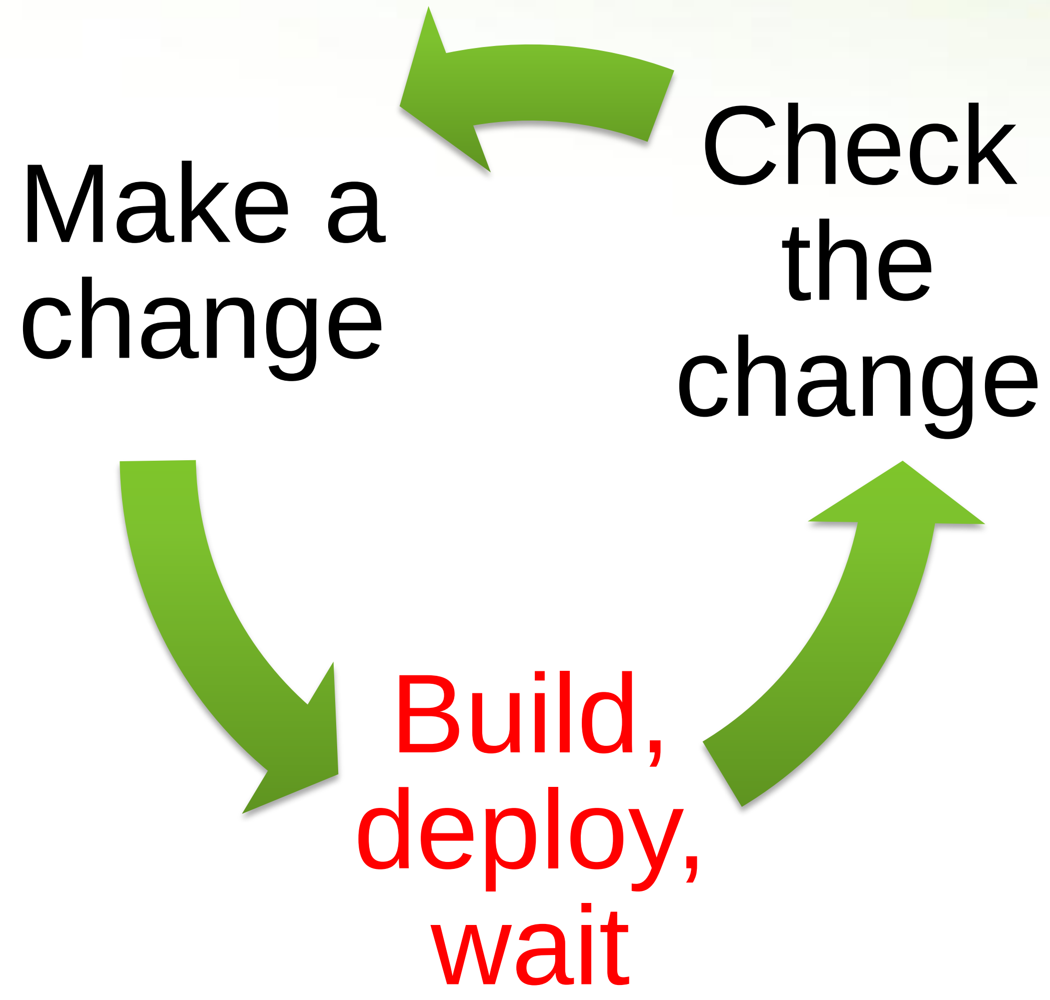
Zero Turnaround in Java

Jevgeni Kabanov

ZeroTurnaround Lead

Aranea and Squill Project Co-Founder

Turnaround cycle



DEMO: SPRING PETCLINIC TURNAROUND

Outline

- Turnaround – Why should you care?
- Trimming Builds
- Reloading Java Code with Class Loaders
- HotSwap, JavaRebel and Beyond

TURNAROUND – WHY SHOULD YOU CARE?

Turnaround Cost

From over 15 projects and 150 people

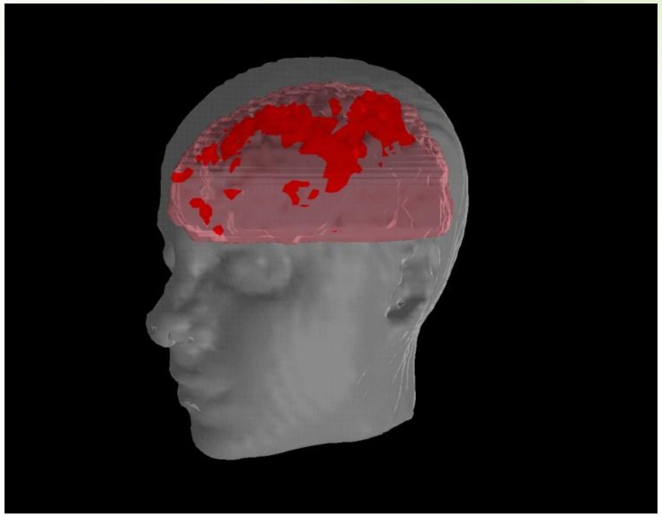
- Average turnaround is at least **1 minute** long
- Done about **5 times an hour**

This sums up to

- **8.3%** of total development time (1*5/60)
- **3.5 hours** a week
- Almost **1 work month a year**

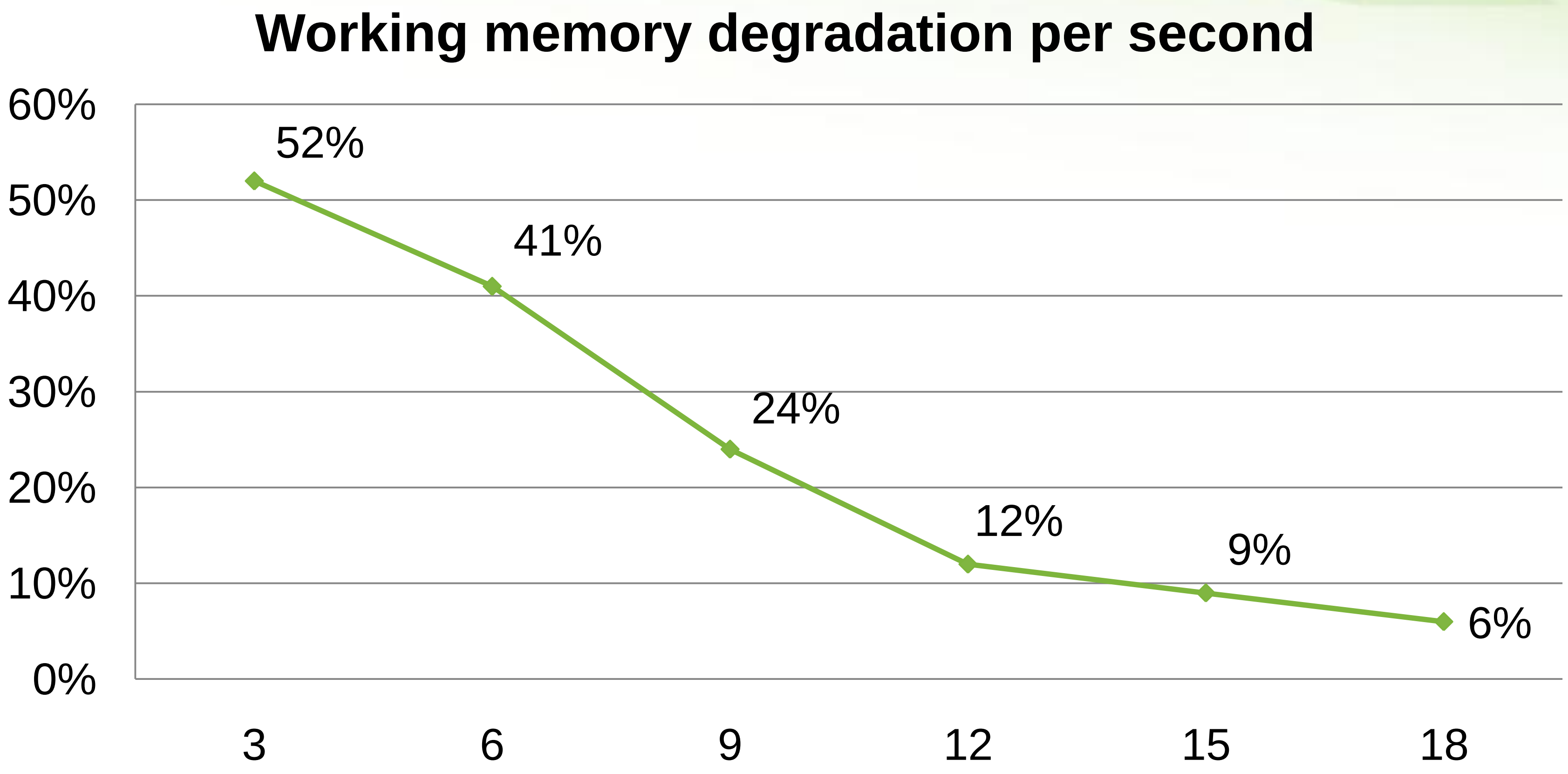
Working Memory

- Programming is an exercise of the working (short-term) memory that holds the current context



