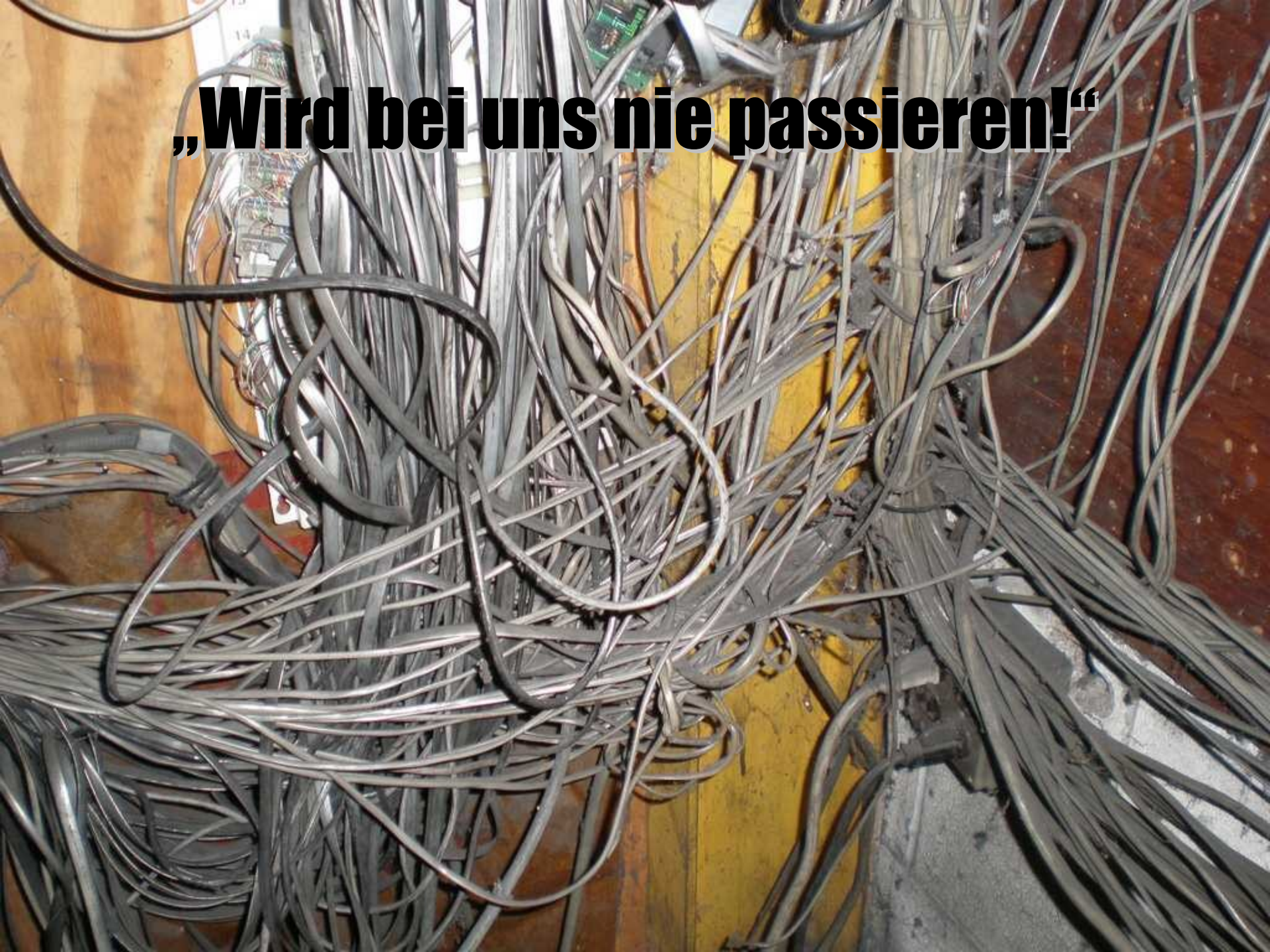
Baby Steps mit Akka

Dr. Stefan Schlott
BeOne Stuttgart GmbH

Ordnung am Anfang


Ein Projekt entwickelt sich...


