Reaktive Programmierung mit Akka

29.07.2014, André Arnold

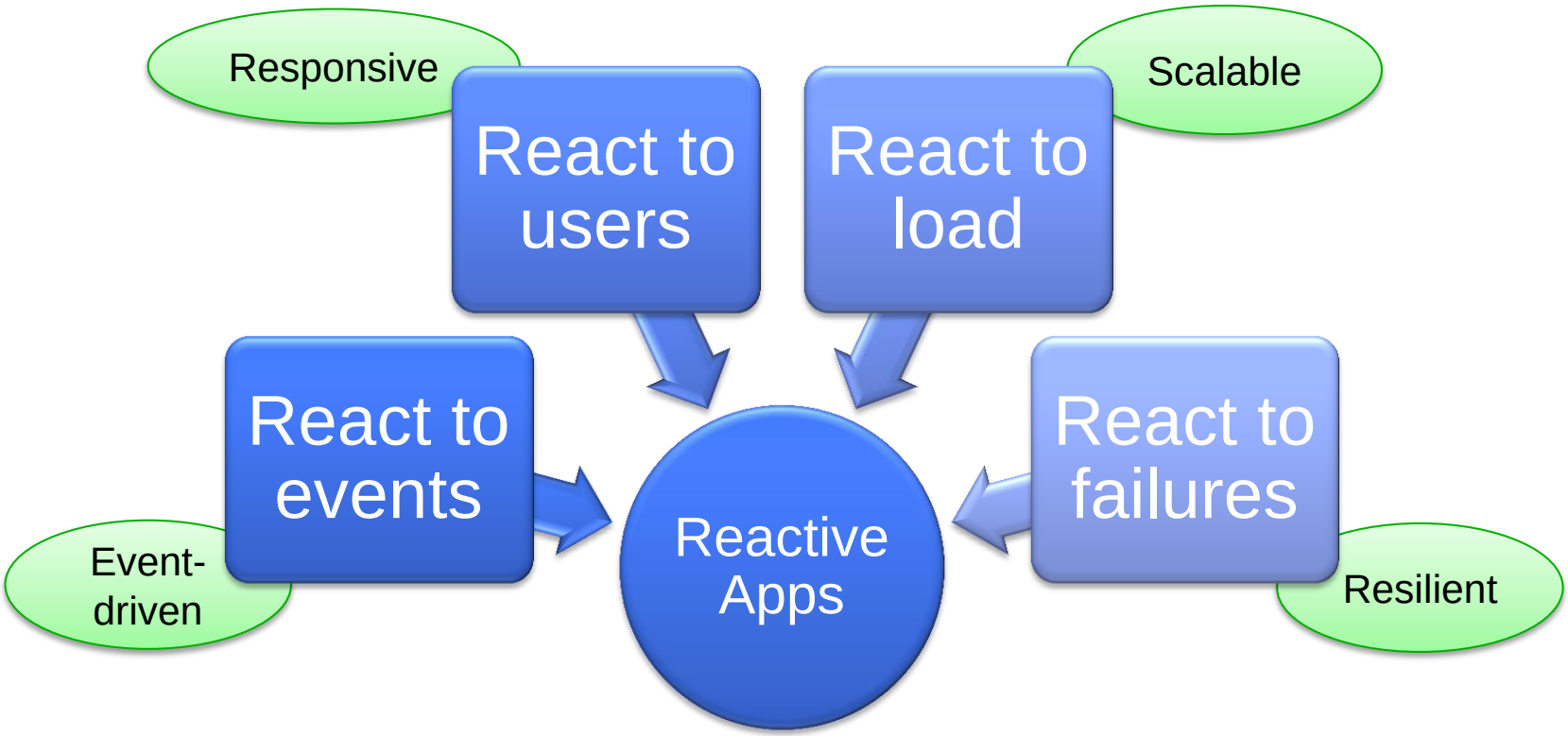
Agenda

1. Reacitve Manifesto: Skalierbarkeit, Concurrency und Resilience
2. Was sind Aktoren?
3. Aktoren in Akka
4. Resilience in Akka
5. Akka Live!

Reactive Manifesto

Merriam-Webster :

