# Exercises for distributed calculation with Actors

TECHNISCHE UNIVERSITÄT DARMSTADT

In this exercise a sample program is developed that uses Actors to calculate the value of $e$.

The concept is to initialize the calculation by a master Actor that creates worker Actors. The master splits the work into chunks which are distributed to the workers. After that it waits for the results of the workers, calculates the result and sends it to a printer which prints it out and stops the ActorSystem.

## Problem 1 Calculation of $e$

### Problem 1.1 Package and Messages

Create a folder according your name in excercises/tutorial. Copy the AkkaTutorial folder into it. Extends the packagede.tkip.akkatutorial.

Create case classes for the needed messages:

- Calculate - send to the master to initialize the calculation
- Work - send from master to the workers and contains the chunk that should be calculated
- Result - send from the workers to the master with the result of their calculation
- eApproximation - contains the result; send from master to the printer

### Problem 1.2 The WorkerActor

Create the class Worker that extends the Actor trait and define the receive method. When a Work message arrives the respective chunk should be calculated and send bach to the Master. As a reminder: $e = \sum_{n=0}^{\infty} \frac{1}{n!}$

### Problem 1.3 Der MasterActor

Write the class Worker. It's constructor should already start some Workers. Additionally, they should be grouped by a load-balancing router. The Master should distribute the calculation to the workers and send the approximation to the printer.

These imports could help you:

```
import akka.actor.{Actor, ActorRef, Props}
import akka.routing.RoundRobinRouter
```

The constructor may have these parameters:

- nrOfWorkers – how much workers to start
- nrOfMessages – how much chunks to send
- nrOfElements – how big the chunks are

### Problem 1.4 The PrinterActor

Write a PrinterActor that prints the content of a received eApproximation and stops the ActorSystem.

### Problem 1.5 Bootstrap Application

Extend the App object with the creation of the needed elements and start the execution.

## Problem 2 Extension with Futures

Now you should use the Akka ask pattern to receive the Results. First of all have a look at the ”Futures und Routing” document to refresh your memory.

### Problem 2.1 Ask-Pattern

Your current solution sends chunks from within a loop. Adapt that loop in a way that it uses the ask pattern and returns Futures with the type Result.

Reduce the sequence of Futures to a single Future[Result].

### Problem 2.2 PipeTo-Pattern

Use the pipeTo-Pattern to send the reduced result to the PrinterActor. Adapt the PrinterActor to receive Result messages instead of eApproximation.