„Wird bei uns nie passieren!“

Java 5: Executioner

Threadpool-Verwaltung

Bringt Arbeitspakete zur Ausführung

Runnable: „Fire and forget“

Callable: Rückgabewert als Future

Java 7: Fork/Join

Ähnlich Executioner, nur mit „Work Stealing“

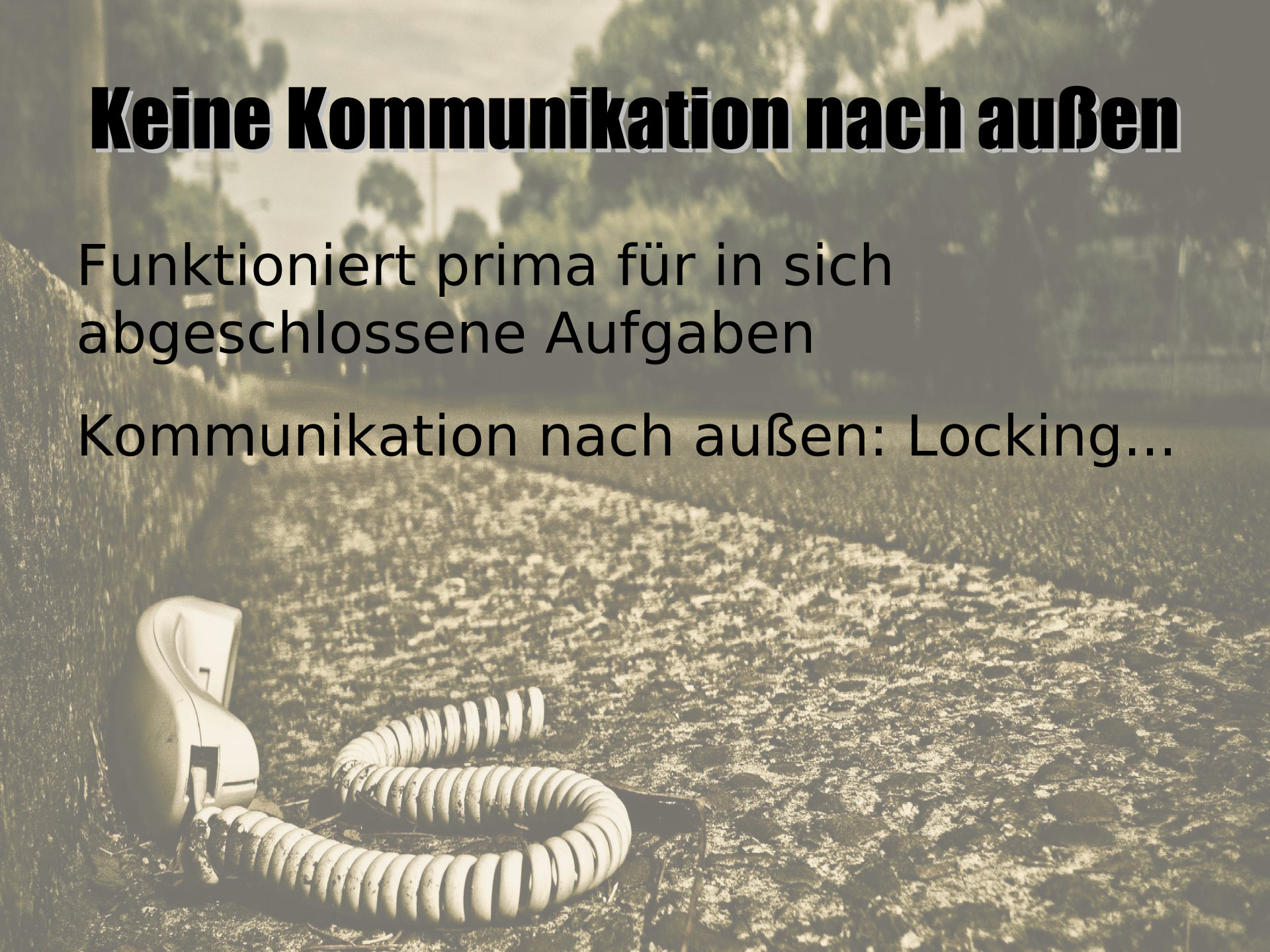
Statt Blockieren: Abarbeiten anderer Tasks

Ideal für Divide-and-Conquer

Keine Kommunikation nach außen

Funktioniert prima für in sich abgeschlossene Aufgaben

Kommunikation nach außen: Locking...