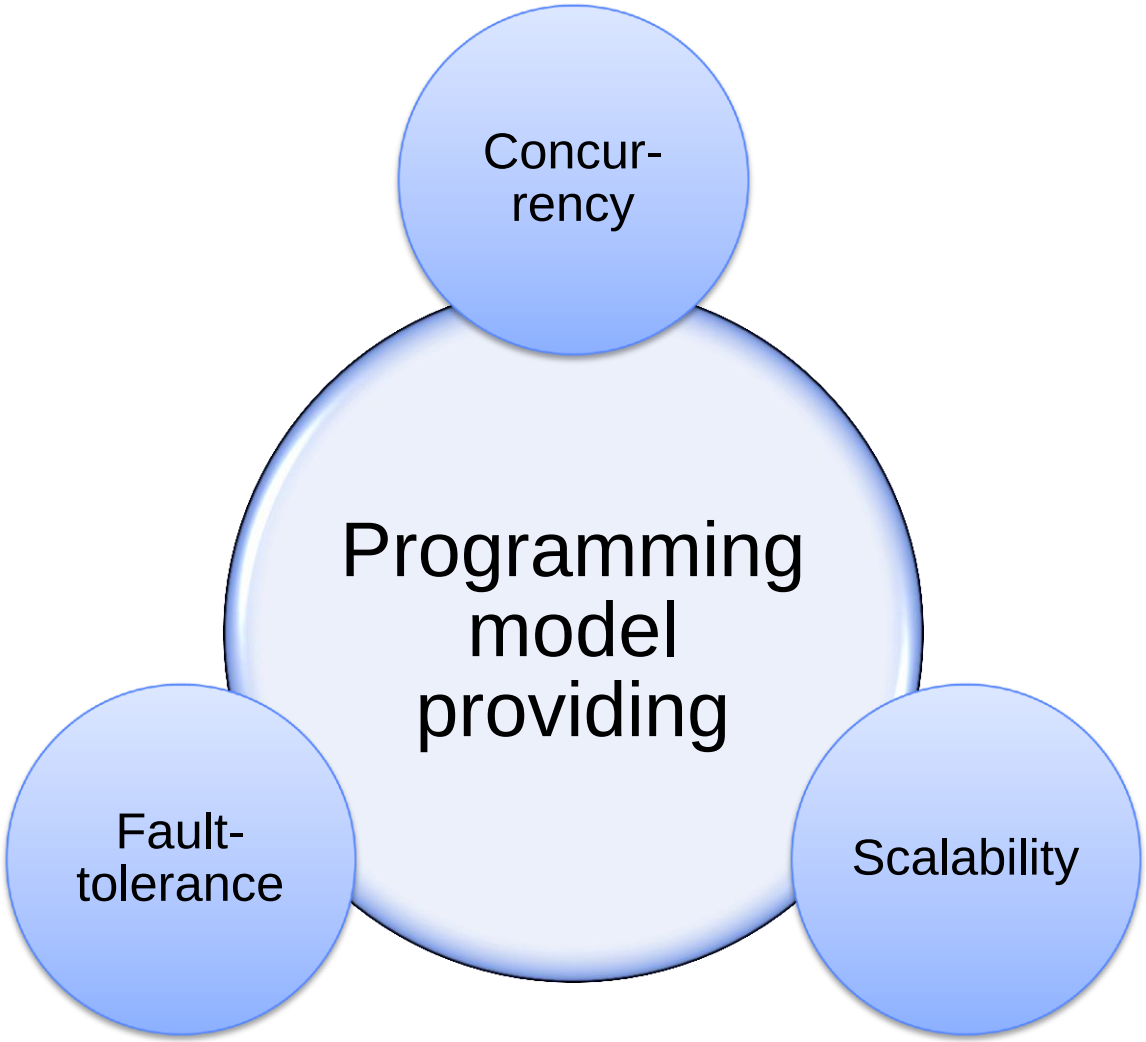
„reactive“ „*readily responsive to a stimulus*”



