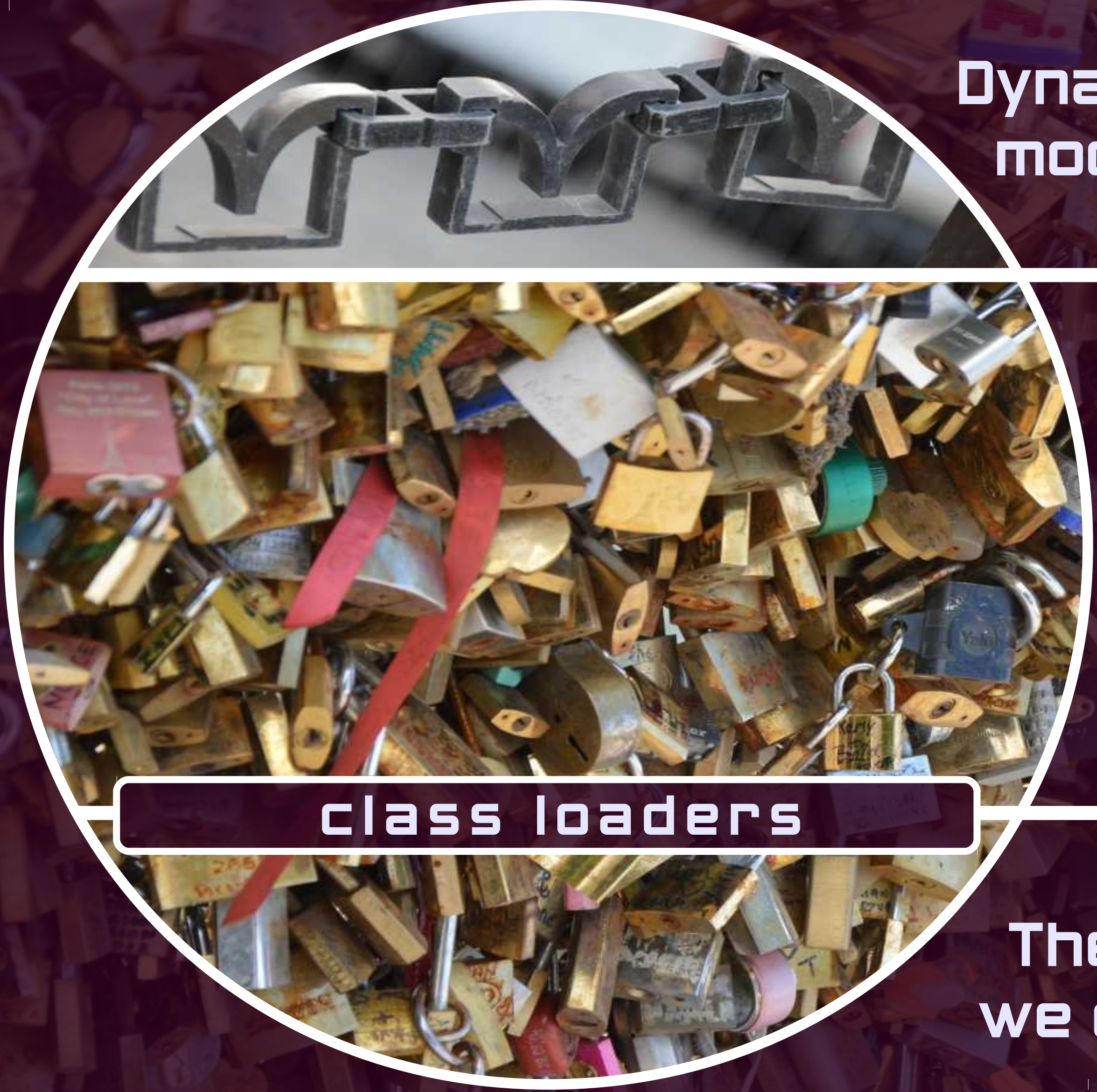
OSGi

There is nothing we can do about it!

class loaders

There is nothing we can do about it!

OSGi



Dynamic multi-layer modular runtime!

There is nothing we can do about it!