- Questions:
  - How fast do you lose that context?
  - How much time does context recovery take?

Working Memory



Source: L. Peterson and M. Peterson "Short-Term Retention of Individual Verbal Items." *Journal of Experimental Psychology*, 1959.

Interruption recovery time

[…] the recovery time after a phone call is at least 15 minutes.

– Interrupts: Just a Minute Never Is, IEEE Software, 1998

The time it takes the employees to recover from an email interrupt […] was found to be on average 64 seconds.

– Case Study: Evaluating the Effect of Email Interruptions within the Workplace, EASE 2002

The recovery time for an instant message was estimated to be between 11 and 25 seconds

– Instant Messaging Implications in the Transition from a Private Consumer Activity to a Communication Tool for Business, Software Quality Management , 2004

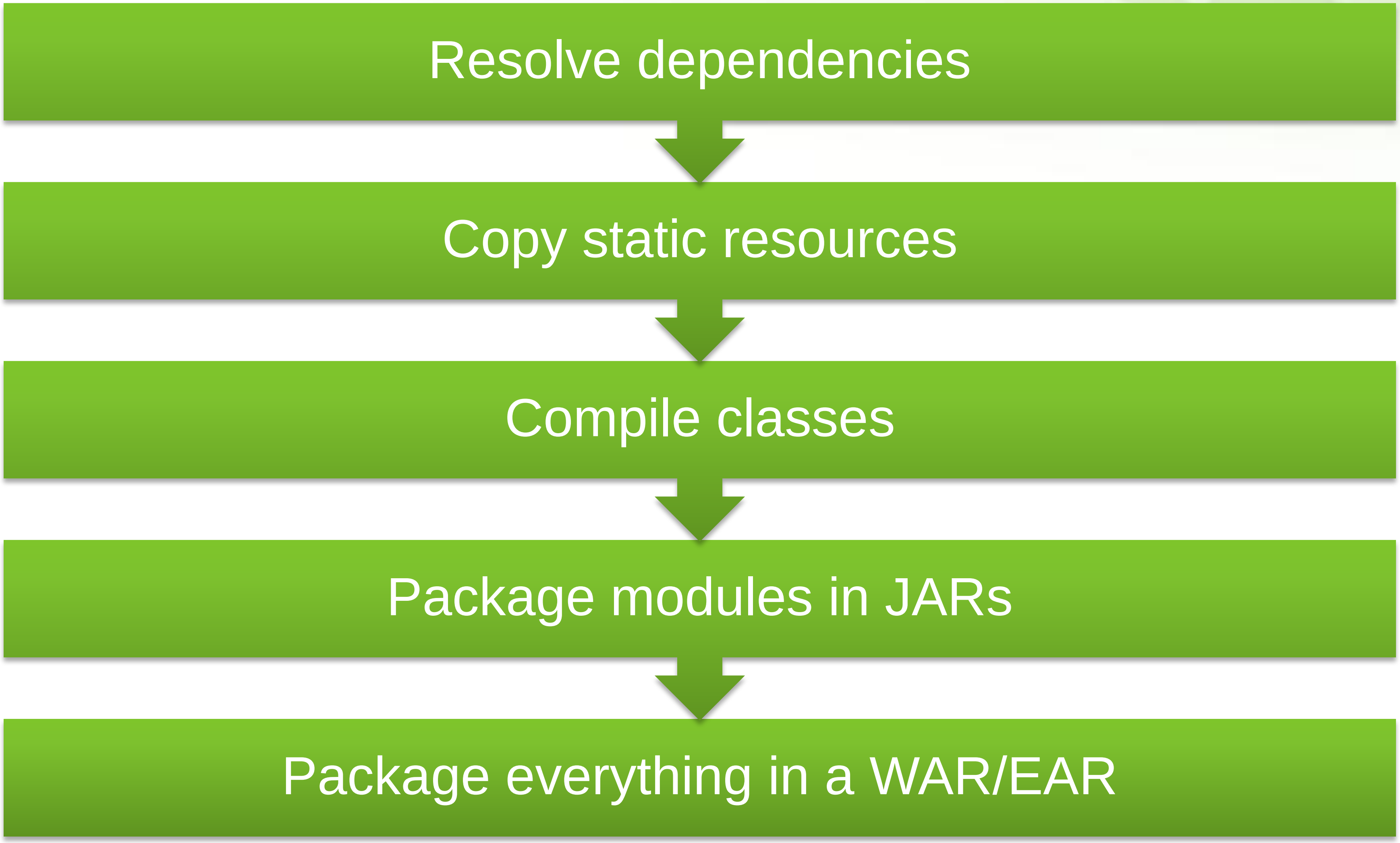
Turnaround Conclusions

1. With the recovery time considered, turnaround can easily cost more than 15% of total development time.
   - ~ 7 hours a week, 7 work weeks a year
   - This does not include the cost of quality degradation.