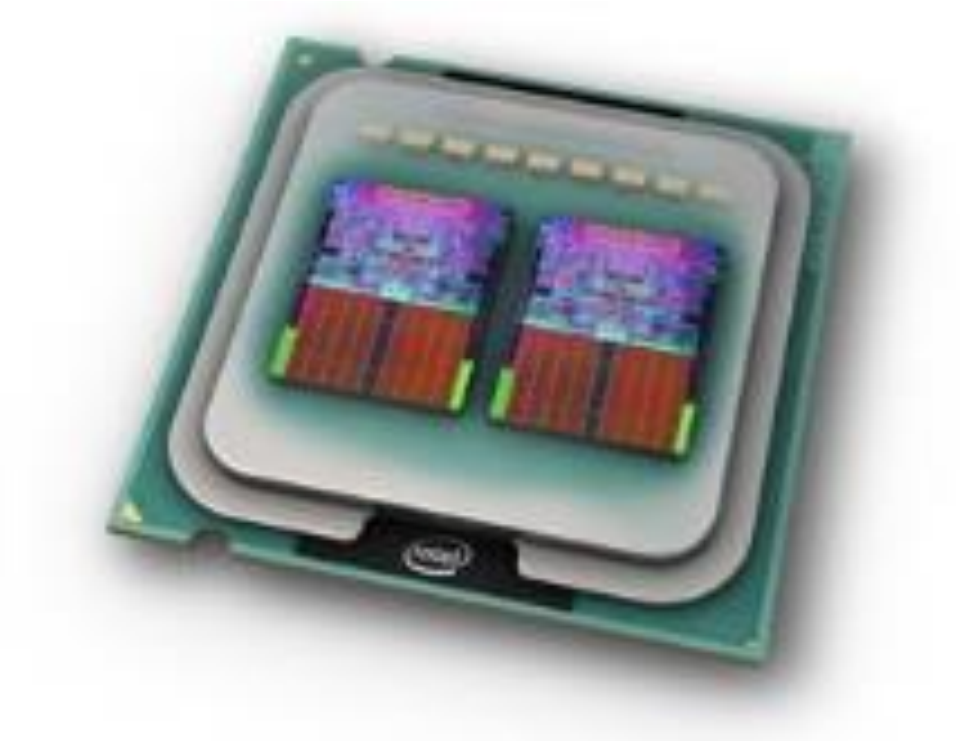
Das Ziel

Scalability + Concurrency

Motivation


Concur-
rency
Programming
model
providing
Fault-
tolerance
Scalability

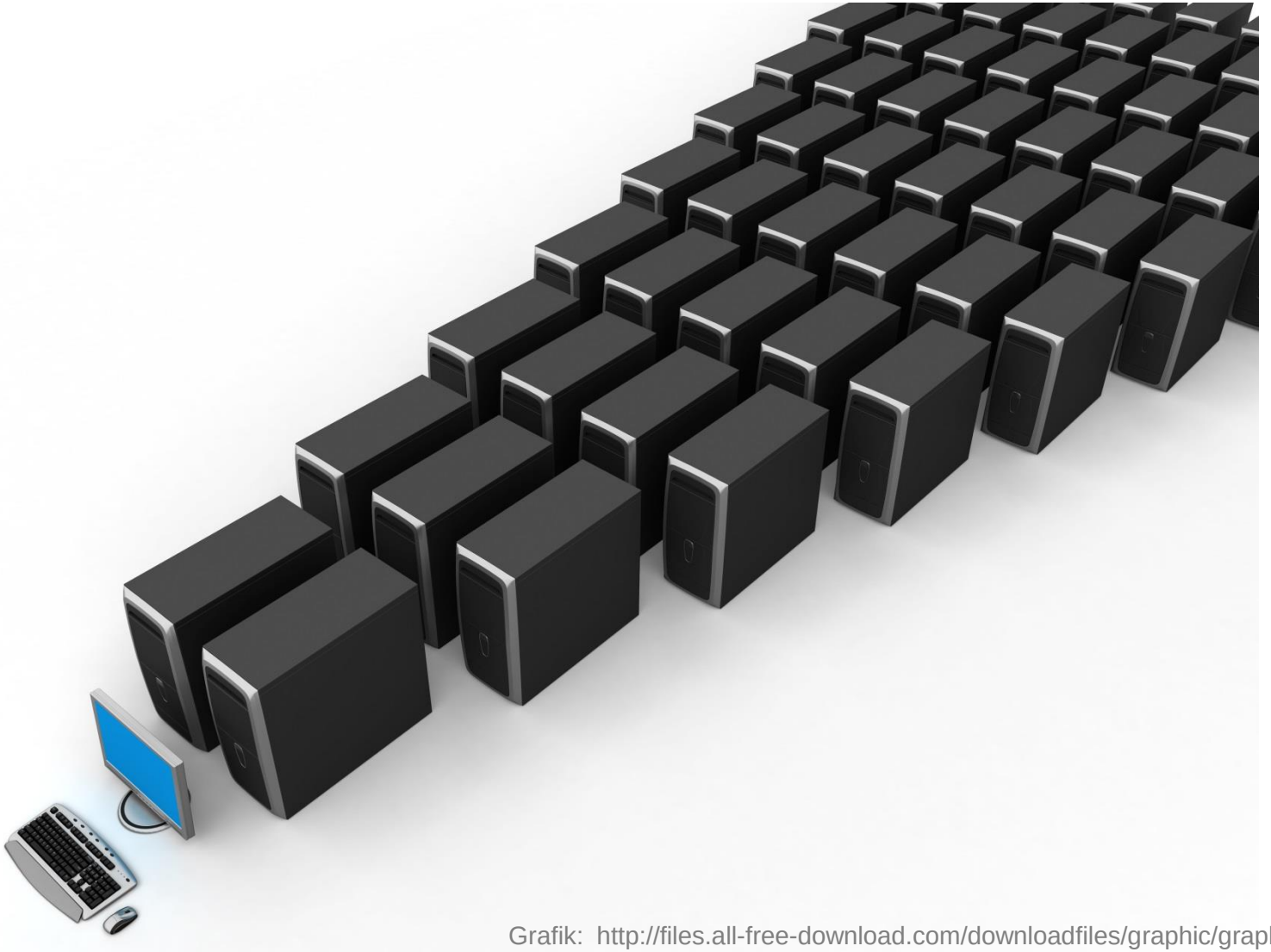
Vertikale Skalierung „Scaling up“



Multicore-Prozessoren