OSGi

## OSGi

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

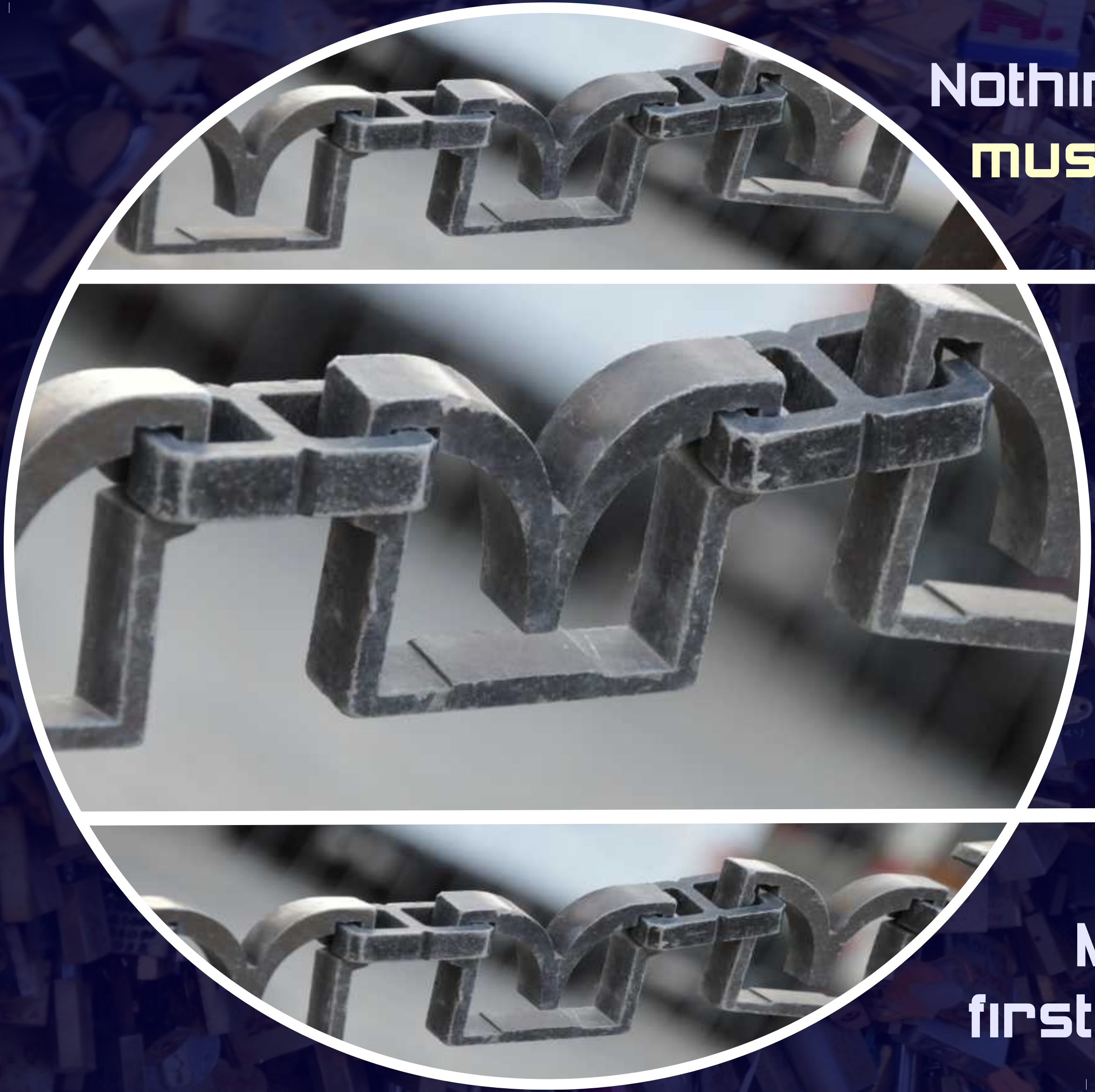
## JSR 376

Modules are
first class citizens!

## JSR 376

Nothing to do about it, **must** use modules!

Modules are first class citizens!



Nothing to do about it, **must** use modules!

It's so easy, everyone **must** release modules!

Modules are first class citizens!

