



Technische Hochschule
Ingolstadt

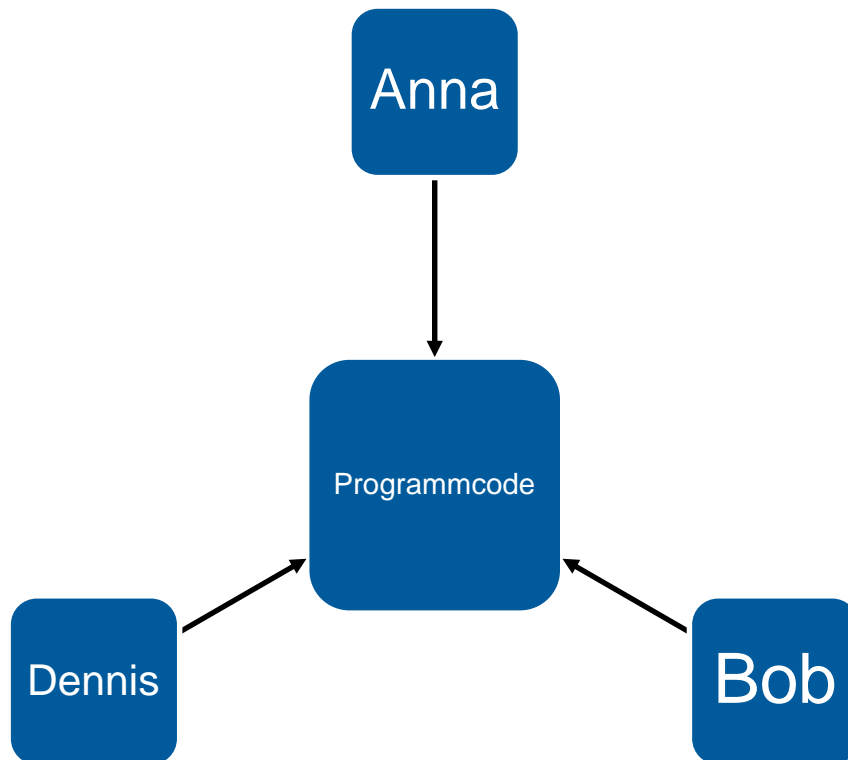
Fakultät für Elektrotechnik
und Informatik

*Zukunft in
Bewegung*

Projektentwicklung mit Git

Matthias Strauß 12.11.2018

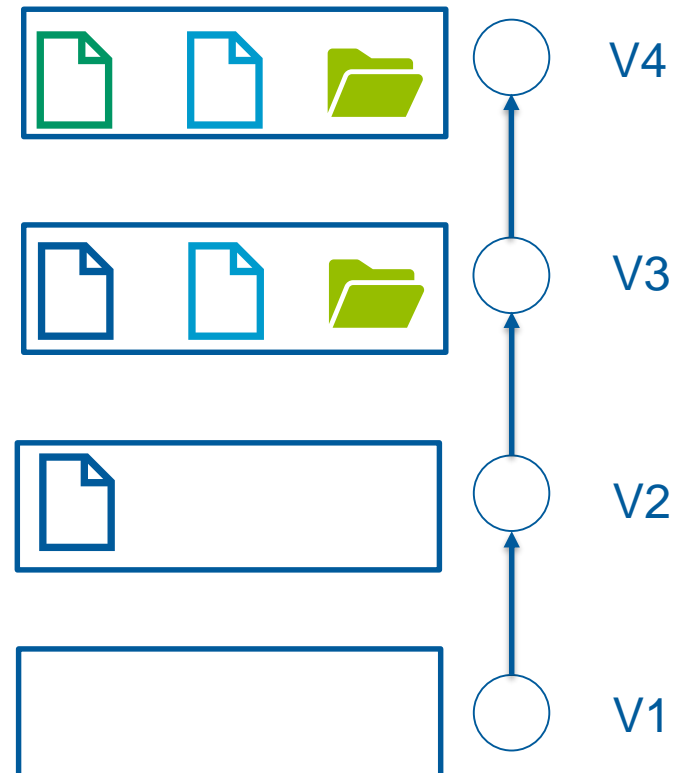


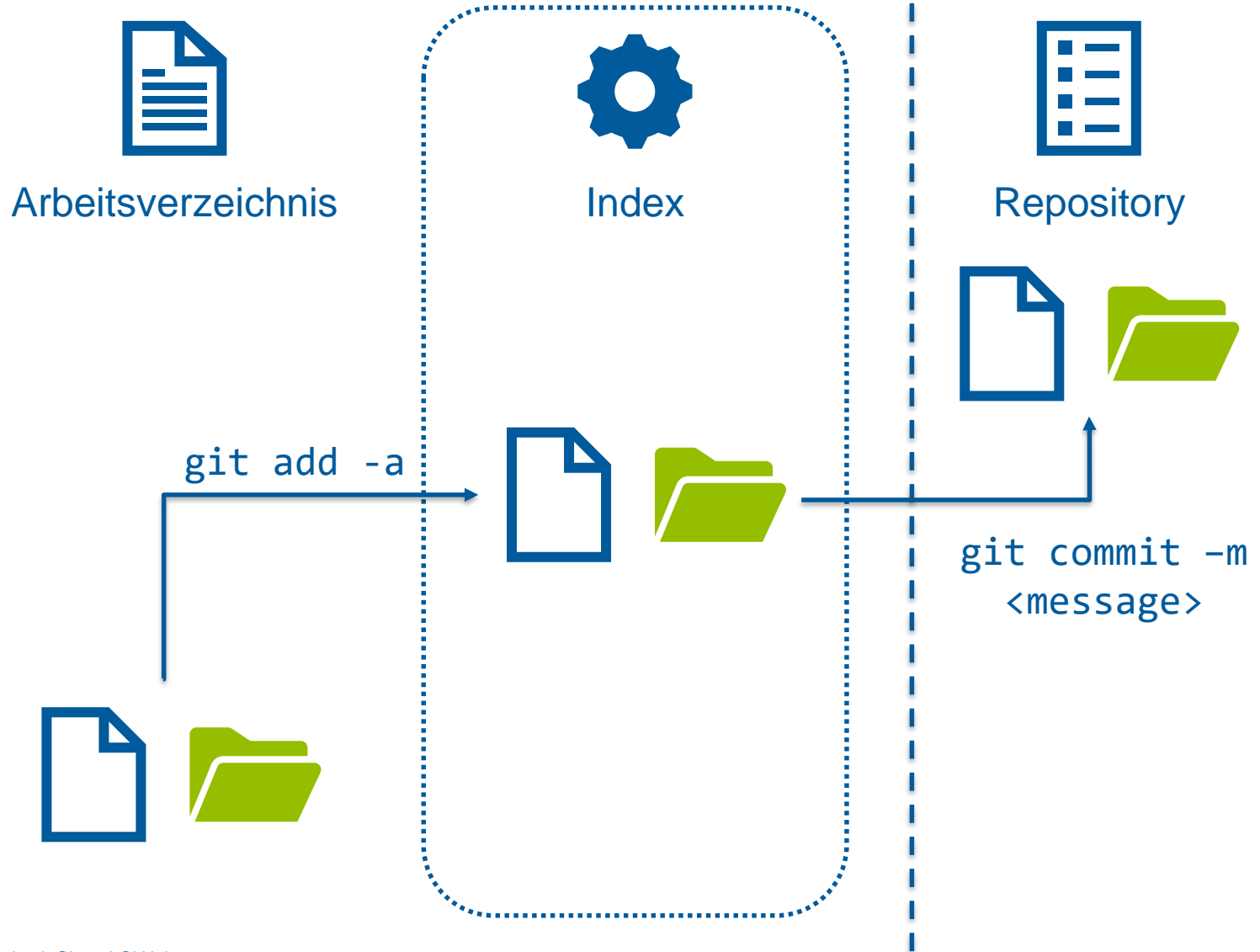


Email?

Cloudspeicher?

- Speichert und verwaltet den Zustand von Dateien
- Ein Abbild heißt Commit und besteht aus
 - Kopien der geänderten Dateien
 - Autor
 - Datum
 - Nachricht
 - Hash (id)

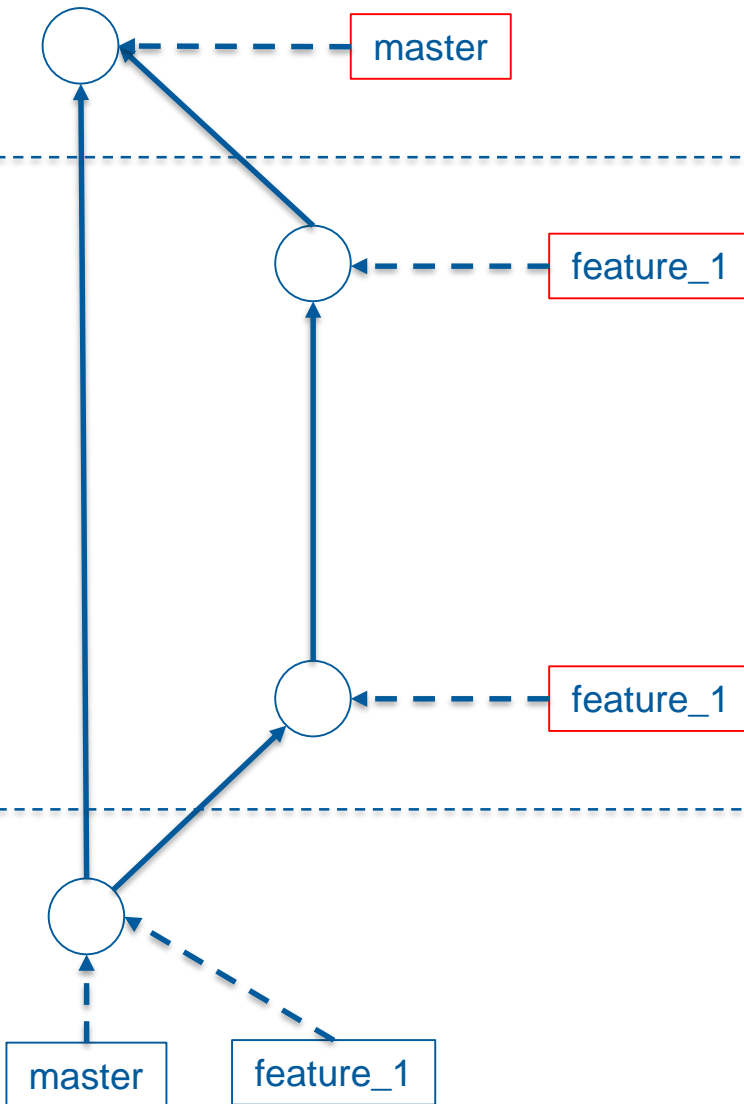






- Git im aktuellem Verzeichnis initialisieren:
 - `git init`
- Aktuellen Status von git anzeigen:
 - `git status`
 - Meldet neue, geänderte und entfernte Dateien
 - Zeigt den aktuellen Branch und Hinweise, wie fortgefahren werden kann
- Commit log anzeigen
 - `git log`

