

# Parallele Programmierung

---

7. Actors-Modell und Akka-Framework

# Überblick (Woche 1)

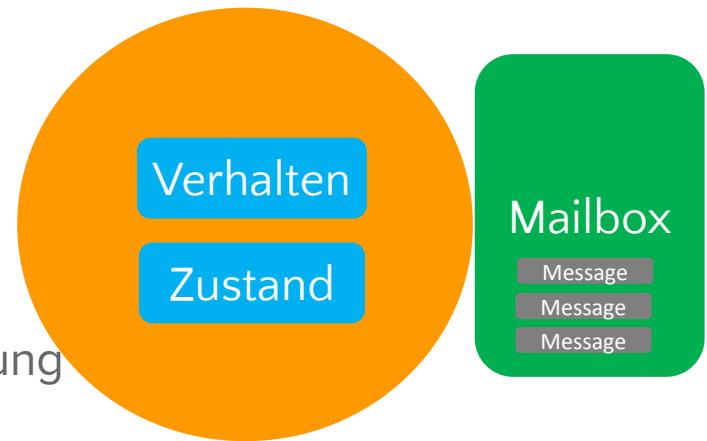
- Actor-Modell
  - Actors, Nachrichten, Nachrichtenverarbeitung
- Actors überwachen Actors
  - “let it crash” statt Exception Handling, Supervisors, Error-Kernel Design Pattern
- Akka als Umsetzung des Actor-Modells
  - Actor programmieren, der auf Nachrichten reagieren kann; Actors erzeugen
  - Hierarchie von Actors, Nachrichten versenden, Monitoring (Supervisor-Actor)
- Router
- Beispiel mit 3 Actor-Typen und 4 unterschiedlichen Nachrichtenarten
- Maven als build Tool
- Konfiguration
- Remote-Actor (Aktoren verteilt im Netzwerk)
- Router-Pool und -Group

# **Actor-Modell**

# Actor-Modell

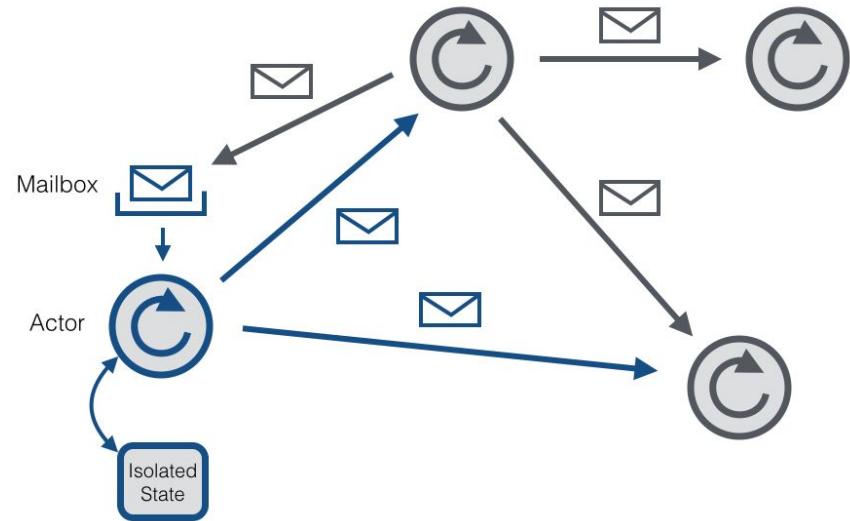
C. Hewitt, P. Bishop und R. Steiger (1973)

- im Prinzip nebenläufige Objekte
- eigener Zustand, eigenes Verhalten
- Kommunikation mit anderen Actors durch Senden von Nachrichten (Methodenaufruf)
- Nachrichtenzustellung durch Ablaufumgebung sichergestellt



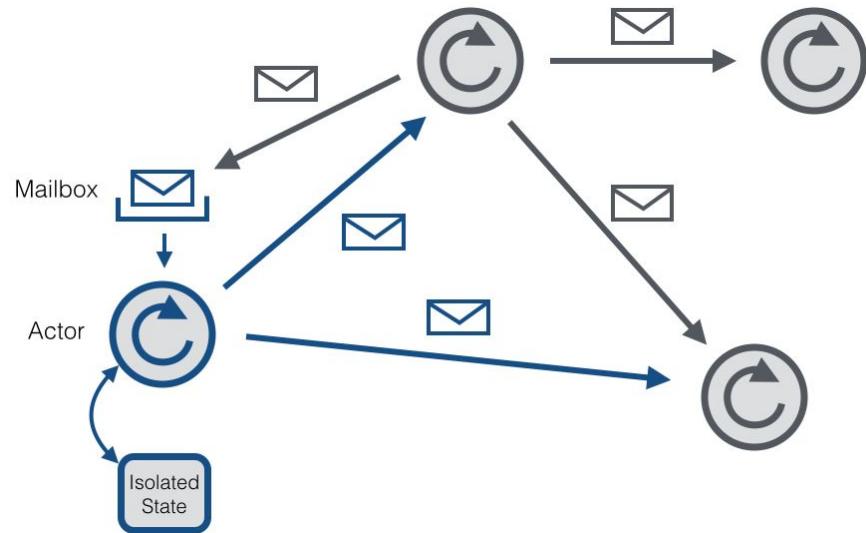
# Actors

- kein geteilter Zustand zwischen Actors
  - keine Synchronisierung erforderlich
- jeder Actor hat eigene Mailbox
  - eingehende Nachrichten
  - Nachrichten werden sequentiell bearbeitet
- Nachrichten werden asynchron zugestellt
- ein Actor kann neue Actors erzeugen
  - Verteilung von Teilaufgaben



# Nachrichtenverarbeitung

- Actors kann Nachricht nehmen und verarbeiten (verbrauchen)
  - oder unverändert weiterleiten
- 
- Alternativ: Verarbeitung = neue Nachricht(en) erzeugen und weiterleiten
  - Actor kann Nachrichten an sich selber senden

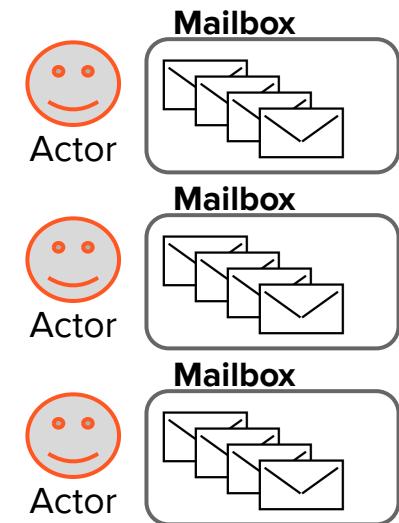
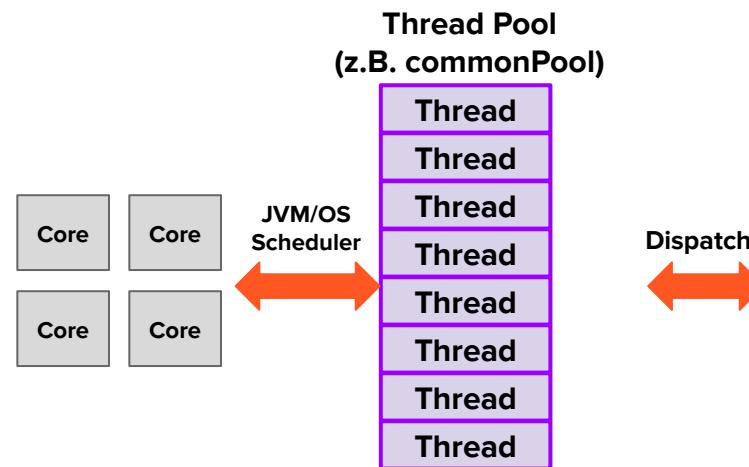


# Nachrichten

- Actors interagieren durch Nachrichtenaustausch
- Nachrichten werden asynchron zugestellt
  - empfänger Actor blockiert nicht bis zum Empfang
- Mailbox != Channel
  - mehrere Produzenten für eine Mailbox möglich
  - ein Empfänger behandelt die Nachrichten mehrere anderer Actors, die in seiner Mailbox ankommen, synchron
  - “put” und “take” sind atomare Operationen
- Nachrichten *immutable*
  - Actors kann Nachricht nehmen und verarbeiten (verbrauchen)
  - oder unverändert weiterleiten
  - Alternativ: Verarbeitung = neue Nachricht(en) erzeugen und weiterleiten
- Actor kann Nachrichten an sich selber senden

# Actors != Threads

- Die Anzahl der Actors kann größer sein, als die Zahl der sinnvoll zu behandelnden Threads (wie bei Fibers, Go-Coroutinen etc.)
- Es gibt eine Komponente im Actor-System (“Dispatcher”), die eine Zuordnung von Actor zu Thread (z.B. aus einem Thread-Pool) übernimmt.
- Die Verarbeitung einer Nachricht aus der Mailbox eines Actors wird vom Dispatcher ausgewählt und in einem Thread ausgeführt



**Actors  
überwachen  
Actors**

# Actor-Programme und Fehler

Actoren überwachen andere Actoren (solche “Überwacher” heißen supersivor).

Statt *defensive programming* oder Exception Handling: “**let it crash**”:

- Worker (Actor, der konkrete Anwendungsaufgabe umsetzt) ist nur für Verarbeitung korrekter Eingabeparameter gemacht.
- Fehlerbehandlung wird in den Supervisor verschoben.

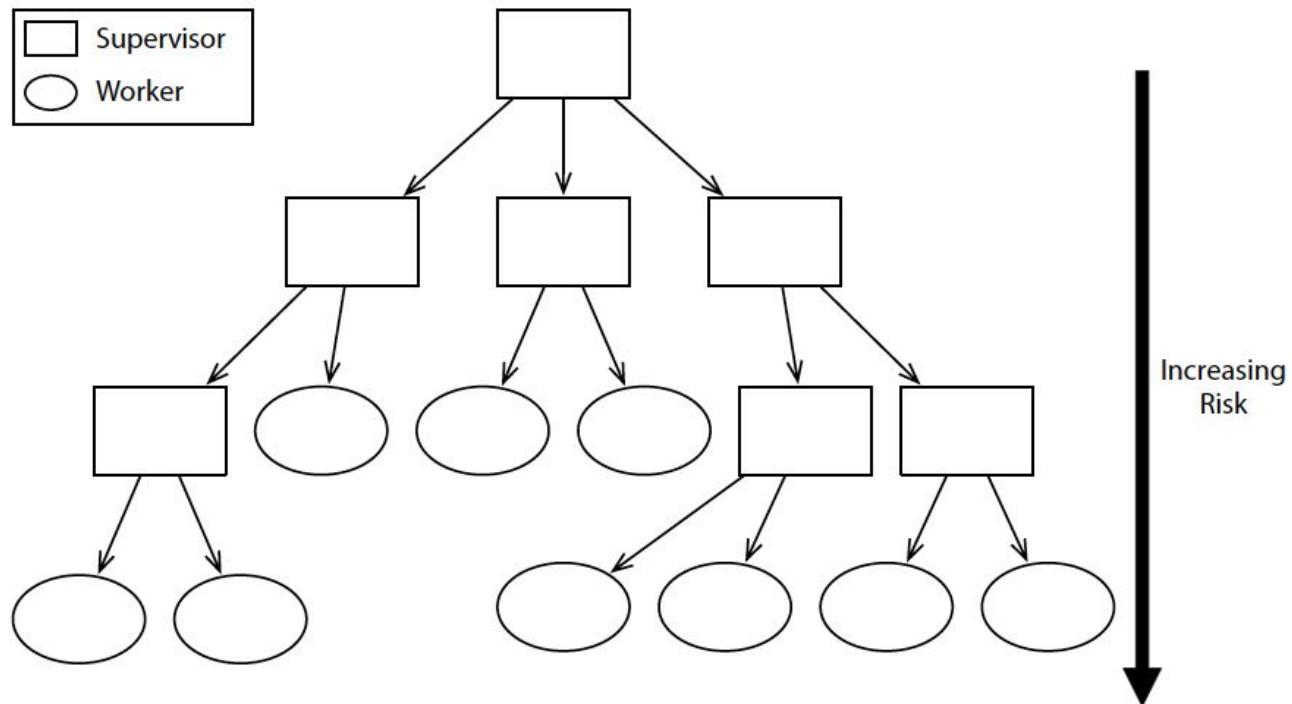
## Vorteile:

- Programme sind einfacher (Fehlerbehandlung ausgelagert).
- Da Actors getrennt sind und keine Variablen teilen, besteht keine Gefahr, dass mehr als ein Programmteil abstürzt (Worker stürzt ab, Supervisor bleibt).
- Supervisor kann weitergehende Fehlerbehandlungsstrategien auch für unvorhergesehene Situationen implementieren und zentrale Protokollierung vornehmen.