JSR 376

not new

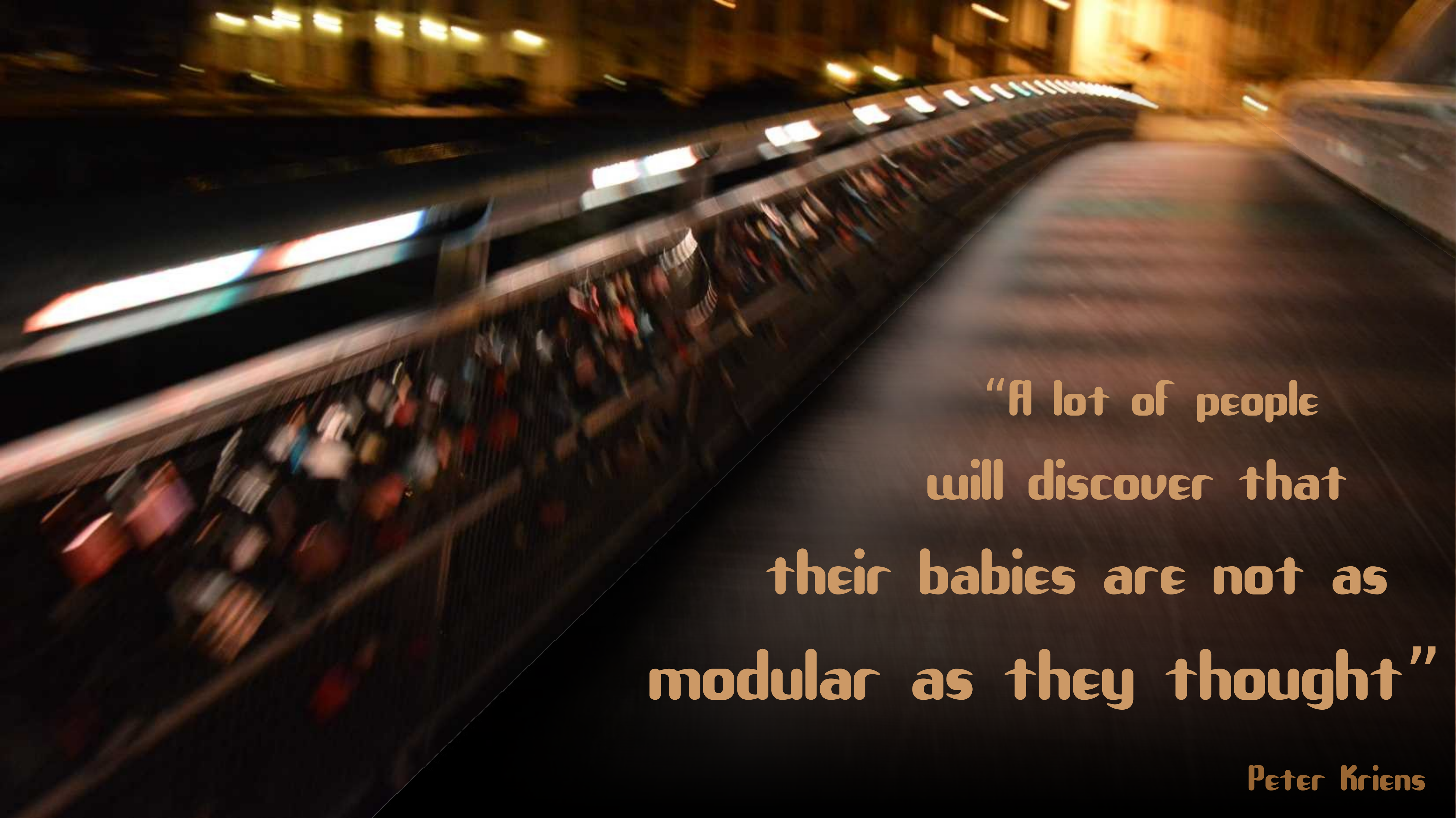
🙂

except now
you kinda
have to

Modular
Java SE
Applications!

new

Modular
Java SE
Platform!