Workflow - Branches



```
git checkout master
```

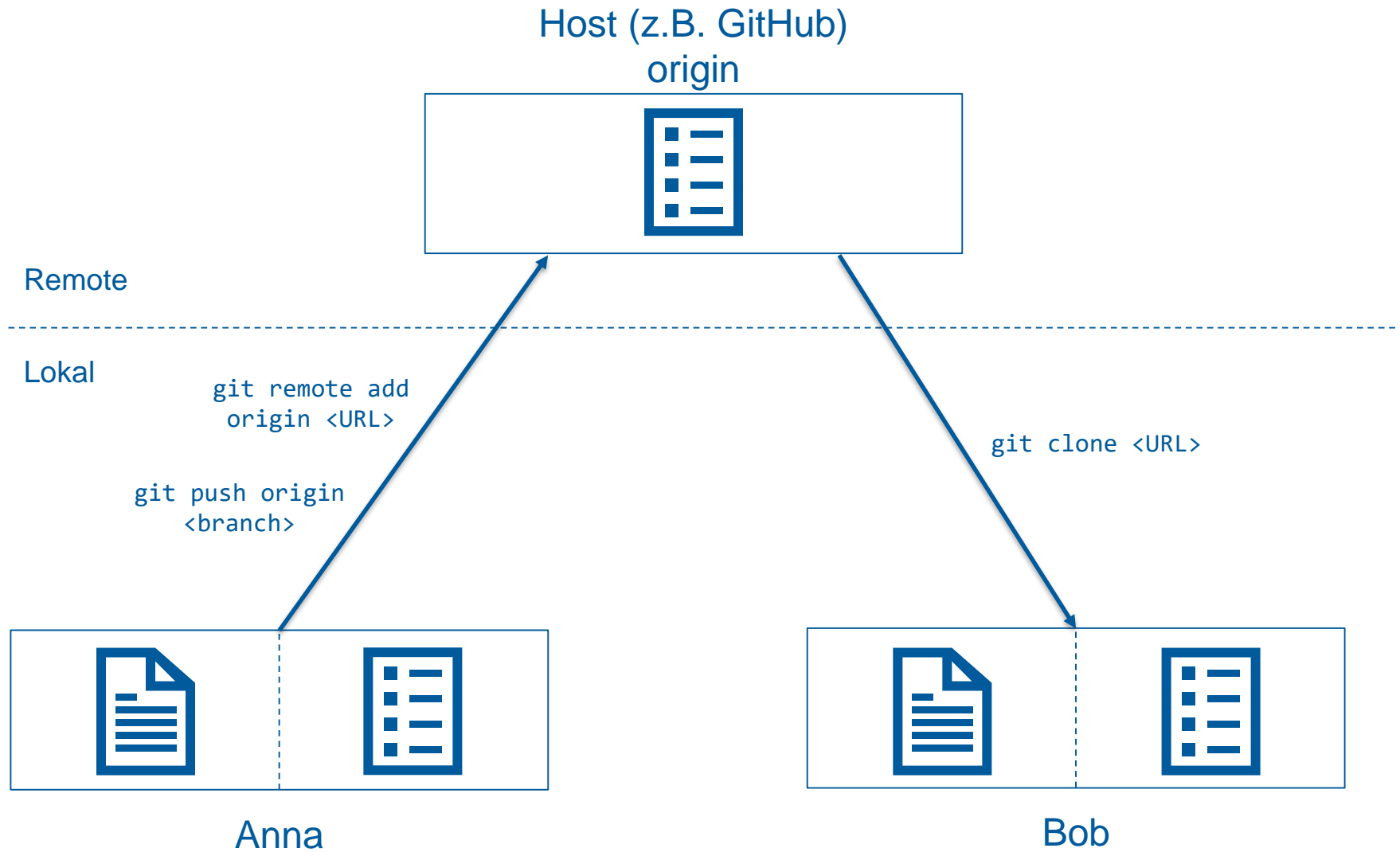
```
git merge -no-ff feature_1
```

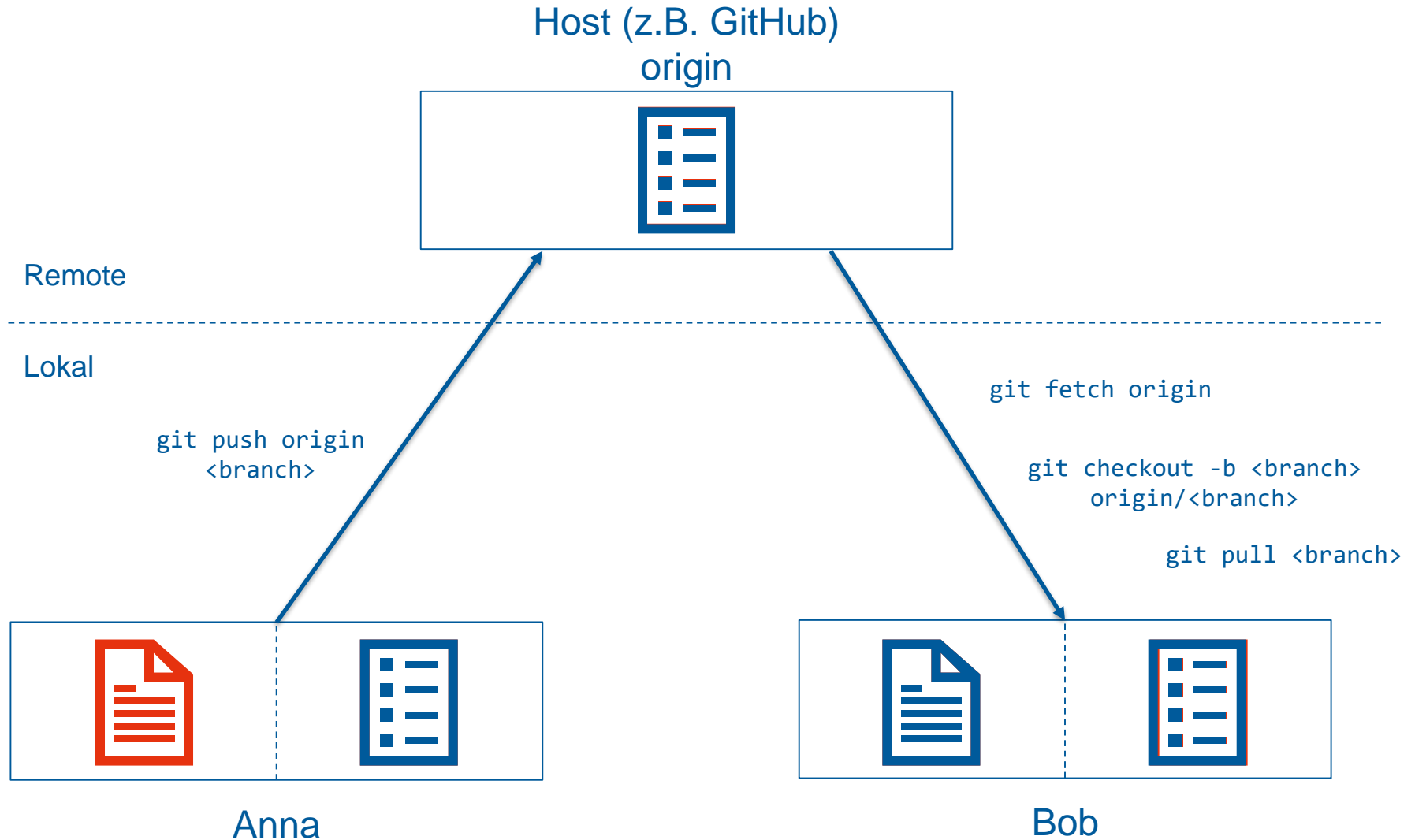
```
git commit -a -m <message>
```

```
git commit -a -m <message>
```

```
git branch feature_1
```

```
git checkout feature_1
```

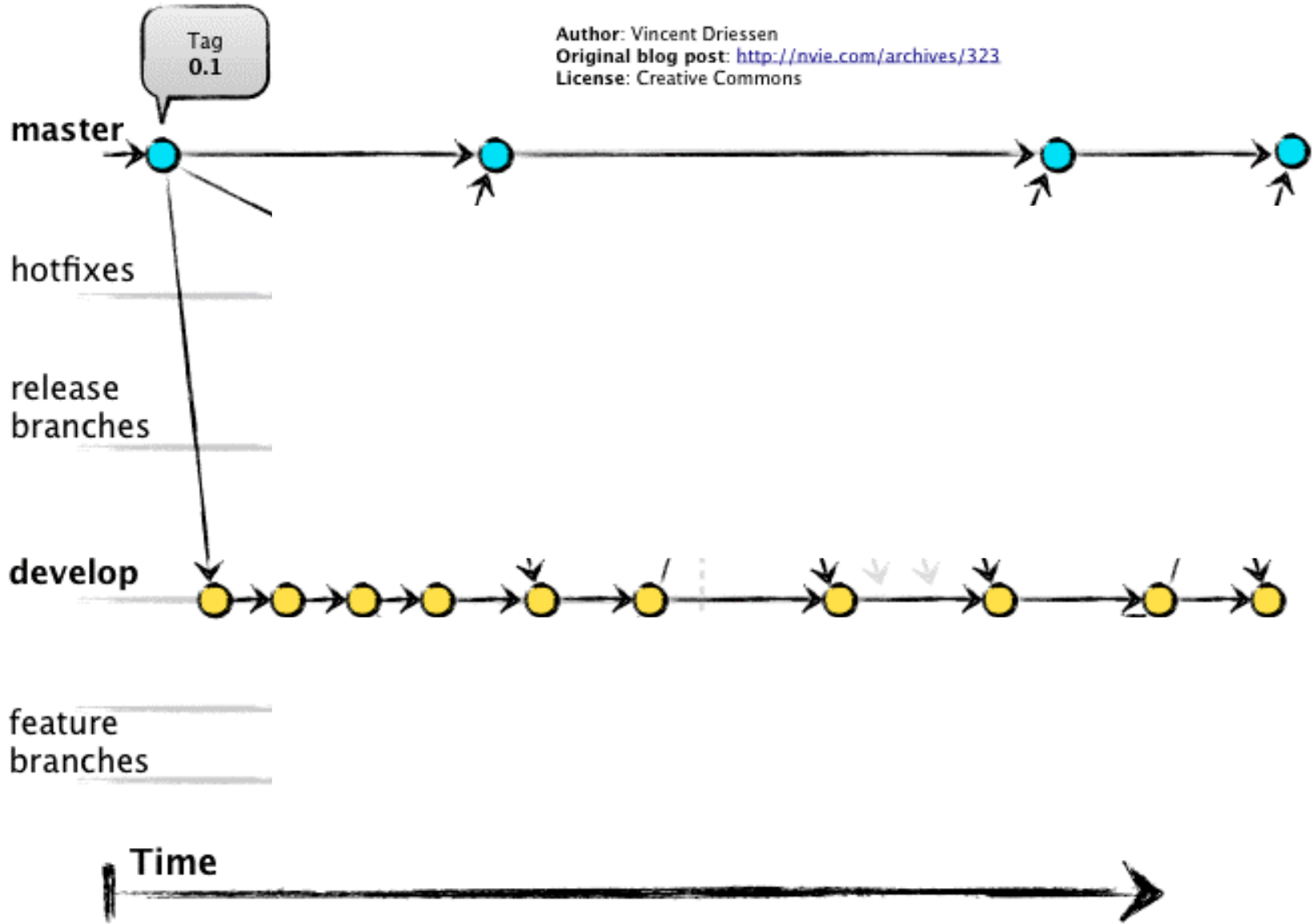




Workflows – Git flow





Author: Vincent Driessen
Original blog post: <http://nvie.com/archives/323>
License: Creative Commons

