Lancaster University Leipzig

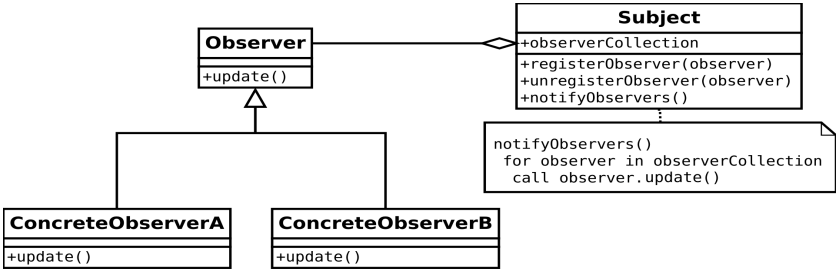
Instrumentierungstechnologien in Java

Dr. **David Georg Reichelt**[1]

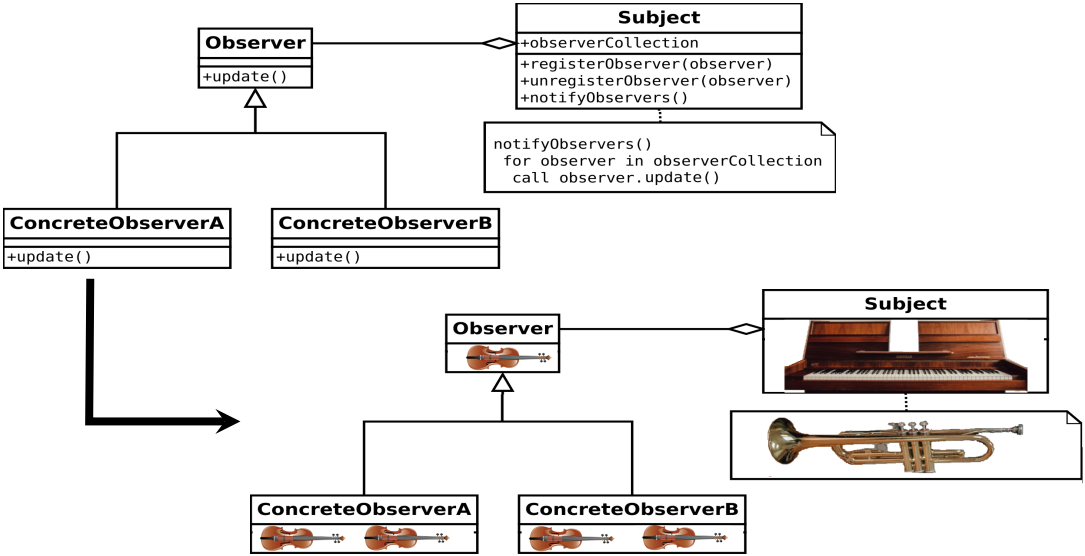
[1]School of Computing and Communications, Lancaster University Leipzig

23.03.2025

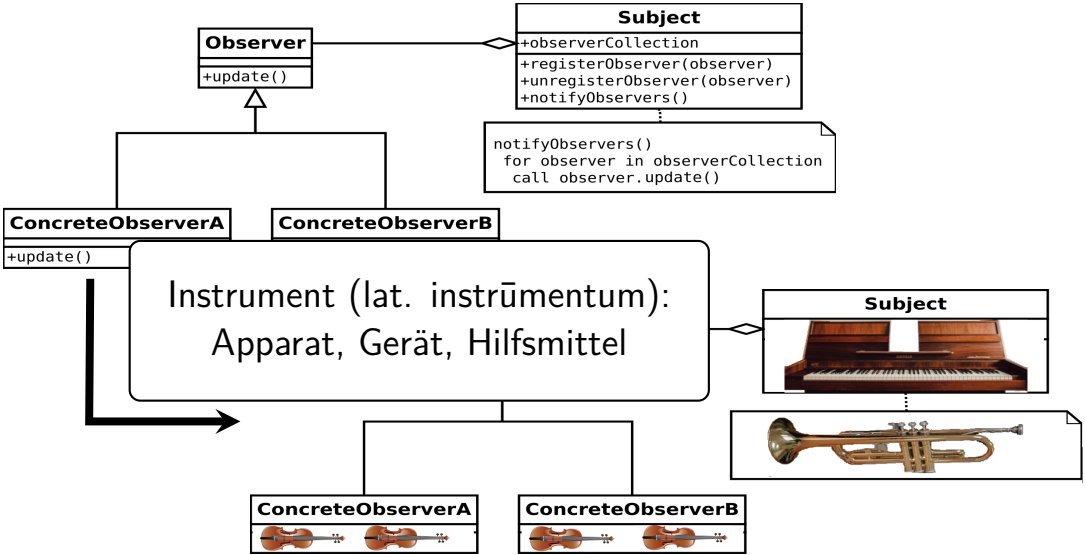
Instrumentierungstechnologien



Instrumentierungstechnologien



Instrumentierungstechnologien



Über mich

Werdegang

- Seit 2022:
  Assistenzprofessor für Softwareengineering
  Lancaster University Leipzig
- Seit 2013:
  Wissenschaftlicher Mitarbeiter
  Universität Leipzig
- Forschungsthemen: Dynamische und statische Analyse von Software

