Building the "Big Brother" for the Java Virtual Machine

Paul René Jørgensen & Steinar Cook

JavaZone - 17.september 2008

Steinar Overbeck Cook

Steinar wrote his first program back in 1979 in APL and have been trying to convince his family that programming is work and not fun ever since.

Special interest in software engineering methods, design patterns and SQL databases.

After establishing the DBMS vendor Informix in Norway in the late 90's, which was later sold to IBM, he founded his second company Million Handshakes, focusing on CRM software.

He is currently involved with his 3rd startup, www.SendRegning.no, focusing on a new SaaS solution for the Scandinavian SMB market

UseMon beta

Paul René Jørgensen

Paul René has worked as a senior consultant at Telenor in Norway for the past 8 years and has now moved back to Trondheim to work for the newspaper Adresseavisen.
He loves to code, and do whatever it takes to get the opportunity to write code, whether it is at his desk, on the bus or in bed. One of his part time projects include a model airplane autopilot written in Java using a Sun Spot.

Usemon has been developed on and off for the past 3 years, but got extra momentum when Steinar joined the project in Q3, 2007.

Challenges

- What goes on in the JVM?
- Who invokes who?
- Runtime dependencies (late binding)
- Who uses the CPU?
- Irregular use of exceptions
- Are the servers balanced in your clusters
- Main call paths through the entire system
- Invocation count and response times
- Only interested in our selected classes and methods

UseMon beta

Possible Solutions

- Many tools
  - Costly
  - Proprietary
    - No extension points
    - Limited to the built in reports
  - Complicated
  - Intrusive?
    - Some may require code modifications