Grafik: http://www.freeware-download.com/blog/wp-content/uploads/2009/03/2006_int_qua_ohne_40x30.jpg

Horizontale Skalierung
Scaling out



Grafik: http://files.all-free-download.com/downloadfiles/graphic/graphic_18/3d_computer_network_connection_picture_6_168630.zip

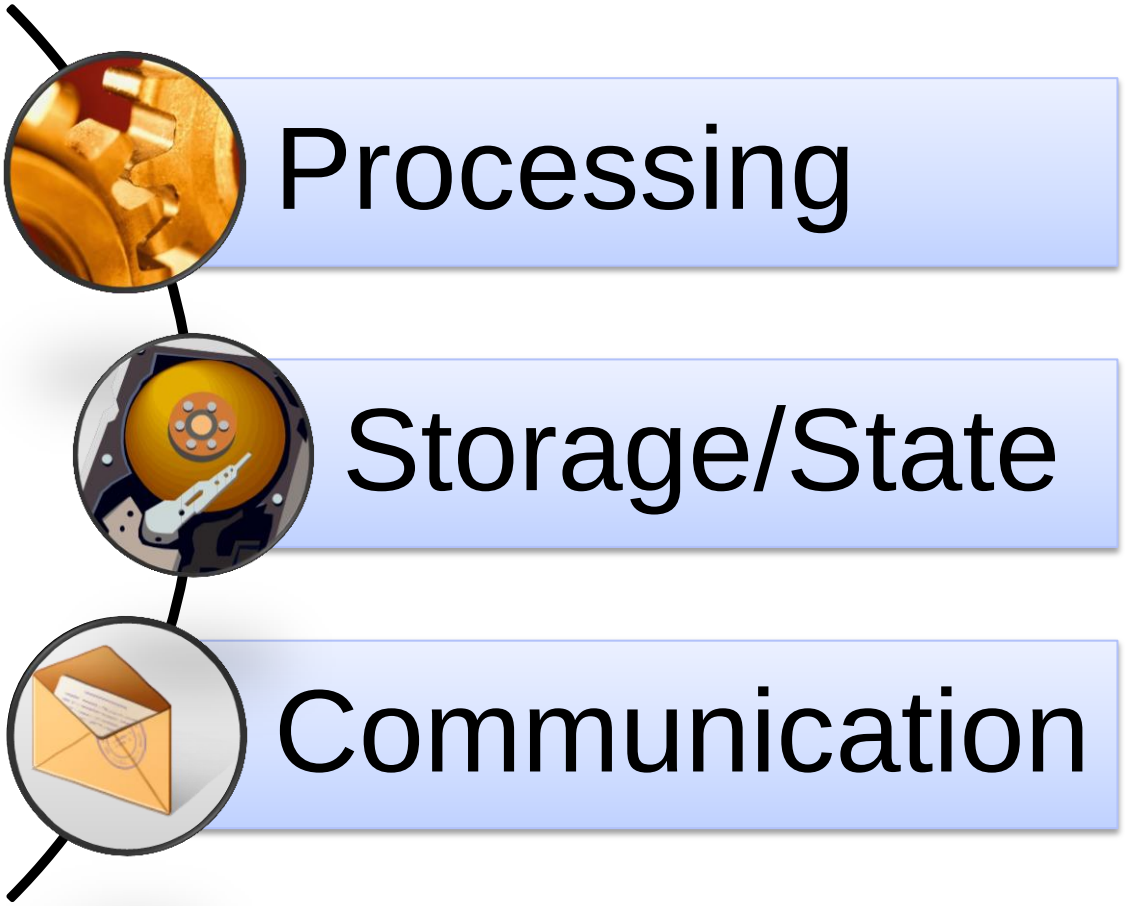
Actor model

Was ist ein Aktor

Actor:

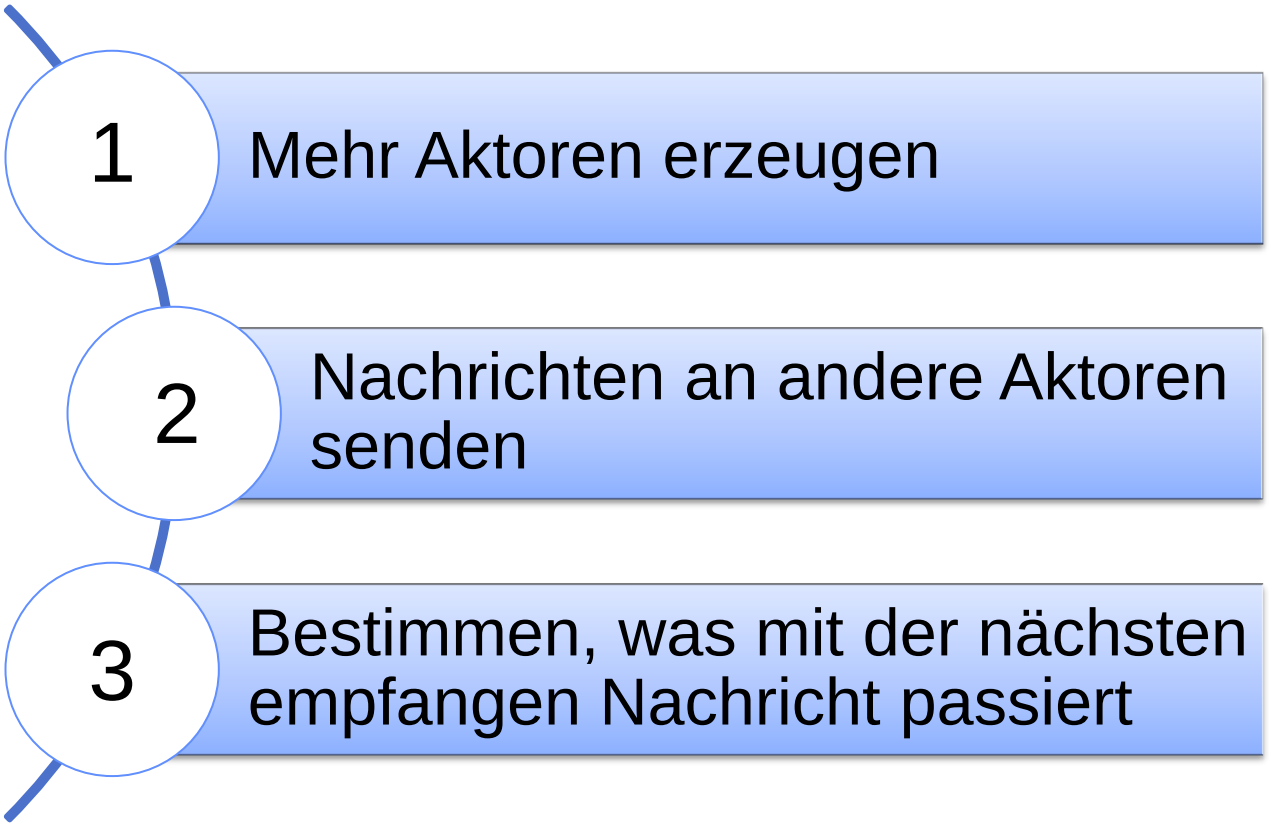
Unit of Computation

(Carl Hewitt 1973)



Was kann ein Actor?

Axiome



Eigenschaften

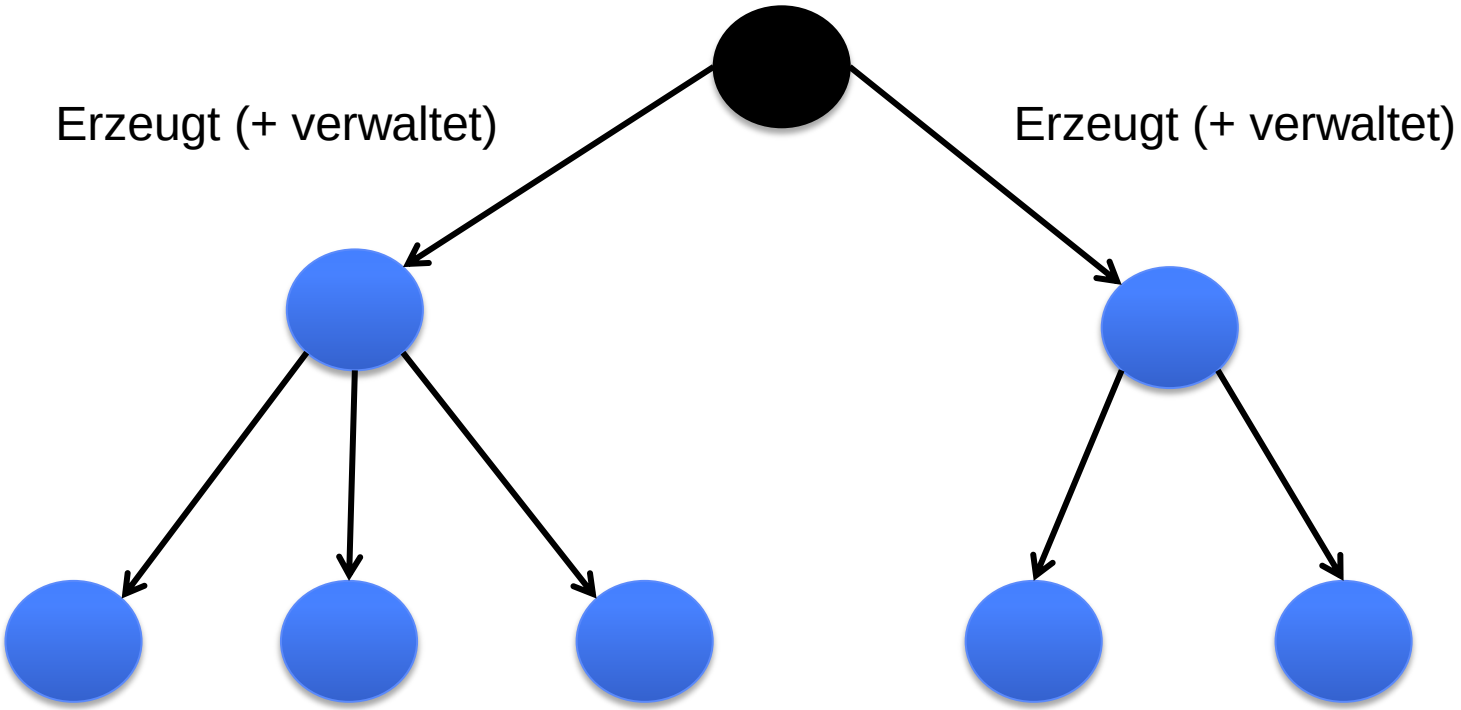
- Isolated black box
- Single threaded illusion
- Lose gekoppelt - Message passing
- „Shared nothing“ architecture
- Reactive
- Lockfree
- Selbstheilend