Publikationen

- D.G. Reichelt, S. Kühne, W. Hasselbring: **Automated Identification of Performance Changes at Code Level**
- Performance-Regressionen frühzeitig erkennen und vermeiden (`https://www.informatik-aktuell.de/entwicklung/methoden/performance-regressionen-fruehze.html`)
- D.G. Reichelt, L. Bulej, R. Jung, A. van Hoorn: **Overhead Comparison of Instrumentation Frameworks**

Über mich

Werdegang

- Seit 2022:
  Assistenzprofessor für Softwareengineering
  Lancaster University Leipzig
- Seit 2013:
  Wissenschaftlicher Mitarbeiter
  Universität Leipzig
- Forschungsthemen: Dynamische und statische Analyse von Software

Publikationen

- D.G. Reichelt, S. Kühne, W. Hasselbring: **Automated Identification of Performance Changes at Code Level**
- Performance-Regressionen frühzeitig erkennen und vermeiden (`https://www.informatik-aktuell.de/entwicklung/methoden/performance-regressionen-fruehze.html`)
- D.G. Reichelt, L. Bulej, R. Jung, A. van Hoorn: **Overhead Comparison of Instrumentation Frameworks**

Überblick

- Instrumentierung in der JVM
- Technologien
  - Javassist
  - AspectJ
  - ByteBuddy
- Overheadvergleich

Überblick

- Instrumentierung in der JVM
- Technologien
  - Javassist
  - AspectJ
  - ByteBuddy
- Overheadvergleich

Instrumentierung: Beispiel

```
1 public int calculateDouble(int y) {
2
3   return y * 2;
4 }
```

```
1 public int calculateDouble(int y) {
2   LOG.info("de.MyClass.calculateDouble(int);");
3   return y * 2;
4 }
```

Instrumentierung: Beispiel

```
1 public int calculateDouble(int y) {
2
3   return y * 2;
4 }
```

```
1 public int calculateDouble(int y) {
2   LOG.info("de.MyClass.calculateDouble(int);");
3   return y * 2;
4 }
```

- Ändern von Bytecode zur Laufzeit: Hinzufügen von Methoden, Klassen, ...

Anwendungsfälle

- Ursprünglich: Separation of Concerns
  - Automatisiertes Logging
  - Sicherheitsüberprüfungen
  - Aspektorientierte Programmierung

Beispiel: Transaktionsmanagement

```
1  public abstract aspect AbstractTransAspect {
2    abstract pointcut transactionOperations();
3      void around() : transactionOperations() {
4    try {
5      InitialContext ic = new InitialContext();
6      UserTransaction ut = (UserTransaction)
7      ic.lookup("java:comp/UserTransaction");
8      ut.begin();
9      proceed();
10     ut.commit();
11   }
12 }
```

Aus: A. Mesbah and A. van Deursen, "Crosscutting concerns in J2EE applications," 7th IEEE ISWSE, https://www.doi.org/10.1109/WSE.2005.4

Beispiel: Transaktionsmanagement II

```
1 public aspect PetStoreTransactionAspect
2   extends AbstractTransAspect {
3     pointcut transactionOperations() :
4   execution(* com..TemplateServlet.insertTemplate(..)
5    throws IOException, ServletException) ||
6   execution(* com..RcvrRequestProcessor.updateAndSend(..)
7     throws IOException, ServletException);
8 }
```

Aus: A. Mesbah and A. van Deursen, "Crosscutting concerns in J2EE applications," 7^{th} IEEE ISWSE, https://www.doi.org/10.1109/WSE.2005.4

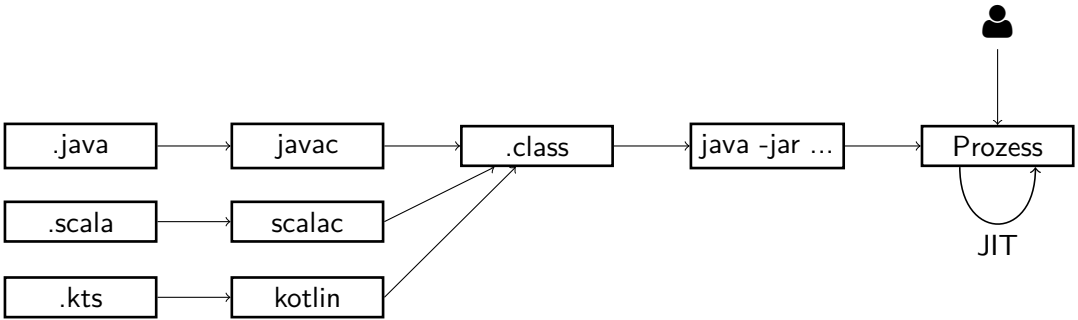
Anwendungsfälle

- Ursprünglich: Separation of Concerns
  - Automatisiertes Logging
  - Sicherheitsüberprüfungen
  - Aspektorientierte Programmierung
- Observability Tools
  - Erfassen von Start- und Endzeiten von Service-Aufrufen, Methodenaufrufen, ...
  - Erkennen von Anomalien
  - Architekturerkennung
- Transaktionsüberwachung
- Mocking von Testobjekten