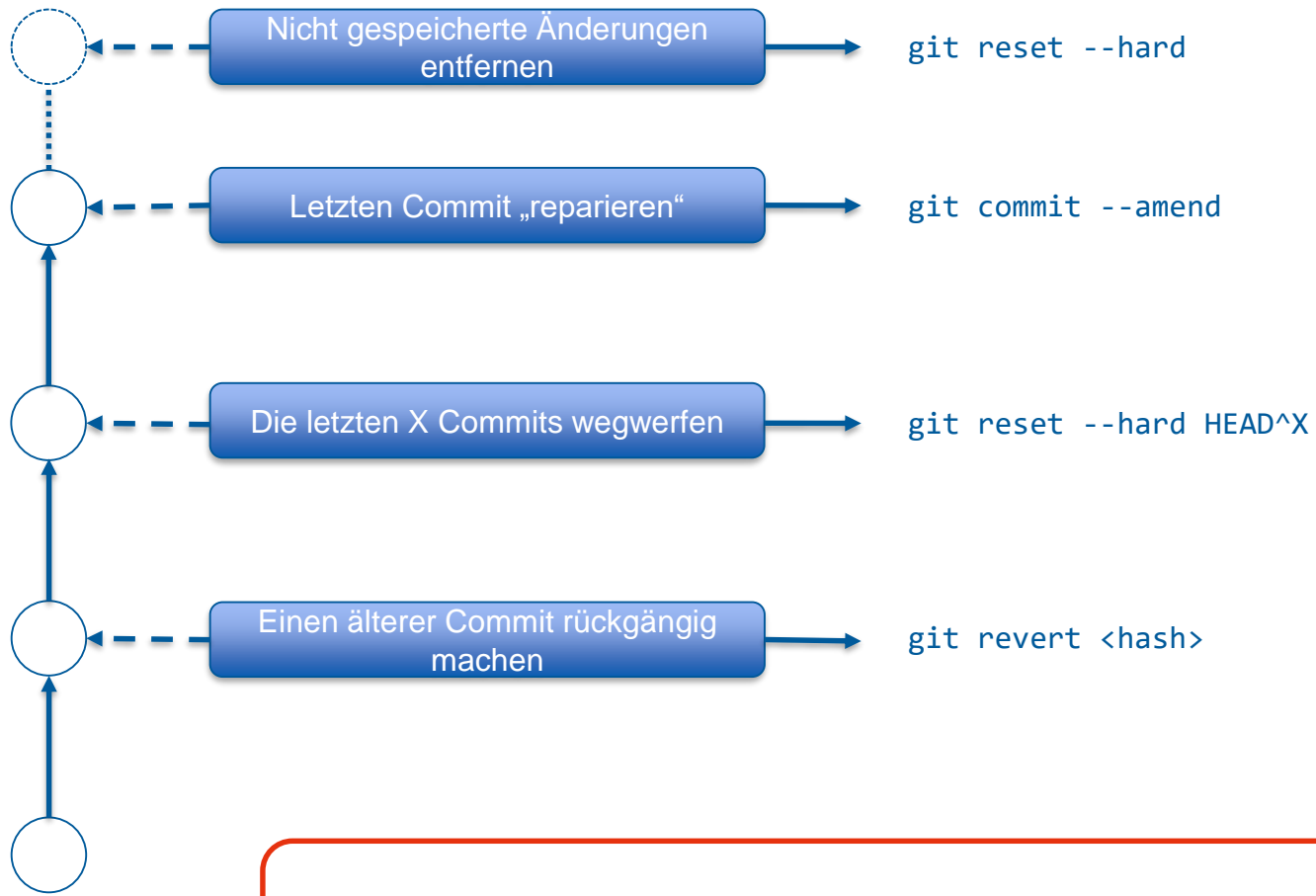




- Fester Workflow, an den sich alle Entwickler halten
- Keine BLOBS, kompilierten Code oder Logs in das Repository aufnehmen
 - .gitignore anlegen
- Datenbank versionieren
- Schöne Commit Nachrichten
 - Erklären, was der Code bewirkt
 - Verweise zu Arbeitsaufgaben

	Strauss, Matthias	b9e1ceec1d	BRO-41 Button-Message geändert, Public und All-Users Einstellungen werden kopiert
	Strauss, Matthias	90f96f23d26	BRO-21 pom.xml nicht mehr in target einfüegen

Erweiterte Git Verwendung – Fehler entfernen



Nur `git revert` verändert die history nicht!



- Git Reference Manual: <https://git-scm.com/docs>
- Das Buch Pro Git von Scott Chacon und Ben Straub: <https://git-scm.com/book> (2. Edition, 2014)
- Git-Guide von Atlassian: <https://www.atlassian.com/git>
- Escape a git mess: <http://justinhileman.info/article/git-pretty/git-pretty.png>
- Git flow: <https://nvie.com/posts/a-successful-git-branching-model/>

Bilder

- Git Flow: Author: [Luca Mezzalira](https://lucamezzalira.com/2014/03/10/git-flow-vs-github-flow/), <https://lucamezzalira.com/2014/03/10/git-flow-vs-github-flow/>, Creative Commons BY-SA



Technische Hochschule
Ingolstadt

Fakultät für Elektrotechnik
und Informatik

*Zukunft in
Bewegung*

Persistenz

JDBC, JPA

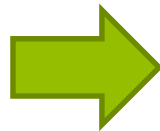
Katrin Krüger 12.11.18



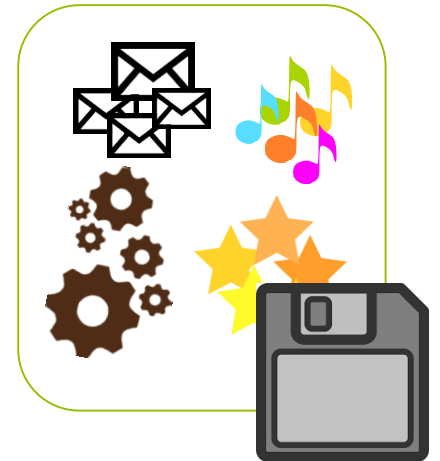
Lat. persistere = bestehen bleiben



Daten



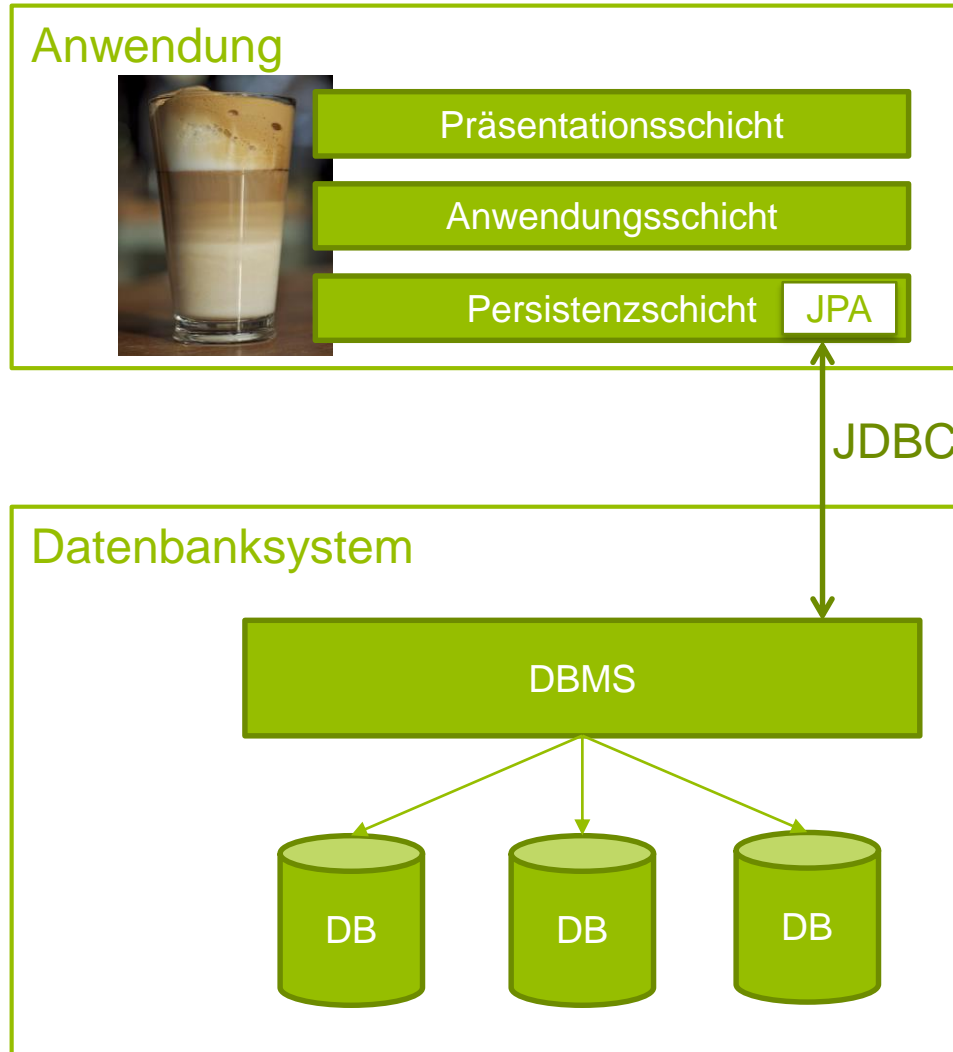
Informationssystem



Informationen

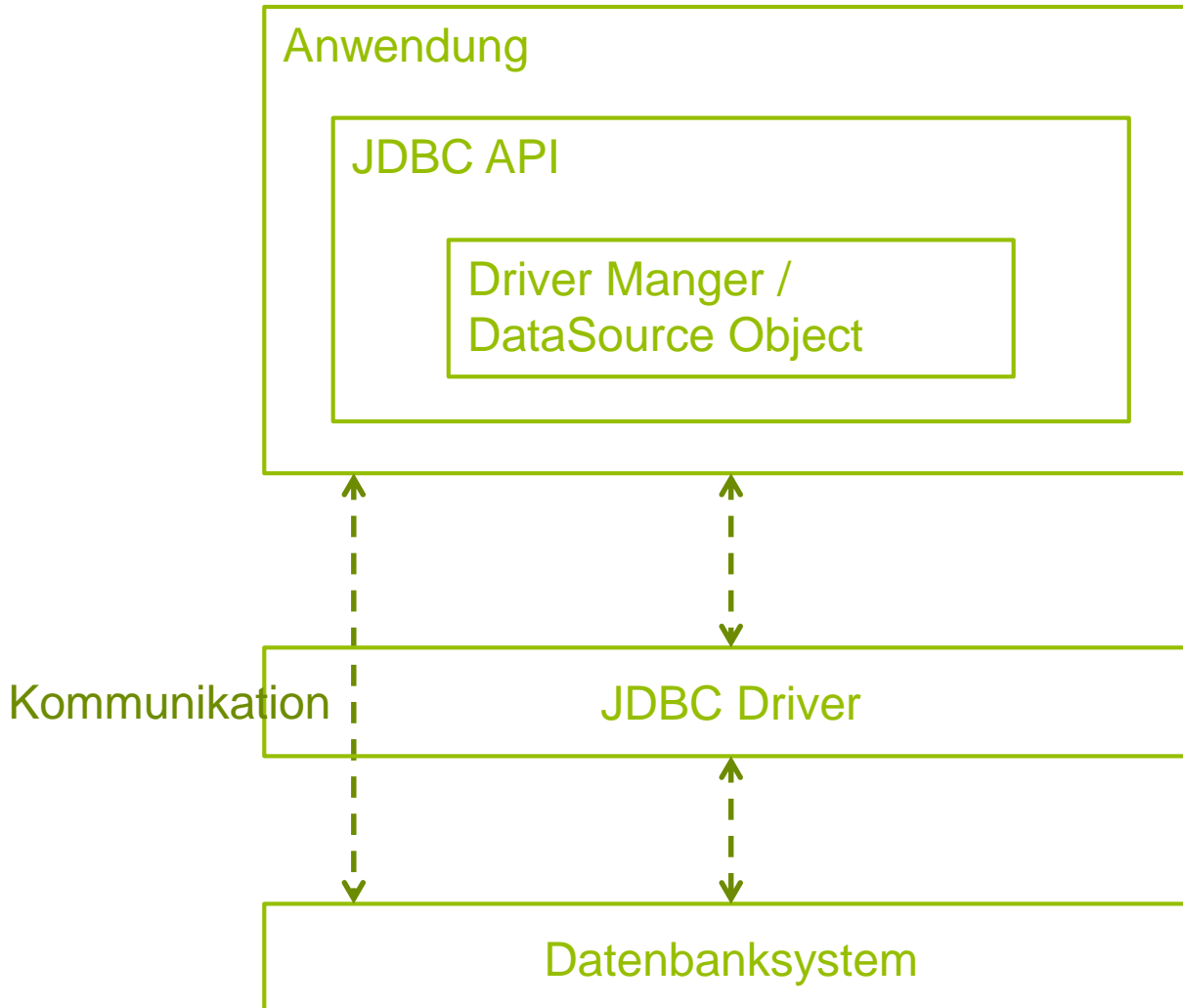
Persistenz

Drei Schichten Architektur



Java Database Connectivity

Motivation & Aufbau





Verbindungsaufbau

```
@Resource(lookup="java:jboss/datasources/Shop")  
private DataSource dataSource;  
Connection connection = dataSource.getConnection();
```