Anwendungsszenarien

- Thread
  - Concurrency + State (ohne State genügen Threads)
- Object + Message Passing
- asynchron
- Callback / Listener
- Service
- Router
- JEE SessionBean aber leichtgewichtig
- Analogie zu Servlet
- Finite State Machine

One actor is no actor!

Actors come in systems!

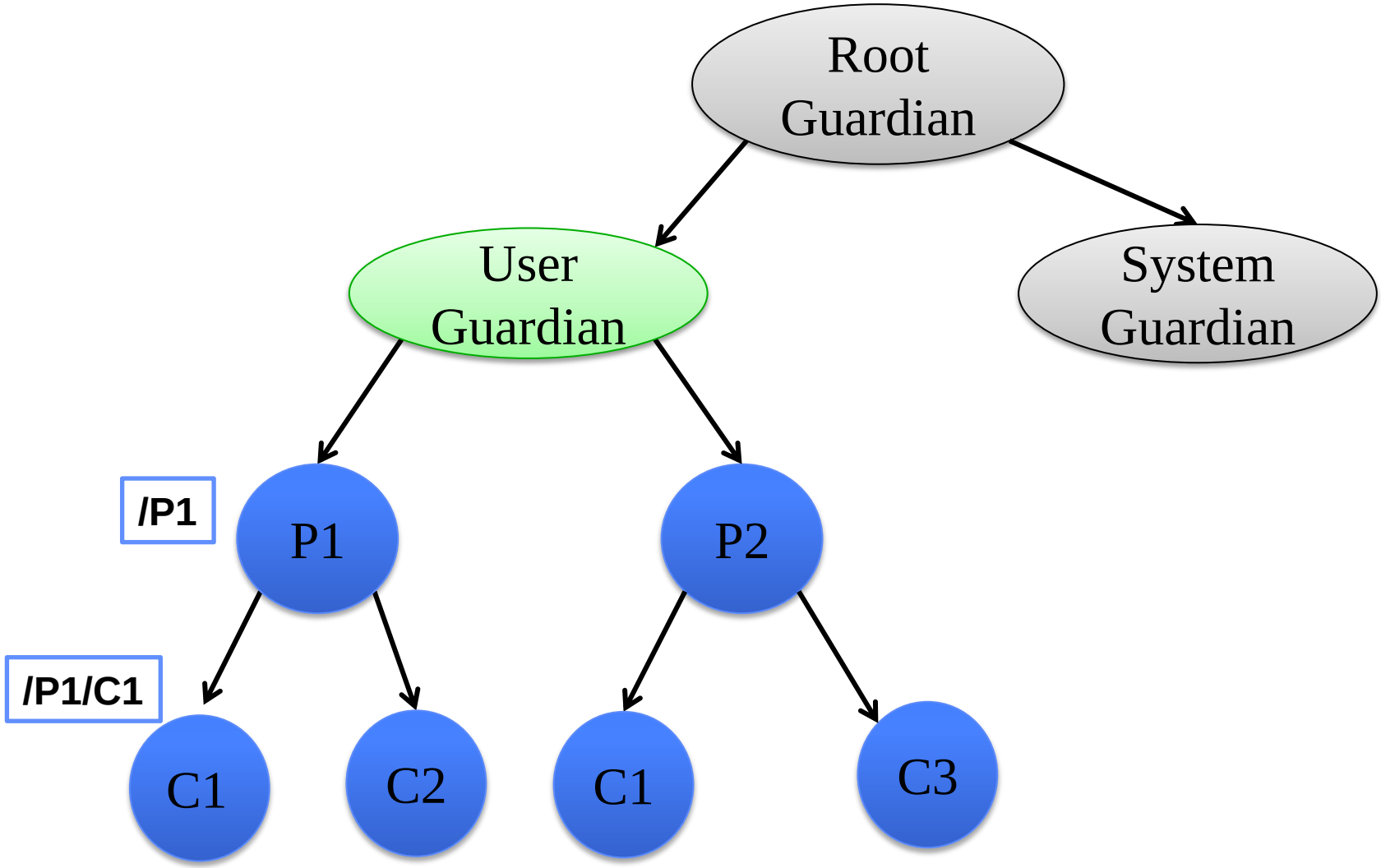


Quelle: https://plus.google.com/+AkkaIoFlow/posts

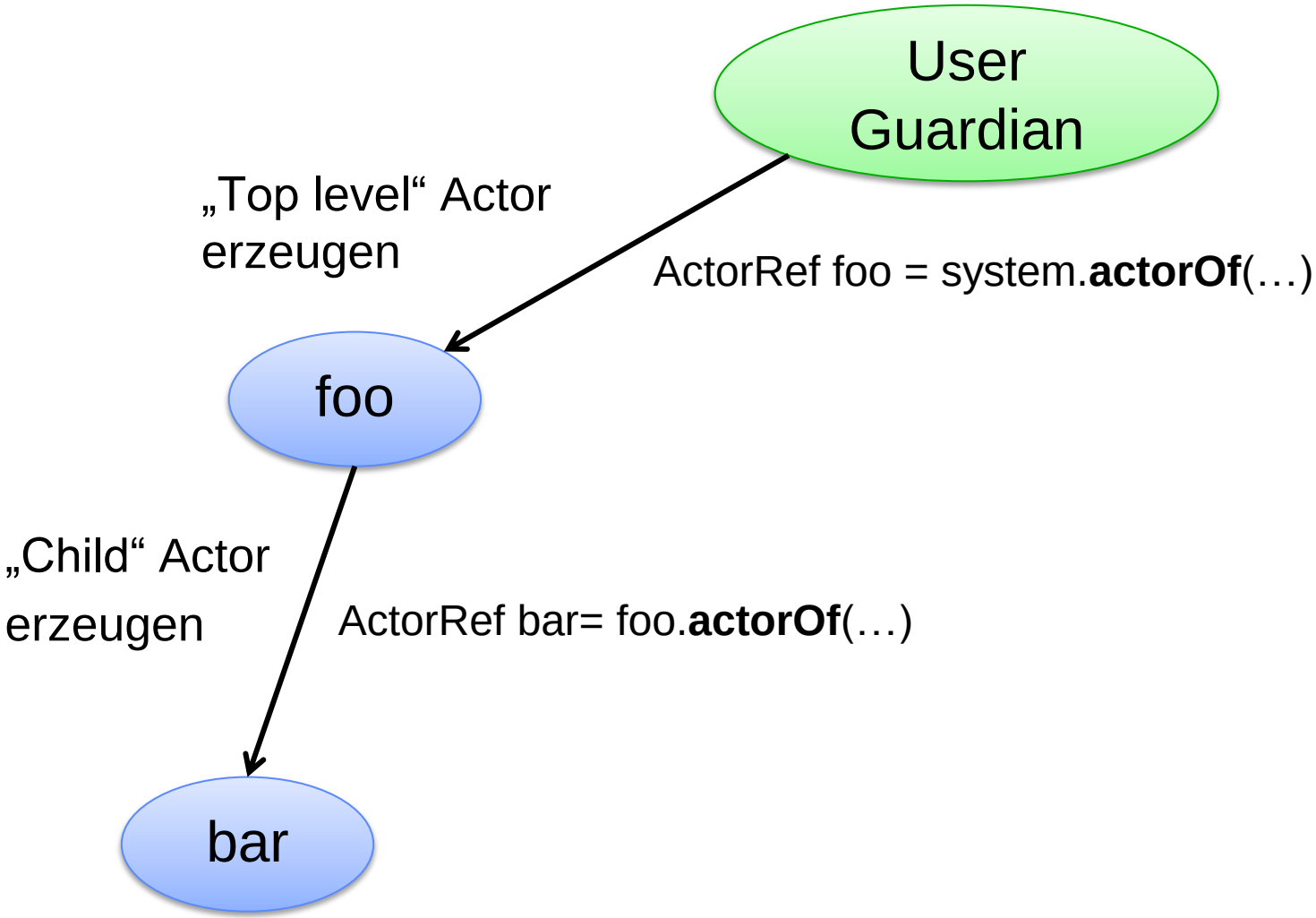
Akka - Intro

- Toolkit zur Nutzung des Actor models
- basiert auf Scala
- Java API
- Verteilbar
- Fault tolerance per Supervision
- Lightweight
  - 2,7 Mio / GB
  - > 50 Mio Message/s on single box