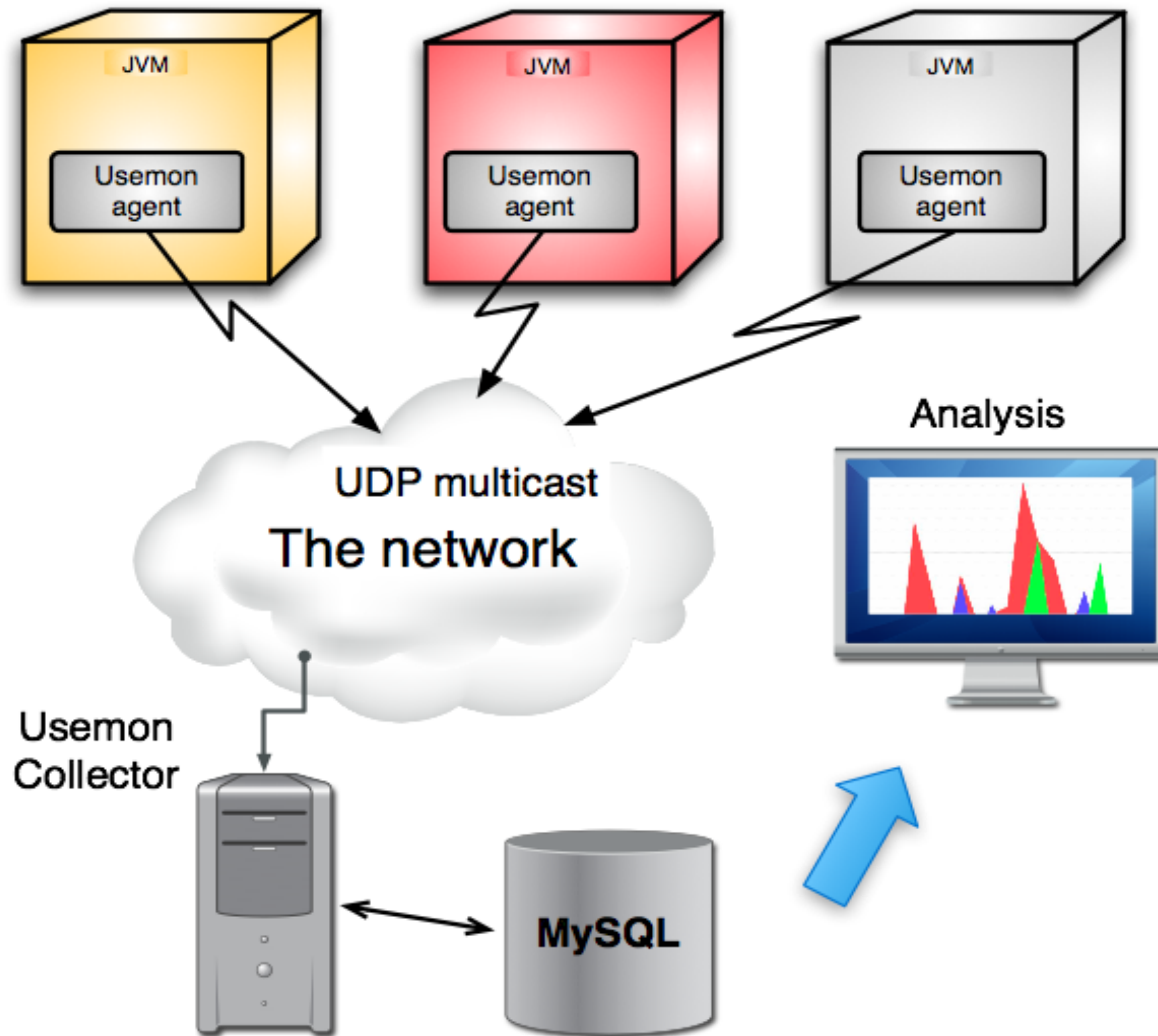
UseMon beta

Possible Solutions (cont.)

I'm a programmer and I want to do this myself!

UseMon beta

Overall Architecture



The agent

- Bootstrapping the JVM
  - JDK >= 5.0
    - Hook into the Java Agent Interface
  - JDK <= 1.4
    - Modify system supplied `java.lang.ClassLoader`
    - Modify the JVM startup
- Modification of byte code during class loading
- Bootstrapping the internal registry and the publisher
- Measure and assemble observations
- Multi casting observations to the collector

Boostrapping JVM >= 5.0

- Starting the JVM with modified class loader

```
java
   -javaagent:usemon-agent.jar
```

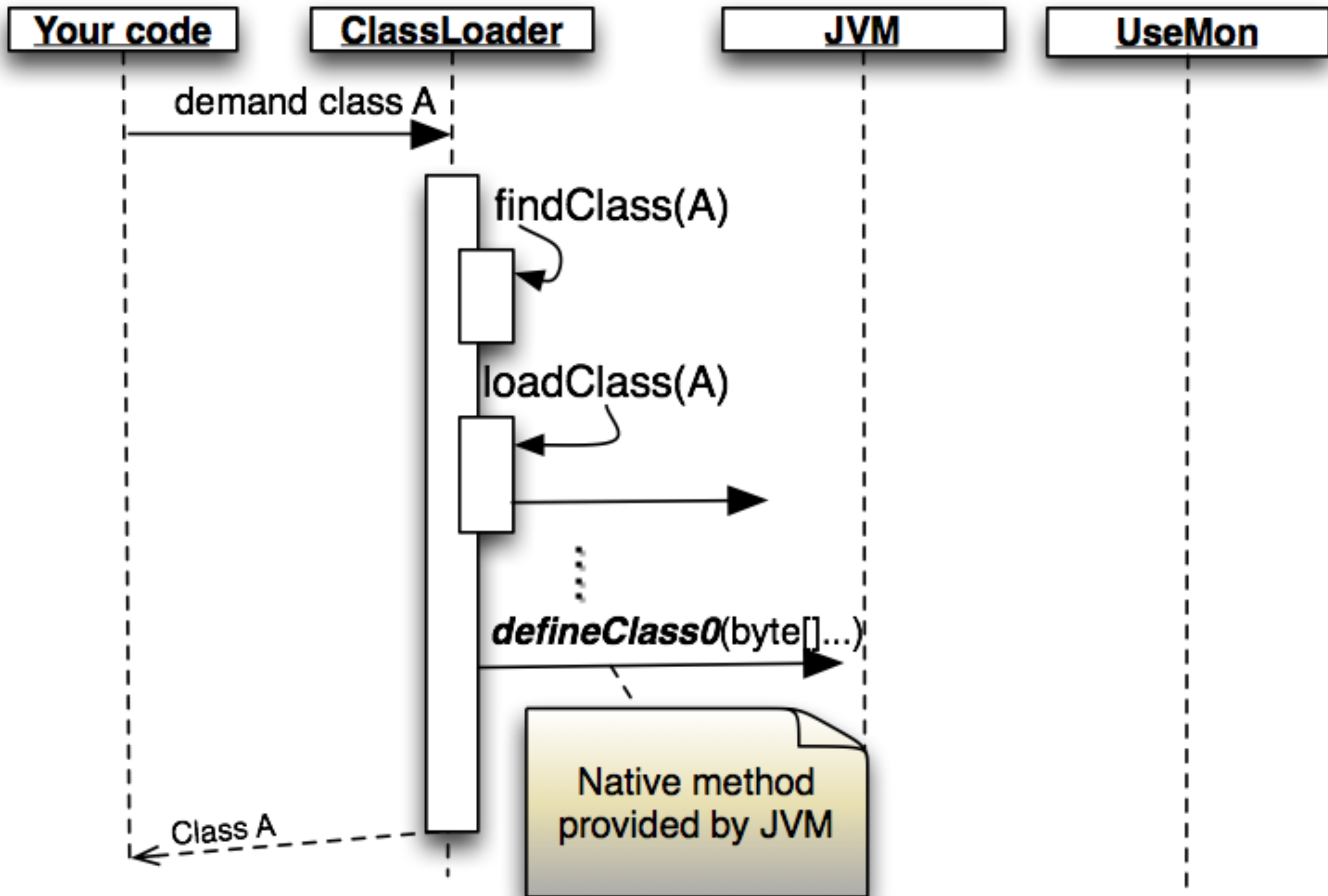
```java
public class JDK5Agent implements ClassFileTransformer {

    /** JDK5+ entry point */
    public static void premain(String agentArgs, Instrumentation inst) {
        inst.addTransformer(new JDK5Agent());
    }
    /** Invoked by the JVM class loader before defining each class */
    public byte[] transform(ClassLoader loader, String className, Class<?> classBeingRedefined,
            ProtectionDomain protectionDomain, byte[] classfileBuffer)
            throws IllegalClassFormatException {
        // Instruments methods on given class
        return RootInstrumentor.transform(loader, className, protectionDomain, classfileBuffer);
    }

}
```