# Hierarchie von Error Kernels

Actors (Supervisor)  
überwachen Actors  
(Worker oder  
Supervisor)

- sie werden über Abstürze informiert
- Ersatz für Exception Handling
- sehr robust (Whatsapp-Architektur)



# Error Kernel Design Pattern

Der Teil eines Programms der absolut korrekt sein muss, da er andere (potenziell unsicherere) Teile überwacht und ggf. neu startet.

Man sollte bestrebt sein, den Error Kernel zu minimieren, damit wenigstens dort Fehlerfreiheit garantiert werden kann.

Im Actor-Paradigma sollte jeder Prozess der einen Anwendungszweck hat (Worker) einen übergeordneten Supervisor-Prozess haben, der als dessen Error-Kernel fungiert.

Es spricht nichts dagegen, auch komplexere Supervisor-Prozesse von einfacheren Error-Kernel-beinhaltenden Meta-Supervisor-Prozessen überwachen zu lassen.

# Design Prinzipien für Actors

- Ein Actor sollte nur eine klar abgegrenzte Aufgabe haben (“**single responsibility**”-Prinzip)
- **Spezifische Supervisor:** Ein Supervisor sollte nur genau eine Art von Worker überwachen.
- **Einfacher Error Kernel:** Die Wurzel in der Überwachungshierarchie sollte so einfach wie möglich sein, um Fehler zu vermeiden, die nicht abgefangen werden können.
- **Fehler-Zonen:** Fehler sollten sich nur auf einen abgegrenzten Teil der Hierarchie von Supervisors/Workers auswirken.

# Akka (Java I/F)

# Auf Nachrichten reagieren

Folie mit  
Anmerkungen

```
public class DemoMessagesActor
    extends AbstractLoggingActor {

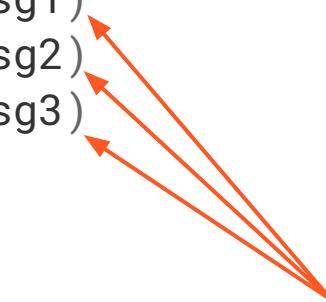
    @Override
    public Receive createReceive() {
        return receiveBuilder()
            .match(Greeting.class, message -> {
                log().info("I was greeted by {}", message.getGreeter());
            })
            .build();
    }
}
```

```
public class Greeting {
    private final String from;
    public Greeting(String from) {
        this.from = from;
    }
    public String getGreeter() {
        return from;
    }
}
```

hier als Lambda-Ausdruck (Java 8)

# Nachrichtenempfang (untersch. Typen)

```
public class NewActor extends AbstractActor {  
    @Override  
    public Receive createReceive() {  
        return receiveBuilder()  
            .match(Msg1.class, this::receiveMsg1)  
            .match(Msg2.class, this::receiveMsg2)  
            .match(Msg3.class, this::receiveMsg3)  
            .build();  
    }  
    private void receiveMsg1(Msg1 msg) {...}  
    private void receiveMsg2(Msg2 msg) {...}  
    private void receiveMsg3(Msg3 msg) {...}  
}
```



hier als Funktionsreferenz (Java 8)  
(könnte auch mit Lambda-Ausdrücken  
und anonymen inneren Klassen  
gemischt werden)

# Actor programmatisch erzeugen

Folie mit Anmerkungen

Dienen der Spezifikation und Konfiguration von Actors.

```
final ActorSystem system = ActorSystem.create();
```

```
ActorRef myActor =
```

```
    system.actorOf(Props.create(MasterActor.class), "master");
```

Basis-Typ für Actor in Akka

“Eltern”-Kontext für den neuen Actor

Implementierungsklasse für diesen Actor

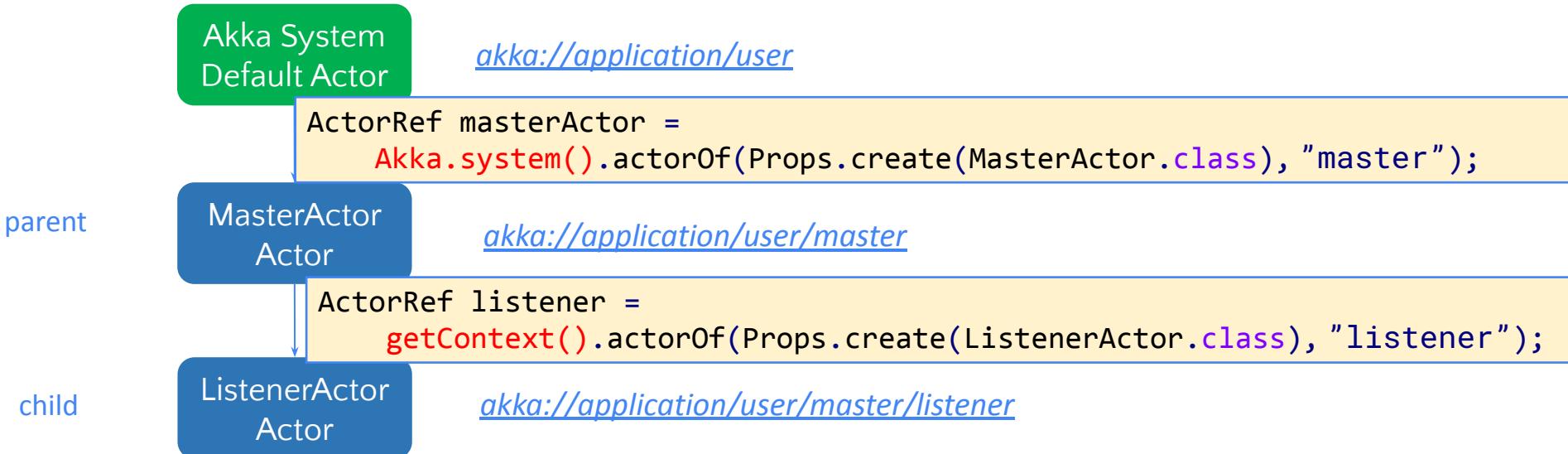
Akka-interner Name, mit dem der Actor identifiziert werden kann

**Kind-Actor in einem Eltern-Actor (getContext()) erzeugen:**

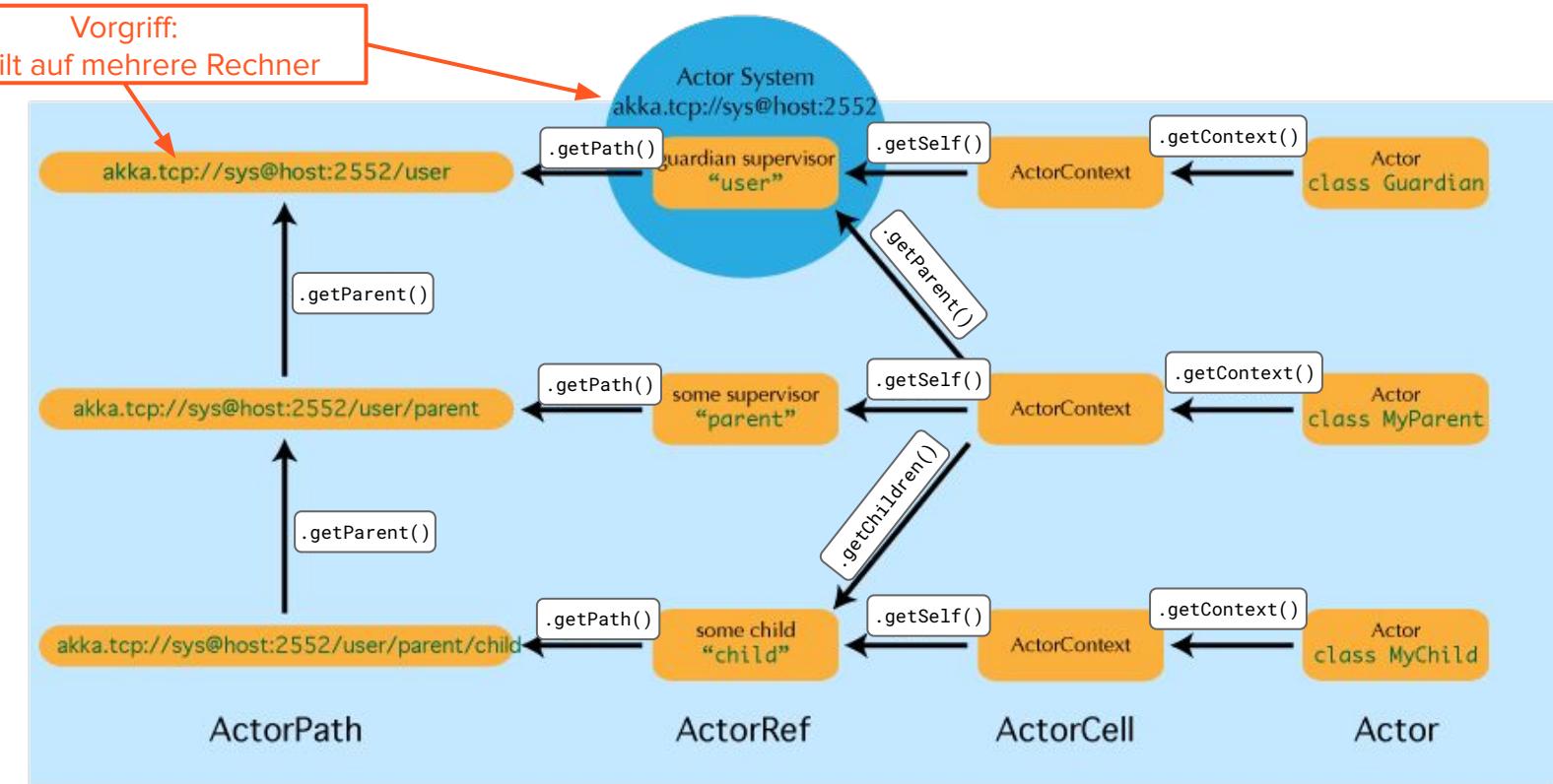
```
ActorRef listener =
```

```
    getContext().actorOf(Props.create(ListenerActor.class), "listener");
```

# Actor-Hierarchie 1/2



# Actor-Hierarchie 2/2



# Nachrichtenversand

```
target.tell("Hello", getSelf());  
getSender().tell(new Messages.ResultMsg(result), getSelf());
```

```
import static akka.pattern.PatternsCS.ask;
```

```
CompletableFuture<Object> future1 = ask(actorA, "request", 1000);  
CompletableFuture<Object> future2 = ask(actorB, "another request", 1000);  
CompletableFuture<Result> transformed =  
CompletableFuture.allOf(future1, future2).thenApply(v -> {  
    String x = (String) future1.join();  
    String s = (String) future2.join();  
    return new Result(x, s);  
});
```

getSender(): welcher Actor hat die letzte Nachricht gesendet?

timeout: 1000ms

Niemals ein CompletableFuture als Nachricht versenden: Wenn Actors auf unterschiedlichen Rechnern sind, müsste das Promise zum Versand serialisiert werden, was nicht funktioniert.