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

Peter Kriens

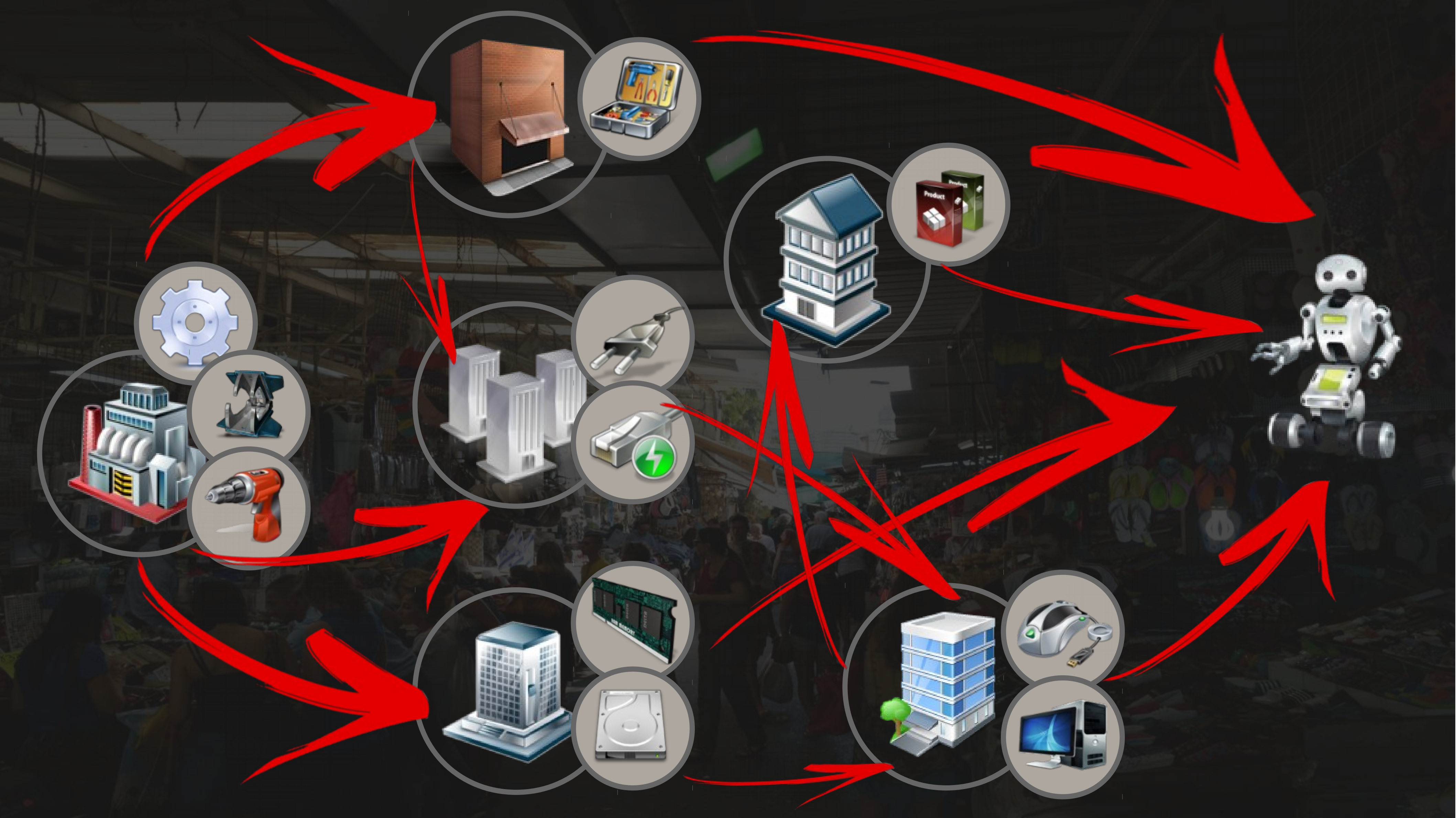
# product

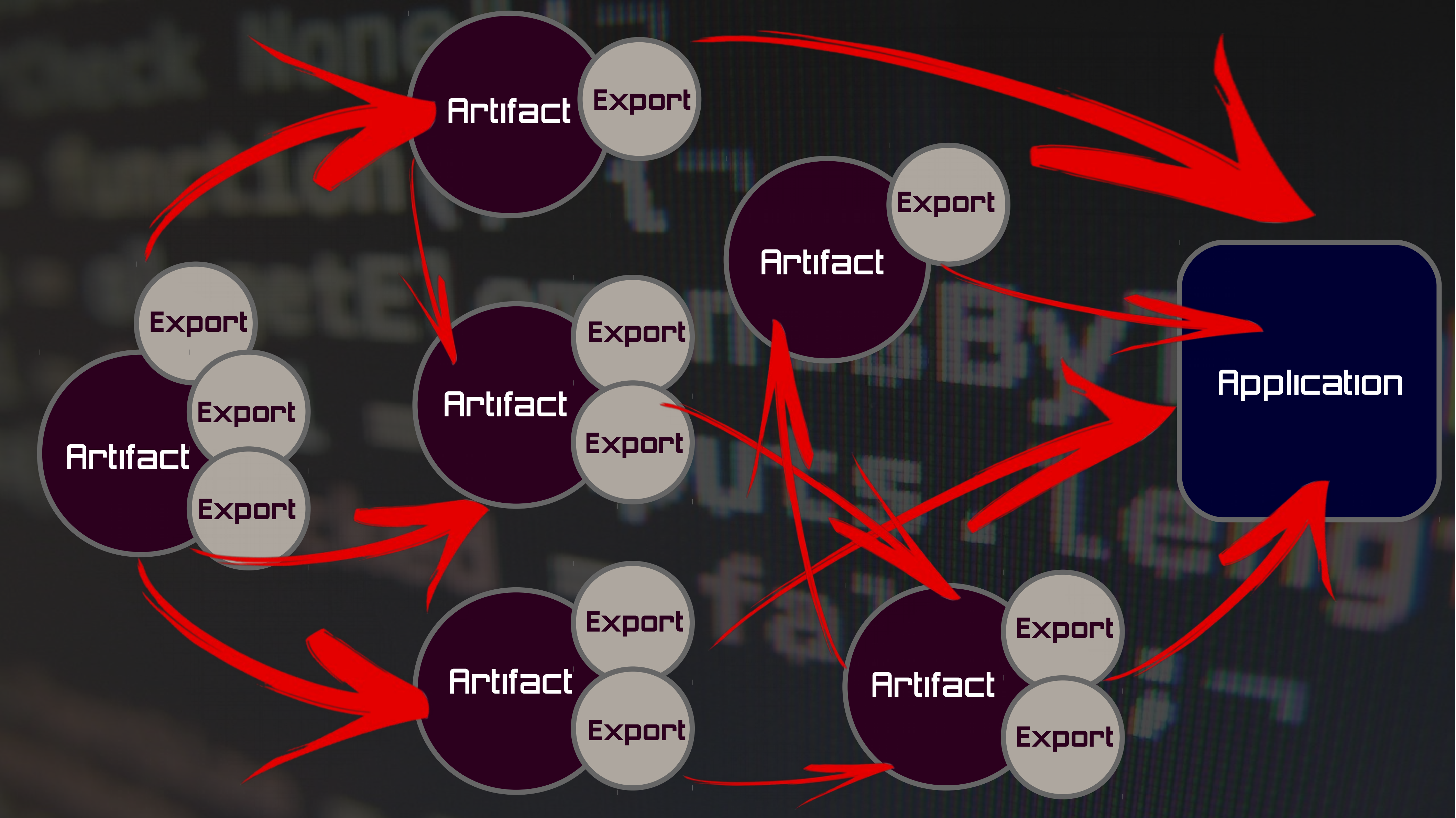
intermediate

intermediate

material

Product
Product


Product
Product


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

what is module ?

Artifact

## OSGi

**MANIFEST.MF**

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

...
```

## JSR 376

**module-info.java**

```
module com.mycompany.mymodule {

...