2. Every second counts! There is a significant difference between a minute, 30, 15, 5 and 1 second turnaround.

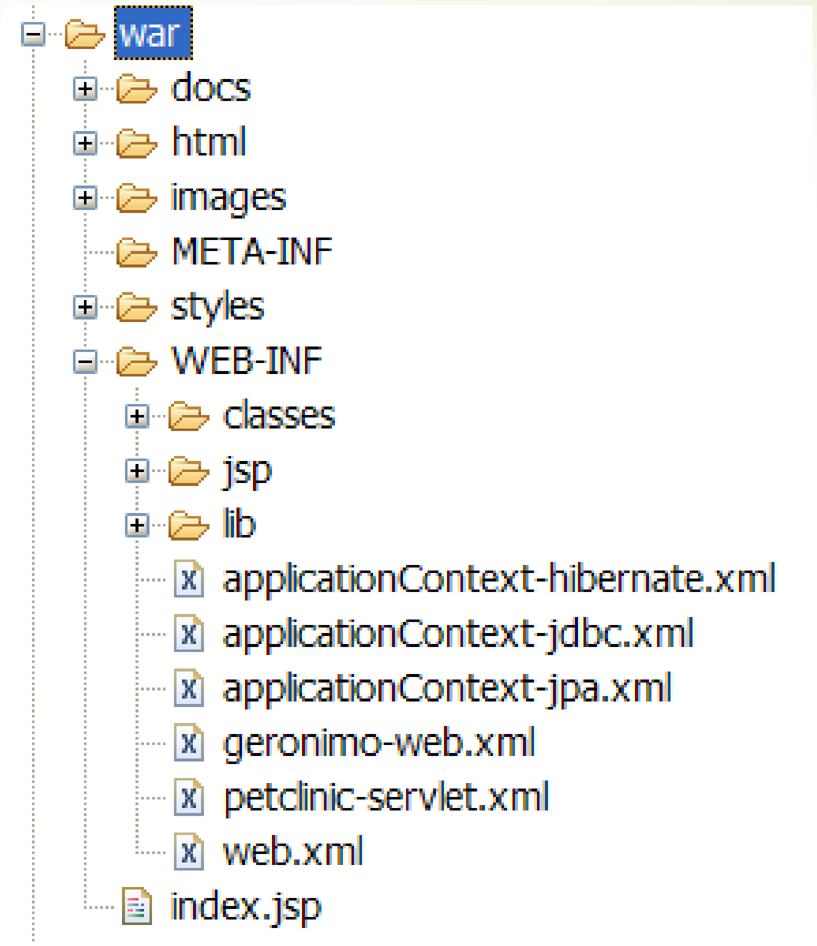
TRIMMING BUILDS

A typical web application build



Exploded layout

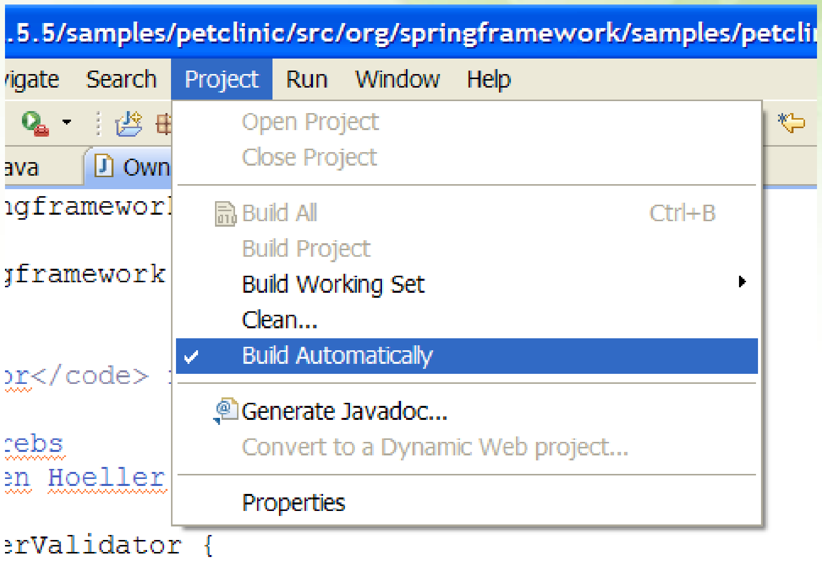
- The project layout exactly follows the deployment layout
- All resources are edited in-place without copying



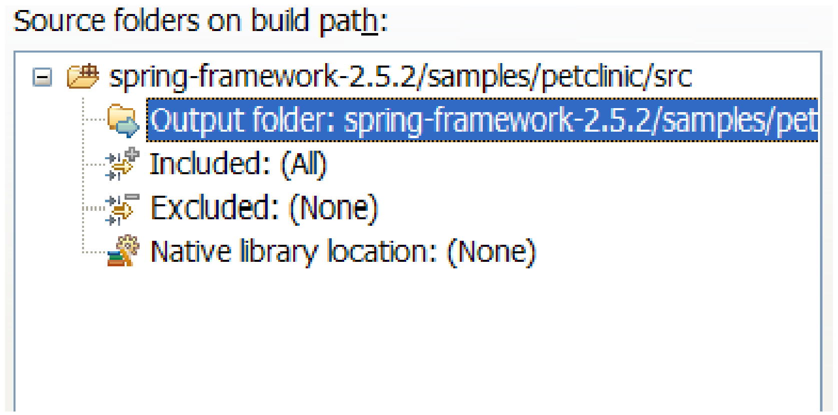
```
war
  docs
  html
  images
  META-INF
  styles
  WEB-INF
    classes
    jsp
    lib
    applicationContext-hibernate.xml
    applicationContext-jdbc.xml
    applicationContext-jpa.xml
    geronimo-web.xml
    petclinic-servlet.xml
    web.xml
  index.jsp
```

Automatic building

- Classes should be compiled automatically by the IDE
- The output should be set directly to WEB-INF/classes or similar



.5.5/samples/petclinic/src/org/springframework/samples/petcli
igate Search Project Run Window Help
Open Project
Close Project
Build All Ctrl+B
Build Project
Build Working Set
Clean...
✓ Build Automatically
Generate Javadoc...
Convert to a Dynamic Web project...
Properties


Source folders on build path:

- spring-framework-2.5.2/samples/petclinic/src
  - Output folder: spring-framework-2.5.2/samples/pet
  - Included: (All)
  - Excluded: (None)
  - Native library location: (None)

Deployment by linking

- The project is deployed by either pointing the container to it or creating a symbolic link in the deployment directory

Linux symbolic links

- ln -s
- Symlinks can point to any file

Windows symbolic links

- Sysinternals **junction** utility on NTFS partitions
- Can only link to local directories and must be careful when deleting

A ~~typical~~ web application build

1. ~~Resolve dependencies~~
2. ~~Copy static resources~~
3. **Compile classes**
4. ~~Package modules in JARs~~
5. ~~Package everything in a WAR/EAR~~

Bootstrapping Builds

- Can’t always use exploded layout
- Instead:
  - Build the WAR/EAR
  - Unzip it to a temp directory
  - Remove some of the folders/jars and symlink them to the project folders
  - Set the project to build automatically
- Easy to automate with a bootstrapping script
- Save on copying resources and packaging classes