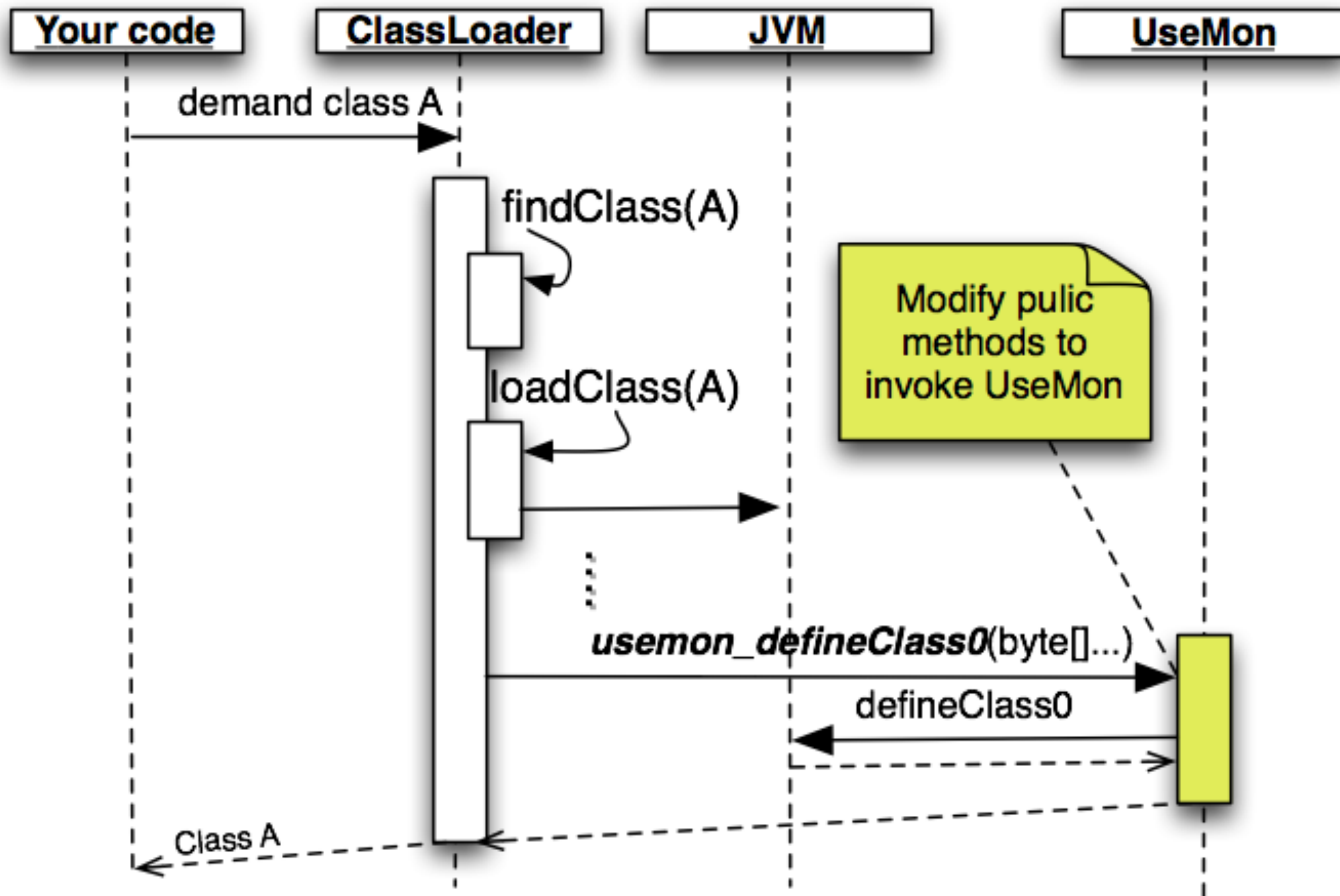
UseMon beta

Java Agent Interface for JVM <= 1.4



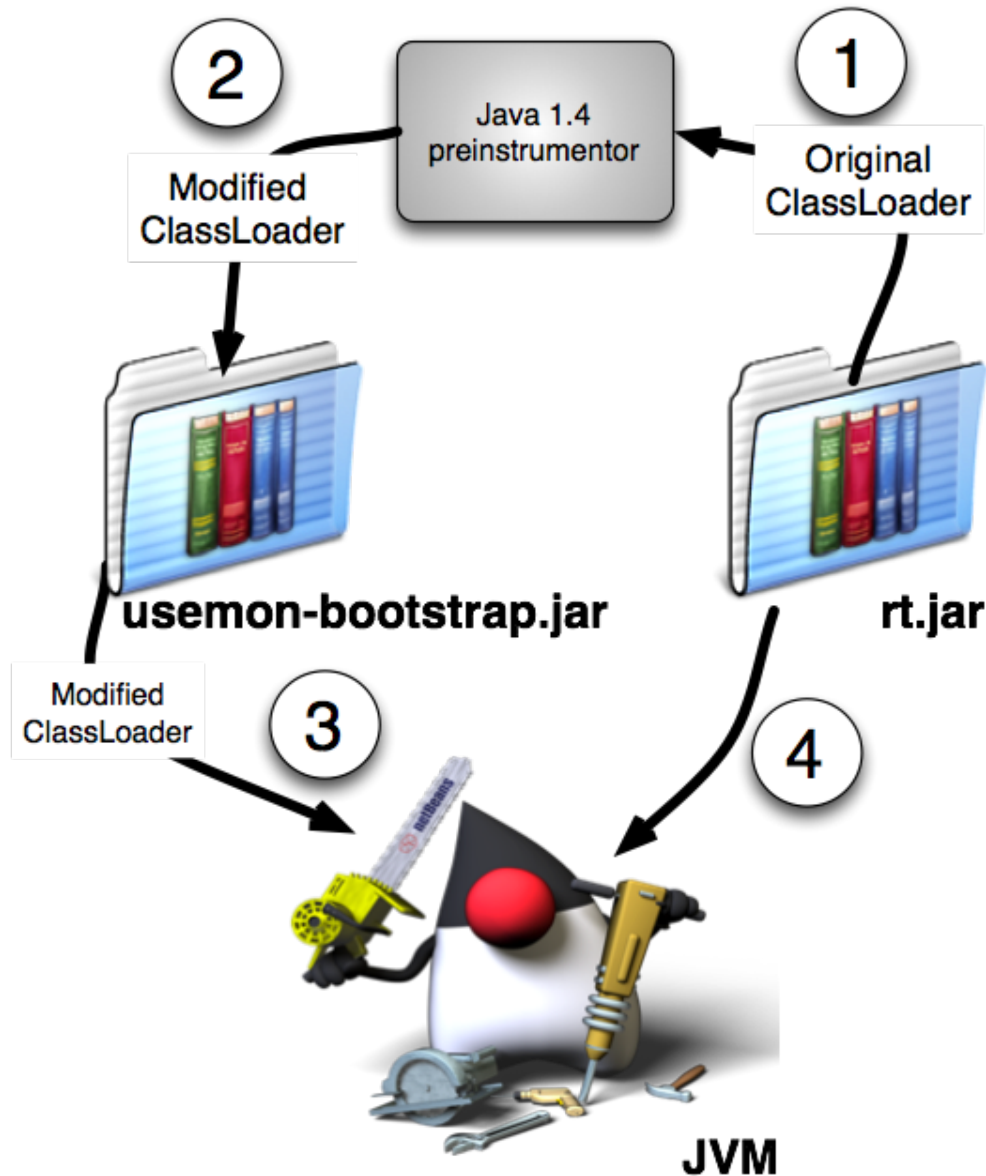
UseMon beta

Emulating Java Agent Interface



UseMon beta

Instrumentation of the JVM <= 1.4



UseMon beta

```java
CtMethod newDefineClass0Method =
    CtNewMethod.copy(method, composeNewMethodName(), javaClass, new ClassMap());
// removes the native flag, as we will actually supply the
// java byte code implementation for it.
newDefineClass0Method.setModifiers(newDefineClass0Method.getModifiers() ^ Modifier.NATIVE);
StringBuffer code = new StringBuffer();
code.append("{");
// $0 is first argument, $1 is second etc.
// When executed, this code will invoke our
// com.usemon.agent.instrumentation.RootInstrumentor.transform() method
code.append("  byte[] newClass = "
        + Constants.instrumentCallback + "($0, $1, $5, $2, $3, $4);");
// if the returned byte array is not null, the class was instrumented
code.append("  if(newClass!=null) {");
// replaces the original byte array with our modified byte array
code.append("    $2 = newClass;");
code.append("    $3 = 0;"); // start of byte array
// the length of our modified byte array
code.append("    $4 = newClass.length;");
code.append("  }");
// invokes the original native code which we act as a proxy for.
code.append("  return defineClass0($$);");
code.append("}");
// In our copy of the defineClass0 method, the body is empty
// since the method was native,
// Fills the empty body with our code
newDefineClass0Method.setBody(code.toString());
newDefineClass0Method.insertBefore(""); // This works, remove at your own risk :-)
javaClass.addMethod(newDefineClass0Method); // adds our new method into the class
```

JVM <= 1.4 - modifying the class loader

UseMon beta

Boostrapping JVM <= 1.4

- Starting the JVM with modified class loader

```
java