# Design Prinzipien für Actors

- **Fire-and-Forget statt Request-Reply:** Nachrichten sollten tendenziell eher nicht auf Aufforderung versendet werden, sondern aufgrund eines Ereignisses/situativen Zustands.
- **tell vor ask bevorzugen:** mit tell kann genauso ein Request-Reply Kommunikationsmuster umgesetzt werden: In vielen Fällen ist es nicht erforderlich eine Korrelation zwischen Anfrage und Antwort herzustellen.
  - besser: Frage in Antwort wiederholen (bzw. wesentliche Elemente daraus), dann kann der Empfänger sie unabhängig von seiner Anfrage interpretieren
  - ermöglicht auch das Antworten ans andere Aktoren als den ursprünglichen Anfrager

# Monitoring

```
public class WatchActor extends AbstractActor {  
    ActorRef child = getContext().actorOf(Props.create(ListenerActor.class), "listener");  
    ActorRef lastSender;  
    public WatchActor() {  
        getContext().watch(this.child);  
    }  
    @Override  
    public Receive createReceive() {  
        return receiveBuilder()  
            .match(PoisonPill.class, (msg) -> {  
                getContext().stop(this.child);  
                this.lastSender = getSender(); })  
            .match(Terminated.class, (msg) -> {  
                if (msg.getActor() == this.child) {  
                    this.lastSender.tell("finished", null);  
                } })  
            .build();  
    }  
}
```

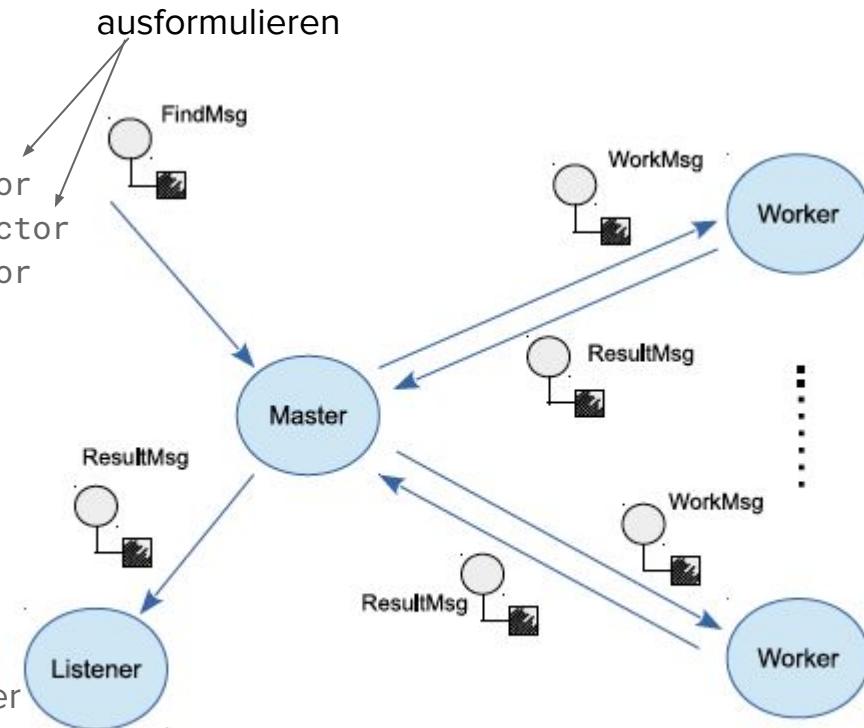
The annotations highlight the following parts of the code:

- A red box surrounds `getContext().actorOf(Props.create(ListenerActor.class), "listener");`. An arrow points from this box to the text: ““listener” wird als Kind von WatchActor erzeugt” (‘listener’ is created as a child of WatchActor).
- A red box surrounds `getContext().watch(this.child);`. An arrow points from this box to the text: “WatchActor wird Supervisor von “listener”” (WatchActor becomes supervisor of “listener”).
- A red box surrounds `Terminated.class` in the `.match(Terminated.class, (msg) -> {` line. An arrow points from this box to the text: “Supervisors werden über Ende von überwachten Actors informiert” (Supervisors are informed about the end of monitored actors).

# Akka-Beispiel

# Systemarchitektur

- pp.07.01-AkkaFindWords
- 3 Actor-Typen:
  - io.dama ffi.actors.find.MasterActor
  - io.dama ffi.actors.find.ListenerActor
  - io.dama ffi.actors.find.WorkerActor
- 4 Message-Typen:  
`io.dama.ffi.actors.find.messages.*`
  - PleaseCleanupAndStop
  - FindMsg
  - WorkMsg
  - ResultMsg
- Master und Listener genau einmal
- “viele” Worker (1 Worker pro Datei)
- Master erzeugt Worker und Listener als Kinder und überwacht sie.



# Router

# Router

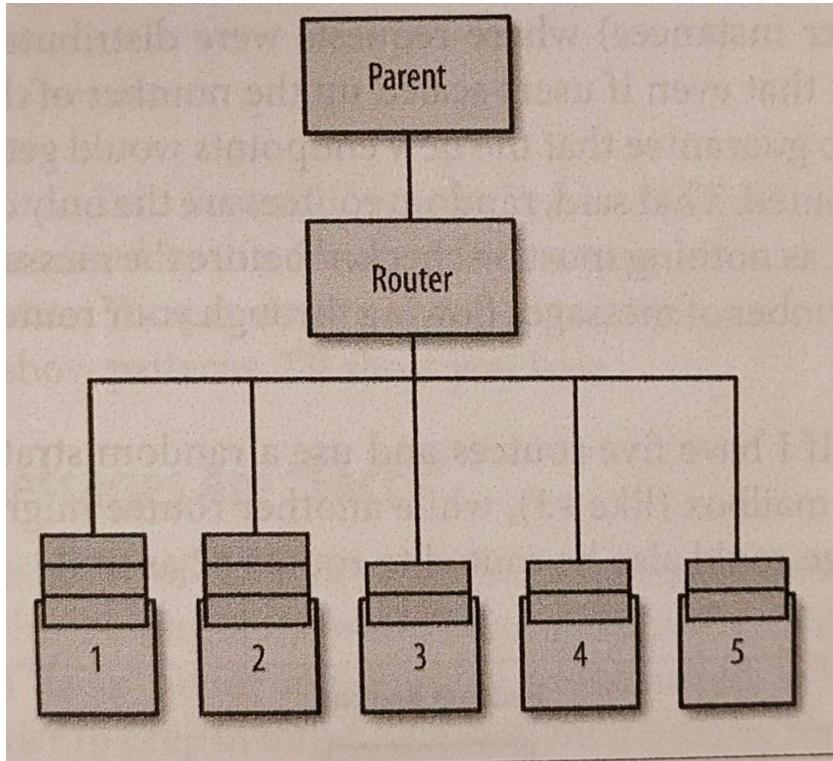
- Ein **Router** ist ein Actor-ähnliches Konzept. Es hat aber keinen eigenen Zustand und Verhalten sondern dient als *Eingang* für eine Reihe nachgeordneter Actors.
- Die nachgeordneten Actors (“**Routees**”) sind in der Regel gleichartig (z.B. von derselben Klasse) und verarbeiten prinzipiell dieselben Nachrichtentypen.
  - Der Router kennt seine Routees.
  - Eine Nachricht eines anderen Actors an einen Routee wird nicht direkt geschickt, sondern über den Router (“**route**”).
  - Der Router ändert die Nachricht nicht, der Absender bleibt der ursprüngliche Sender.
  - Der Router entscheidet aufgrund einer eigenen **Entscheidungslogik**, an welchen Routee die Nachricht weitergeleitet werden soll.
  - Der Router dient als als “**Load-Balancer**” für die Routees.
  - Wenn ein Routee eine Nachricht versendet, sollte der Router als Absender gesetzt werden.

# Router-Verwendung

```
var routees = new ArrayList<Routee>();
for (var i = 0; i < 5; i++) {
    var r = getContext().actorOf(Props.create(Worker.class));
    getContext().watch(r);
    routees.add(new ActorRefRoutee(r));
}
var router = new Router(new RoundRobinRoutingLogic(), routees);

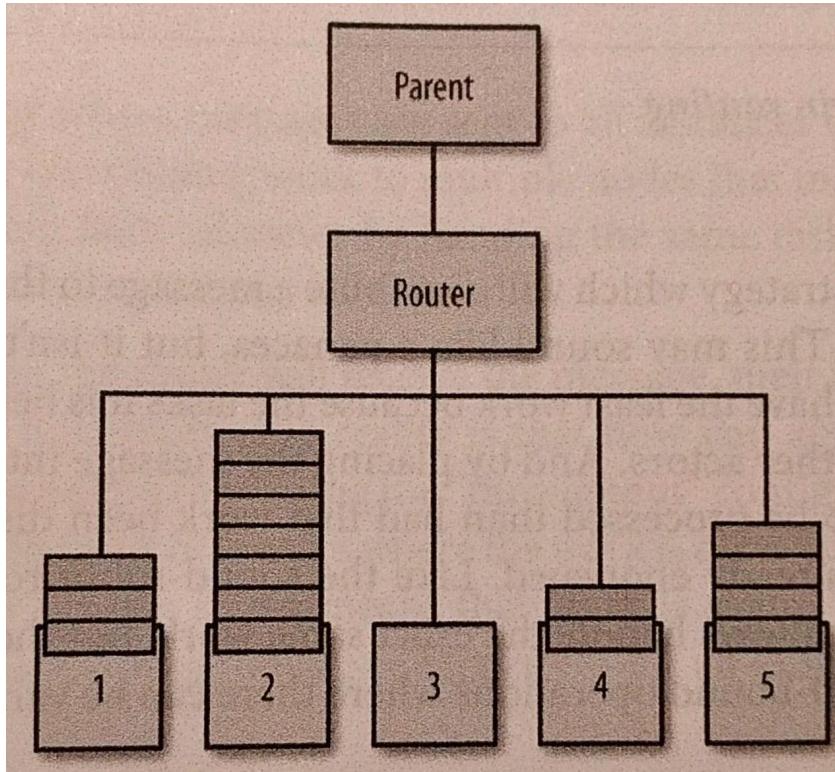
// ...
router.route(message, getSender());
```

# Router



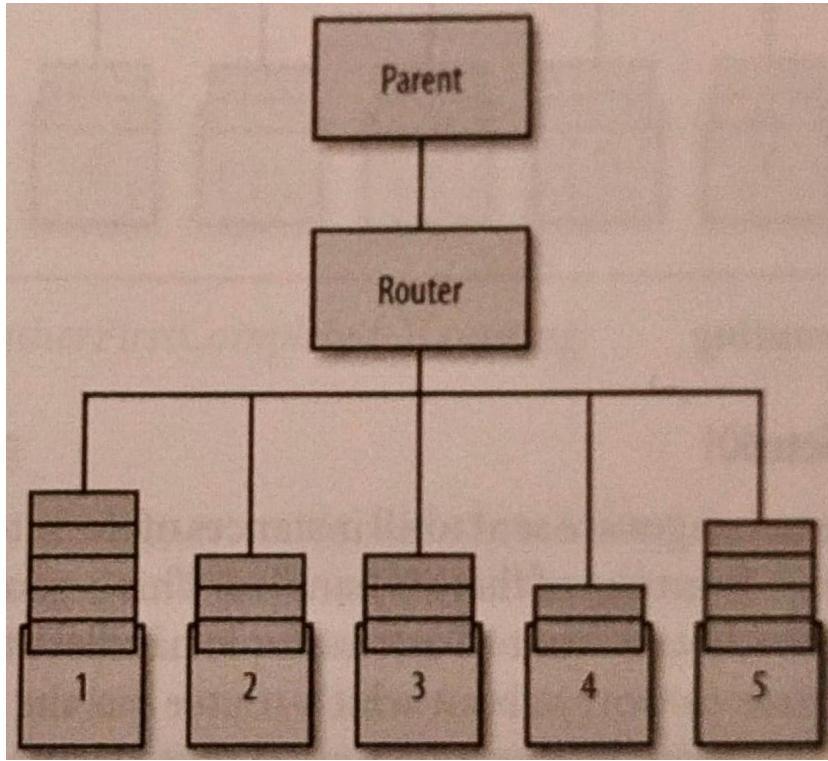
akka.routing.\*  
**RoundRobinRoutingLogic**  
RandomRoutingLogic  
SmallestMailboxRoutingLogic  
BroadcastRoutingLogic  
ScatterGatherFirstCompletedRoutingLogic  
TailChoppingRoutingLogic  
ConsistentHashingRoutingLogic

# Router



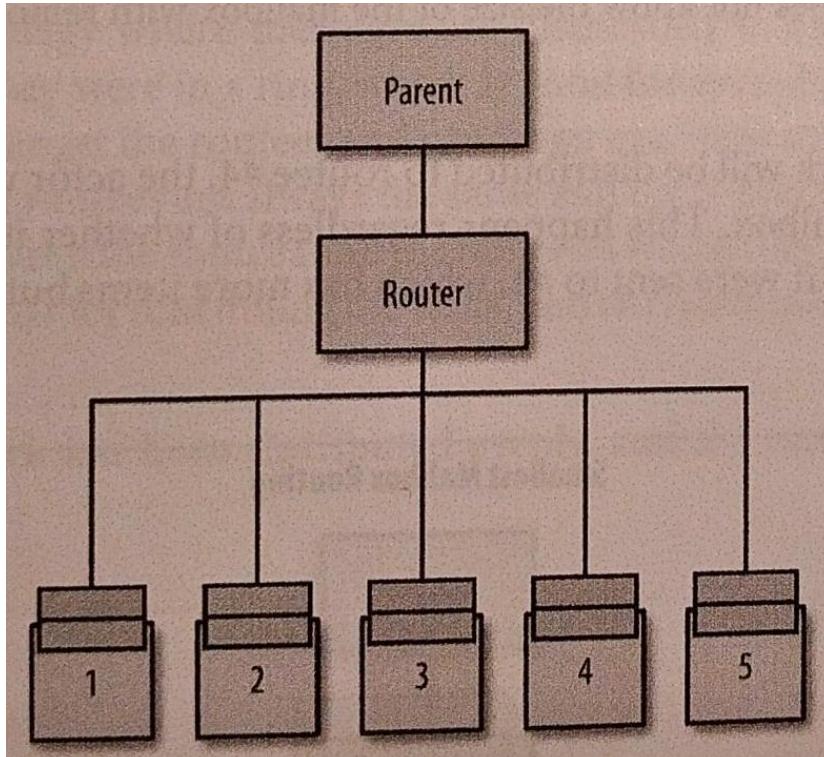
akka.routing.\*  
RoundRobinRoutingLogic  
**RandomRoutingLogic**  
SmallestMailboxRoutingLogic  
BroadcastRoutingLogic  
ScatterGatherFirstCompletedRoutingLogic  
TailChoppingRoutingLogic  
ConsistentHashingRoutingLogic