RELOADING CODE

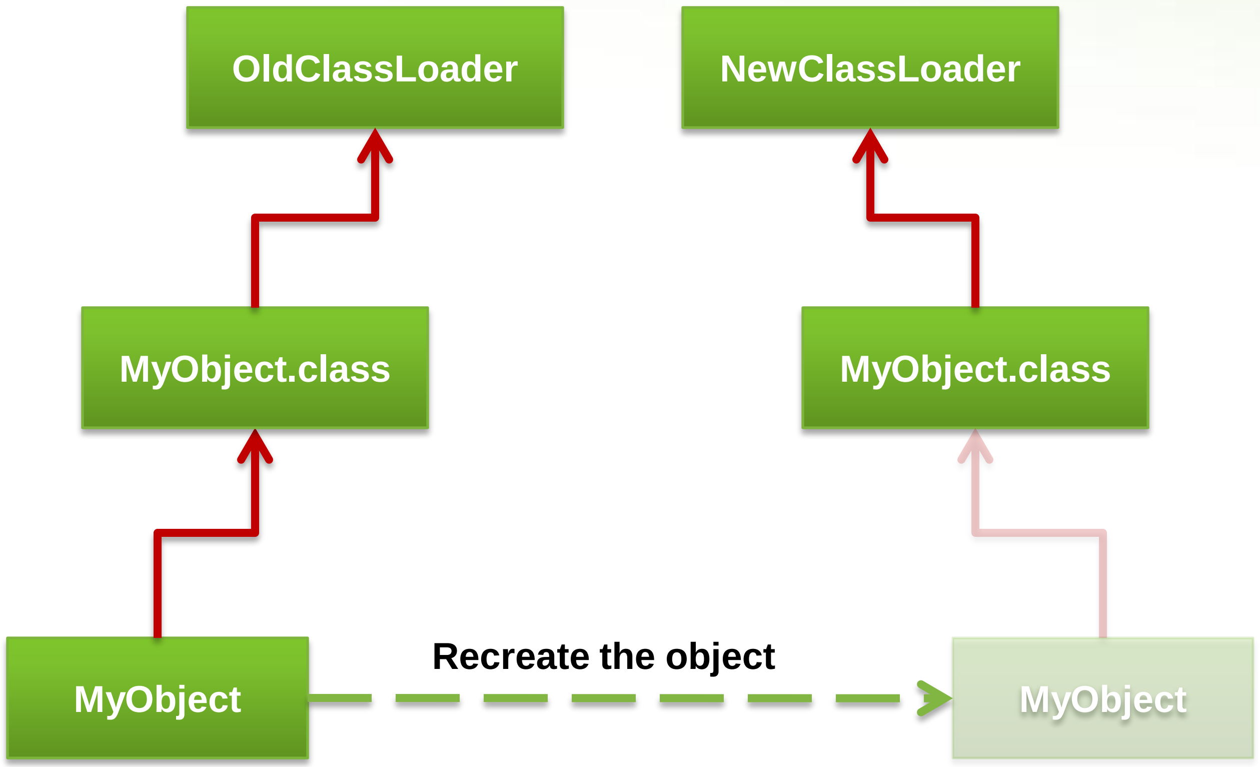
Reloading Code

Objects & Class Loaders

Deployment, OSGi & etc

JVM Dynamic languages

Reloading an Object

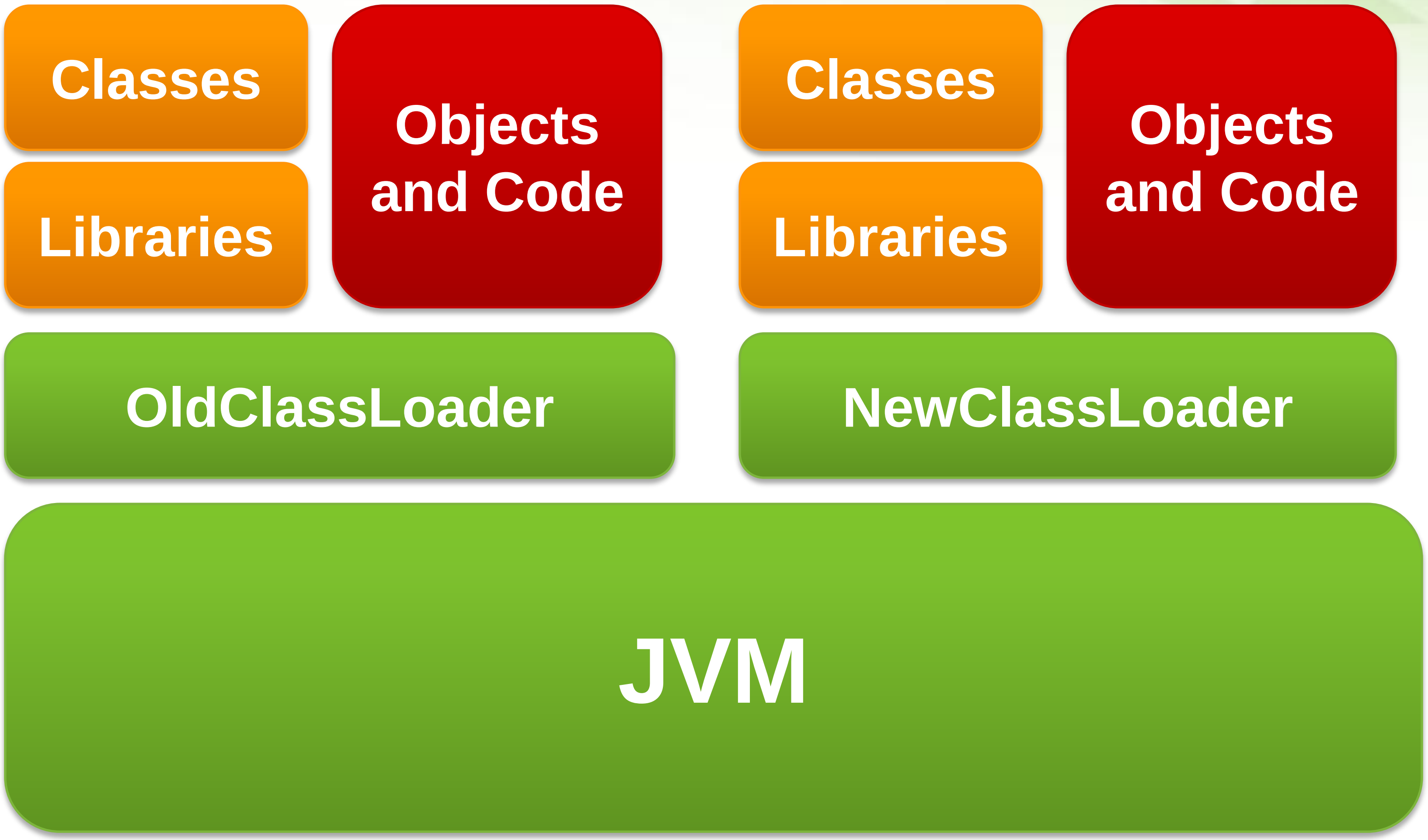


Twin Classes

Wow! You look exactly like me!
Not anymore! HA-HA-HA-HA!
Bang!!!

MyClass (OldClassLoader) **MyClass (NewClassLoader)**

Twin Class Loader



Twin Class Issues

New objects are not instances of old classes

- instanceof returns false
- Casting throws an exception

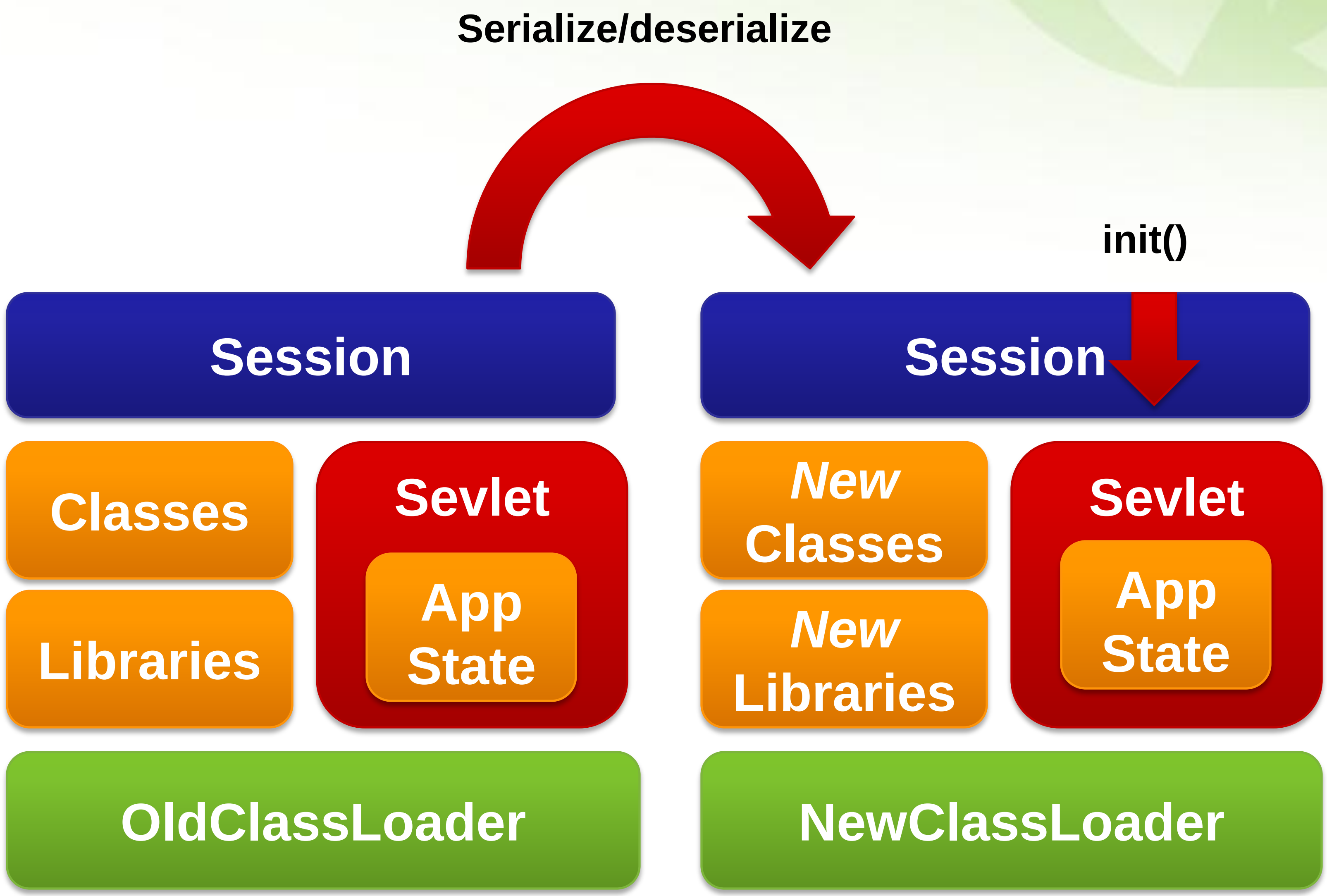
New classes are not members of the old packages