Datenmanipulation

```
String sql = "SELECT * FROM Order";  
Statement st = connection.createStatement();  
ResultSet rs = st.executeQuery(sql);
```

Verbindungsabbau

```
connection.close()
```



Transaktionsmanagement

Explizites Management statt Autocommit

http://openbook.rheinwerk-verlag.de/javainsel9/javainsel_24_009.htm#mj17d275b71ed1cf7b6f2b9511c8b63c58

PreparedStatement

Parametrisierte SQL-Abfragen

http://openbook.rheinwerk-verlag.de/javainsel9/javainsel_24_008.htm#mjdeb4eefa360476894b8cd02a4767f015

Callable Statements

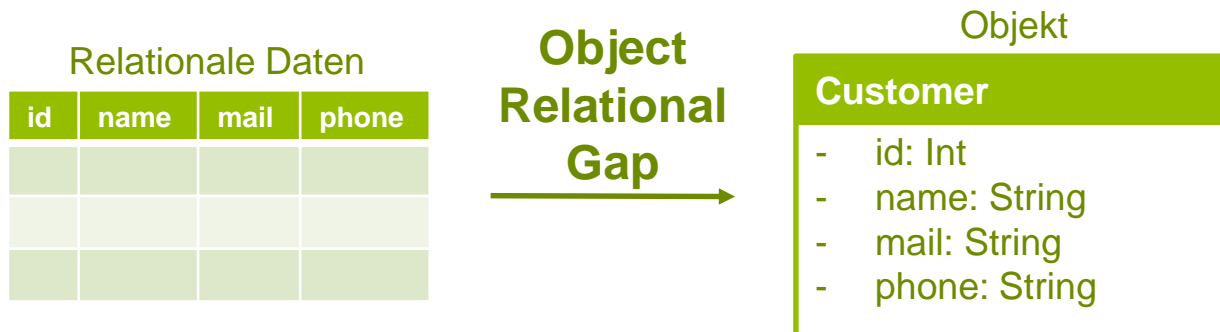
Stored Procedures und Funktionen nutzen & anlegen

<https://www.tutorialspoint.com/jdbc/jdbc-statements.htm>

Bisher:

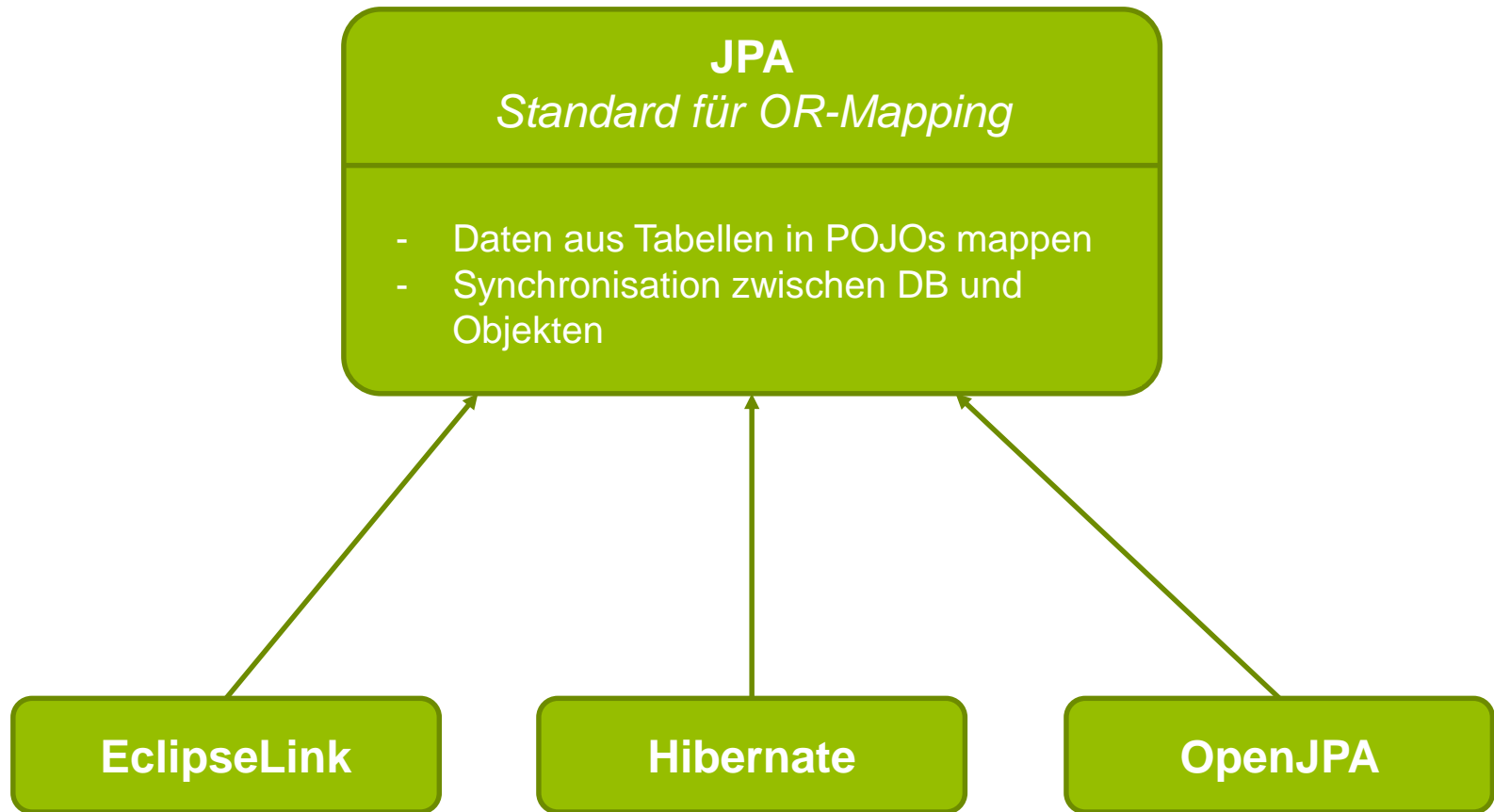


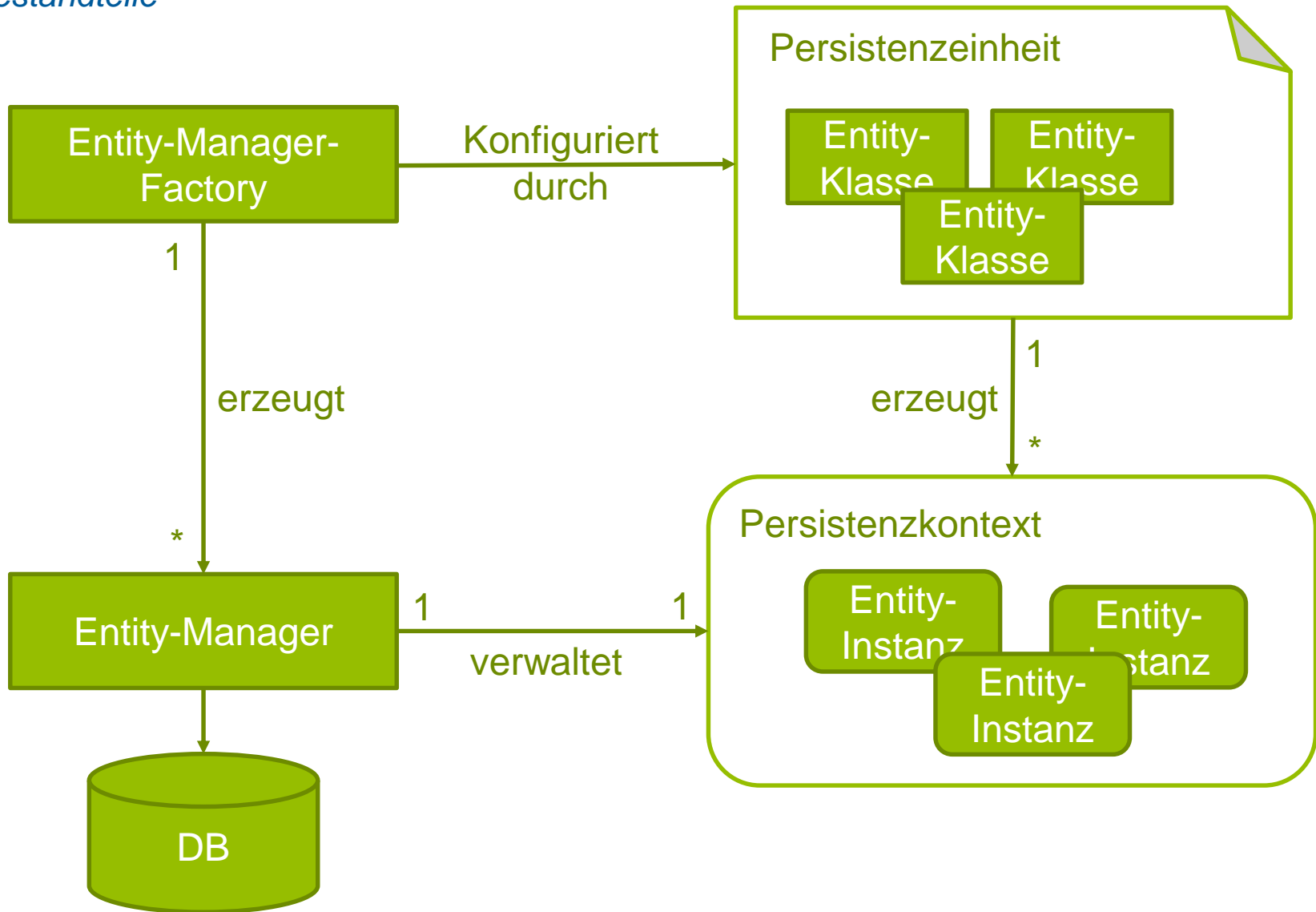
Problem:

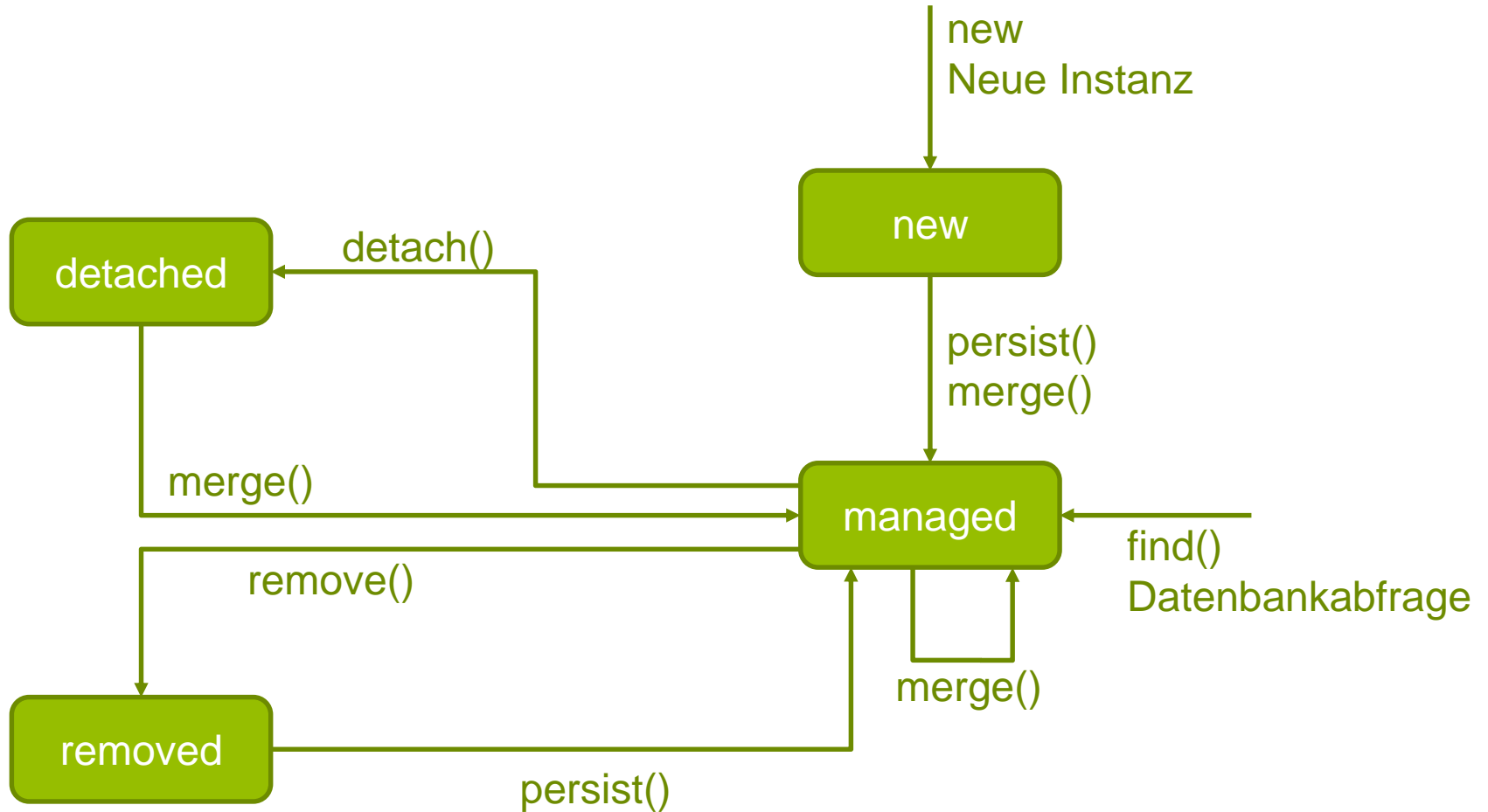


Lösung:

**Object
Relational
Mapping**









Annotation

@Entity

Klassenrumpf

```
public class Customer implements Serializable(){
```

id

```
@Id  
@GeneratedValue(strategy = GenerationType.AUTO)  
private int id;
```

Attribute

```
@Basic //Standardwerte  
private String firstname;  
private String lastname;  
@Transient // Wird nicht persistiert  
private String comment;
```

Leerer Konstruktor

```
public Customer(){  
}
```

Konstruktor

```
public Customer(String firstname, String lastname, String  
comment){  
    this.firstname = firstname;  
    this.lastname = lastname;  
    this.comment = comment;  
}
```



Getter-Methoden

```
public int getId(){
    return id;
}
//...
public String getFirstname(){
    return firstname;
}
```

Setter-Methoden

```
public void setFirstname(String firstname){
    this.firstname = firstname;
}
//...
public void setLastname(String lastname){
    this.lastname = lastname;
}
```

ToString-Methode

```
@Override
public String toString() {
    return "Customer [id=" + id + ", firstname=" + firstname +
        ", lastname=" + lastname + "]";
}
```




HashCode-Methode

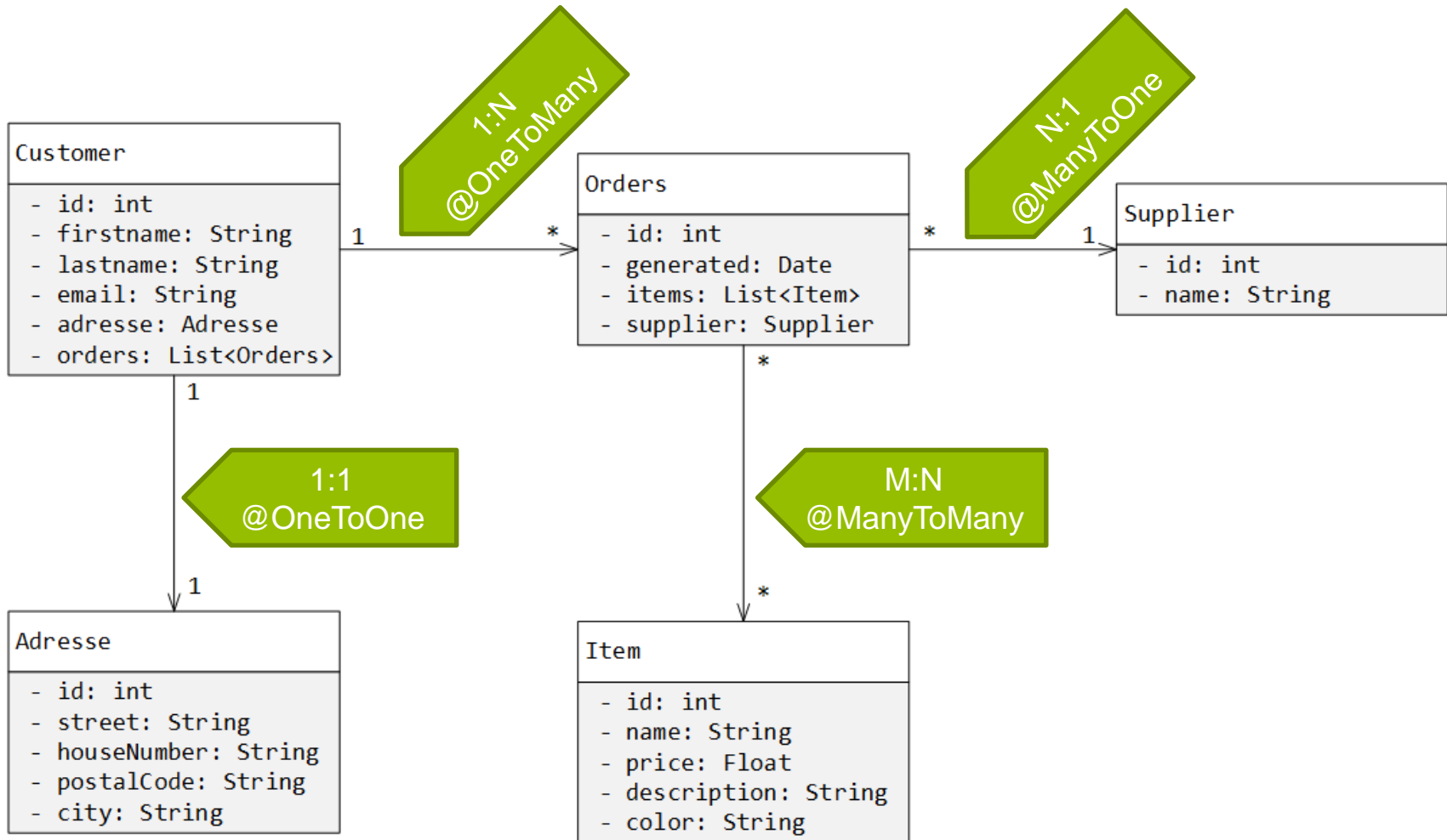
```
@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + id;
    return result;
}
```

Equals-Methode

```
@Override
public boolean equals(Object obj) {
    if (this == obj) {
        return true;
    }
    if (obj == null) {
        return false;
    }
    if (!(obj instanceof Customer)) {
        return false;
    }
    Customer other = (Customer) obj;
    if (id != other.id) {
        return false;
    }
    return true;
}
```

Java Persistence API

Entity-Beziehungen - unidirektional





@OneToOne

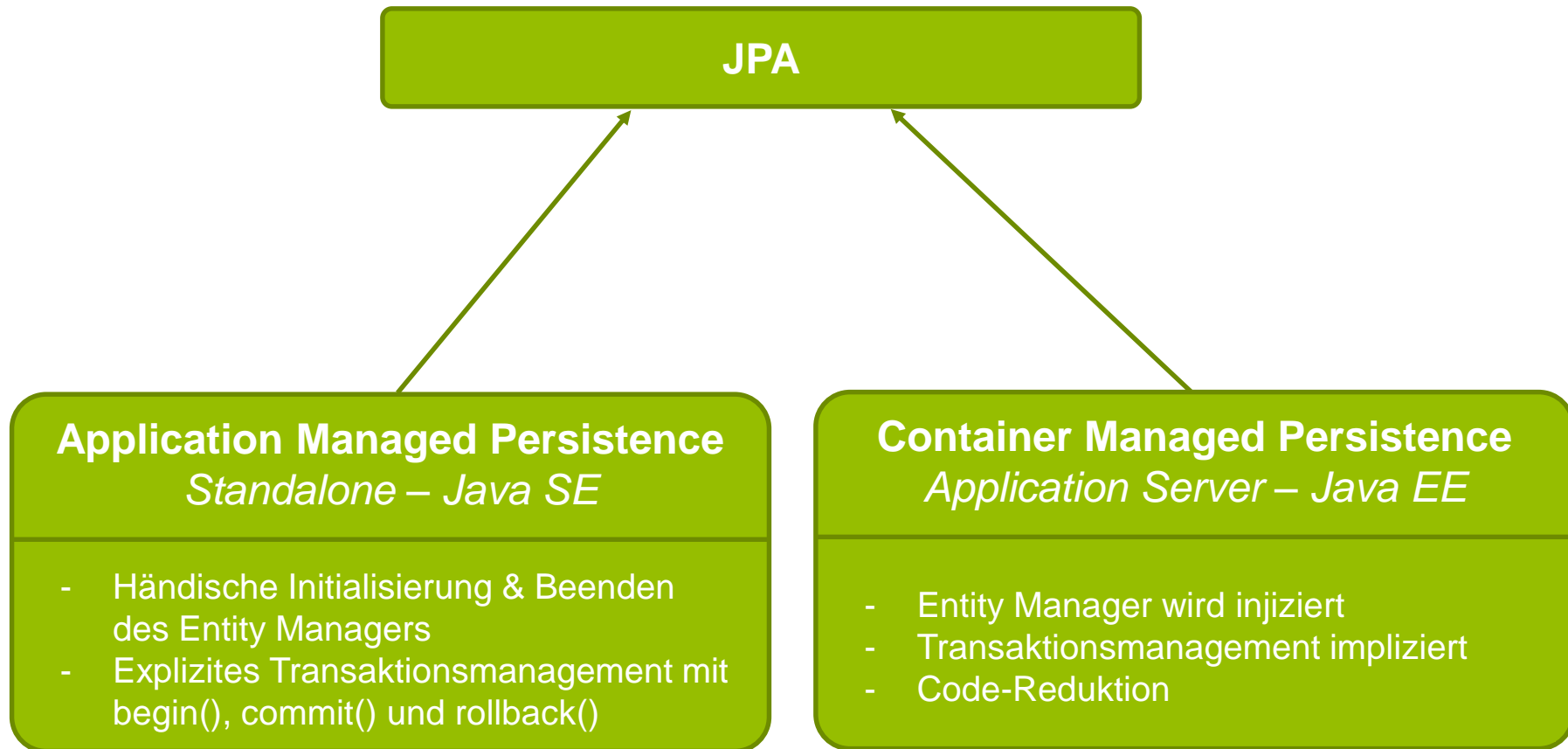
@OneToMany

```
@Entity
public class Customer implements Serializable(){
    //...
    @OneToOne
    private Adresse adresse;
    //...
    @OneToMany
    private List<Orders> orders;
    //...
}
```