# Router



akka.routing.\*  
RoundRobinRoutingLogic  
RandomRoutingLogic  
**SmallestMailboxRoutingLogic**  
BroadcastRoutingLogic  
ScatterGatherFirstCompletedRoutingLogic  
TailChoppingRoutingLogic  
ConsistentHashingRoutingLogic

# Router

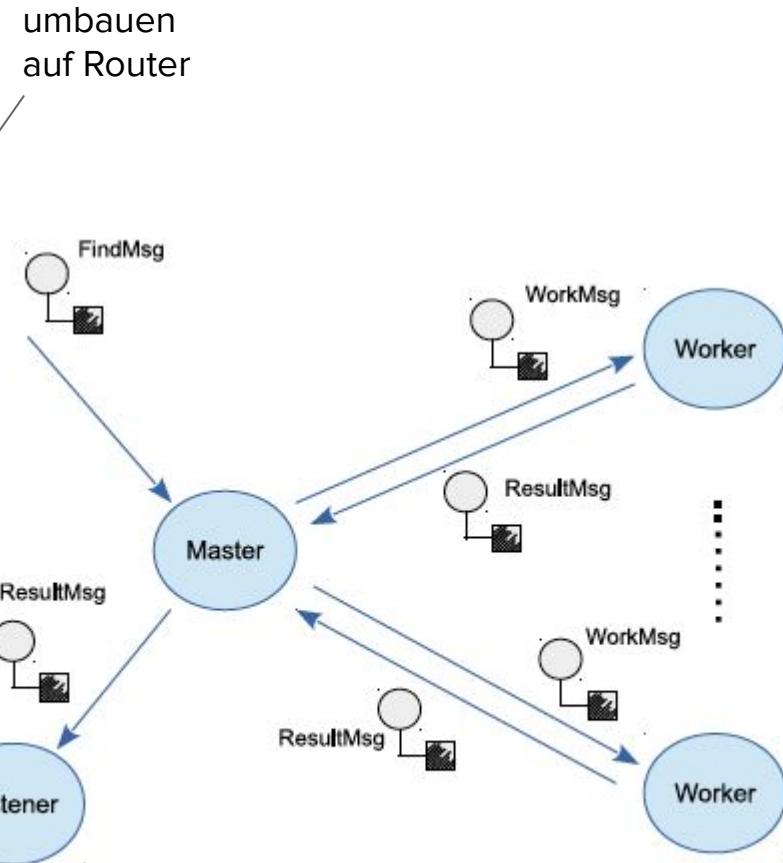


akka.routing.\*  
RoundRobinRoutingLogic  
RandomRoutingLogic  
SmallestMailboxRoutingLogic  
**BroadcastRoutingLogic**  
ScatterGatherFirstCompletedRoutingLogic  
TailChoppingRoutingLogic  
ConsistentHashingRoutingLogic

# Akka-Beispiel

# Systemarchitektur

- pp.07.02-RouterFindWords
- 3 Actor-Typen:
  - io.dama ffi.actors.find.MasterActor
  - io.dama ffi.actors.find.ListenerActor
  - io.dama ffi.actors.find.WorkerActor
- 4 Message-Typen:  
`io.dama.ffi.actors.find.messages.*`
  - PleaseCleanupAndStop
  - FindMsg
  - WorkMsg
  - ResultMsg
- Master und Listener genau einmal
- “viele” Worker als Routees und ein Router
- Master erzeugt Worker und Listener als Kinder und überwacht sie.



nicht prüfungsrelevant!

# **Woche 2: Akka-Actors verteilt im Netzwerk**

# Überblick (Woche 2)

- Actor-Modell
- Actors überwachen Actors
- Akka als Umsetzung des Actor-Modells
- Router
- Beispiel mit 3 Actor-Typen und 4 unterschiedlichen Nachrichtenarten
- Maven als build Tool
  - Umgang mit externen Dependencies, Build und Start
- Konfiguration
  - statt der programmatischen Erzeugung, deklarative Spezifikation über conf-Framework
- Remote-Actor (Aktoren verteilt im Netzwerk)
  -
- Router-Pool und -Group
  - Spezifikation von Router/Routees über config im Netzwerk verteilt

nicht prüfungsrelevant!

# Maven als Build-Tool

# Maven als Build-Tool

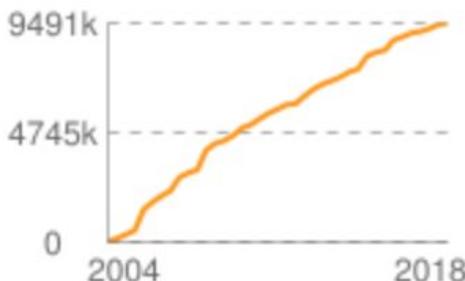
- externe Libraries erforderlich um Akka zu benutzen (ist nicht Teil des JDK-APIs)
- Kann man manuell beim Anbieter downloaden und dem Classpath hinzufügen.
- Wird aber schnell unübersichtlich, da Libraries von anderen Libraries abhängig sein können, die wiederum...
- Build-Tools automatisieren den Download und das Verwalten der Abhängigkeiten (und vieles mehr).
- Maven sucht in einem zentralen Repository nach Libraries (<https://mvnrepository.com/>)
- Die Libraries heißen bei Maven “Artefakte” und werden über eine Pfad, ID und Version identifiziert.
- Für Akka (lokal) wird akka-actor von com.typesafe.akka in Version 2.5.12 benötigt.



akka

Search

## Indexed Artifacts (9.50M)



Found 897 results

Sort: relevance | popular | newest

## 1. Akka Actor

com.typesafe.akka &gt; akka-actor

akka-actor

1,282 usages

Apache

Last Release on Apr 13, 2018

## Popular Categories

Aspect Oriented

Actor Frameworks

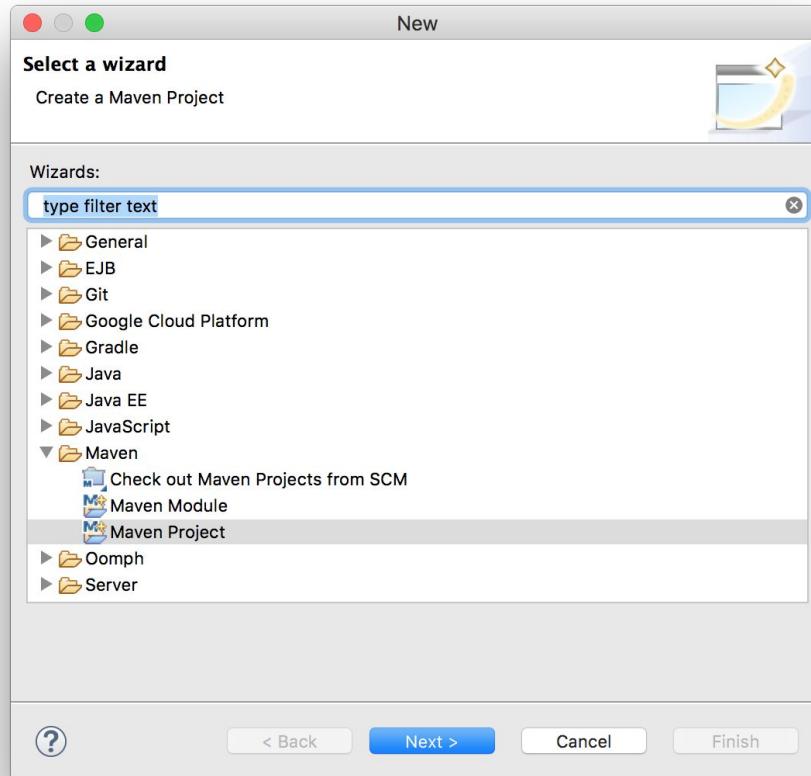


2. Akka Testkit

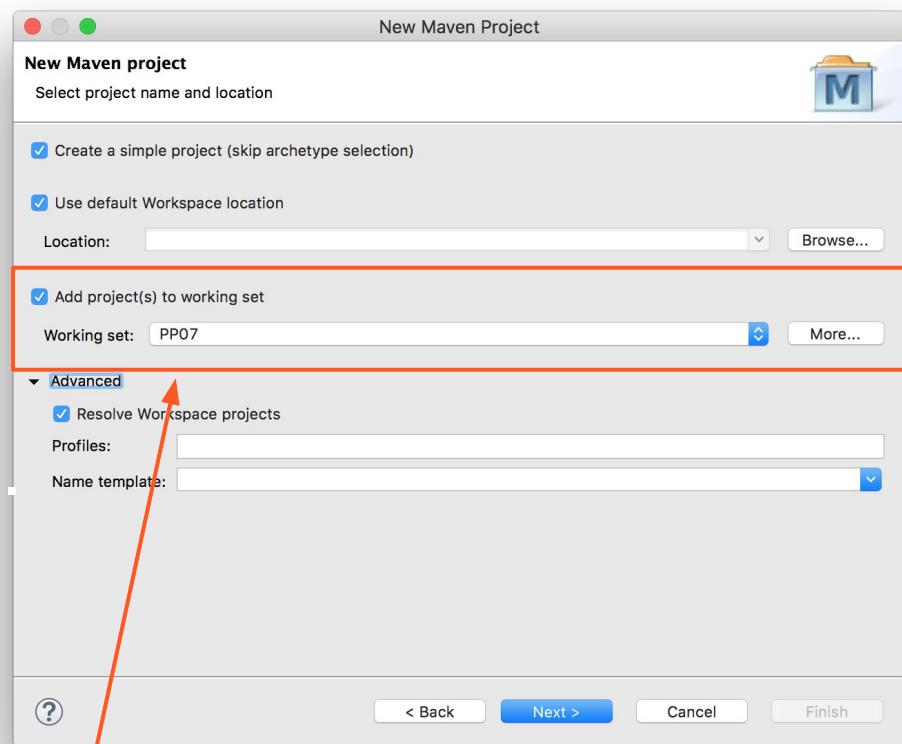
228 usages

# Maven als Build-Tool

- Maven benötigt die Beschreibung eines Projekts in Form einer XML-Datei (pom.xml).
- Darin enthalten sind u.a. die Abhängigkeiten, aber auch andere Informationen.
- Eclipse unterstützt Maven als Build-Tool, man kann aber auch komfortabel auf der Kommandozeile arbeiten, wenn Maven auf dem Rechner installiert ist: Das Maven Kommando ist mvn:
  - mvn clean
  - mvn compile
  - mvn exec:java -Dexec.mainClass=io.dama.par.actor.Main
- In den nächsten Folien wird Schritt für Schritt gezeigt, wie ein Akka-fähiges Maven-Projekt in Eclipse angelegt werden kann...



File→New→Other...



optional



## New Maven Project



### New Maven project

Configure project

#### Artifact

Group Id: io.dama.par

Artifact Id: pp.07.01-FindWords|

Version: 0.0.1-SNAPSHOT

Packaging: jar

Name:

Description:

#### Parent Project

Group Id:

Artifact Id:

Version:

Browse...