  -Xbootclasspath/p:usemon-bootstrap.jar
  -cp usemon-agent.jar
```

- Run through the `java.lang.ClassLoader` code and intercept all calls to `defineClass0`
- Insert code that let the Usemon `RootInstrumentor` class modify the byte code before the original `defineClass0` is invoked

UseMon beta

Byte code modification during class loading

- Identify interesting classes
  - Enterprise Java Beans
    - SessionBeans
    - EntityBeans
  - Message Driven Beans
  - Servlets
  - QueueSenders
  - TopicPublishers
  - SqlStatements
  - SqlConnections
  - Custom classes based on user defined patterns

UseMon beta

Measure and assemble observations

```java
public void doBusiness() {
    long usemon_invoke_time = java.lang.System.currentTimeMillis();
    // Push interesting data like SQL, JMS Queue name, method names
    // etc. on the usemon call stack.

    try {
        // IMPORTANT business code
        // .....

        // Pop Usemon call stack and
        // transfer observations to Usemon registry
        return;
    } catch (Throwable t){
        // Pop Usemon call stack and
        // transfer observations to Usemon registry
        // _including_ the Throwable!
    }
}
```

UseMon beta

Internal registry and the publisher

- Aggregates observations for an interval of 60 seconds
- Yields if the JVM is approaching critical state
  - JVM Garbage Collector removes "Soft references"
  - "Soft references" versus "Weak references"

Remember! We are running inside a production JVM!

Multi casting observations

Goals:

- Must not interfere with business code
- Fault tolerance
- Loose couplings

Assumption:

- Loosing some observations is acceptable

Conclusion; We needed a message queue!

- UDP multicast - simplest form of asynchronous messaging
- Proven in battle by Telenor:
    - Metro logging framework transports several GB per day

