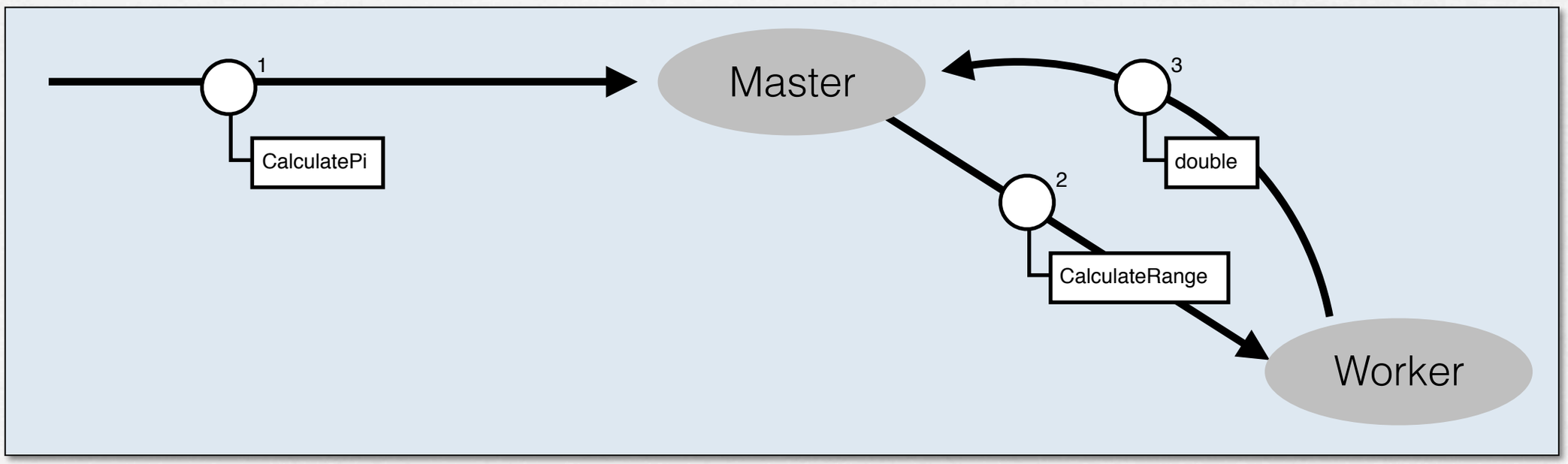
Akka.NET

Aufgabe 5: Pi Berechnung, Multitasking mit Router

Pi Berechnung

- Nur ein Aktor rechnet

$$\sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots = \frac{\pi}{4}.$$



verschachtelte Nachricht

- Kommando-Nachricht Klasse im Aktor
  - \+ weniger Dateien
  - \+ „Kohärenz“
  - – „Unruhe“ in Klassen
  - – Benutzung auf diesen Aktor beschränkt

```
worker.Tell(
  new Worker.CalculateRange()
);
```

```
/// Ein Worker errechnet die Summe über einen gegebenen Bereich...
public class Worker : ReceiveActor
{
    /// Kommando für die Berechnung...
    public class CalculateRange
    {
        public int From { get; private set; }
        public int To { get; private set; }

        public CalculateRange(int from, int to)
        {
            From = from;
            To = to;
        }
    }
```

Berechnung

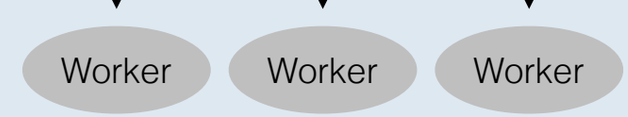
- via Kommando „CalculateRange“
- Berechnung ausführen
- Ergebnis an Absender zurück
- Request/Reply

```
/// Ein Worker errechnet die Summe über einen gegebenen Bereich...
public class Worker : ReceiveActor
{
    /// Kommando für die Berechnung...
    public class CalculateRange...

    public Worker()
    {
        Console.WriteLine("  Starting Worker {0}", Self.Path.Name);

        Receive<CalculateRange>(r => CalculateRangeHandler(r));
    }

    /// Behandlung des Kommandos...
    private void CalculateRangeHandler(CalculateRange range)
    {
        // rechnen
        double sum = 0;
        var factor = Math.Pow(-1, range.From - 1);
        int denom = 2 * range.From - 1;
        for (int k = range.From; k <= range.To; k++)
            sum += (factor *= -1) / (denom += 2);

        // und zurückschicken
        Sender.Tell(sum * 4);
    }
}
```

Nachteil

- Ein Worker Aktor macht alles
- Eine CPU zu 100% belastet, rest frei
- Aufteilung auf mehrere Aktoren

$$\sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots = \frac{\pi}{4}.$$



Router allgemein

- Nachrichten annehmen
- Entscheidung treffen
- Nachricht weiterleiten

Aber...