Clear

► Advanced



< Back

Next >

Cancel

Finish

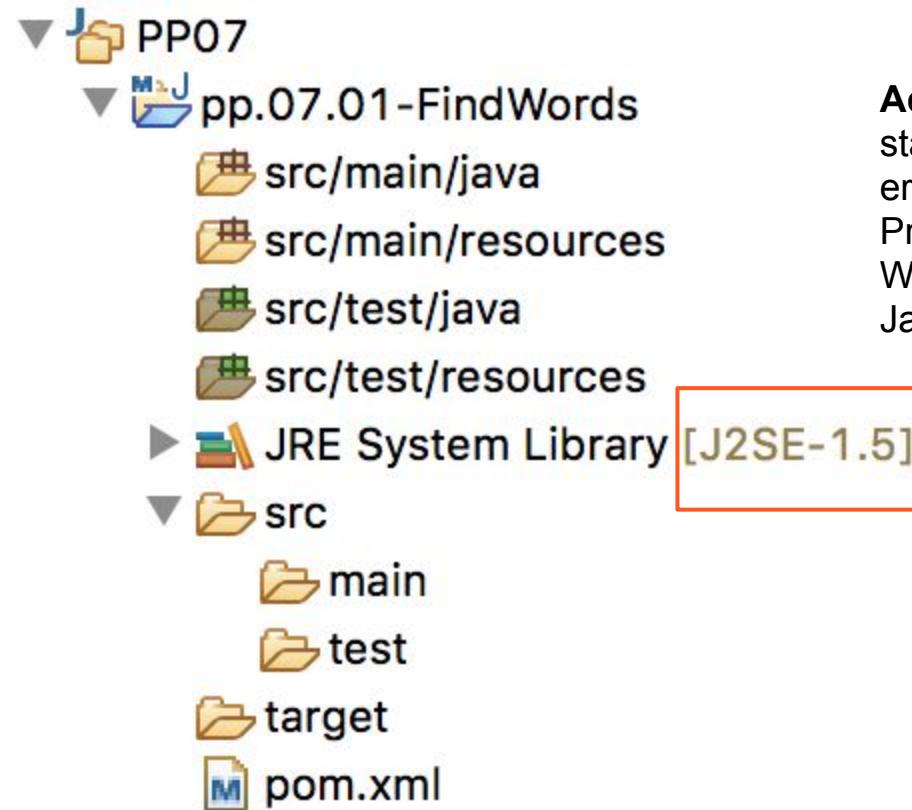
Dies beschreibt das Artefakt, das mit dem Projekt erzeugt werden soll (beispielsweise eine Library, die über Maven Central verteilt werden soll). Die Artefakt ID wird in Eclipse als Projektname verwendet. Die Group ID ist nur relevant, wenn das Artefakt von anderen identifiziert werden muss.

Das Projekt-Layout unterscheidet sich von dem, das sonst in Eclipse benutzt wird. Auf der Wurzelebene des Projekts gibt es nur das pom.xml, und die Verzeichnisse src/ und target/ (wird später beim Compilieren generiert)

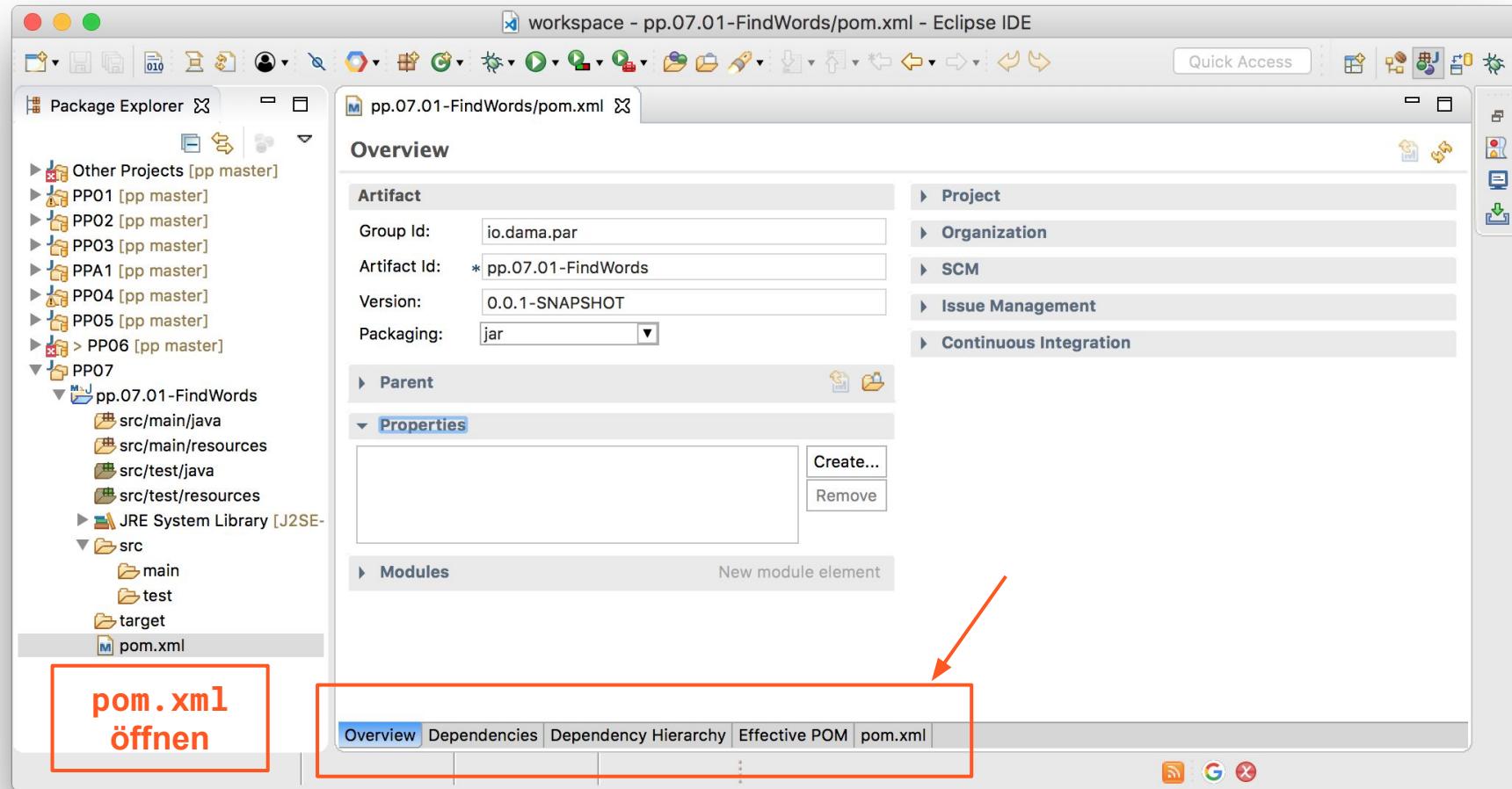
In src/ gibt es zwei Zweige: einen für das eigentliche Artefakt (main/) und einen für Unit-Tests (test/).

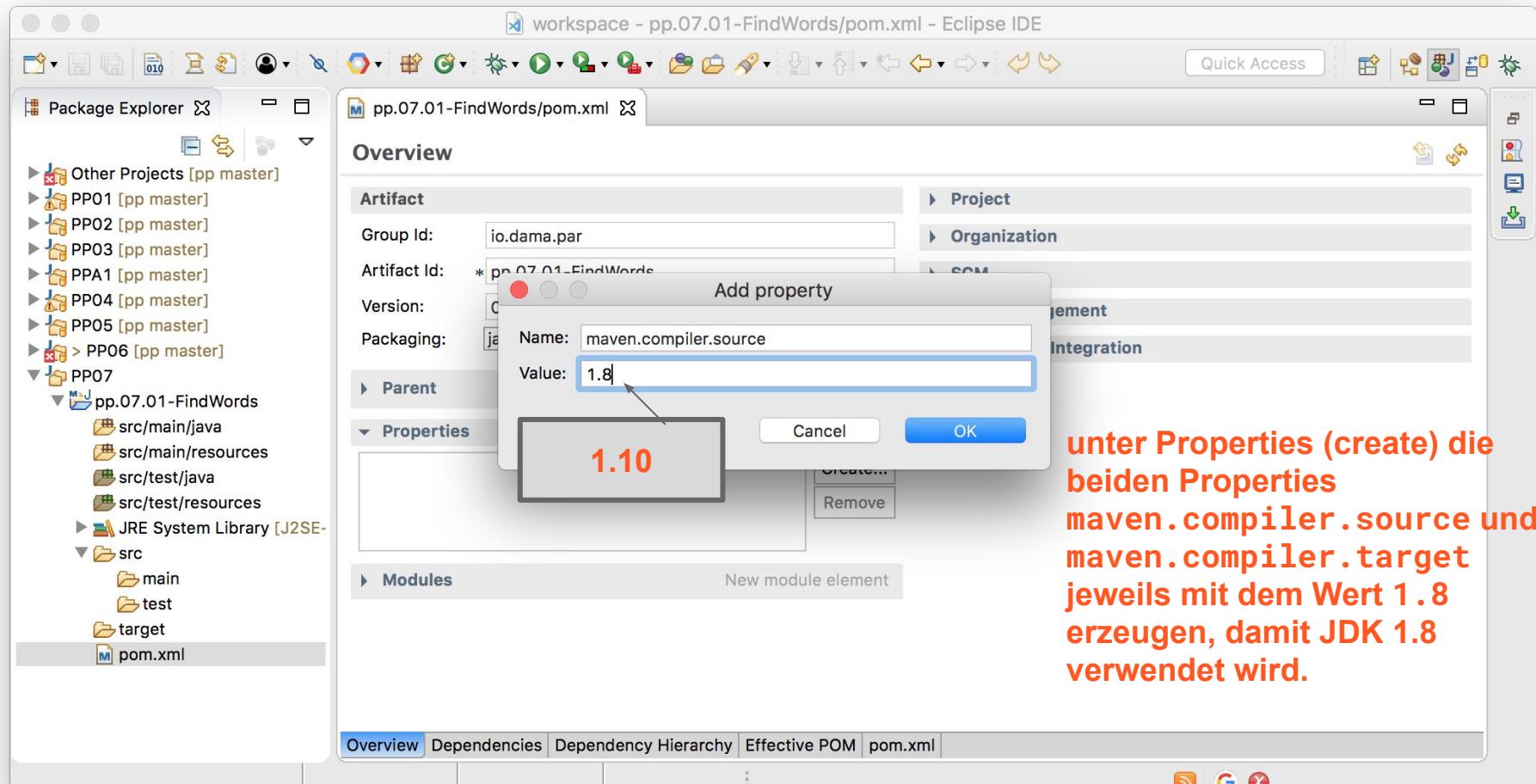
Im Fall eines Java-Projekts gibt es unter src/main/ und src/test/ jeweils die Verzeichnisse java/ und resources/. src/main/java/ ist das Grundverzeichnis für die Java-Klassen (ab hier kommen die Verzeichnisse für Packages). In resources/ liegen andere Dateien, die von Java heraus eingelesen werden sollen.

In target/ gibt es eine Reihe von Unterverzeichnissen, die spezifisch für die Art des Maven-Projekts sind. Im Regelfall liegen in target/classes/ alle compilierten class-Files und eine Kopie des Inhalts aus resources/.



**Achtung:**  
standardmäßig erzeugt Eclipse ein Projekt für JDK 1.5.  
Wir brauchen für Akka Java 8 (JDK 1.8)





unter Properties (create) die beiden Properties maven.compiler.source und maven.compiler.target jeweils mit dem Wert 1.8 erzeugen, damit JDK 1.8 verwendet wird.