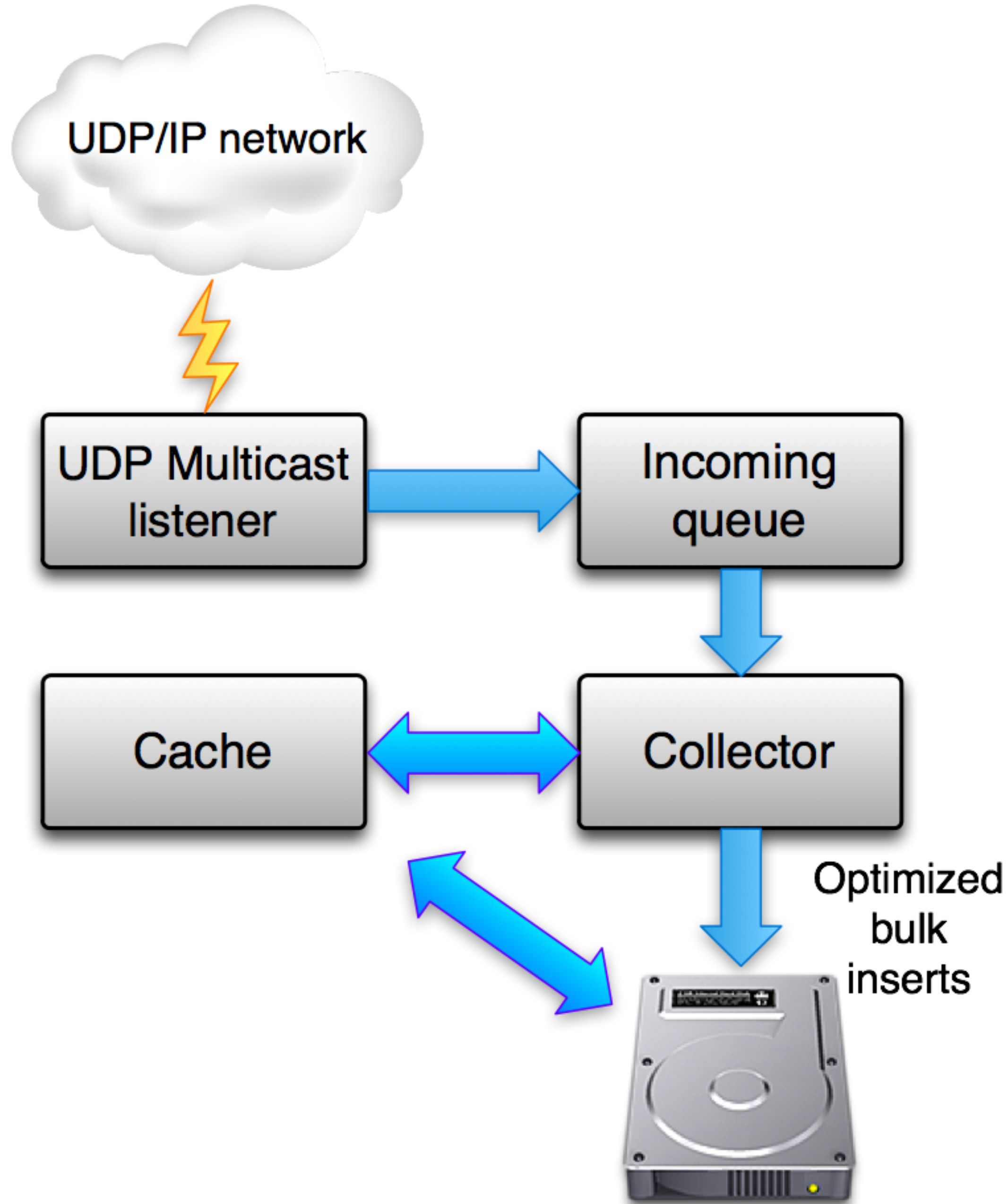
UseMon beta

The collector

- Stand alone Java process
- Receive multi casted observations
  - Hop count
  - Format: Java or JSON
- Cache observations to increase database insert performance
- Reorganize and store
- Drop observations if heap space falls below threshold
  - Receive rate higher than storage rate
- Monitoring and management through JMX

UseMon beta

Collector components



The repository of collected data

- SQL "star schema" with 3 facts:
  - Method invocations
  - Method dependencies
  - Heap usage
- Available dimensions:
  - Location (platform, cluster, server)
  - Package
  - Class
  - Method (with signature)
  - Principal
  - Date and time

UseMon beta

Method measurement fact

class usemon

class

package

method

d_date
«column»
year_v
mont_v
day_v
day_of_week_v
date_v

method_measurement_fact
«column»
invocation_count
max_response_time
avg_response_time
checked_exceptions
unchecked_exceptions
period_length