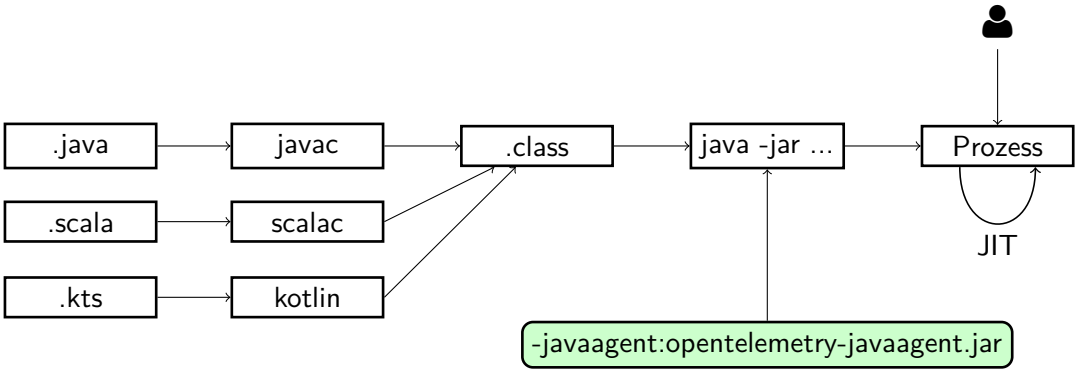
Anwendungsfälle

- Ursprünglich: Separation of Concerns
  - Automatisiertes Logging
  - Sicherheitsüberprüfungen
  - Aspektorientierte Programmierung
- Observability Tools
  - Erfassen von Start- und Endzeiten von Service-Aufrufen, Methodenaufrufen, ...
  - Erkennen von Anomalien
  - **Architekturerkennung**
- Transaktionsüberwachung
- Mocking von Testobjekten