Eclipse IDE workspace - pp.07.01-FindWords/pom.xml

Package Explorer

- Other Projects [pp master]
- PP01 [pp master]
- PP02 [pp master]
- PP03 [pp master]
- PPA1 [pp master]
- PP04 [pp master]
- PP05 [pp master]
- PP06 [pp master]
- PP07
  - pp.07.01-FindWords
    - src/main/java
    - src/main/resources
    - src/test/java
    - src/test/resources
  - JRE System Library [J2SE-]
  - src
    - main
    - test
  - target
  - pom.xml

\*pp.07.01-FindWords/pom.xml

### Overview

**Artifact**

Group Id: io.dama.par  
Artifact Id: \* pp.07.01-FindWords  
Version: 0.0.1-SNAPSHOT  
Packaging: jar

**Parent**

**Properties**

- maven.compiler.source : 1.8
- maven.compiler.target : 1.8

**Modules**

1.10 New module element

Quick Access

Project Organization SCM Issue Management Continuous Integration

Overview Dependencies Dependency Hierarchy Effective POM pom.xml

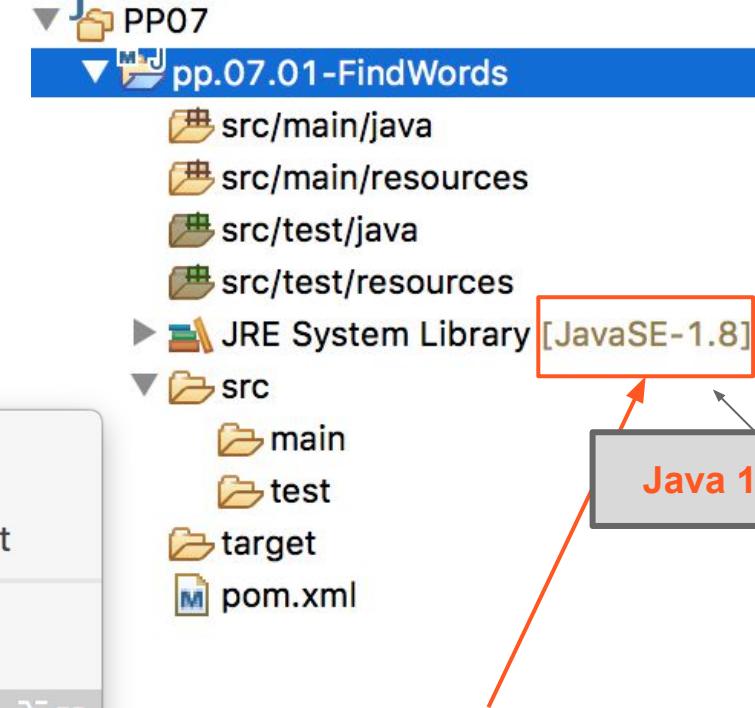
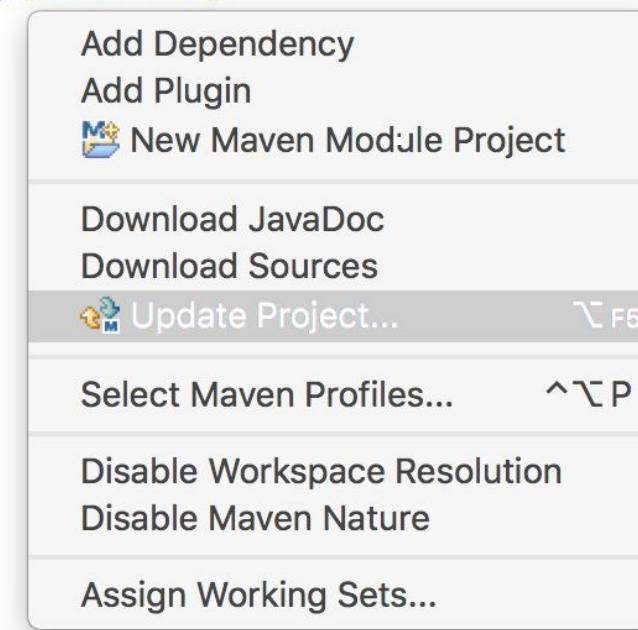
project

45

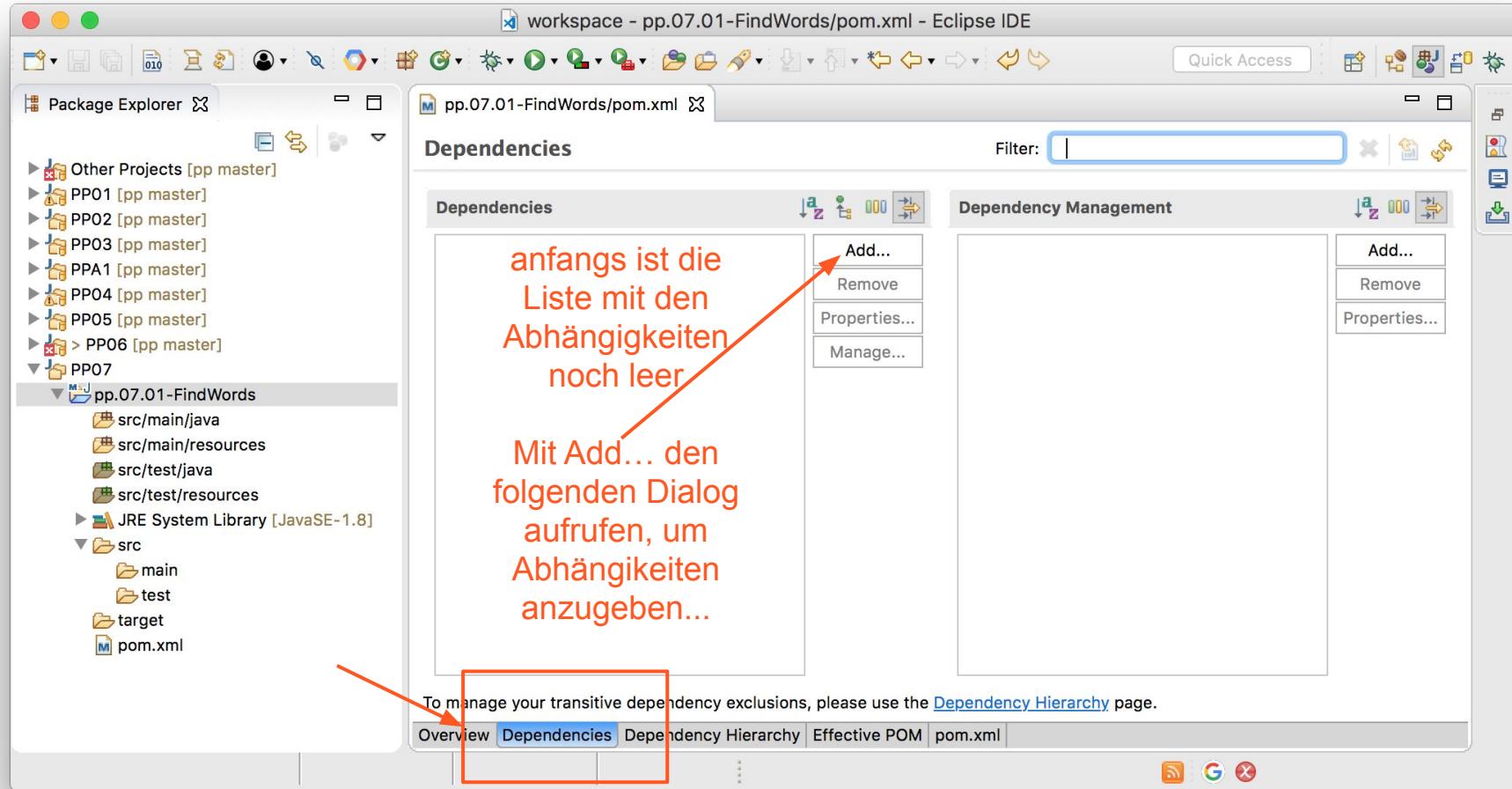
- src/main/java
- src/main/resources
- src/test/java
- src/test/resources
- JRE System Library [J2SE-1.5]
- ▼ src
  - main
  - test
- target
- pom.xml

Das pom.xml mit JDK 1.8 muss noch gespeichert werden, damit das Update gelingt!

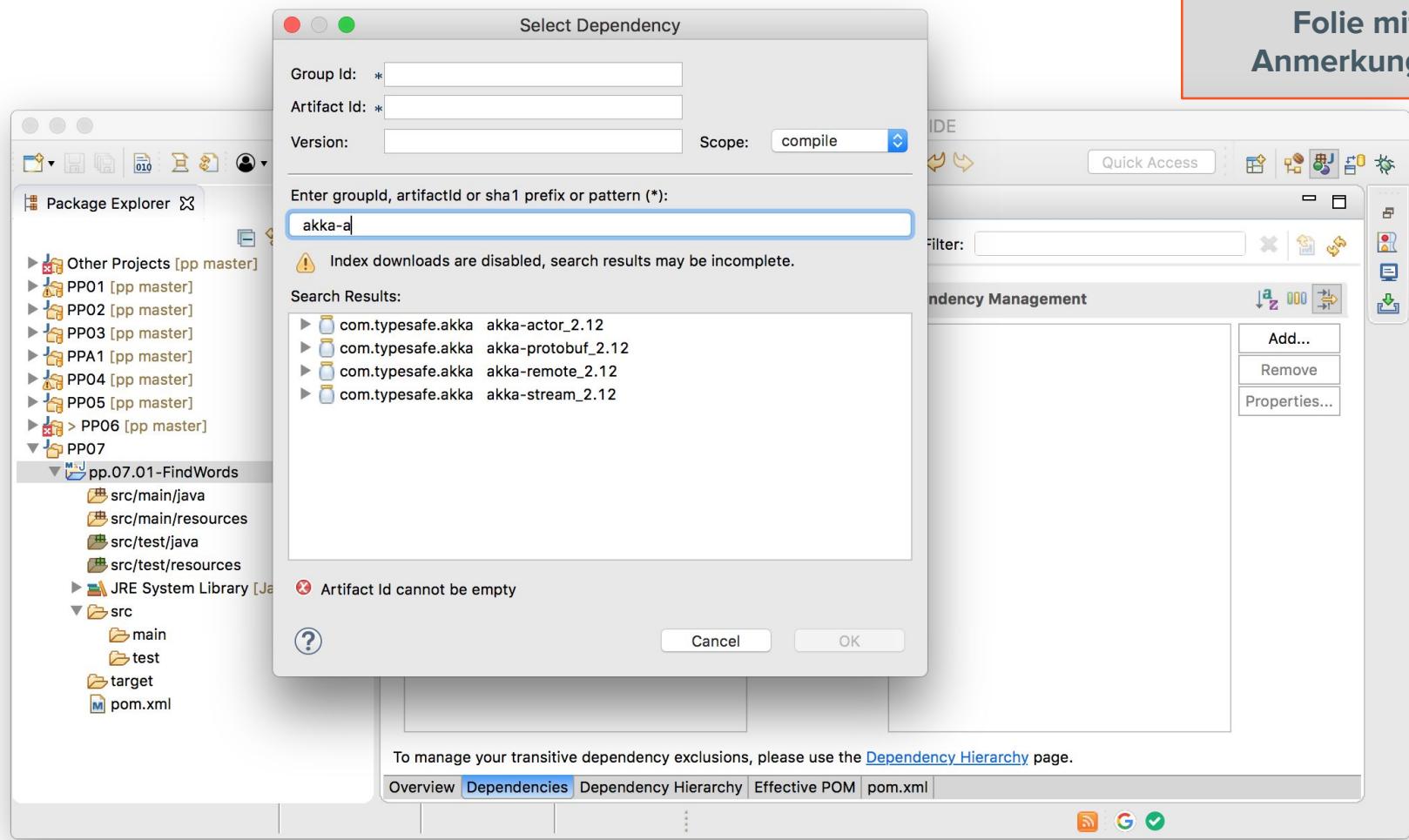
Im Kontextmenü (Rechtsklick) der Projektwurzel Maven... auswählen:



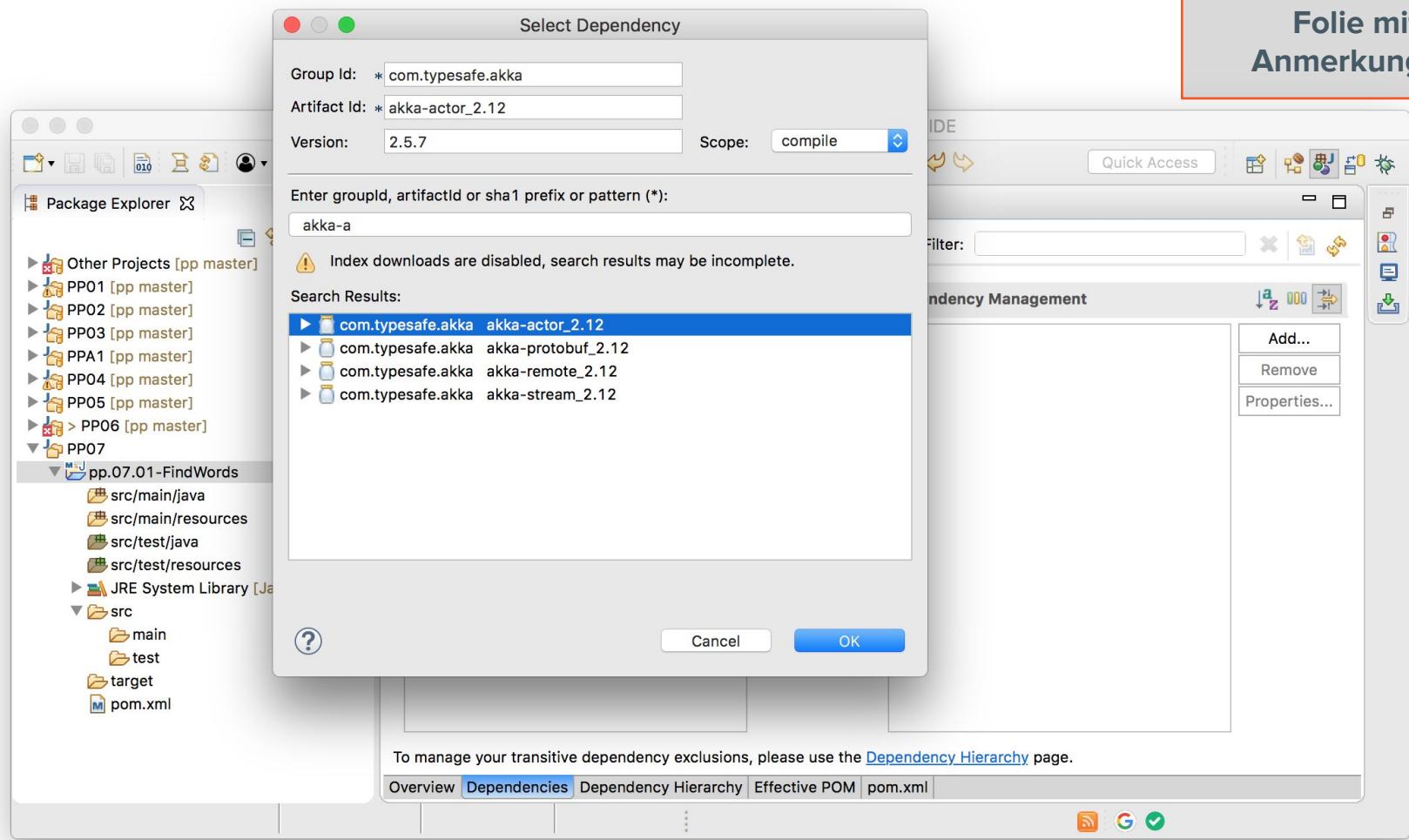
Update Project... macht aus dem Java 5-Projekt dauerhaft ein Java 8-Projekt.