- Can get an IllegalAccessException when calling a perfectly legal method

Memory leaks are easy

- If you hold a reference to any object in the old classloader you will hold all old classes (including their static fields)

Web Deployment

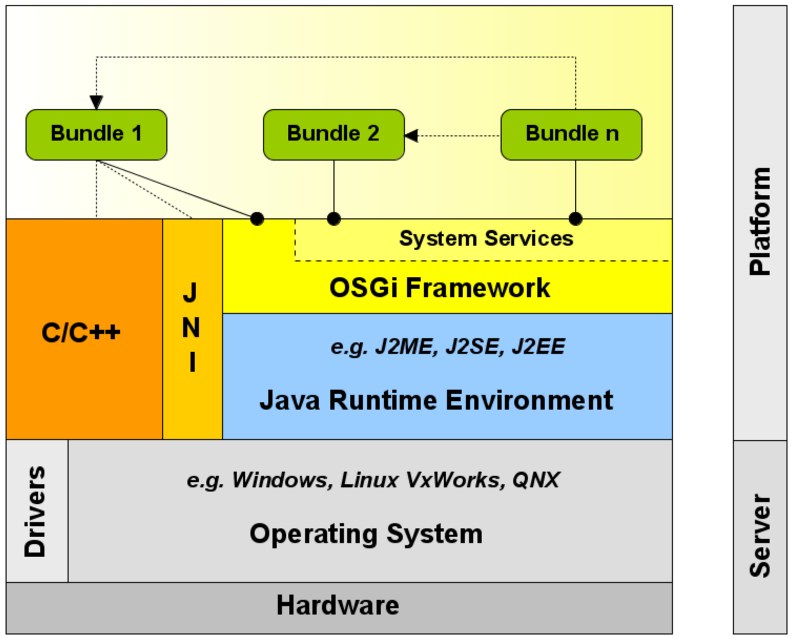


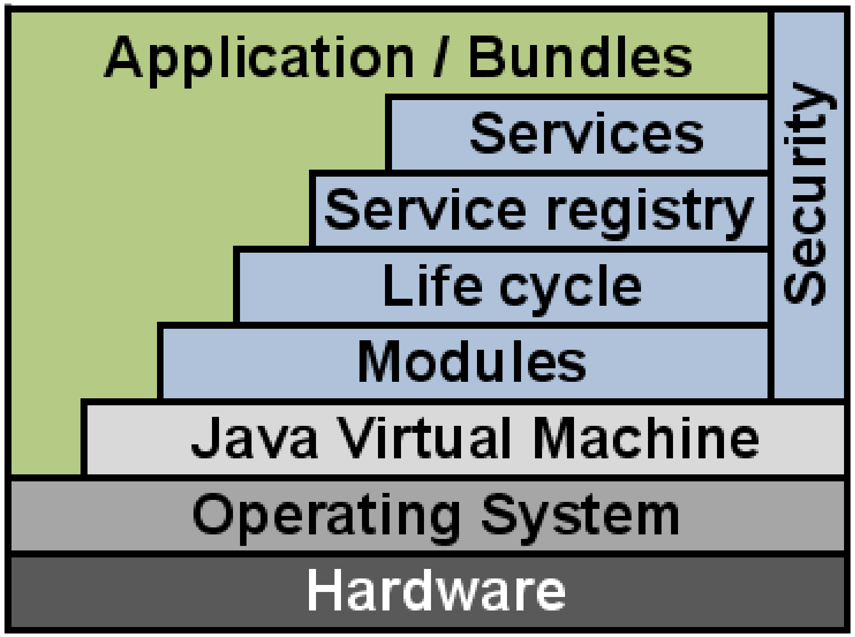
Web Deployment

Class loader scope	• Every deployed application gets a dedicated class loader
State recreation	• Application state is recovered by reinitialization • Session state is (optionally) serialized and deserialized in the new class loader
Reloading time	• Applications reinitialization time, typically around one minute
Problems	• Leaks memory • Lazy caches need to be warmed up every time

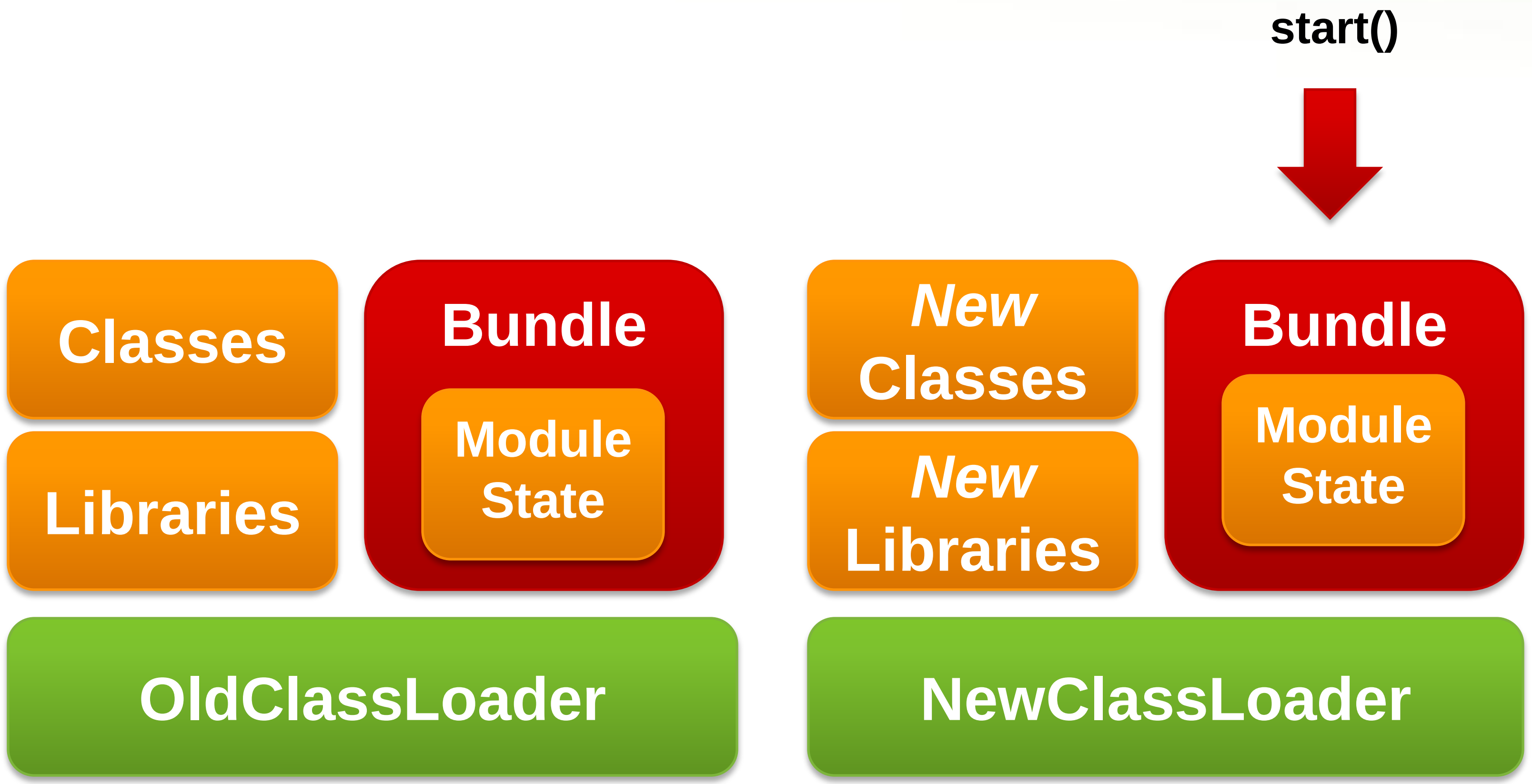
OSGi

- Frameworks that implement the OSGi standard provide an environment for the modularization of applications into smaller bundles. [Wikipedia]