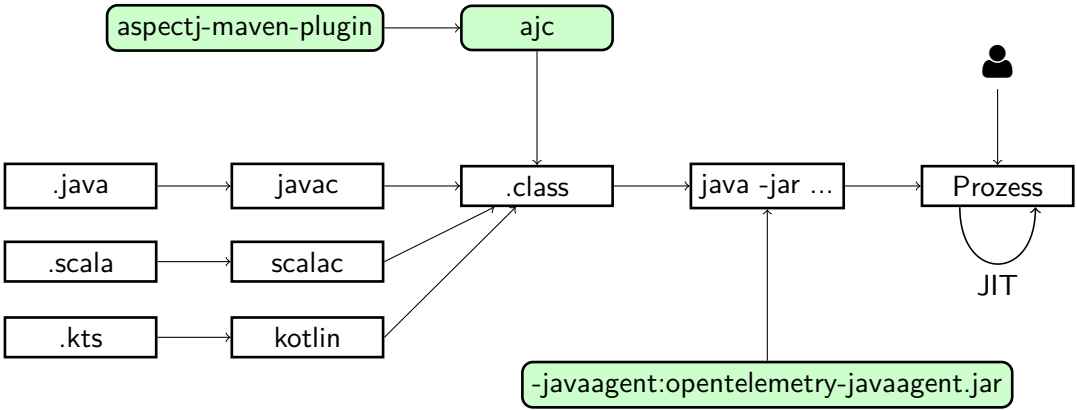
ByteCode and JIT



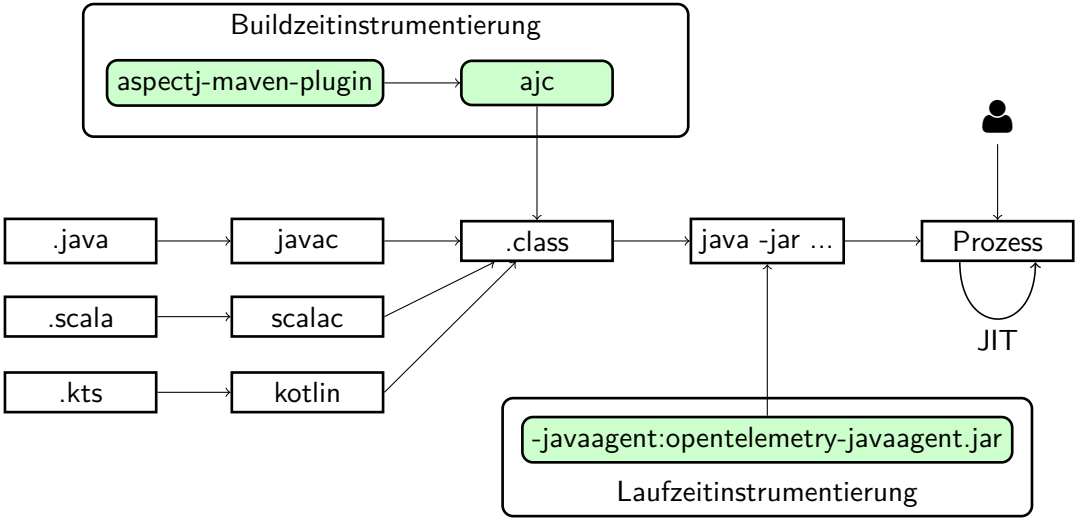
ByteCode and JIT



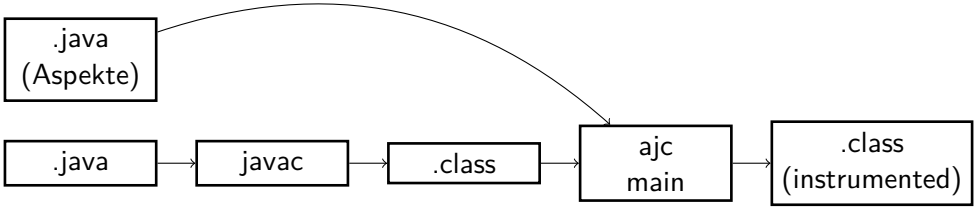
ByteCode and JIT



ByteCode and JIT



Buildzeitinstrumentierung



- Änderung des Bytecodes vor der Ausführung
  - Weaving-Code muss zur Buildzeit bekannt sein
  - Reduziert Overhead während des Classloadings
  - Erfordert Anpassung des Buildprozesses

Laufzeitinstrumentierung

- Wird beim Start übergeben, bspw.
  `java -javaagent:opentelemetry-javaagent.jar -jar tomcat.jar`
- Führt zum Aufruf der `premain` vor der `main` Methode der jar
- Kann einen `ClassFileTransformer` registrieren

```
1 public final class PremainClass {
2   public static void premain(String agentArgs,
       Instrumentation inst) {
3     ClassFileTransformer t = new Transformer();
4     inst.addTransformer(t);
5   }
6 }
```

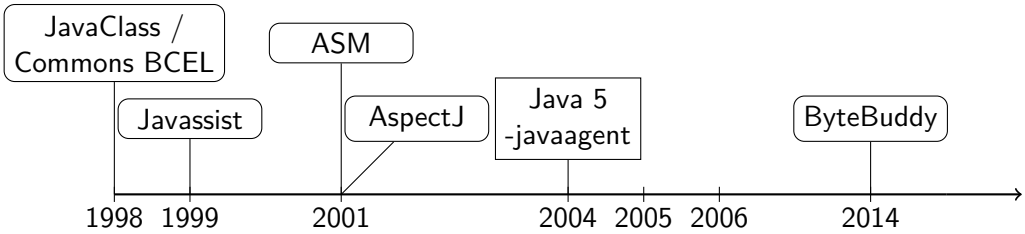
Laufzeitinstrumentierung

```
1  public class Transformer implements ClassFileTransformer{
2    @Override
3    public byte[] transform(
4      ClassLoader loader,
5          String className,
6          Class<?> classBeingRedefined,
7          ProtectionDomain protectionDomain,
8          byte[] classfileBuffer) {
9          ...
10   }
11 }
```

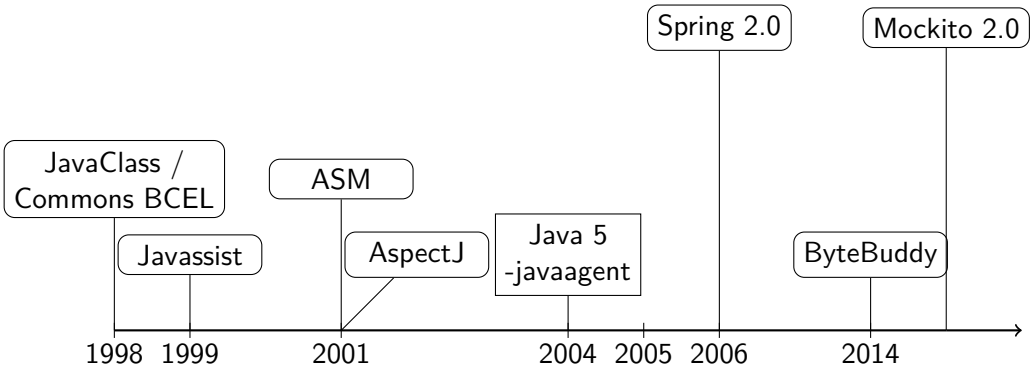
Laufzeitinstrumentierung

```
1  public class Transformer implements ClassFileTransformer{
2    @Override
3    public byte[] transform(
4      ClassLoader loader,
5          String className,
6          Class<?> classBeingRedefined,
7          ProtectionDomain protectionDomain,
8          byte[] classfileBuffer) {
9          ...
10   }
11 }
```