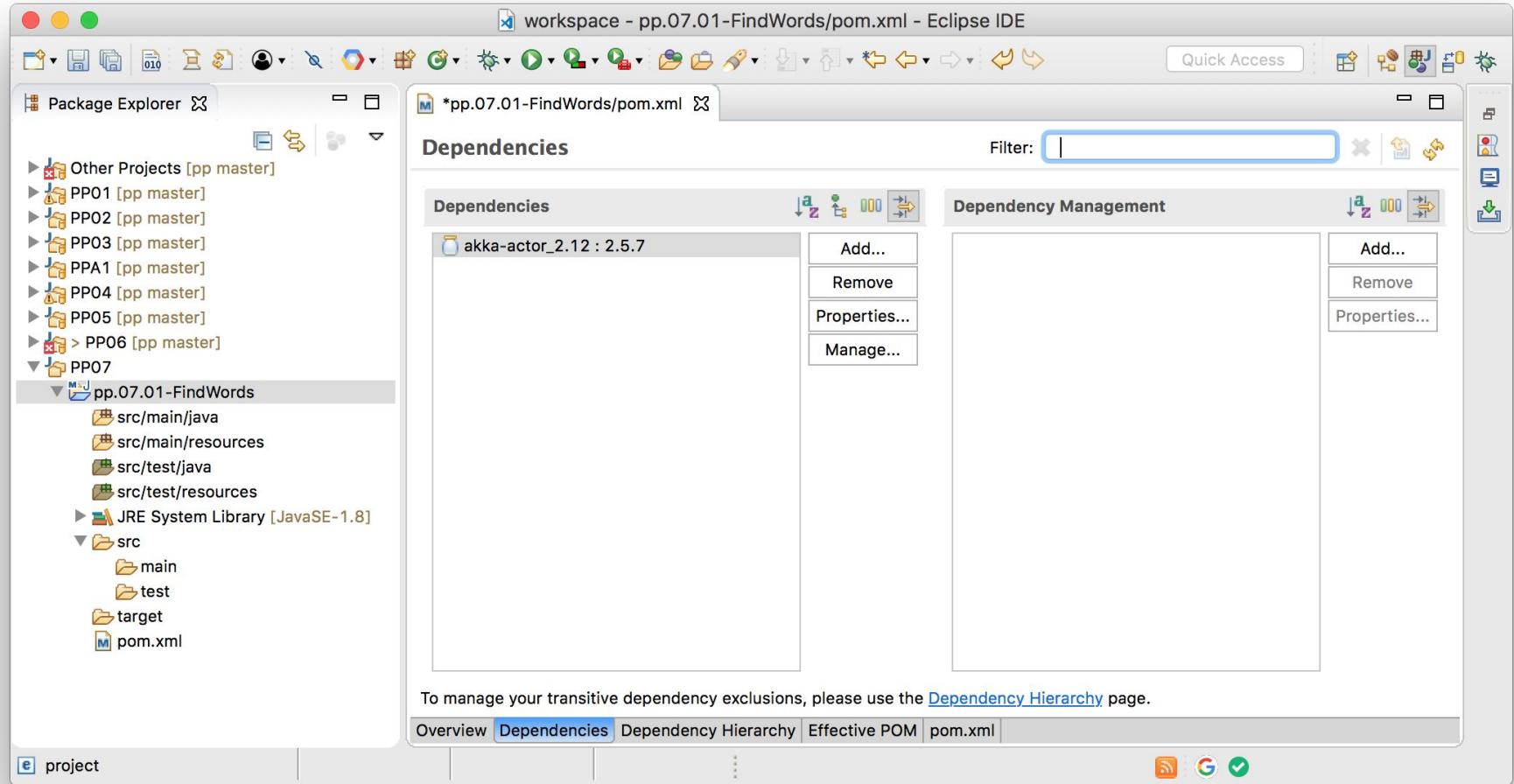


## Folie mit Anmerkungen

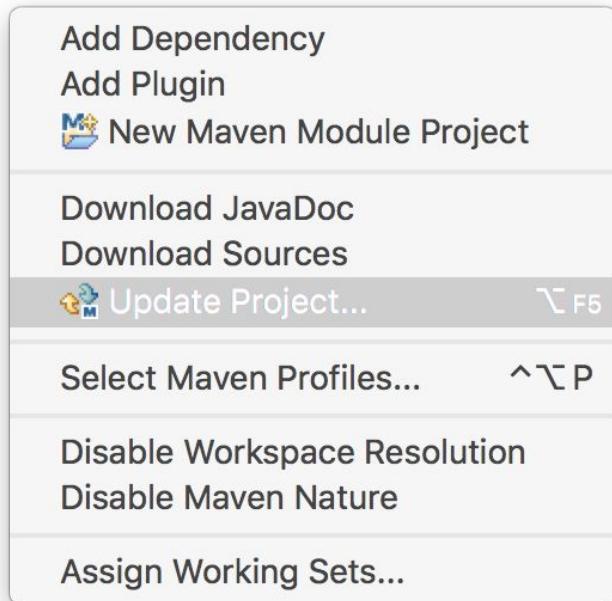


## Folie mit Anmerkungen





es werden noch die Abhangigkeiten der Abhangigkeiten benotigt. Dazu wieder an der Projektwurzel Maven... und Update Project... aufrufen:



workspace - pp.07.01-FindWords/pom.xml - Eclipse IDE

Package Explorer    pom.xml

Dependency Hierarchy [test]

Filter:

Dependency Hierarchy

- akka-actor\_2.12 : 2.5.7 [compile]
  - scala-library : 2.12.4 [compile]
  - config : 1.3.2 [compile]
- scala-java8-compat\_2.12 : 0.8.0 [compile]
  - scala-library : 2.12.0 (omitted for conflict with 2.12.4)

Resolved Dependencies

- akka-actor\_2.12 : 2.5.7 [compile]
- config : 1.3.2 [compile]
- scala-java8-compat\_2.12 : 0.8.0 [compile]
- scala-library : 2.12.4 [compile]

pp.07.01-FindWords

- src/main/java
- src/main/resources
- src/test/java
- src/test/resources

JRE System Library [JavaSE-1.8]

Maven Dependencies

- src
- main
- test

target

pom.xml

akka-actor ist abhängig von scala-library, config, scala-java8-compat, was wiederum von scala-library abhängig ist.

Diese vier Libraries werden in Form von JAR-Dateien von Maven Central runtergeladen und dem Classpath hinzugefügt.

Overview Dependencies Dependency Hierarchy Effective POM pom.xml

pp.07.01-FindWords

Quick Access

52

speichern nicht vergessen! Das resultierende pom.xml ist...

# Maven: pom.xml für Akka (lokal)

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>io.dama.par</groupId>
  <artifactId>pp.07.01-FindWords_solution</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>
  <dependencies>
    <dependency>
      <groupId>com.typesafe.akka</groupId>
      <artifactId>akka-actor_2.12</artifactId>
      <version>2.5.7</version>
    </dependency>
  </dependencies>
</project>
```

# Projekt-Layout bei Maven

```
MeinProjekt/  
└── pom.xml  
└── src  
    ├── main  
    │   └── java  
    │       └── resources  
    └── test  
        └── java  
            └── resources
```



```
cd MeinProjekt/  
mvn compile
```



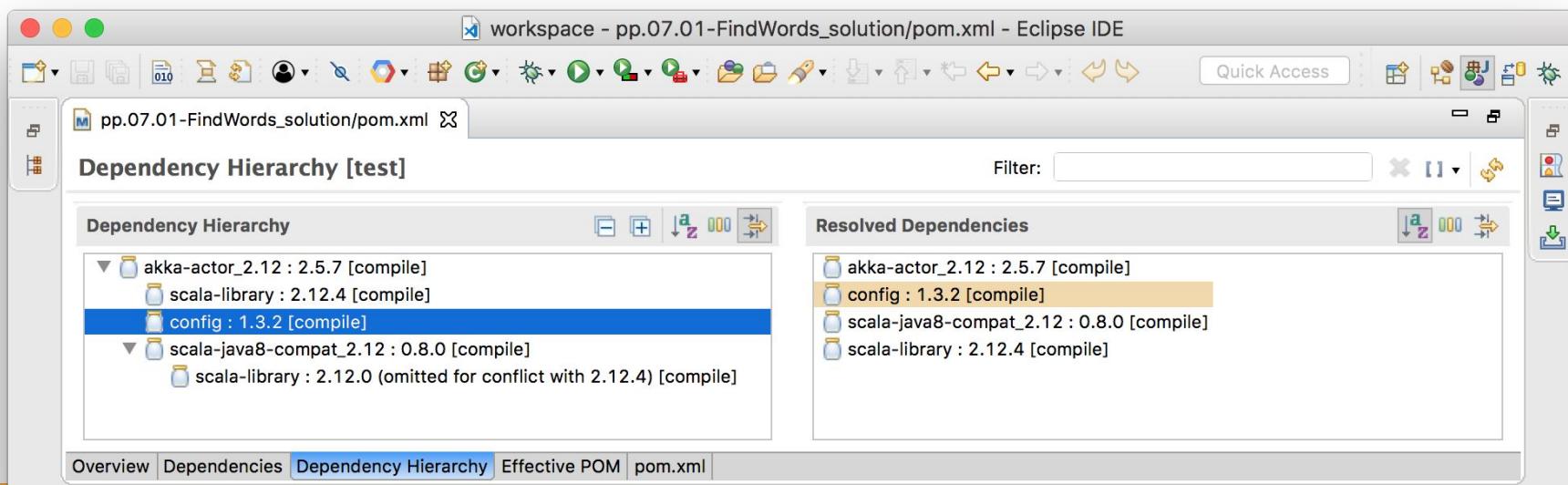
```
MeinProjekt/  
└── pom.xml  
└── src  
    ├── main  
    │   └── java  
    │       └── resources  
    └── test  
        └── java  
            └── resources  
└── target  
    ├── classes  
    └── maven-status  
        └── maven-compiler-plugin  
            └── compile  
                └── default-compile  
                    └── inputFiles.lst
```

nicht prüfungsrelevant!

**Remote (auf  
mehrere  
Rechner  
verteilte Actors)**

# Konfiguration über config 1/2

Das config-Framework ist ein moderner Ersatz von Properties-Dateien, ist aber nicht Teil des Java-Standard API's, sondern muss als Library eingebunden werden. Es wird von Akka intensiv genutzt. config ist daher Teil der Abhängigkeiten ("dependency") von Akka.



# Konfiguration über config 2/2