OSGi Redeployment

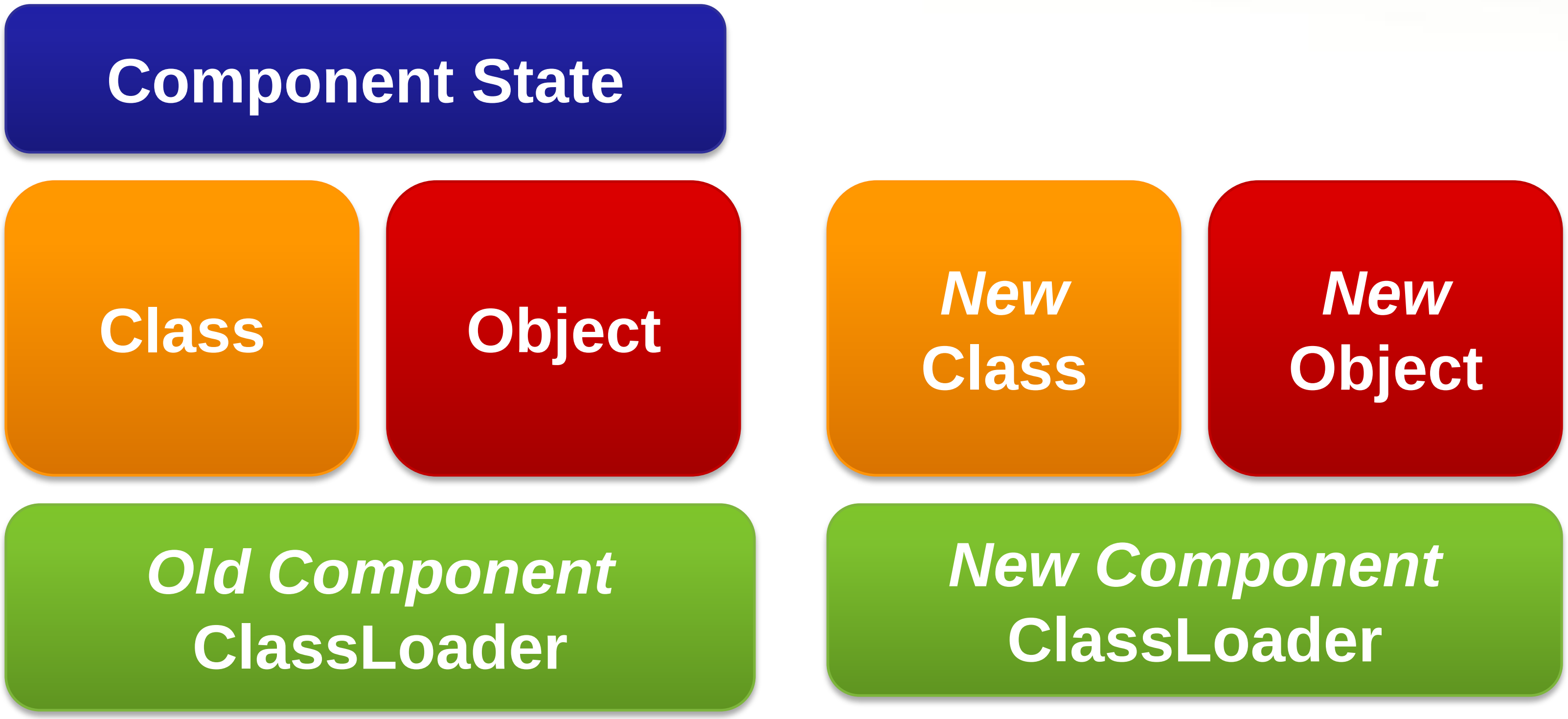


OSGi

Class loader scope	• Dedicated class loader per application module
State recreation	• Module state is recovered by reinitialization
Reloading time	• Module reinitialization time, usually less than whole application reinitialization
Problems	• Applications must be designed with OSGi in mind • Overhead interface definitions • Module export interfaces cannot be changed without redeploying the application

Fine-grained Class Loaders

- Wrap a class loader around components
  - E.g. Tapestry 5, RIFE
- Very fast reloading
  - Few classes at a time
  - Components managed by the framework are usually easy to recreate


Component State
Class
Object
New Class
New Object
Old Component ClassLoader
New Component ClassLoader

Fine-grained Class Loaders

Class loader scope	• Class loader per component/service
State recreation	• State restored by framework (component/service recreated)
Reloading time	• (Almost) Instant
Problems	• Only managed components can be reloaded • Managed components referring unmanaged code can be a problem (twin class issues)

Some Conclusions

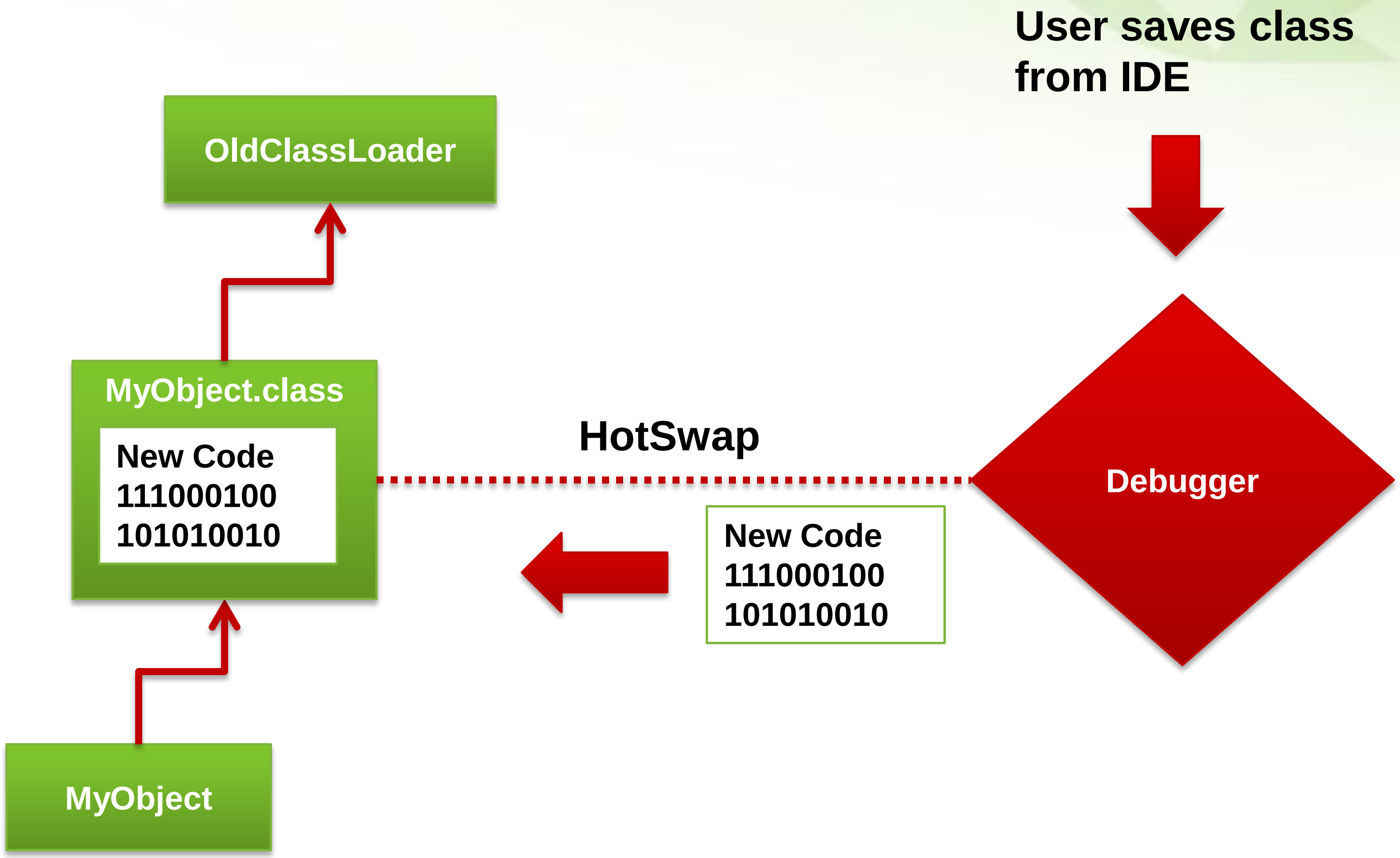
- Recreating the state is the breaking point of reloading a class
- Coarse-grained class loaders take too much time to recreate the state
- Fine-grained class loaders exhibit the twin class problem and are not universally applicable
- Both are useful, but not really a solution to the zero turnaround problem