STM erkennt Änderungen, die in Konflikt stehen

STM = Software Transactional Memory

Idee: Transaktionen ähnlich wie bei Datenbanken

Im Konfliktfall: Abbruch, Rollback, keine Änderungen exponiert

→ Problem bei simpler Retry-Strategie

Dataflow parallelisiert Zugriffe „nach Bedarf“

Idee: Dataflow-Variablen

- Nur 1x zuweisen
- Berechnung: Kein „mutable State“, Zugriff nur auf Konstanten oder andere Dataflow-Variablen
- Parallelisierbar, äquivalent zur seriellen Ausführung

Aktoren: Leichtgewichtige, Nachrichten-basierte Prozesse

Aktoren: Abgeschlossene Einheit, kein Zugriff auf Interna von außen

Kommunikation mit (unveränderbaren!) Nachrichten

Immer nur eine Nachricht in Bearbeitung, keine Nebenläufigkeit

Mailbox

...schon mal dagewesen?

Akka: Mischung mehrerer Ansätze

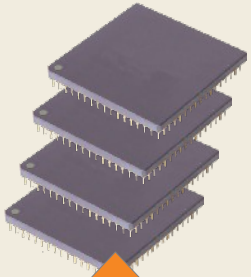
Kernansatz: Aktoren, Nachrichten

STM und „Transaktoren“

Ökosystem 'drumherum, z.B.:

- Logging
- Scheduler
- Event Bus
- Agents

Akka: All-in-one Skalierungslösung



Scale Up

akka

Scale Out

Beispiel: Verzeichnisbaum lesen

Synchronisation zwischen Laptop und Desktop

Verzeichnisse rekursiv abarbieten

Manche Verzeichnisse besonders behandeln (z.B. git-Repos)


Scala

Framework-Rumpf: ActorSystem

Initialisieren von Akka: ActorSystem erzeugen

```
ActorSystem system =
     ActorSystem.create("actors");
```

(theoretisch mehrere möglich)

Context/System kreiert Aktoren

Aktor erzeugen (programmatisch):

```
ActorRef ref = system.actorOf(
      new Props(MyActor.class), "myactor");
```

Props ≈ Factory für Actor-Instanz

Aktoren bilden Hierarchie

Zugriff immer über ActorRef

Aktoren behandeln Nachrichten

UntypedActor

- Einzelne „onReceive“-Methode

TypedActor

- Interface-Implementierungs-Paar
- „Aktoren-untypisch“, aber schöne Schnittstelle nach außen