```
import com.typesafe.config.ConfigFactory;

public class MainClassOfApp {

    public static void main(final String[] args) {
        final ActorSystem system = ActorSystem.create("main", ConfigFactory.load("config"));
        // ...
    }
}
```

Sucht die Konfigurationsdatei **config.conf** im *current working directory* (das Verzeichnis, von dem aus die Anwendung gestartet wird (nicht unbedingt das Verzeichnis, in dem die Klasse liegt)).

Bei Verwendung von Maven:

src/main/resources/config.conf

# Neue Dependency: Akka-Remote

Eclipse IDE workspace - pp.07b.02-FindWordsRouterRemote\_solution/pom.xml

pp.07b.02-FindWordsRouterRemote\_solution/pom.xml

Dependency Hierarchy [test]

Dependency Hierarchy

- akka-remote\_2.12 : 2.5.12 [compile]
  - scala-library : 2.12.5 [compile]
- akka-actor\_2.12 : 2.5.12 [compile]
  - scala-library : 2.12.5 (omitted for conflict with 2.12.5) [compile]
  - config : 1.3.2 [compile]
- scala-java8-compat\_2.12 : 0.8.0 [compile]
  - scala-library : 2.12.0 (omitted for conflict with 2.12.5) [compile]
- akka-stream\_2.12 : 2.5.12 [compile]
  - scala-library : 2.12.5 (omitted for conflict with 2.12.5) [compile]
  - akka-actor\_2.12 : 2.5.12 (omitted for conflict with 2.5.12) [compile]
  - akka-protobuf\_2.12 : 2.5.12 (omitted for conflict with 2.5.12) [com
  - reactive-streams : 1.0.2 [compile]
- ssl-config-core\_2.12 : 0.2.3 [compile]
  - scala-library : 2.12.5 (omitted for conflict with 2.12.5) [compile]
  - config : 1.2.0 (omitted for conflict with 1.3.2) [compile]
- scala-parser-combinators\_2.12 : 1.1.0 [compile]
  - scala-library : 2.12.4 (omitted for conflict with 2.12.5) [comp
- akka-protobuf\_2.12 : 2.5.12 [compile]
  - scala-library : 2.12.5 (omitted for conflict with 2.12.5) [compile]
- netty : 3.10.6.Final [compile]
- aeron-driver : 1.7.0 [compile]
  - aeron-client : 1.7.0 (omitted for conflict with 1.7.0) [compile]
- aeron-client : 1.7.0 [compile]
- agrona : 0.9.12 [compile]

Resolved Dependencies

- aeron-client : 1.7.0 [compile]
- aeron-driver : 1.7.0 [compile]
- agrona : 0.9.12 [compile]
- akka-actor\_2.12 : 2.5.12 [compile]
- akka-protobuf\_2.12 : 2.5.12 [compile]
- akka-remote\_2.12 : 2.5.12 [compile]
- akka-stream\_2.12 : 2.5.12 [compile]
- config : 1.3.2 [compile]
- netty : 3.10.6.Final [compile]
- reactive-streams : 1.0.2 [compile]
- scala-java8-compat\_2.12 : 0.8.0 [compile]
- scala-library : 2.12.5 [compile]
- scala-parser-combinators\_2.12 : 1.1.0 [compile]
- ssl-config-core\_2.12 : 0.2.3 [compile]

Maven Repository: com.typesafe.akka/akka-remote\_2.12/2.5.12

Sicher | https://mvnrepository.com/artifact/com.typesafe.akka/akka-remote\_2.12/2.5.12

MVNREPOSITORY

Indexed Artifacts (9.53M)

Popular Categories

- Aspect Oriented
- Actor Frameworks
- Application Metrics
- Build Tools
- Bytecode Libraries
- Command Line Parsers
- Cache Implementations
- Cloud Computing
- Code Analyzers
- Collections
- Configuration Libraries
- Core Utilities
- Date and Time Utilities
- Dependency Injection
- Embedded SQL Databases

Akka Remote > 2.5.12

akka-remote

License	Apache 2.0
Categories	Distributed Communication
Organization	Lightbend Inc.
HomePage	<a href="http://akka.io/">http://akka.io/</a>
Date	(Apr 13, 2018)
Files	<a href="#">pom (4 KB)</a> <a href="#">jar (2.2 MB)</a> <a href="#">View All</a>
Repositories	Central
Used By	156 artifacts
Scala Target	Scala 2.12 (View all targets)

Maven Gradle SBT Ivy Graal Leiningen Buildr

```
<!-- https://mvnrepository.com/artifact/com.typesafe.akka/akka-remote -->
<dependency>
    <groupId>com.typesafe.akka</groupId>
    <artifactId>akka-remote_2.12</artifactId>
    <version>2.5.12</version>
</dependency>
```

Include comment with link to declaration

# Remote benutzen: Rechner 10.136.117.1

Folie mit  
Anmerkungen

```
import com.typesafe.config.ConfigFactory;

public class Main {
    public static void main(final String[] args) {
        final ActorSystem system = ActorSystem.create("sys1", ConfigFactory.load('app'));
        //...
    }
}

akka {
    actor {
        provider = remote
    }
    remote {
        enabled-transports = [ "akka.remote.netty.tcp" ]
        netty.tcp {
            hostname = "10.136.117.1"
            port = 4711
        }
    }
}
```

Main.java

app.conf

# Remote benutzen: Rechner 10.136.117.2

Folie mit  
Anmerkungen

```
import com.typesafe.config.ConfigFactory;

public class Main {
    public static void main(final String[] args) {
        final ActorSystem system = ActorSystem.create("sys1", ConfigFactory.load('app'));
        //...
    }
}

akka {
    actor {
        provider = remote
    }
    remote {
        enabled-transports = [ "akka.remote.netty.tcp" ]
        netty.tcp {
            hostname = "10.136.117.2"
            port = 8080
        }
    }
}
```

Main.java

app.conf

# Entfernte Actors

URL: akka.<protocol>://<actor system name>@<hostname>:<port>/<actor path>

Entfernten Actor von 10.136.117.1 aus auf 10.136.117.2 finden:

- auf Rechner 10.136.117.1 wird folgendes ausgeführt:

```
ActorSelection remoteActor =  
    getContext().actorSelection("akka.tcp://sys1@10.136.117.2:8080/user/worker");  
remoteActor.tell("Hallo von 10.136.117.1");
```

- Nachricht "Hallo von 10.136.117.1" wird an Actor /user/worker auf dem Rechner 10.136.117.2 geschickt.

Entfernten Actor von 10.136.117.1 aus auf 10.136.117.2 erzeugen:

- auf Rechner 10.136.117.1 wird folgendes ausgeführt:

```
ActorRef remoteActor = system.actorOf(Props.create(WorkerActor.class), "worker");  
remoteActor.tell("Hallo von 10.136.117.1", ActorRef.noSender());
```

- Actor /user/worker wird auf dem Rechner 10.136.117.2 erzeugt und Nachricht "Hallo von 10.136.117.1" wird an ihn geschickt.

# Alternativ über Conf auf 10.136.117.1

```
akka {  
    actor {  
        deployment {  
            "remote/*" {  
                remote = "akka.tcp://sys1@10.136.117.2:8080"  
            }  
        }  
    }  
}
```

Wenn lokal (auf 10.136.117.1) ein Actor namens `remote / ...` erzeugt oder auf ihn zugegriffen wird, wird er effektiv auf Rechner 10.136.117.2 erzeugt bzw. benutzt.

Wildcard-Symbol `*` funktioniert nur innerhalb von `"..."`. Ein partielles Matching wie z.B. `"worker-*"` funktioniert leider nicht. Es kann vielmehr immer nur ein Actor-Name (was zwischen `/` steht) ersetzt werden.

# Remote-Actors und Nachrichten

Nachrichten-Objekte müssen von einer serialisierbaren Klasse instanziert werden, damit sie als Byte-Abfolge über eine Netzwerkverbindung versendet werden können:

```
import java.io.Serializable;

public class Messages {
    public static class PleaseCleanupAndStop implements Serializable {
        private static final long serialVersionUID = 7273183276322548603L;
    }
}
```

Die serialVersionUID ist eine eindeutige Zahl, mit der die Klasse beim Empfänger wiedererkannt werden kann (hilft bei Doppelbenennungen und Versionskonflikten).

nicht prüfungsrelevant!

# **Router: Pool und Group**

# Router-Pool

Im Fall eines **Router-Pools** sind *Router* und *Routees* eng gekoppelt:

- Der Router erzeugt “seine” Routees-Actoren selber und überwacht sie.
- Sollte ein Routee terminieren, entfernt der Router sie aus seiner Routee-Liste.

# Router-Pool erzeugen 1/3

Variante 1: lokal programmatisch

```
public class Main {  
    public static void main(final String[] args) {  
        final ActorSystem sys = ActorSystem.create("sys");  
        ActorRef r = sys.actorOf(new RoundRobinPool(5).props(Props.create(Worker.class)), "router");  
    }  
}
```

# Router-Pool erzeugen 2/3

## Variante 2: lokal mit Konfiguration

```
akka.actor.deployment {  
    router {  
        router = round-robin-pool  
        nr-of-instances = 5  
    }  
}
```

---

```
public class Main {  
    public static void main(final String[] args) {  
        final ActorSystem sys = ActorSystem.create("sys", ConfigFactory.load("app"));  
        ActorRef r = sys.actorOf(FromConfig.getInstance().props(Props.create(Worker.class)), "router");  
    }  
}
```

app.conf

Main.java

# Router-Pool erzeugen 3/3

Variante 3: remote mit Konfiguration (10 Worker auf host2 und host3 verteilt)

```
akka.actor.deployment {  
    remotePool {  
        router = round-robin-pool  
        nr-of-instances = 10  
        target.nodes = ["akka.tcp://app@host2:4711", "akka.tcp://app@host3:4711"]  
    }  
}
```

app.conf

```
public class Main {  
    public static void main(final String[] args) {  
        final ActorSystem sys = ActorSystem.create("sys", ConfigFactory.load("app"));  
        ActorRef r = sys.actorOf(FromConfig.getInstance().props(Props.create(Worker.class)), "remotePool");  
    }  
}
```

Main.java

# BalancingPool

Beim **BalancingPool** teilen alle *Routees* eine *Mailbox*.

Die Arbeit (Nachrichten in der Mailbox) werden also optimal auf die zur Verfügung stehenden (Worker-) Ressourcen aufgeteilt: Kein Routee muss auf Nachrichten warten während andere Routees mit der Abarbeitung der Nachrichten nicht hinterherkommen.

# Router-Group

Im Fall einer **Router-Group** sind *Router* und *Routees* weniger eng gekoppelt als bei einem Router-Pool:

- Der Router erzeugt “seine” Routees-Actoren nicht selber, sondern bekommt sie mitgeteilt.
- Er überwacht die Routees nicht selber.

# Router-Group erzeugen 1/2

## Variante 2: lokal mit Konfiguration

```
akka.actor.deployment {  
    router {  
        router = round-robin-group  
        routees.paths = [ "/user/worker1", "/user/worker2", "/user/worker3" ]  
    }  
}
```

```
public class Main {  
    public static void main(final String[] args) {  
        final ActorSystem sys = ActorSystem.create("sys", ConfigFactory.load("app"));  
        sys.actorOf(Props.create(WorkerActor.class), "worker1");  
        sys.actorOf(Props.create(WorkerActor.class), "worker2");  
        sys.actorOf(Props.create(WorkerActor.class), "worker3");  
        ActorRef r = sys.actorOf(FromConfig.getInstance().props(), "router");  
    }  
}
```

app.conf

Main.java

# Router-Group erzeugen 2/2

## Variante 2: remote mit Konfiguration

```
akka.actor.deployment {  
    router {  
        router = round-robin-group  
        routees.paths = ["akka.tcp://sys@host2:2552/user/worker1", "akka.tcp://sys@host2:2552/user/worker2"]  
    }  
}
```

---

```
public class Main {  
    public static void main(final String[] args) {  
        final ActorSystem sys = ActorSystem.create("sys", ConfigFactory.load("app"));  
        sys.actorOf(Props.create(WorkerActor.class), "worker1");  
        sys.actorOf(Props.create(WorkerActor.class), "worker2");  
        ActorRef r = sys.actorOf(FromConfig.getInstance().props(), "router");  
    }  
}
```

app.conf

Main.java

# Zusammenfassung

- Actors-Modell für Nebenläufigkeit
- kein geteilter Zustand (Variablen), daher keine Erfordernis Zugriffe zu koordinieren
- stattdessen Zusammenarbeit über asynchronen Nachrichtenversand
- In Akka
  - Actors sind in Hierarchie und können sich überwachen
  - Nachrichten sind beliebige Objekte
  - Versenden mit `tell` (“fire and forget”) und `ask` (liefert Future, an dem die Antwort in der Zukunft realisiert wird)
  - Router erlauben Aufteilung der Arbeit auf eine Gruppe von gleichartigen Routees (erlaubt die Ressourcen besser auszunutzen, insbesondere auf in verteilten Systemen)
- in Eclipse kaum noch manuell zu bewältigen. Daher Maven als Build-Tool
  - neues Projektlayout + pom.xml