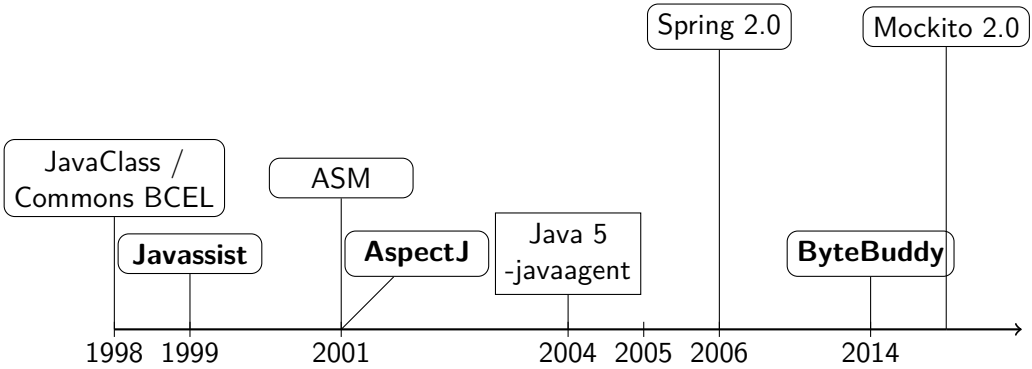
Entwicklung von Instrumentierungstechnologien



Entwicklung von Instrumentierungstechnologien



Entwicklung von Instrumentierungstechnologien



Alternative: Annotation Processing API

- Definition eigener Annotation

```
1 @SupportedAnnotationTypes("*")
2 public class LombokProcessor extends
     AbstractProcessor {
3     @Override
4     public boolean process(
5       Set<? extends TypeElement> annotations,
6       RoundEnvironment roundEnv) {
7         ...
8     }
9 }
```

- Wird zur Kompilierzeit eingesetzt, bspw. in Lombok

Überblick

- Instrumentierung in der JVM
- Technologien
  - Javassist
  - AspectJ
  - ByteBuddy
- Overheadvergleich

Überblick

- Instrumentierung in der JVM
- Technologien
  - Javassist
  - AspectJ
  - ByteBuddy
- Overheadvergleich

Javassist: Dynamische Bytecode-Manipulation

- API zur Bytecode-Manipulation
  - Javaagent (premain) und Selektion müssen selbst implementiert werden
- Manipulation von Klassen durch Strings
  - Einfache Erstellung
  - Keine Syntaxprüfung bis zur Ausführung
  - Kein Autoboxing:
    `Integer i = 3;` ⇒ `Integer i = new Integer(3);`
- Repository wird gepflegt
- Mozilla Public License

Beispiel

```
1  final ClassPool pool = ClassPool.getDefault();
2  final CtClass clazz = pool.get(realClassName);
3
4  for (final CtMethod method : clazz.getDeclaredMethods())
5  {
6    if (!method.isEmpty()) {
7      String fqn = cc.getName() + "." + method.getName();
8      method.insertBefore("javassistAgent.CWM.call(\""
9        + fqn + "\");");
10   }
11 }
12
13 final byte[] byteCode = clazz.toBytecode();
14 clazz.detach();
```

Beispiel

```
1  final ClassPool pool = ClassPool.getDefault();
2  final CtClass clazz = pool.get(realClassName);
3
4  for (final CtMethod method : clazz.getDeclaredMethods())
5  {
6    if (!method.isEmpty()) {
7      String fqn = cc.getName() + "." + method.getName();
8      method.insertBefore("javassistAgent.CWM.call(\""
9        + fqn + "\");");
10   }
11 }
12
13 final byte[] byteCode = clazz.toBytecode();
14 clazz.detach();
```

Beispiel

```
1  final ClassPool pool = ClassPool.getDefault();
2  final CtClass clazz = pool.get(realClassName);
3
4  for (final CtMethod method : clazz.getDeclaredMethods())
5  {
6    if (!method.isEmpty()) {
7      String fqn = cc.getName() + "." + method.getName();
8      method.insertBefore("javassistAgent.CWM.call(\""
9        + fqn + "\");");
10   }
11 }
12
13 final byte[] byteCode = clazz.toBytecode();
14 clazz.detach();
```

Beispiel

```
1  final ClassPool pool = ClassPool.getDefault();
2  final CtClass clazz = pool.get(realClassName);
3
4  for (final CtMethod method : clazz.getDeclaredMethods())
5  {
6    if (!method.isEmpty()) {
7      String fqn = cc.getName() + "." + method.getName();
8      method.insertBefore("javassistAgent.CWM.call(\""
9        + fqn + "\");");
10   }
11 }
12
13 final byte[] byteCode = clazz.toBytecode();
14 clazz.detach();
```

Überblick

- Instrumentierung in der JVM
- Technologien
  - Javassist
  - AspectJ
  - ByteBuddy
- Overheadvergleich

AspectJ

- API für Bytecodemanipulation und Selektion von Instrumentierungspunkten
  - AspectJ javaagent kann Erweiterungen auf Basis von `aop.xml` selbst definieren
  - `ajc` ermöglicht Kompilierung
- Manipulation von Methodenausführungen durch Aspekte
  - Syntaxprüfung zur Kompilierzeit
  - Vererbung etc. ist möglich
- Eclipse Public License

AspectJ – Joinpoints und Pointcuts

```
1       public void callingMethod() {
2         myMethod();
3       }
4       public void myMethod() {
5         // Original code
6       }
```

- Join points: Punkte in der Programmausführung

AspectJ – Joinpoints und Pointcuts

```
1     public void callingMethod() {
2       myMethod();
3     }
4     public void myMethod() {
5       // Original code
6     }
```