}
```

## modules can be producers

Export
Artifact

## OSGi

MANIFEST.MF

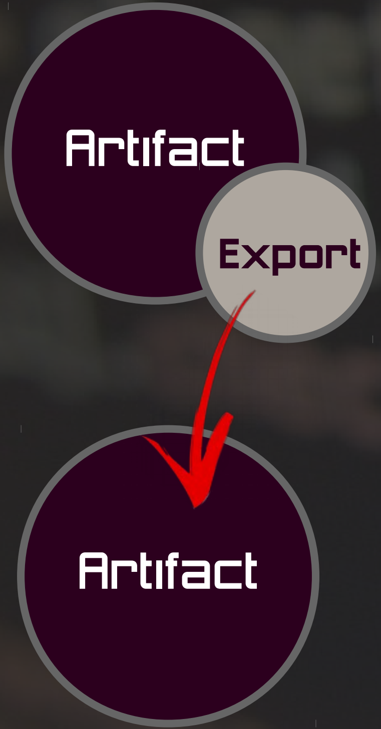
```
Manifest-Version: 1.0
Bundle-SymbolicName: \
      com.mycompany.mymodule

Export-Package: \
      com.mycompany.mypackage

...
```

## JSR 376

module-info.java

```
module com.mycompany.mymodule {

  exports com.mycompany.mypackage;

...

}
```

## modules can also be consumers



## OSGi

MANIFEST.MF

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

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

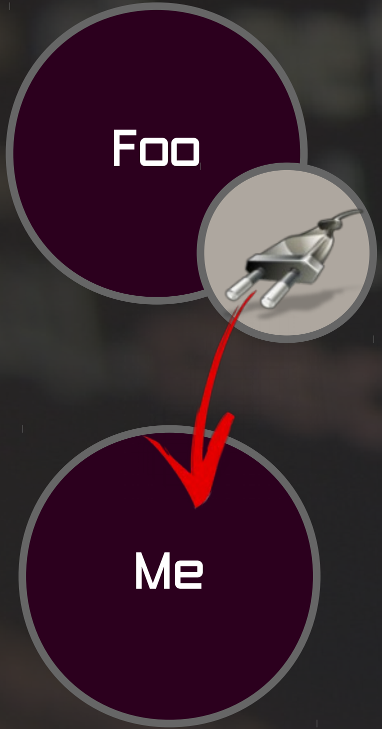
```
Import-Package: \
      com.some.package

...
```

## JSR 376

module-info.java

```
module com.mycompany.mymodule {

  requires other.module;

...

}
```

## modules can also be consumers



## OSGi

MANIFEST.MF

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

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

Import-Package: \
     com.intermediate.powerplug

...
```

I need power plug!

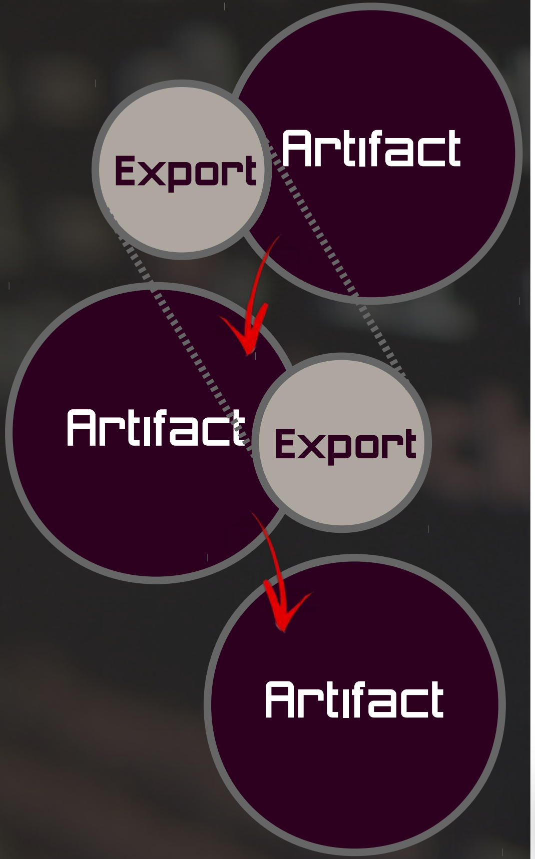
## JSR 376

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!