Dynamic Languages

- Class-based languages have same limitations as Java
  - Groovy
  - Jython
- Non-class based languages can have better support
  - JRuby
  - Clojure

HOTSWAP AND JAVAREBEL

HotSwap


User saves class from IDE
OldClassLoader
MyObject.class
New Code
111000100
101010010
HotSwap
Debugger
New Code
111000100
101010010
MyObject

HotSwap

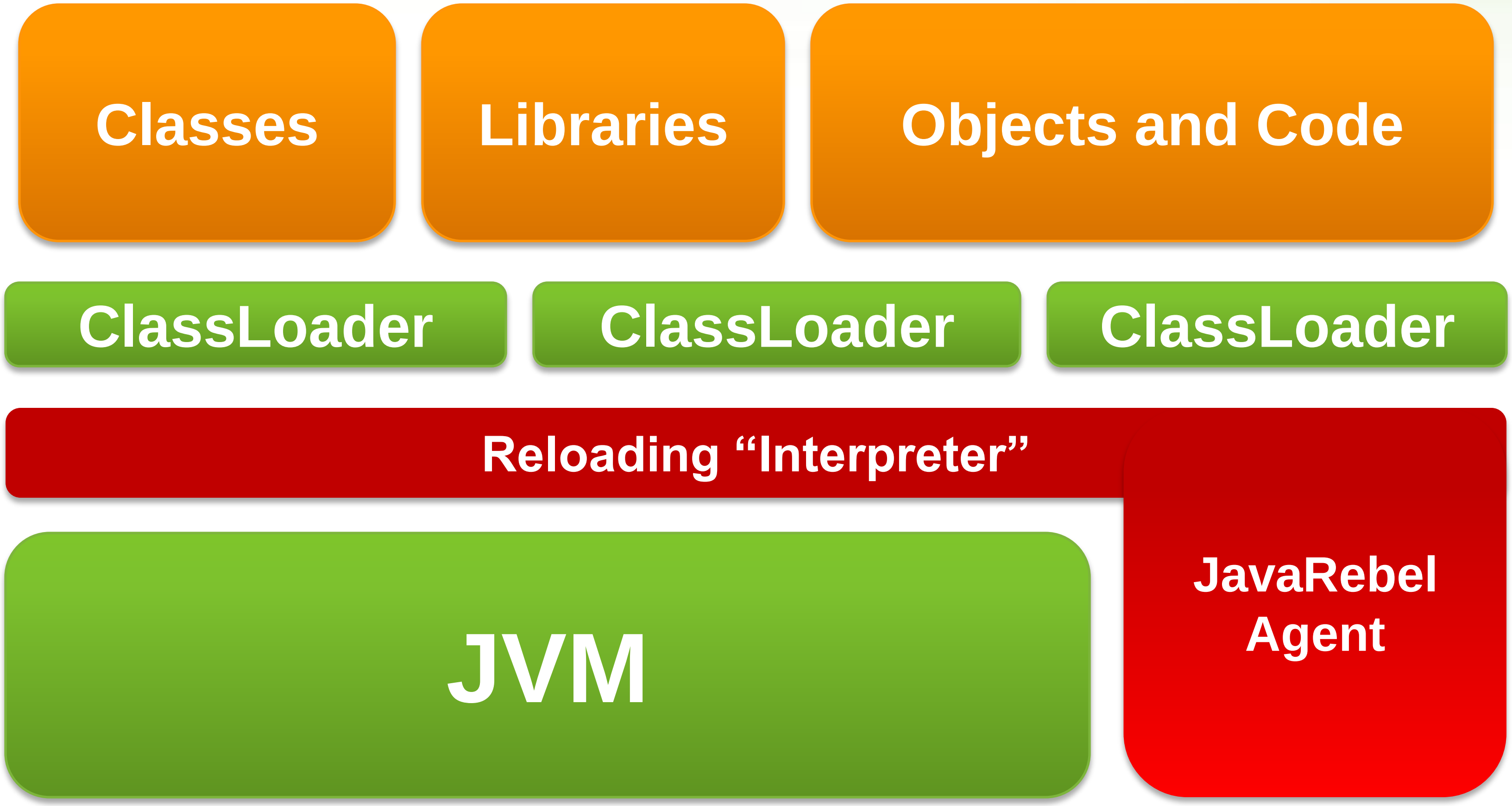
Updates classes and objects

- Almost instantly
- Can be attached remotely

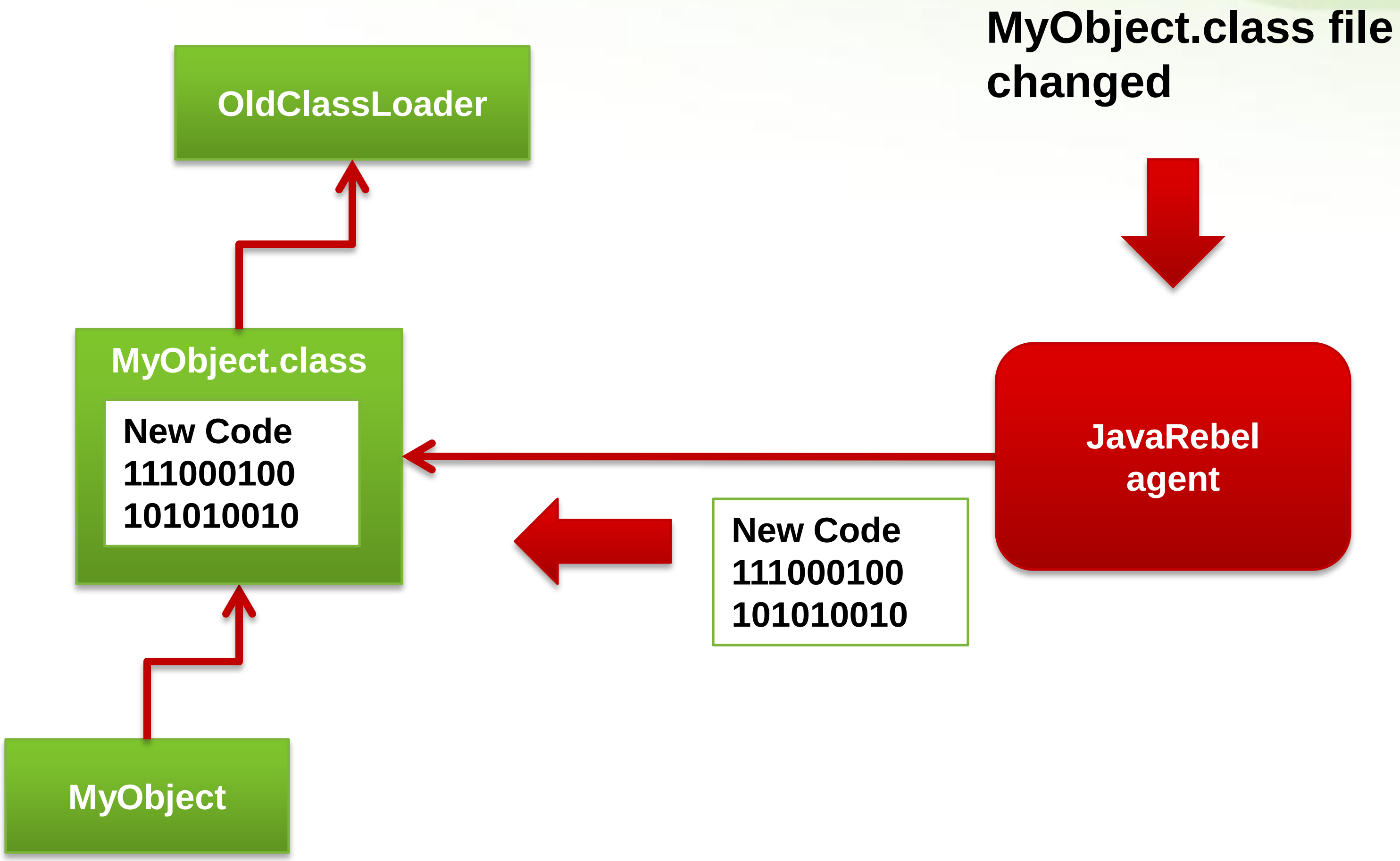
Very limited

- Only updates method bodies, no new fields, methods or classes
- Needs a debugger session running, slow and prone to error