Nachrichten senden

Ohne Rückgabewert

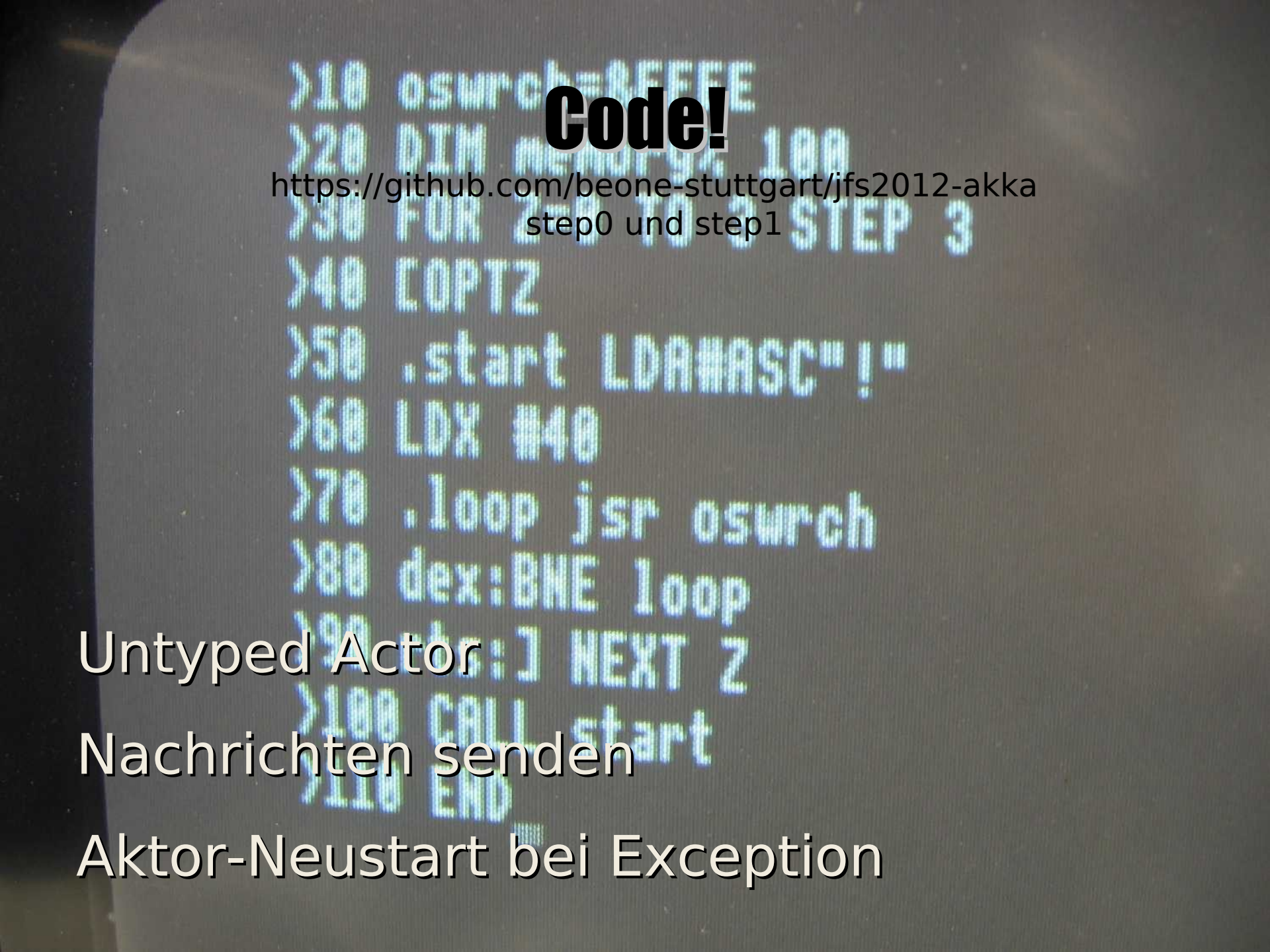
```
actorref.tell(msg, getSelf());
```

Mit Rückgabewert

```
Future<Object> reply =
     Patterns.ask(actor, msg, timout);

getSender().tell(msg);
```

Antwort (auch im Fehlerfall!) muß selbst generiert werden


Code!
https://github.com/beone-stuttgart/jfs2012-akka
step0 und step1
Untyped Actor
Nachrichten senden
Aktor-Neustart bei Exception

Akka startet im Fehlerfall Aktoren neu

One-for-One (default): Aktor (und dessen Kinder) werden neu gestartet

All-for-One: Zusätzlich Neustart der Geschwister

Aktuelle Nachricht wird verworfen, restliche Mailbox bleibt erhalten

Lifecycle-Methoden von Aktoren

preStart, postStop

preRestart: Auf der alten Instanz

postRestart: Auf der neuen Instanz

Aktoren-Referenzen immer gültig

Referenz: Kein direkter Zugriff auf Instanz!

...auch gültig nach Aktor-Restart

...auch gültig nach Aktor-Stop (führt dann zum DeadLetter-Aktor)