location
«column»
location
platform
cluster
server

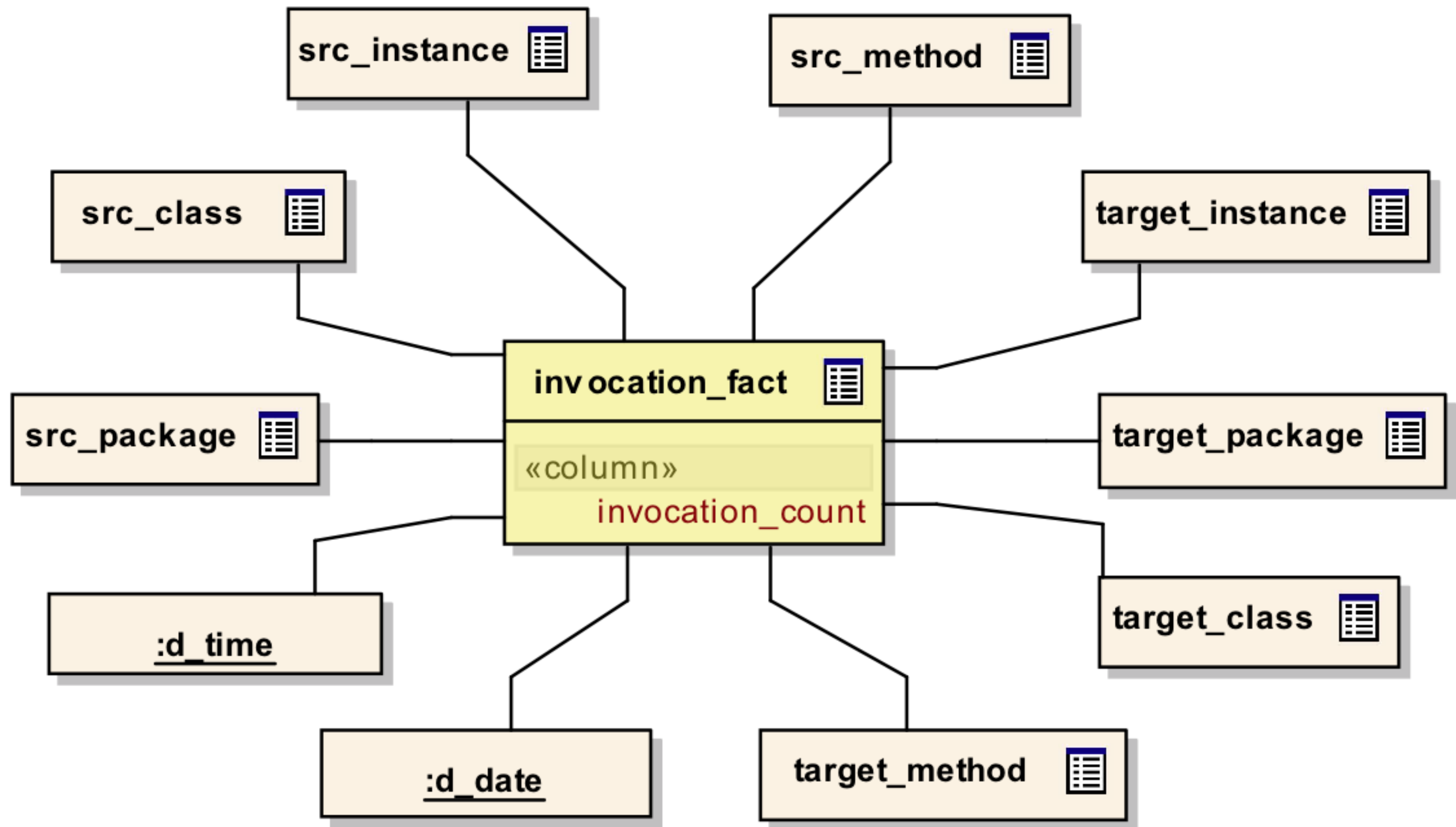
d_time
«column»
hh
mm
ss
time_v

channel

principal

UseMon beta

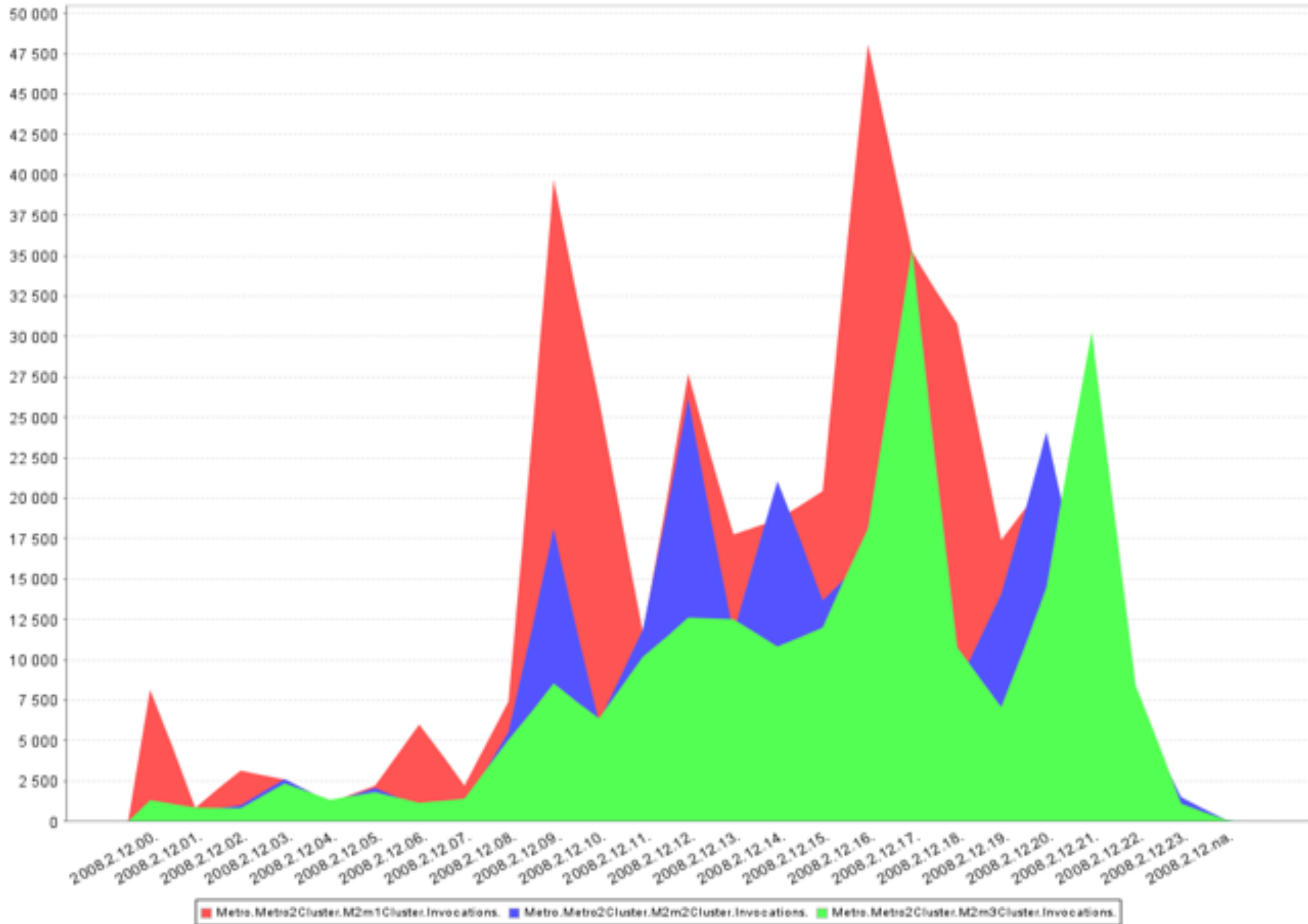
Method dependency fact



OLAP / BI

- Several OLAP/BI tools available
  - Mostly commercial and expensive
  - A few OSS tools based upon *mondrian*
- Requires detailed knowledge of the OLAP cube model and MDX query language
  - Cube model based upon Usemon dimensional model
    - Simplifies the writing of queries
    - Slice & dice is much simpler with MDX than SQL
- Jasper Server & Jasper Analysis

Invocations split over servers for 24h



Exceptional exits

Class	Method	Measures		
Class	Method	Invocations	CPU Time	Exceptional Exits
OrdercoordinatorServiceBean	k2Search	1	0.00h	100.00%
SaturnBean	checkApproveInstalment	168	0.00h	99.40%
InvoiceServiceBean	checkInstallmentApprovalAcceptReminder	158	0.00h	99.37%
TkanalApplicationBean	getNonInvoiceBalance	7	0.00h	85.71%
	getAccountStatus	11	0.00h	63.64%
	getInvoiceInformationArray	8	0.00h	62.50%
ShortMessageSenderBean	send	2	0.00h	50.00%
XdslActivationManagerBean	activateBatchjobSingle	2	0.00h	50.00%
TelsisLKSystemBean	getProductCodes	61	0.00h	31.15%
TkanalApplicationBean	getAllSubscribedProducts	12	0.00h	25.00%
CustomerServiceBean	getAddressInformation	25	0.00h	16.00%