@ManyToOne

@ManyToMany

```
@Entity
public class Orders implements Serializable(){
    //...
    @ManyToOne
    private Supplier supplier;
    //...
    @ManyToMany
    private List<Item> items;
}
```



EMF injizieren

```
@PersistenceUnit  
private EntityManagerFactory emf;
```

UserTransaction
injizieren

```
@Resource  
private UserTransaction utx;
```

EM erstellen

```
EntityManager em = emf.createEntityManager();
```

Datenbankzugriff
mit Transaktionen

```
Orders o = new Orders(new Date(...));  
try {  
    utx.begin();           // Transaktionsstart  
    em.persist(o);  
    utx.commit();          // Transaktion ausführen  
} catch (Exception e) {  
    try {  
        utx.rollback(); // im Fehlerfall Rollback  
    } catch (Exception e1) {  
        e.printStackTrace();  
    }  
    throw new ServletException(e.getMessage());  
}
```

EM beenden

```
finally {  
    em.close();  
}
```



Entity Manager
injizieren

```
@PersistenceContext  
private EntityManager em;
```

Datenbankzugriff

```
em.persist(order);
```



```
@PersistenceContext  
private EntityManager em;
```

CREATE

```
Customer customer = new Customer("Hans", "Peter", "");  
em.persist(customer);    //Speichern  
em.flush();              //Synchronisieren mit DB
```

READ

```
Int id = 3;  
Customer c = em.find(Customer.class, id);
```

UPDATE

```
customer.setFirstname("Gustav");  
em.merge(customer);
```

DELETE

```
em.remove(customer);
```



```
@PersistenceContext  
private EntityManager em;
```

SELECT all

```
String psql = "SELECT c FROM Customer c";  
TypedQuery<Customer> query = em.createQuery(psql, Customer.class);  
  
List<Customer> = query.getResultList();
```

WHERE

```
String fname = "Peter";  
String psql = "SELECT c FROM Customer c WHERE c.firstname = :fname";  
TypedQuery<Customer> query = em.createQuery(psql, Customer.class);  
  
query.setParameter("fname", fname);  
List<Customer> = query.getResultList();
```




Bidirektionale Beziehungen

<https://www.javaworld.com/article/2077819/java-se/understanding-jpa-part-2-relationships-the-jpa-way.html>

JPQL weiterführend

NamedQueries, NativeQueries, JOIN, Datenaggregation

Müller, Wehr (2012): Java Persistence API 2, Kapitel 7

Zusammengesetzte Primärschlüssel

Mit @EmbeddedId und @Embeddable Spaltenfelder auf Klassen verteilen

Müller, Wehr (2012): Java Persistence API 2, Kapitel 2

Laudon, Laudon, Schoder: *Wirtschaftsinformatik – Eine Einführung*. 3. Auflage

Abts, D. (2016): *Grundkurs Java. Von den Grundlagen bis zu Datenbank- und Netzanwendungen*. Wiesbaden : Springer Fachmedien

Adams, R. (2016): *SQL. Der Grundkurs für Ausbildung und Praxis. Mit Beispielen in MySQL/MariaDB*. Carl Hanser Verlag GmbH & Co. KG

Kleuker, S. (2013): *Grundkurs Datenbankentwicklung. Von der Anforderungsanalyse zur komplexen Datenbankabfrage*. Wiesbaden: Springer Fachmedien, 3. Auflage

Unterstein, Matthiessen (2013): *Anwendungsentwicklung mit Datenbanken*. Heidelberg: Springer, 5.Auflage

Sharan, K. (2018): *Java APIs, Extensions and Libraries. With JavaFX, JDBC, jmod, jlink, Networking, and the Process API*. New York: Springer Science + Business.

Müller, Wehr (2012): *Java Persistence API 2. Hibernate, EclipseLink, OpenJPA und Erweiterungen*. München: Carl Hanser Verlag.

Java Documentation: <https://docs.oracle.com/javase/tutorial/jdbc/overview/index.html>,
<https://docs.oracle.com/javase/tutorial/jdbc/basics/index.html>



Technische Hochschule
Ingolstadt

Fakultät für Elektrotechnik
und Informatik

*Zukunft in
Bewegung*

Maven

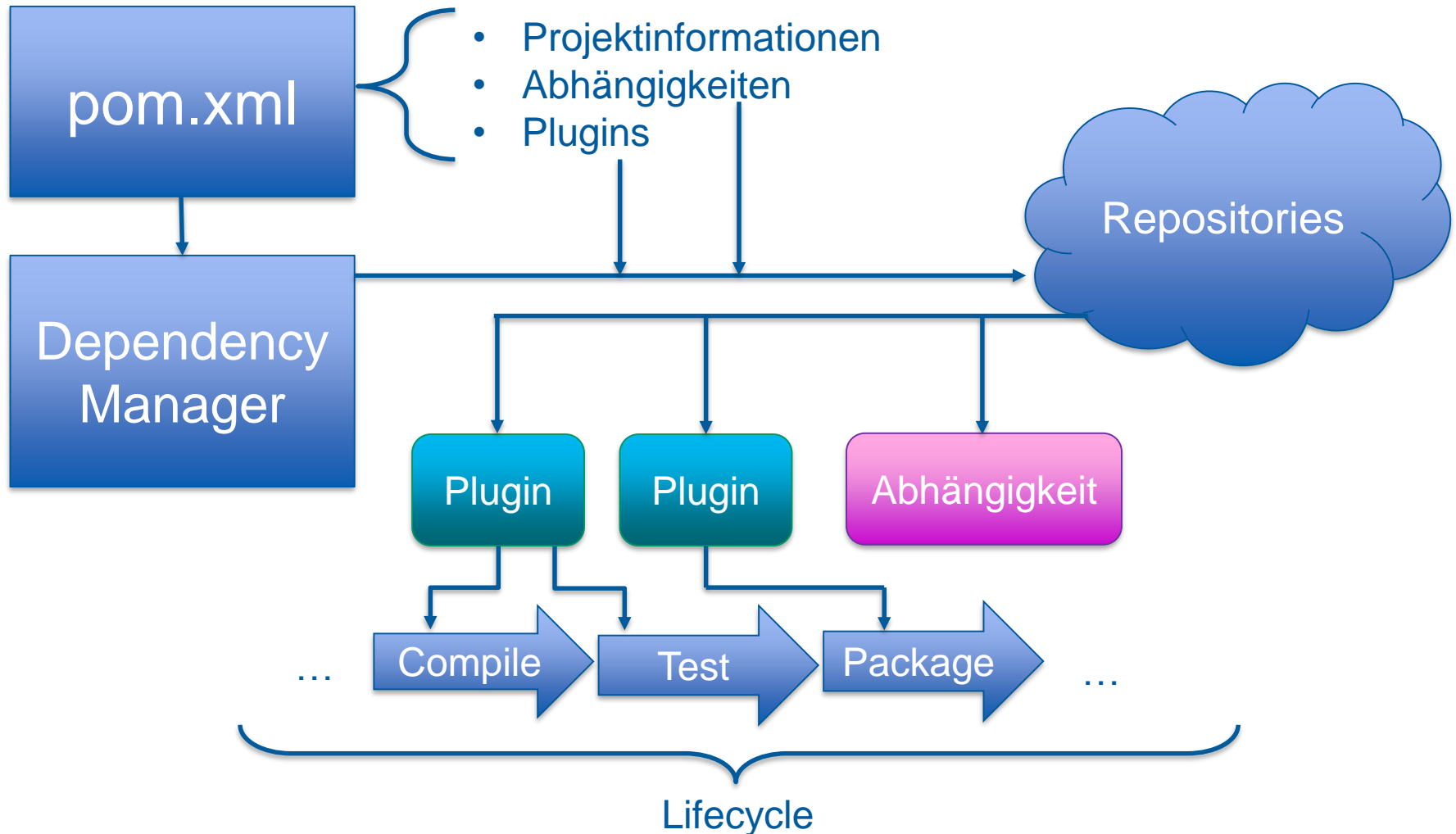
Paket- und Buildsystem

Michael Höpp 12.11.18



- Build Tool
- Abhängigkeits-Management Tool
- Projekt-Management Tool





pom.xml

- Projektinformationen
- Abhängigkeiten
- Plugins

```
<?xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
  <dependencies>
    <dependency>
      <!-- process engine, needs to be provided -->
      <groupId>org.camunda.bpm</groupId>
      <artifactId>camunda-engine</artifactId>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <!-- decision engine -->
      <groupId>org.camunda.bpm.dmn</groupId>
      <artifactId>camunda-engine-dmn</artifactId>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <!-- AssertJ Testing Library -->
      <groupId>org.camunda.bpm.extension</groupId>
      <artifactId>camunda-bpm-assert</artifactId>
      <version>1.2</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <repositories>
    <repository>
      <id>camunda-bpm-nexus</id>
      <name>Camunda Maven Repository</name>
      <url>https://app.camunda.com/nexus/content/groups/public</url>
    </repository>
    <!-- enable this for EE dependencies (requires credentials in ~/.m2/settings.xml) -->
    <repository>
      <id>camunda-bpm-nexus-ee</id>
      <name>Camunda Enterprise Maven Repository</name>
      <url>https://app.camunda.com/nexus/content/repositories/camunda-bpm-ee</url>
    </repository>
  </repositories>
  <build>
    <finalName>${project.artifactId}</finalName>
    <plugins>
      <plugin>
        <!-- Deploy to Tomcat using: mvn clean package antrun:run -->
        <!-- Follow the instructions in build.properties.example to make it work! -->
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-antrun-plugin</artifactId>
        <configuration>
          <tasks>
            <ant antfile="${basedir}/build.xml">
              <target name="copy.war.into.tomcat" />
            </ant>
          </tasks>
        </configuration>
      </plugin>
    </plugins>
  </build>
  <failOnMissingWebXml>false</failOnMissingWebXml>
</project>
```

pom.xml

- Projektinformationen
- Abhängigkeiten
- Plugins

Vereinfachte Bearbeitung via Eclipse integration:

Dependency Properties

Group Id: * log4j

Artifact Id: * log4j

Version:

Classifier:

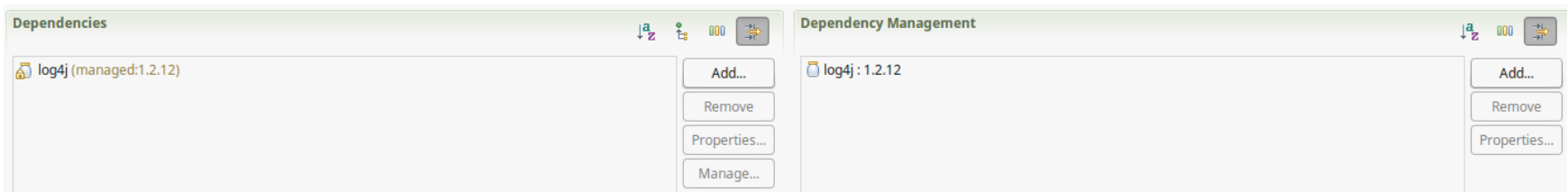
Type: jar

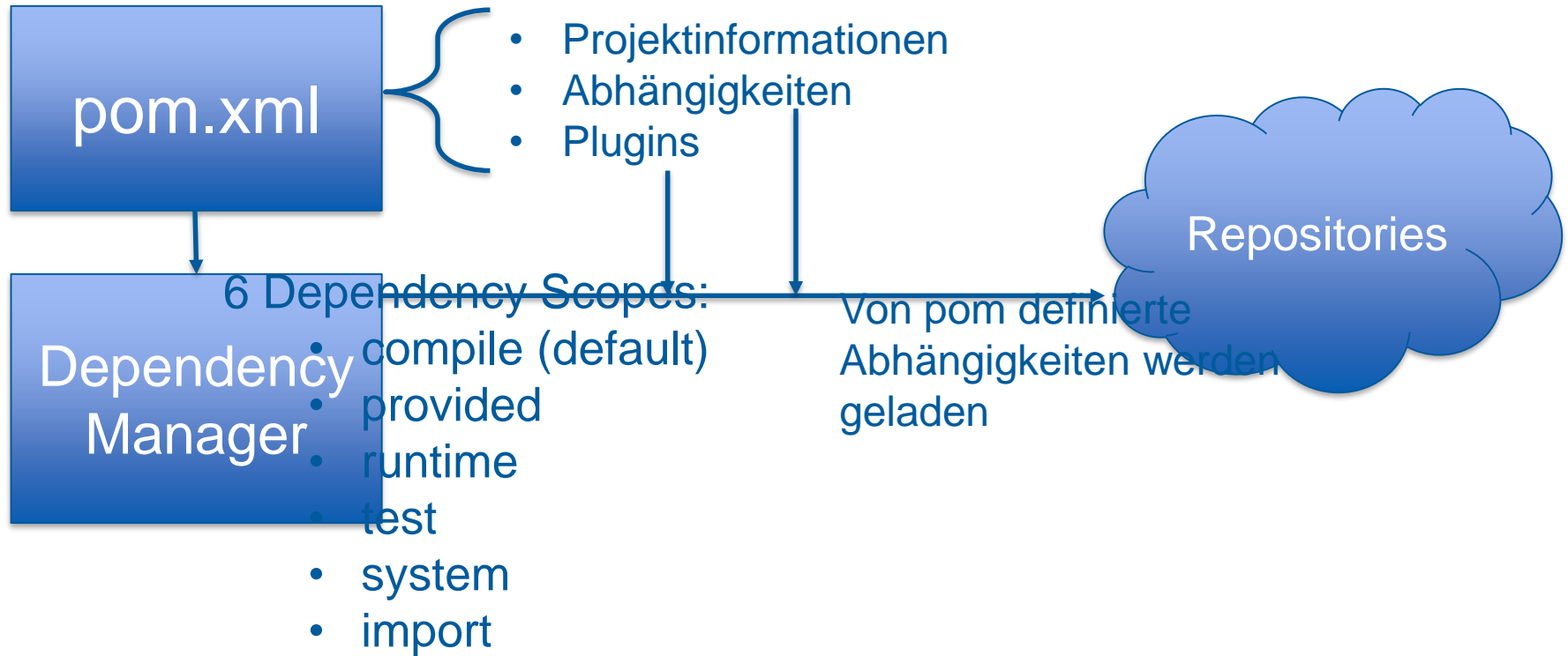
Scope: compile

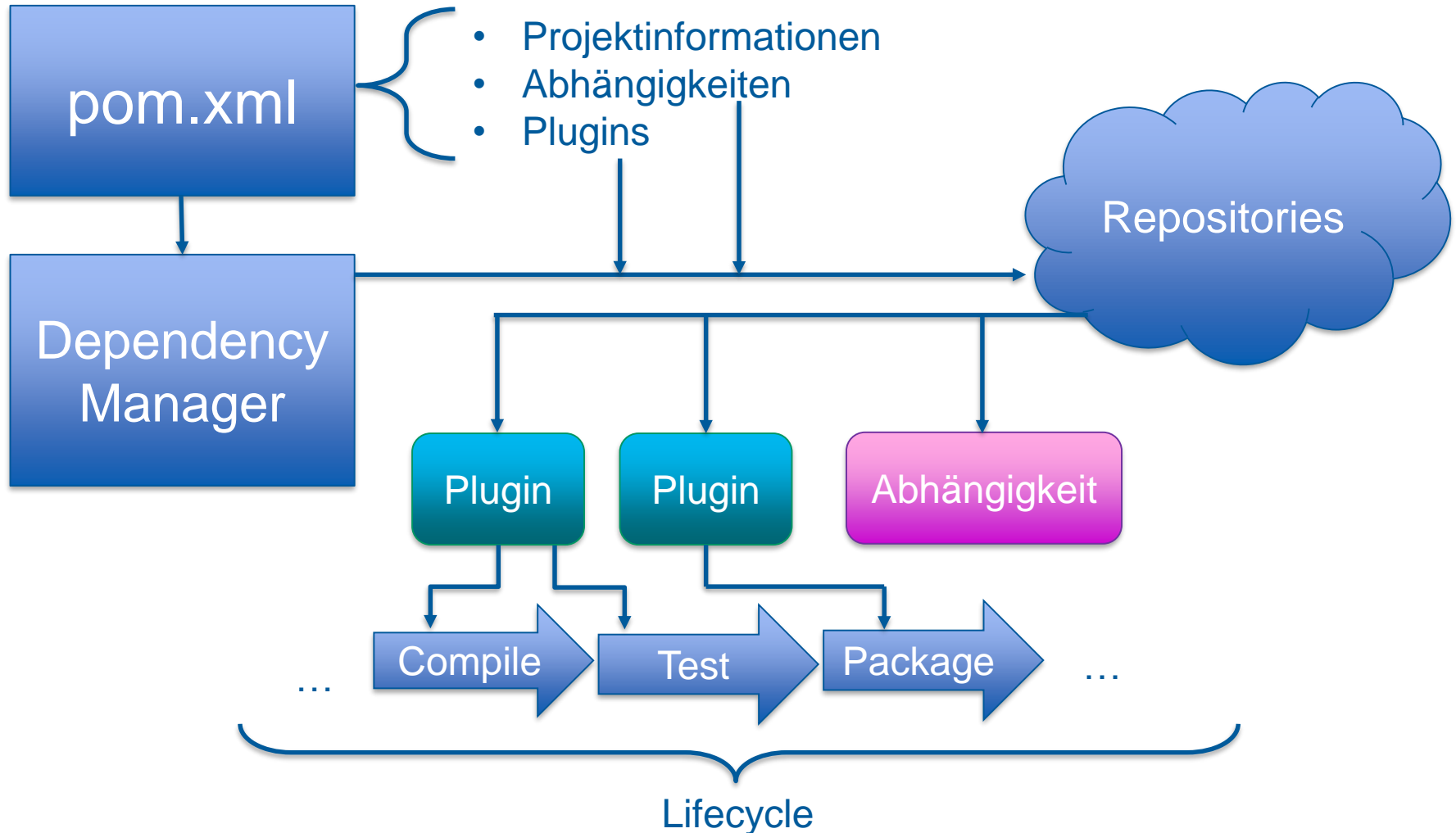
System Path:

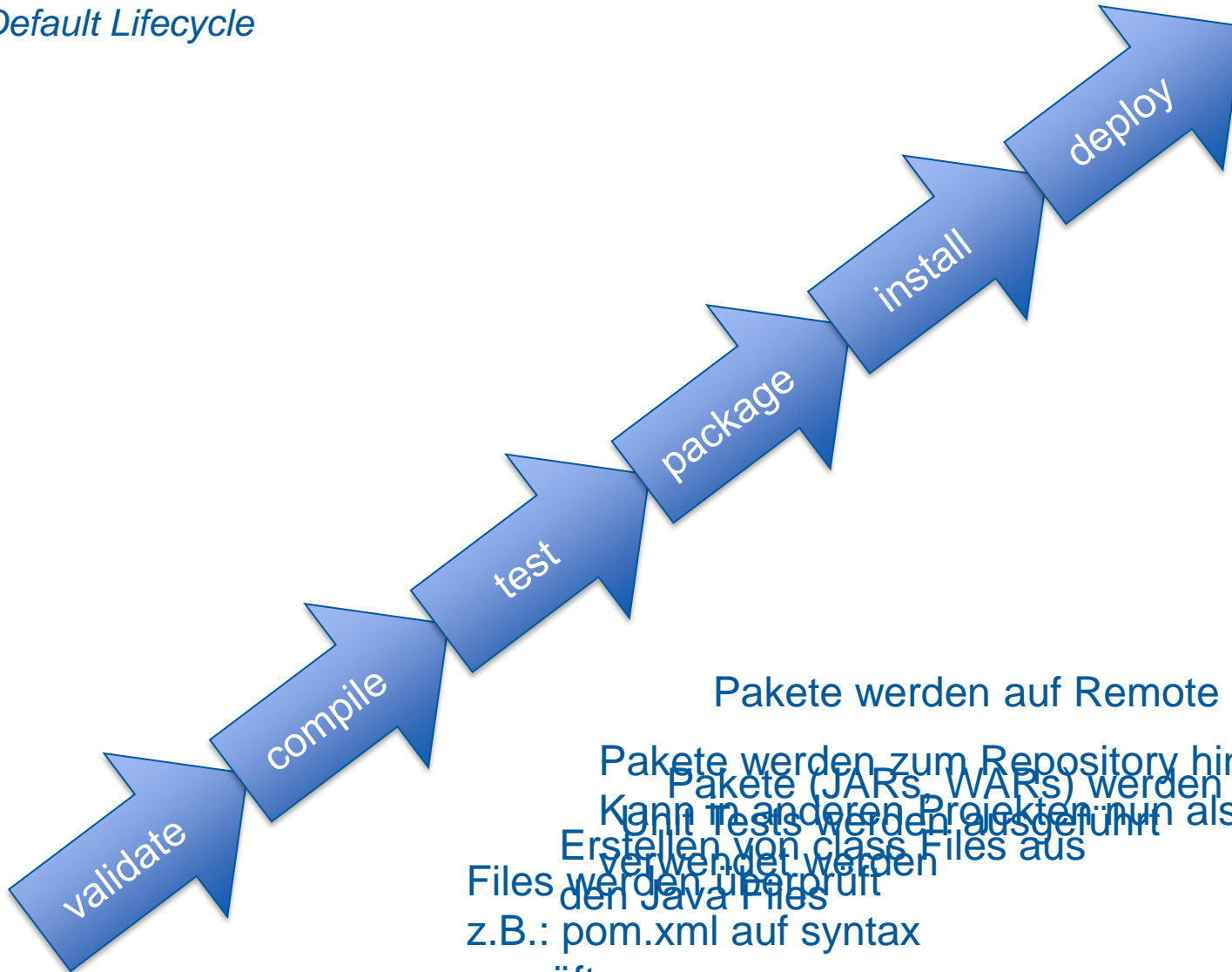
☐ Optional

Cancel OK







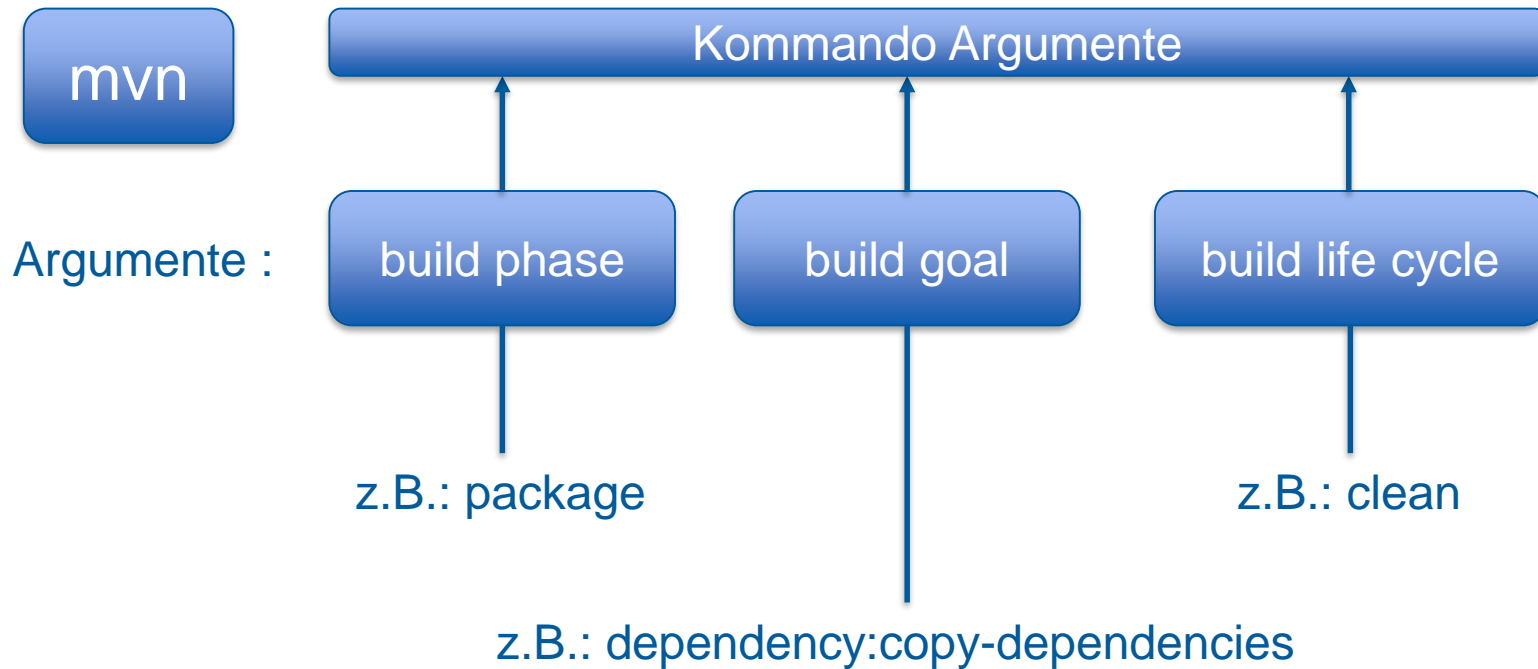


Pakete werden auf Remote kopiert

Pakete werden zum Repository hinzugefügt
Pakete (JARs, WARs) werden gebunden
Kann in anderen Projekten nun als Abhängigkeit
Erstellen von class Files aus
verwendet werden
Files werden überprüft
z.B.: pom.xml auf syntax
geprüft



- Clean Lifecycle:
 - pre-clean
 - clean
 - post-clean
- Default Lifecycle
- Site Lifecycle
 - pre-site
 - Site
 - post-site
 - site-deploy



Maven

Quellen



<https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html>

<https://maven.apache.org/what-is-maven.html>

<https://www.torsten-horn.de/techdocs/maven.htm>



Technische Hochschule
Ingolstadt

Fakultät für Elektrotechnik
und Informatik

*Zukunft in
Bewegung*

JUnit

Automatisiertes Testen

Michael Höpp 12.11.18





- Framework zum automatisierten Testen von Software
- Unit Tests sind spezielle, durch Annotationen markierte Methoden
- Einheiten (Methoden oder Klassen) sollen einzeln getestet werden
- JUnit ist standardmäßig in Eclipse integriert
- Validierung ob der Test fehlschlägt durch verschiedene assert Aufrufe

Mögliche Ergebnisse

Pass

ProgramTests [Runner: JUnit 4] (0,001 s)

- testGreaterThan_Greater (0,000 s)
- testGreaterThan_Equal (0,000 s)
- testGreaterThan_Lower (0,001 s)
- testGreaterThan_NegativeValues (0,000 s)

Fail

ProgramTests [Runner: JUnit 4] (0,000 s)

- testGreaterThan_Greater (0,000 s)
- testGreaterThan_Equal (0,000 s)
- testGreaterThan_Lower (0,000 s)
- testGreaterThan_NegativeValues (0,000 s)

Error

ProgramTests [Runner: JUnit 4] (0,021 s)

- testGreaterThan_Greater (0,019 s)
- testGreaterThan_Equal (0,001 s)
- testGreaterThan_Lower (0,000 s)
- testGreaterThan_NegativeValues (0,000 s)

Failure Trace

java.lang.AssertionError: expected:<false> but was:<true>
at ProgramTests.testGreaterThan_Greater(ProgramTests.java:11)

Failure Trace

java.lang.NullPointerException
at ProgramTests.testGreaterThan_Greater(ProgramTests.java:13)



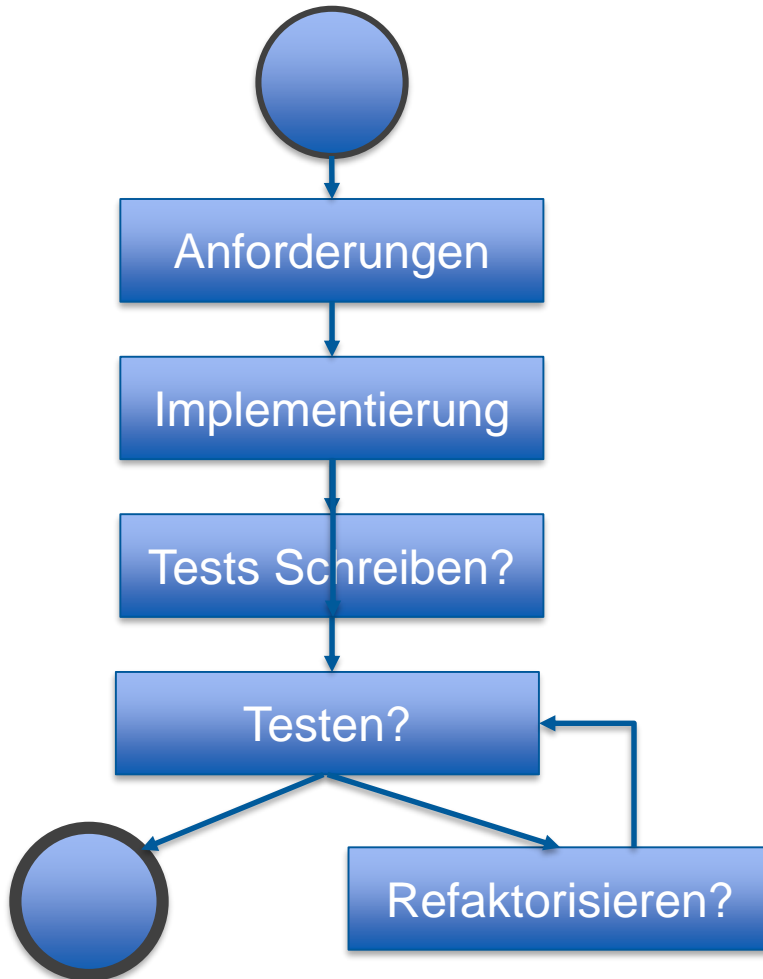
Methodenname	Parameter	Beschreibung
assertEquals	Object, Object	Erwartet, dass die Objekte gleich (Equal Operator) sind
assertEquals	double, double, double	Erwartet, dass die ersten beiden double weniger als der dritte double abweichen
assertArrayEquals	Object[], Object[]	Erwartet, dass die Elemente in den beiden Arrays jeweils Equal sind
assertNull	Object	Erwartet einen Null Wert
assertSame	Object, Object	Erwartet, dass die Objekte die selben sind (selbe Referenz)
assertNotSame	Object, Object	Erwartet unterschiedliche Objekte
assertThat	T, Matcher<T>	Nutzt einen benutzerdefinierten Matcher für das Objekt der Klasse T
assertFalse	boolean	Erwartet, dass der boolean false ist
assertTrue	boolean	Erwartet, dass der boolean true ist



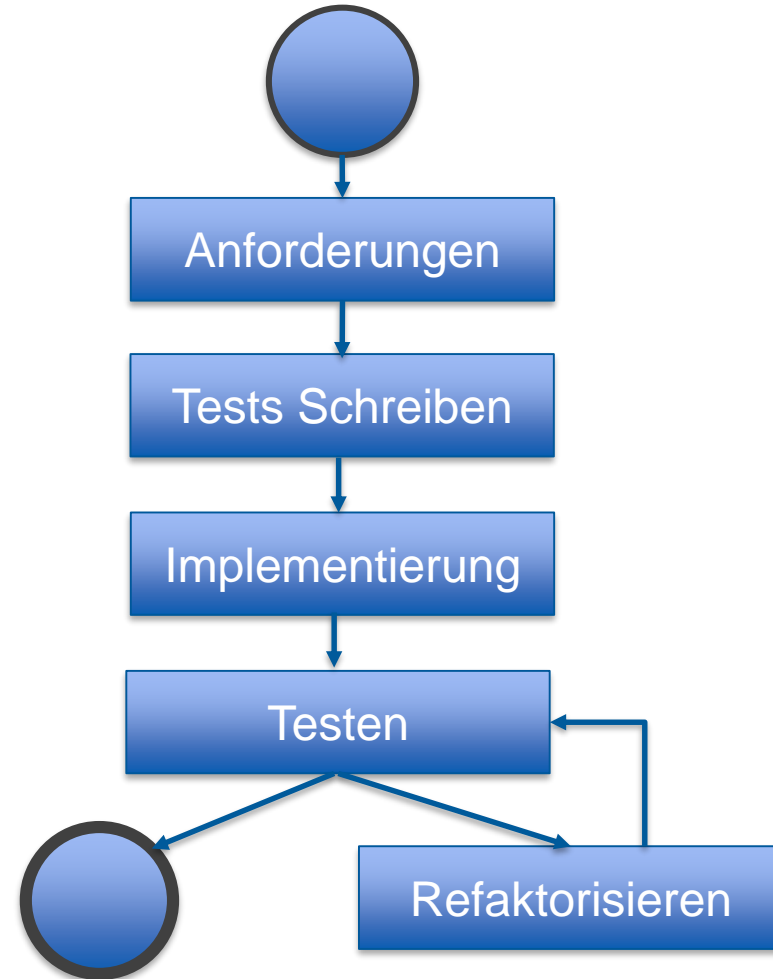
```
@Test
public void testGreaterThan()
{
    Program prog = new Program();
    assertEquals(prog.GreaterThan(2,1),true);
    assertFalse(prog.GreaterThan(2,2));
    assertTrue(prog.GreaterThan(-1,-2));
    if(prog.GreaterThan(2,500))
    { fail(); }
}
```

- Fail() ermöglicht einen sofortigen Fehlschlag des Tests
- Asserts können mit einem zusätzlichen String Parameter aufgerufen werden -> Nachricht wird bei Fail angezeigt

Traditional Development



TDD





- Anforderungen werden als Tests formuliert
- Code wird geschrieben um die Tests (und somit Anforderungen) zu erfüllen
- Nachprüfbar welche Anforderungen erfüllt wurden
- KEINE Garantie alle Fehler zu finden
- Potenzielle Fehlerfälle müssen erkannt werden und Tests dafür geschrieben werden



- Unit Tests sollen isoliert sein
- Unit Tests sollen deterministisch sein (bei gleichen Bedingungen gleiches Ergebnis)
- Unit Tests sollen sprechend benannt sein und intuitiv verstehbares Feedback liefern
- Für verschiedene Testfälle sollen neue Testmethoden erstellt werden
- Unit Tests sollen Ergebnisse testen, nicht die Implementierung
- Überspezifizierung sollte vermieden werden
- Unit Tests sollen möglichst wenig Abhängigkeiten haben
- Artikel hierzu: <https://esj.com/Articles/2012/09/24/Better-Unit-Testing.aspx>

JUnit

Quellen



<https://junit.org/junit4/javadoc/4.8/>

<https://esj.com/Articles/2012/09/24/Better-Unit-Testing.aspx>

<https://www.it-agile.de/wissen/agiles-engineering/testgetriebene-entwicklung-tdd/>

<http://www.vogella.com/tutorials/JUnit/article.html>