Verbesserung der Implementierung

Trennen zwischen Koordination und eigentlichen Workern

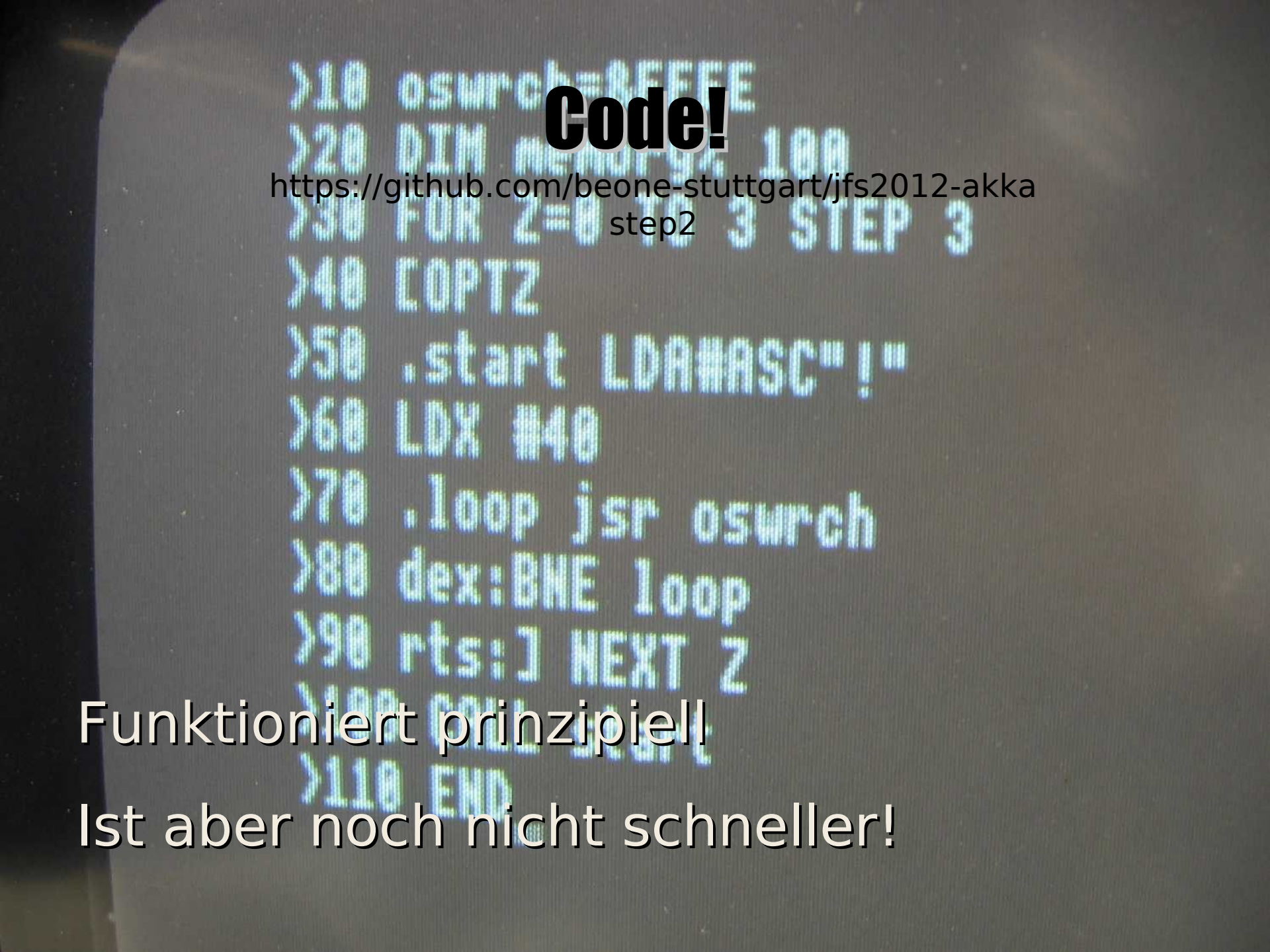
Shutdown der Akka-Bibliothek

Saubere Nachrichtenobjekte

...toString hilft beim Debuggen ungemein

Code!

https://github.com/beone-stuttgart/jfs2012-akka
step2



Funktioniert prinzipiell

Ist aber noch nicht schneller!

Router: (Fast) Aktoren ohne Nachrichten-Behandlung

Router: Leiten Nachrichten an untergeordnete Aktoren weiter

Nicht transparent: Sind in der Hierarchie sichtbar

```
actorref = getContext().actorOf(
  new Props(MyActor.class).withRouter(
    new SmallestMailboxRouter(5)),
  "myactors");
```

Aktoren ohne Standardkonstruktor

```
actorref = system.actorOf(new Props(
  new UntypedActorFactory() {
  @Override public Actor create() {
    return new MyActor(1,2,3);
  }
}), "myactor");
```

Mögliche Alternative: akka-Konfigurations-Datei


Code!
https://github.com/beone-stuttgart/jfs2012-akka
step3
Speed! \o/

Logging: Blockierende Operation

Aufrufe in Logging-Framework: Synchrone Operation

...auch System.out blockiert unnötig

```
LoggingAdapter log = Logging.getLogger(
  getContext().system(), this);
```

Logging über Nachrichten

Handler hierfür austauschbar (z.B. slf4j)