- dann ginge die Antwort an den Router
- unnötiger Nachrichtenversand

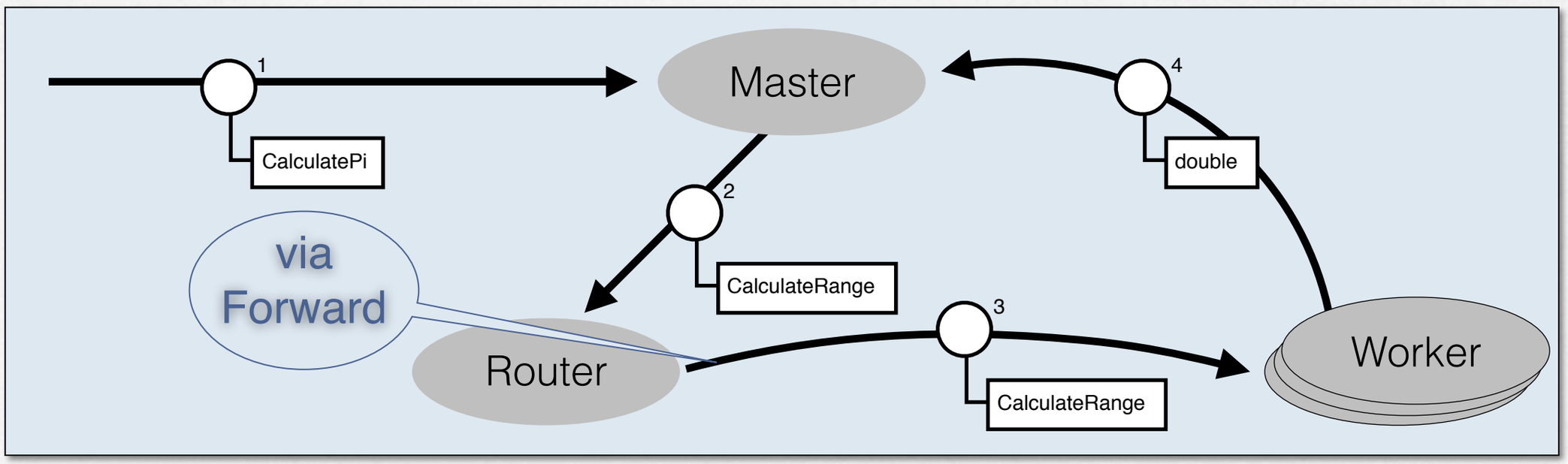
- Lösung: Weiterleitung

```
// innerhalb Router
worker.Forward(…)
```

```
/// Behandlung des Kommandos...
private void CalculateRangeHandler(CalculateRange range)
{
    // rechnen
    double sum = 0;
    var factor = Math.Pow(-1, range.From - 1);
    int denom = 2 * range.From - 1;
    for (int k = range.From; k <= range.To; k++)
        sum += (factor *= -1) / (denom += 2);

    // und zurückschicken
    Sender.Tell(sum * 4);
}
```

Router im Einsatz

- Berechnung Delegieren auf Router
- Router entscheidet wer rechnet
- Router leitet weiter, Worker antwortet



unser Router im Detail

- Master: Router anlegen

```
router = Context.ActorOf(Props.Create<Router>(nrWorkers));
```

- Master: an Router

```
foreach (var calculateRange in Distribute(calculatePi))
    router.Tell(calculateRange);
```

- Router: empfangen
- Router: weiterleiten

```
public class Router : ReceiveActor
{
    // Anzahl der Worker
    private int nrWorkers;

    // Round Robin Pool an Worker Instanzen
    private IList<IActorRef> workers;

    // nächster zu nutzender Worker im Pool
    private int nextWorker;

    public Router(int nrWorkers)
    {
        this.nrWorkers = nrWorkers;
        workers = new List<IActorRef>();
        nextWorker = 0;

        for (var i = 0; i < nrWorkers; i++)
            workers.Add(
                Context.ActorOf(Props.Create<Worker>())
            );

        // einfach mal alles weiter leiten :-)
        Receive<object>(msg =>
            workers[nextWorker++ % nrWorkers].Forward(msg)
        );
    }
}
```

es geht einfacher

- via Code

```
worker = Context.ActorOf(
    Props.Create<Worker>()
         .WithRouter(new RoundRobinPool(nrWorkers))
);
```

```
foreach (var calculateRange in Distribute(calculatePi))
    worker.Tell(calculateRange);
```

- via Config

```
akka {
    actor.deployment {
        /ConfigRoutingMaster/Worker {
            router = round-robin-pool
            nr-of-instances = 3
        }
    }
```

```
worker = Context.ActorOf(
    Props.Create<Worker>()
         .WithRouter(FromConfig.Instance),
    "Worker"
);
```

```
foreach (var calculateRange in Distribute(calculatePi))
    worker.Tell(calculateRange);
```

Akka.NET Routertypen

- Pools
  - erzeugen und verwalten ihre Aktoren
- Groups
  - bekommen Aktoren zur Benutzung

Akka.NET Routerstrategien

- RoundRobin – der Reihe nach
- Broadcast – gleichzeitig
- Random – zufällig
- ConsistentHashing - inhaltsabhängig
- TailChopping – bei Fehler nochmal
- ScatterGatherFirstCompleted – erste Antwort
- SmallestMailbox – „wer eher frei ist“

Projekt „PiCalculator“

- Ein-CPU Berechnung ausführen, evtl. Konstante „Series“ verändern
- Andere Strategie wählen und staunen
- Zahl der Worker erhöhen bis über die Anzahl der CPUs. Was passiert?
- Zahl der Arbeitspakete extrem erhöhen. Was passiert?