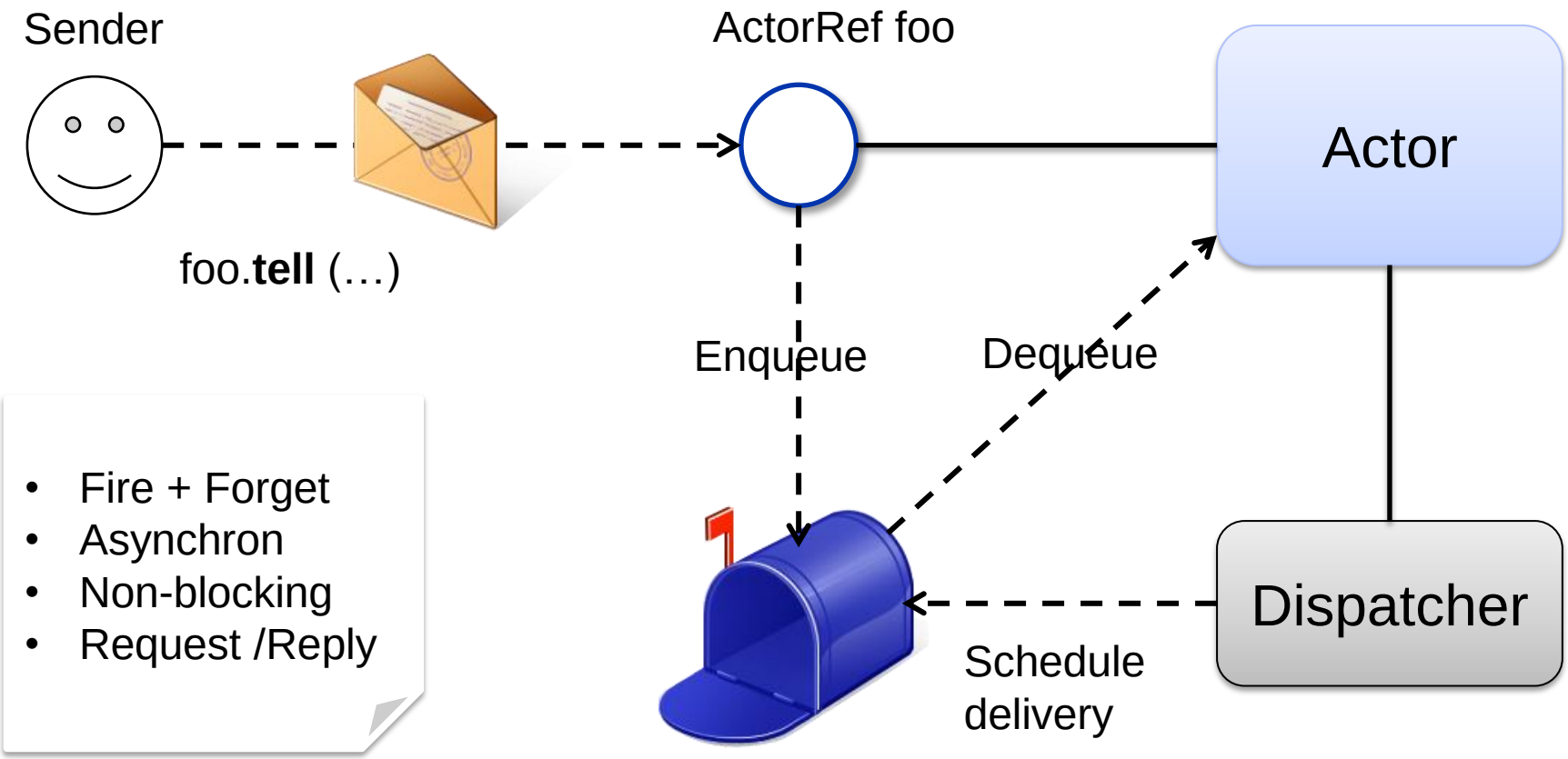
Akka ActorSystem



Create


User Guardian
„Top level“ Actor erzeugen
ActorRef foo = system.**actorOf**(…)
foo
„Child“ Actor erzeugen
ActorRef bar= foo.**actorOf**(…)
bar

Send Message



- Fire + Forget
- Asynchron
- Non-blocking
- Request /Reply

Become

- Verhalten ändern
  - Verhalten f. nächste Message festlegen
  - Ändert Interface + Verhalten
- Aufhebung mit unbecome()
- Stapelbar (Push + Pop)

Fehlertoleranz + Robustheit

Supervision (Fault-tolerance)

Fehler anderer Aktoren behandeln

- Fortsetzen
- Delegieren
- Actor neustarten
- Actor Stoppen

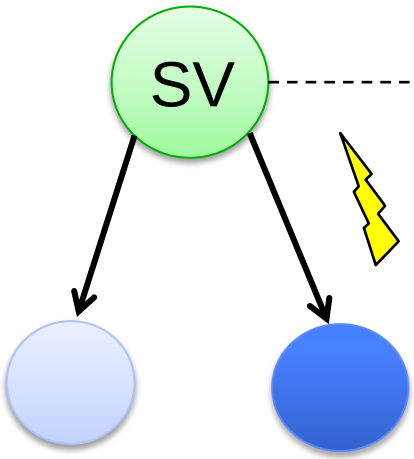
Schalenmodell:

Error Kernel übernimmt,
wenn sich keiner kümmert

SupervisorStrategy

- Definiert Fehlerverhalten

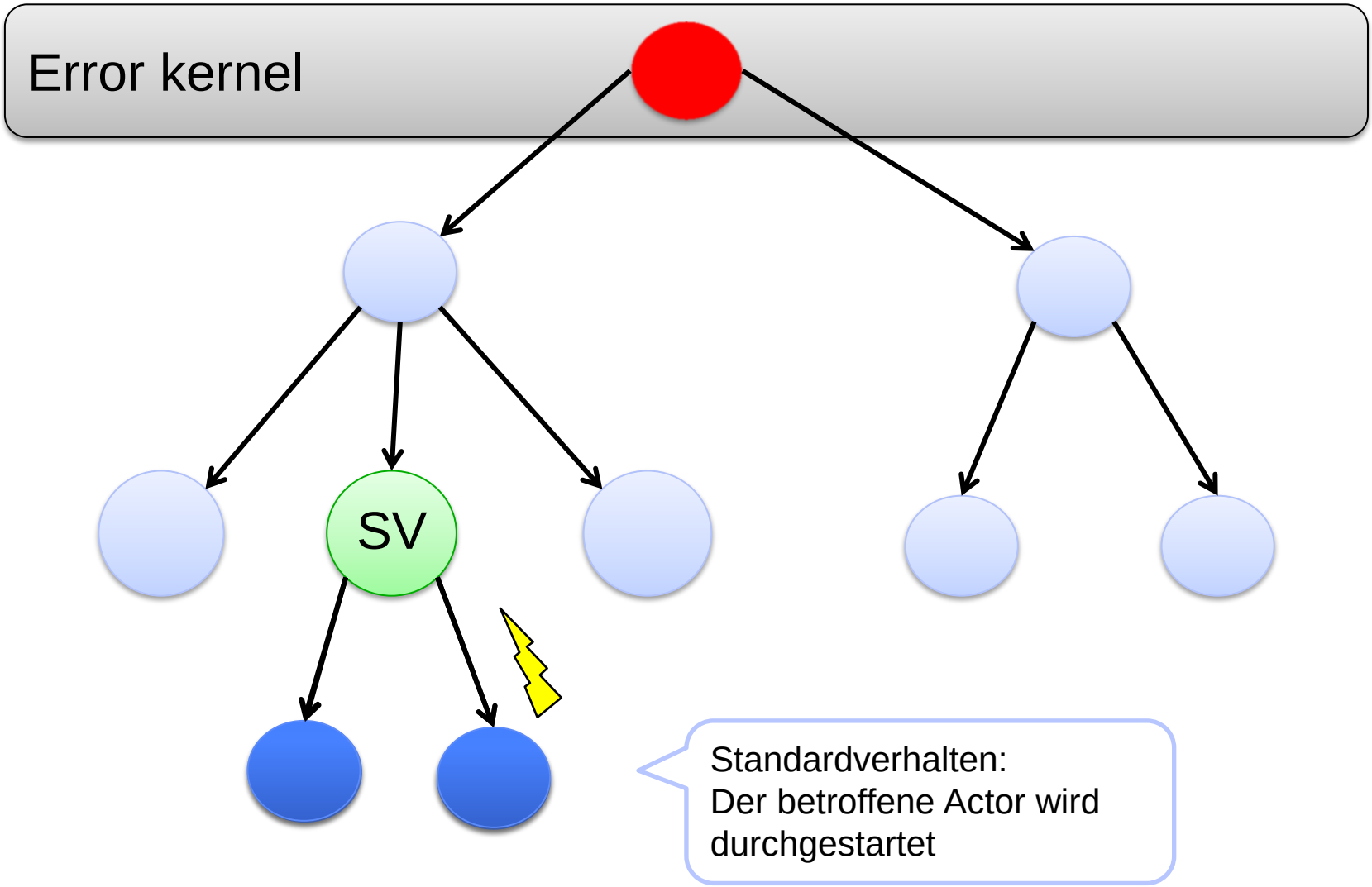
  Standardverhalten: Neustart des Actors