## modules can also be resellers



## OSGi

MANIFEST.MF

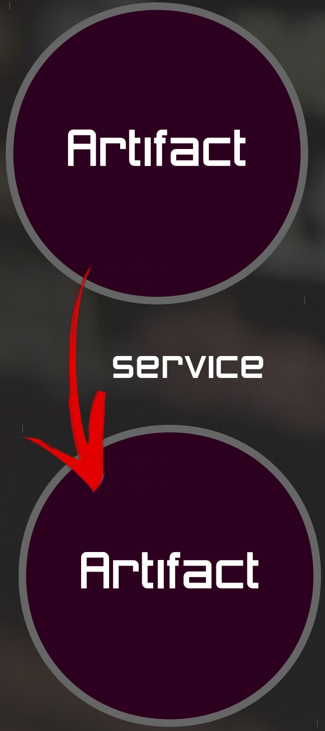
```
Manifest-Version: 1.0
Bundle-SymbolicName: \
     com.mycompany.mymodule

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;

  ...

}
```

# modules and services



## OSGi

**ServiceImpl.java**

```
@Component
public class MyServiceImpl
    implements MyService {

    ...

}
```

**ServiceConsumer.java**

```
public class MyServiceConsumer {

 @Reference
 public MyService bind(
   MyService myservice) {
    ...
 }
...
}
```

## JSR 376

**module-info.java**

```
module com.mycompany.mymodule {

  provides com.some.Service
      with com.some.ServiceImpl;

...

}
```

**module-info.java**

```
module com.mycompany.myapp {

  usess com.some.Service;

...

}
```

## What else ?

Dynamism

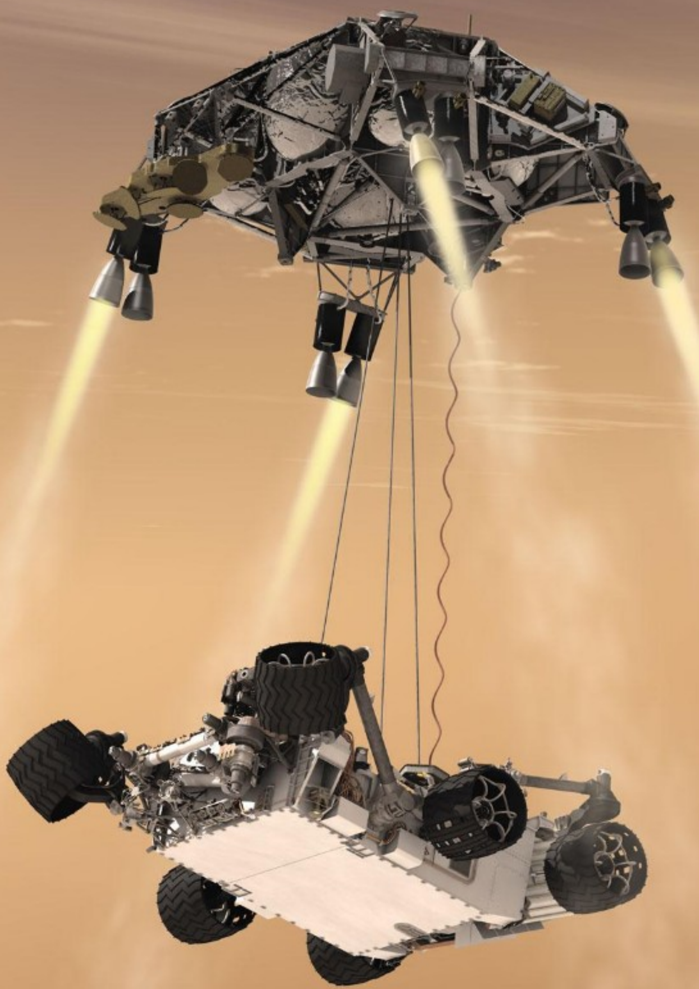
Versions/Layers

Java EE

Build tools

Declaration first vs code first approach

JSR 376 and OSGi interoperability



"... 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

## Modular Java Track

@

# DEVCON

LIFERAY DEVELOPER CONFERENCE

www.liferay.com/devcon

SAVE THE DATE: November 14 - 16, 2016 | Darmstadtium (Germany)

milen.dyankov@liferay.com
@MilenDyankov

http://www.liferay.com
@Liferay