- Join points: Punkte in der Programmausführung

AspectJ – Joinpoints und Pointcuts

```
1     public void callingMethod() {
2       myMethod();
3     }
4     public void myMethod() {
5       // Original code
6     }
```

- Join points: Punkte in der Programmausführung
  - Vor der Ausführung von `myMethod`

AspectJ – Joinpoints und Pointcuts

```
1     public void callingMethod() {
2       myMethod();
3     }
4     public void myMethod() {
5       // Original code
6     }
```

- Join points: Punkte in der Programmausführung
  - Vor der Ausführung von `myMethod`

AspectJ – Joinpoints und Pointcuts

```
1       public void callingMethod() {
2         myMethod();
3       }
4       public void myMethod() {
5         // Original code
6       }
```

- Join points: Punkte in der Programmausführung
  - Vor der Ausführung von `myMethod`
  - Nach der Ausführung von `myMethod`

AspectJ – Joinpoints und Pointcuts

```
1     public void callingMethod() {
2       myMethod();
3     }
4     public void myMethod() {
5       // Original code
6     }
```

- Join points: Punkte in der Programmausführung
  - Vor der Ausführung von `myMethod`
  - Nach der Ausführung von `myMethod`

AspectJ – Joinpoints und Pointcuts

```
1       public void callingMethod() {
2         myMethod();
3       }
4       public void myMethod() {
5         // Original code
6       }
```

- Join points: Punkte in der Programmausführung
  - Vor der Ausführung von `myMethod` – **Pointcut:** `execution(* *.myMethod())`
  - Nach der Ausführung von `myMethod` – **Pointcut:** `execution(* *.myMethod())`

AspectJ – Joinpoints und Pointcuts

- Join points: Punkte in der Programmausführung
  - Vor der Ausführung von `myMethod` – **Pointcut:** `execution(* *.myMethod())`
  - Nach der Ausführung von `myMethod` – **Pointcut:** `execution(* *.myMethod())`

AspectJ – Joinpoints und Pointcuts

- Join points: Punkte in der Programmausführung
  - Vor der Ausführung von `myMethod` – **Pointcut:** `execution(* *.myMethod())`
  - Nach der Ausführung von `myMethod` – **Pointcut:** `execution(* *.myMethod())`
- Aspect: Pointcut + advice

```
1  @Pointcut("execution(* *.myMethod())")
2  public void monitoredOperation() { }
3
4  @Before("monitoredOperation()")
5  public void beforeOperation(final JoinPoint jp) {
6     Signature signature = thisJoinPoint.getSignature();
7     String operationSignature =
8        signature.getDeclaringTypeName()
9        + "." + signature.getName();
10 }
```

AspectJ – Joinpoints und Pointcuts

- Join points: Punkte in der Programmausführung
  - Vor der Ausführung von `myMethod` – **Pointcut:** `execution(* *.myMethod())`
  - Nach der Ausführung von `myMethod` – **Pointcut:** `execution(* *.myMethod())`
- Aspect: Pointcut + advice

```
1  @Pointcut("execution(* *.myMethod())")
2  public void monitoredOperation() { }
3
4  @Before("monitoredOperation()")
5  public void beforeOperation(final JoinPoint jp) {
6     Signature signature = thisJoinPoint.getSignature();
7     String operationSignature =
8        signature.getDeclaringTypeName()
9        + "." + signature.getName();
10 }
```

- `aop.xml` wird genutzt um zu definieren, welche Aspekte für welche Klassen

aop.xml

```
1 <aspectj>
2   <weaver>
3     <include within="*"/>
4     <exclude within="org.slf4j..*" />
5   </weaver>
6   <aspects>
7     <aspect name="de.ittage.ExampleAspect">
8   </aspects>
9 </aspectj>
```

- Definition der Anwendung der Aspekte
  - Klassen, auf die die Aspekte angewandt werden (`include`)
  - Pointcut, nach denen in diesen Klassen gesucht wird
  - Instrumentierungscode (Advice), der angewandt wird

AspectJ und Spring

- Spring enthält eine reduzierte Menge von Pointcut-Ausdrücke
  - execution
  - within
  - @annotation
- AspectJ enthält weitere Pointcut-Ausdrücke
  - call
  - get
  - initialization

Überblick

- Instrumentierung in der JVM
- Technologien
  - Javassist
  - AspectJ
  - ByteBuddy
- Overheadvergleich

ByteBuddy

- API zur Bytecode-Manipulation
  - Javaagent (premain) muss selbst implementiert werden
  - Selektion von instrumentierten Programmteilen wird unterstützt
- Spezifikation der Advices ähnelt AspectJ
  - Syntaxprüfung zur Kompilierzeit
  - Vererbung, etc. möglich
- Apache License

ByteBuddy Advices

```
1  @Advice.OnMethodEnter
2  public static void enter(@Advice.Origin final String
       operationSignature,
3    @Advice.FieldValue(value = "_CALLS_", readOnly =
       false) Set<String> _CALLS_) {
4      if (_CALLS_ == null) {
5        _CALLS_ = new HashSet<String>();
6      }
7      ...
8    }
```