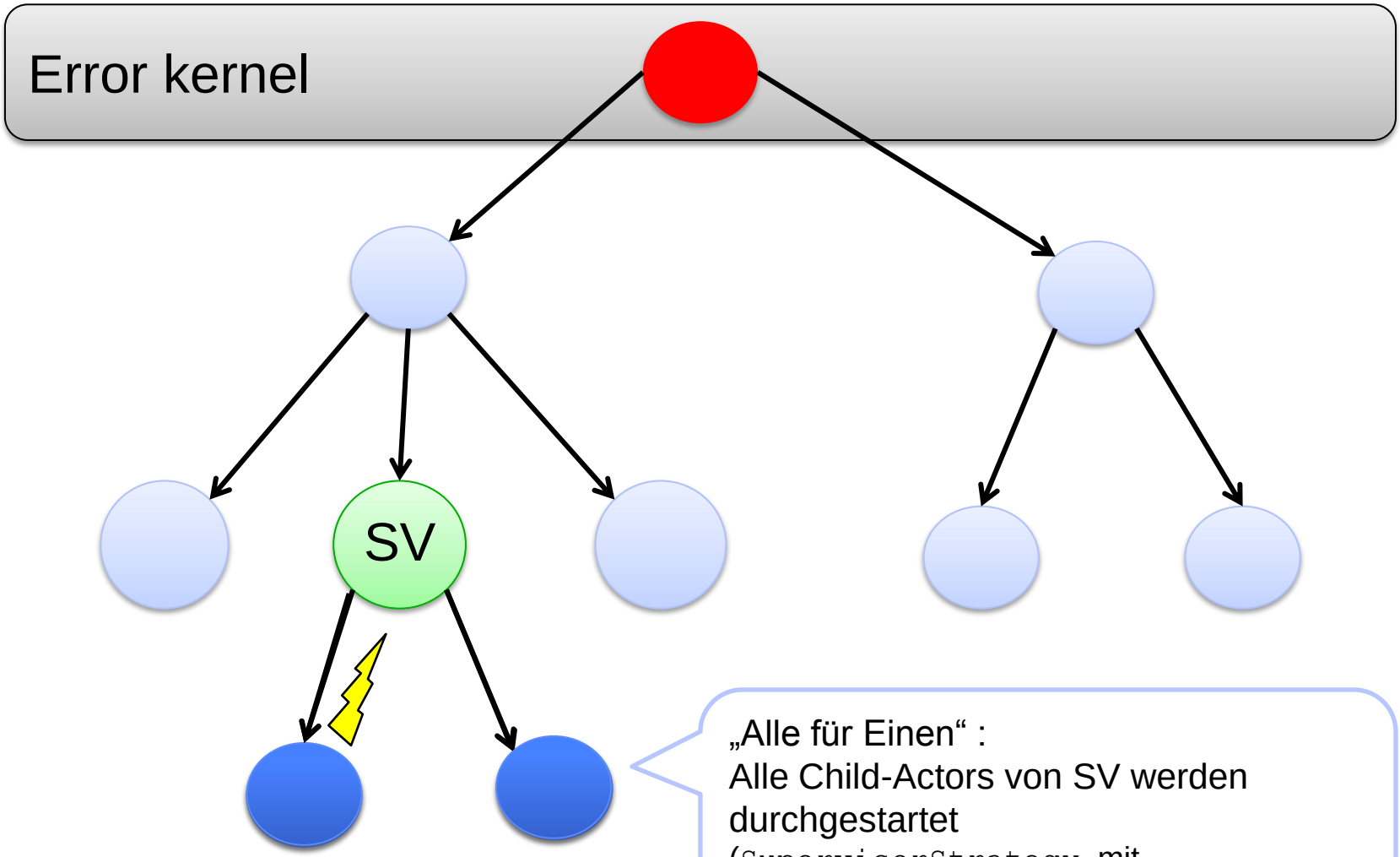

```
private static SupervisorStrategy strategy
  = new OneForOneStrategy (-1, Duration.Inf(),
      new Function<Throwable, Directive>() {

        public Directive apply(Throwable t) {
          if (t instanceof SomeException) {

                return resume();

          } else {
                return escalate();
          }
        }
      });
```

```
@Override
public SupervisorStrategy supervisorStrategy() {
      return strategy;
}
```

Selbstheilung
One for one strategy



Selbstheilung
All for one strategy



„Alle für Einen“ :
Alle Child-Actors von SV werden durchgestartet
(`SupervisorStrategy` mit `OneForOneStrategy` überschrieben)