Aktoren sollen nicht blockieren

Blockierende Operationen: Durch Nachrichten und Antworten ersetzen

Lang dauernde Aktionen: Außerhalb von Akka abhandeln

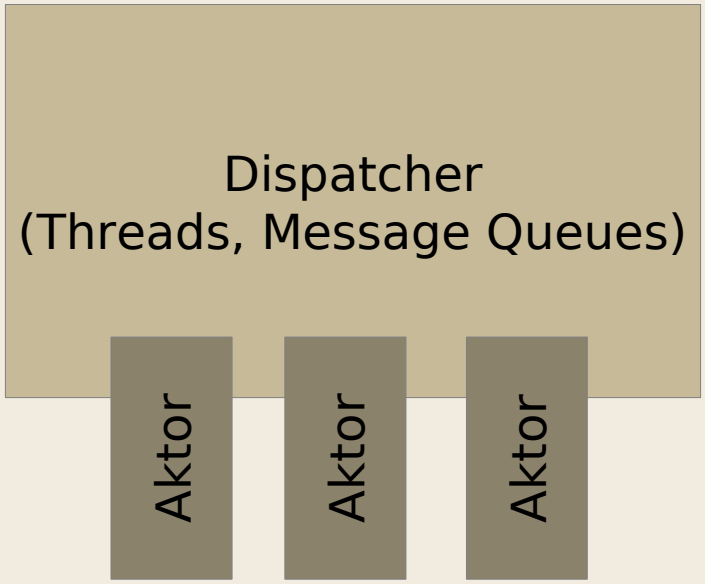
→ Was tun mit der eigentlichen Synchronisation im Beispiel?

Dispatcher führen Aktoren aus

Dispatcher: „Motor“ von Akka

- Verwaltet Nachrichten
- Bringt Aktoren zur Ausführung

Default Dispatcher: Wird auch für Interna von Akka verwendet



Separater Dispatcher für Sync

Konfiguration über Akka-Config-File:

```
workerdispatcher {
  type = PinnedDispatcher
  executor = "thread-pool-executor"
}
```

Auswahl beim Erzeugen des Aktors:

```
actorref = getContext().actorOf(
  new Props(SyncActor.class).
    withDispatcher("workerdispatcher"),
  "sync");
```

Scheduler für verzögerten / wiederholten Nachrichtenversand

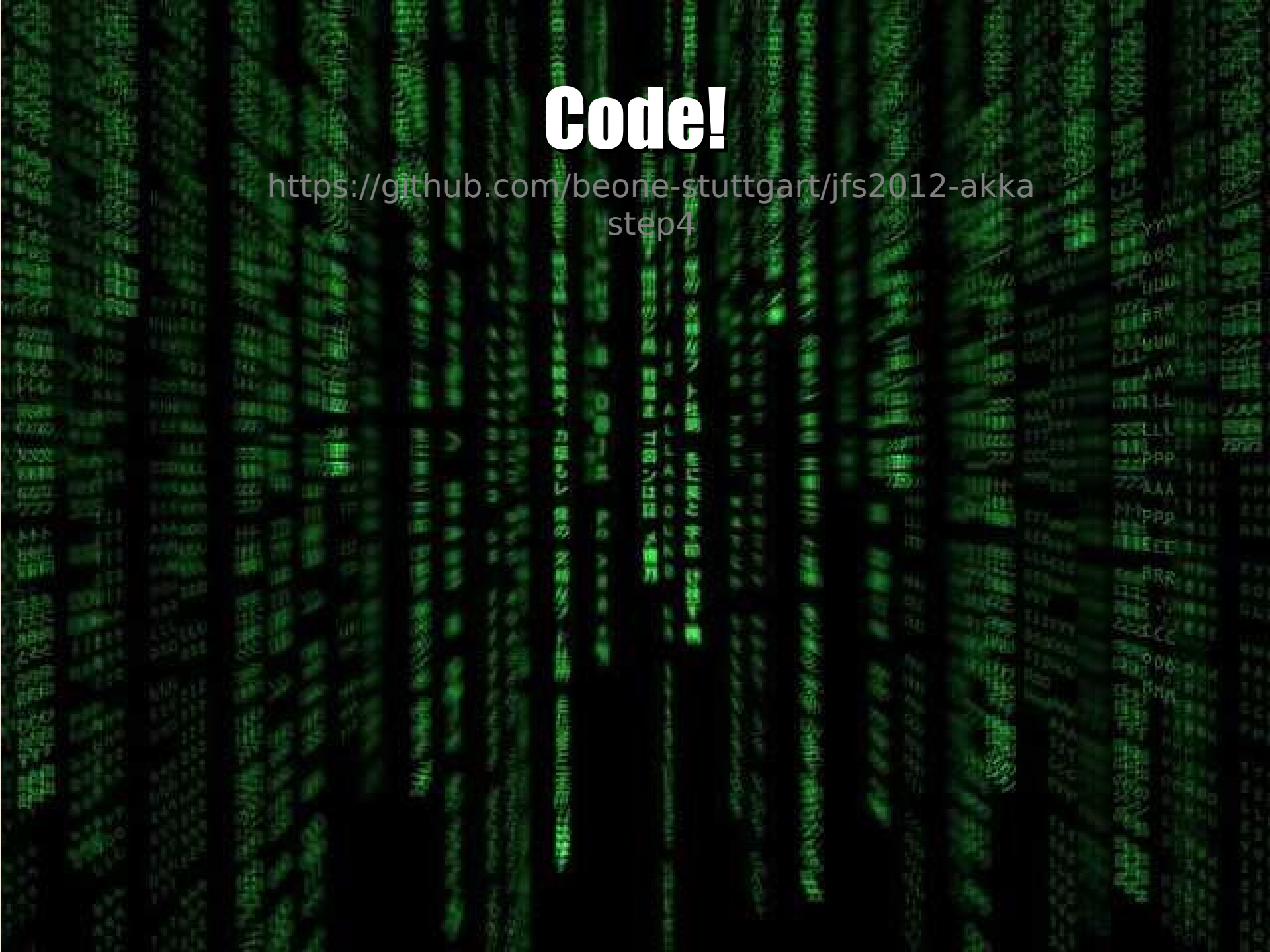
Nachricht verzögert senden:

```
system.scheduler().schedule(Duration.create(250,
  TimeUnit.MILLISECONDS), actorRef, msg);
```