ByteBuddy ElementMatcher

```
1 public static void premain(String agentArgs,
     Instrumentation inst) {
2   new AgentBuilder.Default()
3     .method(ElementMatchers.named("myMethod")
4       .and(ElementMatchers.hasAnnotation(MyAnnotation.class))
5     .and(ElementMatchers.returns(String.class)))
6     .intercept(MethodDelegation.to(MyInterceptor.class))
7     .installOn(inst);
8 }
```

ByteBuddy in Frameworks

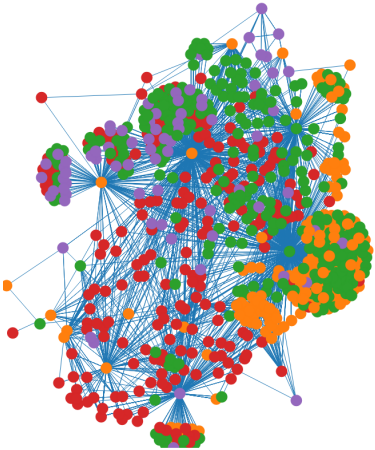
- ByteBuddy wird von verschiedenen Frameworks benutzt
- Änderung von Rückgabewerten
  - Mockito: Ersetzen von Methodenrückgabewerten mit Mock-Werten

```
1 @Advice.OnMethodExit
2 private static void exit(
3   @Advice.Return(readOnly = false, ..) Object returned,
4   @Advice.Enter Callable<?> mocked)
5     throws Throwable {
6   if (mocked != null) {
7     returned = mocked.call();
8   }
9 }
```

- Hibernate: Ersetzen von Daten mit Datenbankinhalten
- Observability von Software: OpenTelemetry, Elastic APM Agent, ...

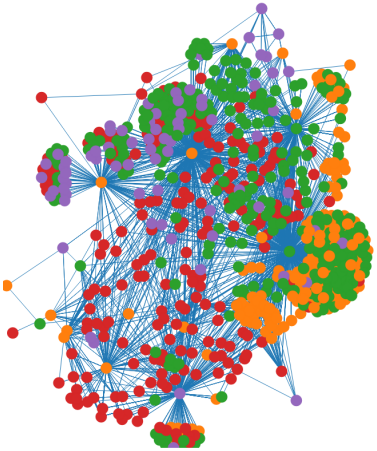
(Funktionaler) Vergleich der Technologien

- Demo: `https://github.com/DaGeRe/demo-agents`



(Funktionaler) Vergleich der Technologien

- Demo: `https://github.com/DaGeRe/demo-agents`



	Javassist	AspectJ	ByteBuddy
Implementierung	Strings	Aspekte (Code)	Aspekte (Code)
Instrumentierung	–	ajc und javaagent	–
Matching	–	`aop.xml`	Matcher
Granularität	Methoden	Methoden + Lambda	Methoden

Überblick

- Instrumentierung in der JVM
- Technologien
  - Javassist
  - AspectJ
  - ByteBuddy
- Overheadvergleich

MooBench

- MooBench: Benchmark für Overhead von Tracing in Observability Tools (Kieker, OpenTelemetry, inspectIT)
- Quellen für Overhead
  - **Instrumentierung der Anwendung**
  - Erfassen der Daten
  - Verarbeitung der Daten (Schreiben in Queue, Serialisierung, . . .)

MooBench

- MooBench: Benchmark für Overhead von Tracing in Observability Tools (Kieker, OpenTelemetry, inspectIT)
- Quellen für Overhead
  - **Instrumentierung der Anwendung**
  - Erfassen der Daten
  - Verarbeitung der Daten (Schreiben in Queue, Serialisierung, ...)

- Ansatz: Vergleich der Instrumentierungstechnologien
  - Ausführung desselben Workloads
  - Tracing derselben Methoden
  - Austausch der Instrumentierungstechnologie

Performancevergleich (Laufzeit)

Benchmark	Mean	Standard Deviation	Mean	Standard Deviation
	i7-4770		Raspberry Pi	
Baseline	0.05	0.00	0.16	0.00
Kieker-java-aspectj (deactivated)	0.21	0.00	0.93	0.01
-aspectj (full)	3.35	0.07	12.69	2.21
-bytebuddy (deactivated)	0.07	0.00	0.31	0.01
-bytebuddy (full)	2.61	0.13	8.10	0.47
-disl (deactivated)	0.07	0.00	0.31	0.02
-disl (full)	2.75	0.18	8.30	0.67
-javassist (deactivated)	0.06	0.00	0.27	0.01
-javassist (full)	2.58	0.17	8.25	0.58

Performancevergleich (Buildzeit)