JavaRebel Approach



JavaRebel


MyObject.class file changed
OldClassLoader
MyObject.class
New Code 111000100 101010010
JavaRebel agent
New Code 111000100 101010010
MyObject

JavaRebel Features

	HotSwap	JavaRebel
Changing method bodies	+	+
Adding/removing methods	-	+
Adding/removing constructors	-	+
Adding/removing fields	-	+
Adding/removing classes	-	+
Adding/removing annotations	-	+
Replacing superclass	-	-
Adding/removing implemented interfaces	-	-

JavaRebel Installation

- -noverify -javaagent:/path/to/javarebel.jar
  - Enables the JavaRebel agent
  - All ***.class** files in the classpath will be monitored for changes automatically
- (Optional) -Drebel.dirs=folder1,folder2,…
  - Specifies IDE output folders or just class folders
  - Can deploy a WAR/EAR and still get instant updates to code

DEMO: PETCLINIC WITH JAVAREBEL

JavaRebel

Just works

- Runs on all JVMs starting with 1.4
- Supports all major containers
- Supports standalone Java applications and OSGi
- Easy to extend with an open-source SDK and plugin system

Full reflection support

- New methods and fields are visible in the reflection
- Changes to annotations and new annotations are propagated

JavaRebel

- Commercial tool, free 30 day trial
- No free/open source analogs
- Get it from: www.zeroturnaround.com

  or just google “javarebel”

- Personal license:

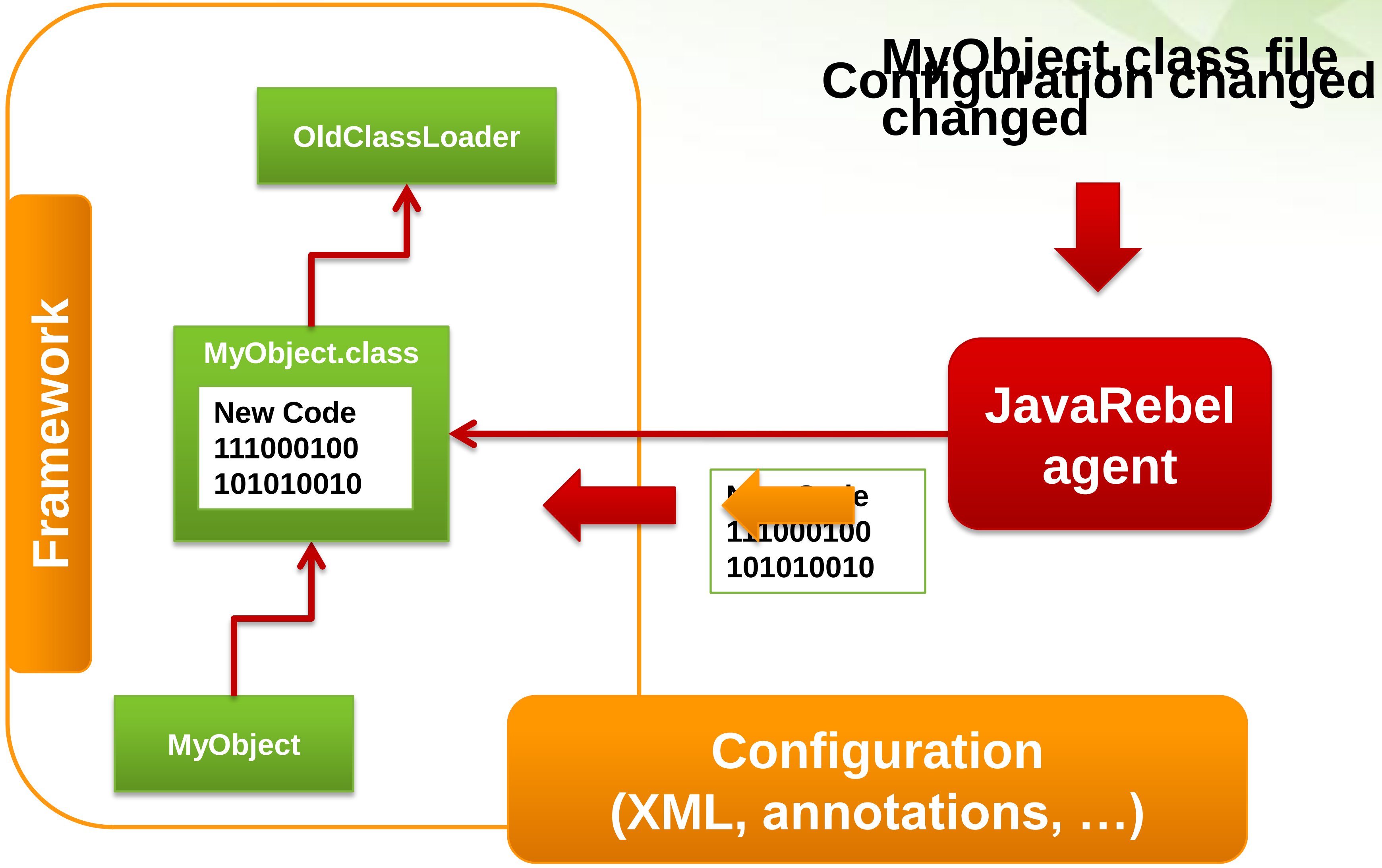
- Commercial license:

JavaRebel History

- JavaRebel 1.0 released in **December, 2007**
- Today over **10 000** licensed users
- Big Java shops with everyone using JavaRebel:
  - **LinkedIn**
  - NHN Corporation
  - Immobilien Scout GmbH
  - Reaktor Innovations
  - GT Nexus, Inc.
  - Teranet Inc.

AND BEYOND

JavaRebel


MyObject.class file changed
Configuration changed
OldClassLoader
Framework
MyObject.class
New Code
111000100
101010010
JavaRebel agent
New Code
111000100
101010010
MyObject
Configuration
(XML, annotations, …)

Types of Configuration

Service Glue	• EJB 2.0/3.0 • Spring • Guice
Web Controller	• Struts 1.0/2.0 • Stripes • Spring MVC
ORM	• Hibernate • TopLink • JPA

JavaRebel Plugins

Open Source JavaRebel SDK

- Plugins are autostarted from classpath
- Javassist support allows patching framework classes
- API to react on class reloads

Spring Plugin

- Adding/removing beans dependencies via setters/fields
- Adding new beans via XML or annotations
- Adding new MVC Controllers and Handlers

DEMO: PETCLINIC WITH JAVAREBEL SPRING PLUGIN

JavaRebel Future

Virtual Resource System, Q4 2008

- All the benefits of exploded development with unexploded one
- Automatically maps propagates class and resource updates to the deployed application
- Will need some user help to configure

New plugins, Q4 2008

- Guice, Stripes, Wicket, Struts, Hibernate, …

Production support, Q1 2009

- Instant automatic production server updates and rollbacks with a press of a button
- Tools for update verification

Take Away

- Every next second spent on turnaround costs **more**!
- **Builds** should be as slim as possible, **symlink** is your best friend
- Existing code reloading solutions have **severe limitations** in reloading time or applicability
- **JavaRebel** solves most of turnaround problems for a cost, plugins support configuration reloads