Nachrichten wiederholt senden:

```
system.scheduler().schedule(Duration.Zero(),
  Duration.create(250, TimeUnit.MILLISECONDS),
  actorRef, msg);
```

→ Regelmäßige Fortschrittsanzeige


Code!
https://github.com/beone-stuttgart/jfs2012-akka
step4

Ausblick

Weitere Features:

- Definition von Aktoren in Akka-Config
- Remoting: Lediglich Änderung der URI in Akka-Config!
- STM/Transaktoren
- Agents, State Machines, Behavior-Change uvm.

Fazit: Ein Blick auf Akka lohnt!

Abstraktion der Parallelisierung

- Testbare (lineare) Programmierung
- Definierte Punkte zur Behandlung von Parallelisierungsproblemen

Je nach Bedarf: Sehr systemnah bis hoher Abstraktionslevel

Akka Cluster, Integration mit nio, etc. läßt auf eine große Zukunft hoffen

Dr. Stefan Schlott

Senior Consultant, BeOne Stuttgart GmbH

http://www.beone-group.com/

Mail: stefan.schlott@beone-group.com

Twitter: @_skyr

Bildnachweise

Logos: (C) der entsprechenden Firmen

"Linn going for a walk" http://www.flickr.com/photos/tlman/4861217206/ by Tilman Zitzmann CC BY-NC

"Wool gathering" http://www.flickr.com/photos/jimmead/6832246175/ by Jim Mead CC BY-SA

"after" http://www.flickr.com/photos/crumpart/3158169788/ by crumpart CC BY-NC

"P4030004" http://www.flickr.com/photos/chiacomo/2385647786/ by Nathan CC BY

"executioner's day off" http://www.flickr.com/photos/54459164@N00/6207683991/ by Johnson Cameraface CC BY-NC-SA

"Forks" http://www.flickr.com/photos/harshwcam3/3444273878/ by HarshWCAM3 CC BY-SA

"disconnected" http://www.flickr.com/photos/mrdarkroom/5357276454/ by Michael CC BY-NC-ND

"DeLorean Time Machine" http://www.flickr.com/photos/animenut/6046397402/ by Alan CC BY-NC-ND

"ST 486 DX4" http://www.flickr.com/photos/wimox/5209368794/ by Henry Mühlpfordt CC BY-SA

"Server room at CERN" http://www.flickr.com/photos/torkildr/3462606643/ by Torkild Retvedt CC BY-SA

"Computer PC Tower Desktop" http://www.flickr.com/photos/jul3sg33/581464364/ by jules:g CC BY-NC-SA

"Piece of code" http://www.flickr.com/photos/nox_noctis_silentium/2829906581/ by Timitrius CC BY-SA

"Matrix Core" http://www.flickr.com/photos/bobloo17/188697388/ by Sourabh Rath CC BY-NC-SA

Lizenz