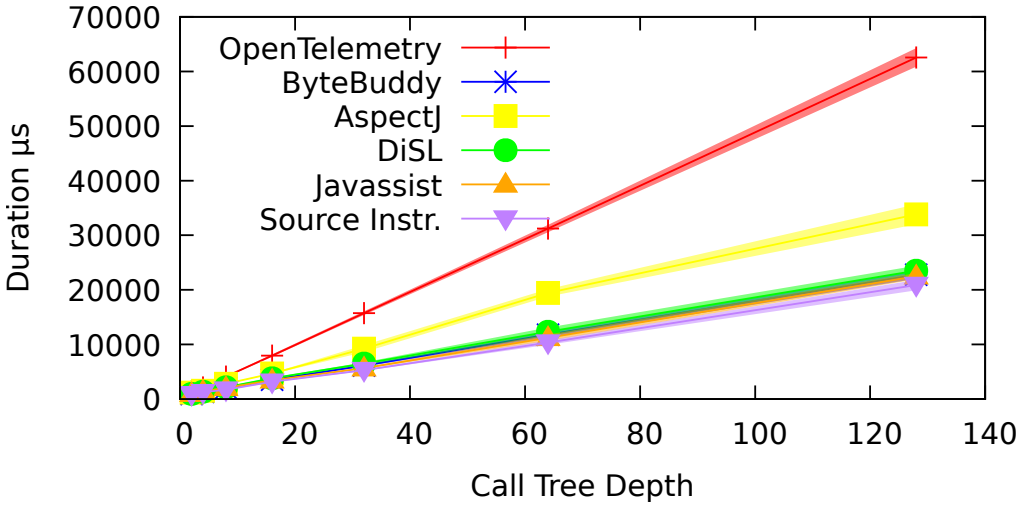
Benchmark	Mean	Standard Deviation	Mean	Standard Deviation
	i7-4770		Raspberry Pi	
Baseline	0.05	0.00	0.16	0.01
Kieker-java				
-aspectj (deactivated)	0.22	0.00	1.17	0.05
-aspectj (full)	3.35	0.12	13.75	3.63
-bytebuddy (deactivated)	0.06	0.00	0.27	0.01
-bytebuddy (full)	2.53	0.12	8.44	0.89
-javassist (deactivated)	0.06	0.00	0.27	0.01
-javassist (full)	2.50	0.06	8.19	0.58
-sourceinstrumentation				
(deactivated)	0.07	0.00	0.28	0.01
(full)	2.32	0.15	7.20	0.55

Performancevergleich (Buildzeit)

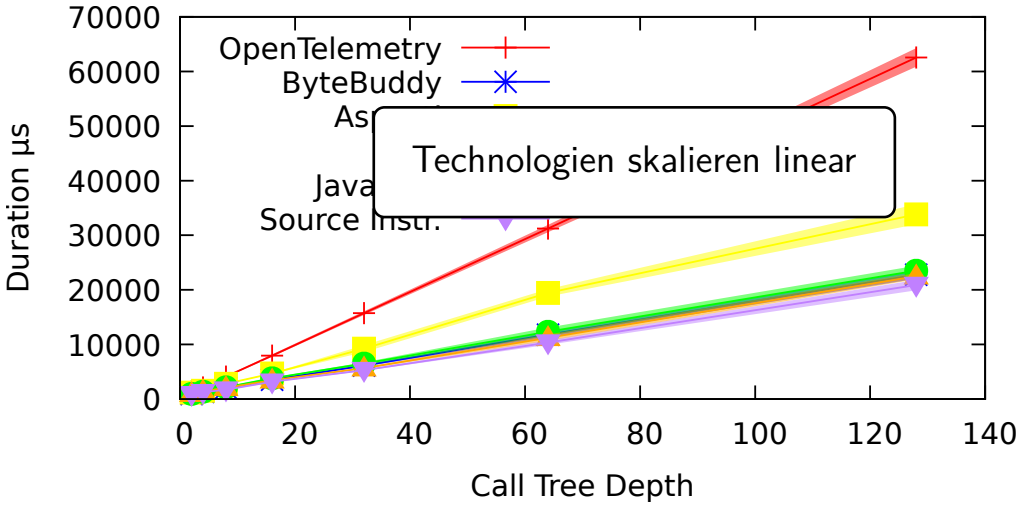
Direkt in den Quelltext schreiben ist am schnellsten
Javassist und ByteBuddy sind vergleichbar schnell

Baseline	0.05	0.00	0.10	0.01
Kieker-java				
-aspectj (deactivated)	0.22	0.00	1.17	0.05
-aspectj (full)	3.35	0.12	13.75	3.63
-bytebuddy (deactivated)	0.06	0.00	0.27	0.01
-bytebuddy (full)	2.53	0.12	8.44	0.89
-javassist (deactivated)	0.06	0.00	0.27	0.01
-javassist (full)	2.50	0.06	8.19	0.58
-sourceinstrumentation				
(deactivated)	0.07	0.00	0.28	0.01
(full)	2.32	0.15	7.20	0.55

Skalierbarkeit



Skalierbarkeit



Zusammenfassung

- Instrumentierungstechnologien...
  - ändern Code zur Build- oder Laufzeit
  - können Querschnittsthemen abbilden
- Implementierung...
  - erfolgt durch `premain` Methode
  - erfordert das Anpassen des `byte[]` der .class Datei
- Javassist, AspectJ und ByteBuddy ermöglichen Instrumentierung
  - AspectJ bietet umfassendste Implementierung
  - Javassist und ByteBuddy haben geringsten Overhead

Vielen Dank für Ihre Aufmerksamkeit!

- David Georg Reichelt
- Assistenzprofessor für Softwareengineering
- Lancaster University Leipzig
- Kontakt: d.g.reichelt [a t] lancaster.ac.uk