K2Bean	getUsersByKof2AccountID	45	0.00h	6.67%
K2ServiceBean	getUsers	45	0.00h	6.67%
AALUseCaseBean	sendTekniskK2	24	0.19h	4.17%
AccountServiceBean	getTransactionsForAccount	115	0.02h	3.48%
	getTransactionsForAccount2	115	0.02h	3.48%
NitraBean	getInterfaceDataBras	75	0.02h	1.33%
NitraBeanImpl	createInterfaceDataBrasOutput	75	0.00h	1.33%
K2Bean	reopenCustomer	84	0.05h	1.19%
K2ServiceBean	reopenCustomer	84	0.05h	1.19%
K2Bean	closeCustomer	200	0.14h	1.00%
K2ServiceBean	closeCustomer	200	0.14h	1.00%

Current usage at Telenor

Usemon is now being used in ongoing projects to discover potential improvements in more than 80 enterprise applications at Telenor

We are trying to identify the worst pieces of code

- Often used code with bad performance
- Big percentage exception exits
- Dead code

This is done at the application level rather than the usual low level java.lang.*<#:-)

Usemon|Live

- A proof of concept real time UI
- Based on the Processing data visualization framework
  - http://processing.org
- Made for big screen presentation
- Animates class dependencies as graphs
  - Green arrows represents invocations
  - Spring layout
- Demo

Where can I find it?

Project site is hosted on Google Code under BSD license

http://usemon.org

Presentation slides and demo video

http://paulrene.no

UseMon beta