AW 2
Software Concurrency
Related Work

Kjell Otto

26.05.2010

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Gliederung

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

AW1 Rückblick

- Wo liegt das Problem?
  - Trend der Hardwareentwicklung
  - Programmierkonzepte langsamer als Hardwareentwicklung
  - Komplexität wächst mit der Aufgabe

AW1 Rückblick

- Wo liegt das Problem?
  - Trend der Hardwareentwicklung
  - Programmierkonzepte langsamer als Hardwareentwicklung
  - Komplexität wächst mit der Aufgabe
- Was bietet Clojure im Hinblick auf Lösungen?
  Wie funktionieren diese Lösungen?
  - Agent - asynchrone Updates
  - Atom - unkoordinierte, synchrone Updates
  - Ref - koordinierte, synchrone Updates [3]

AW1 Rückblick

- Wo liegt das Problem?
  - Trend der Hardwareentwicklung
  - Programmierkonzepte langsamer als Hardwareentwicklung
  - Komplexität wächst mit der Aufgabe
- Was bietet Clojure im Hinblick auf Lösungen?
  Wie funktionieren diese Lösungen?
  - Agent - asynchrone Updates
  - Atom - unkoordinierte, synchrone Updates
  - Ref - koordinierte, synchrone Updates [3]
- Was bieten andere Projekte?

Grundlagen

Man Unterscheidet grundsätzlich zwischen: [2]

- Multicore
  - lokale CPUs
  - lokaler RAM
  - gemeinsamer Speicher

Grundlagen

Man Unterscheidet grundsätzlich zwischen: [2]

- Multicore
  - lokale CPUs
  - lokaler RAM
  - gemeinsamer Speicher
- Multinode
  - verteilte CPUs
  - verteiler RAM
  - getrennter Speicher

Parallel Programming Models

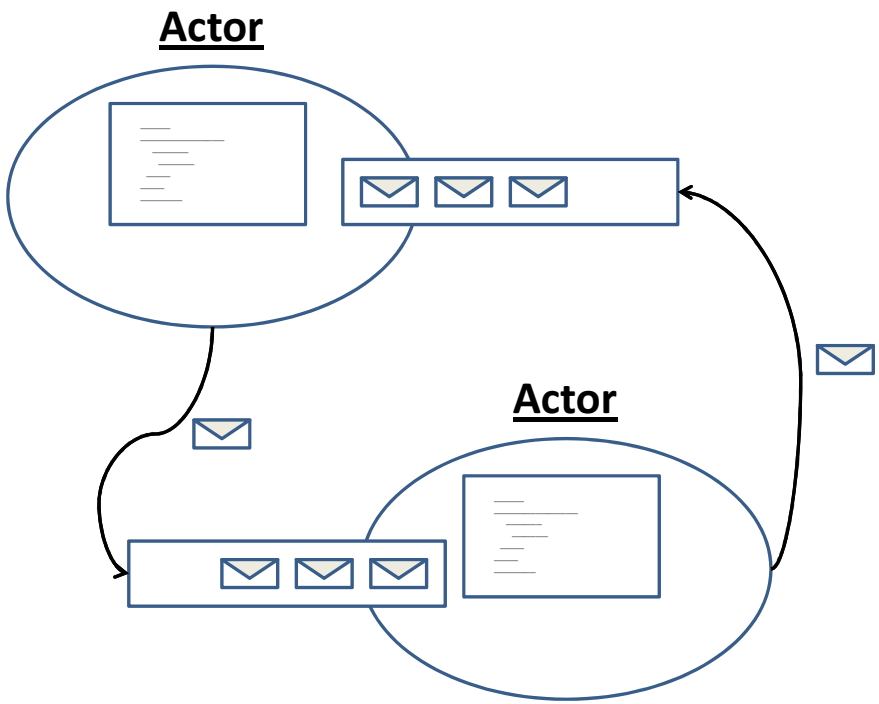
- Modelle sind prinzipiell unabhängig von der zugrunde liegenden Hardware.

Parallel Programming Models

- Modelle sind prinzipiell unabhängig von der zugrunde liegenden Hardware.
- Alternative Programmiermodelle:
  - Message Passing mit Aktoren [8]
  - Software Transactional Memory [1]
  - MapReduce [4]

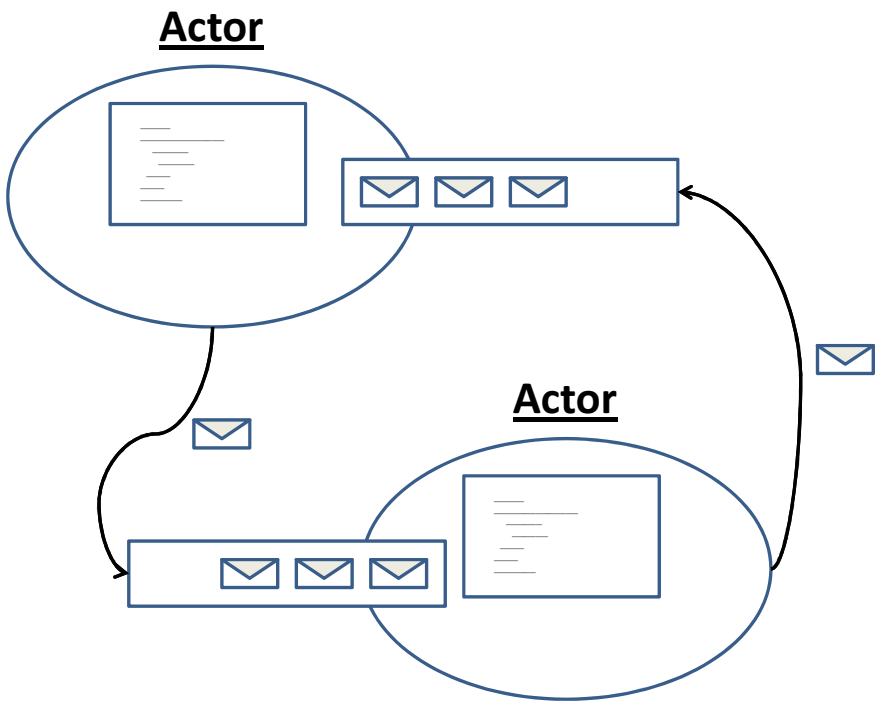
Parallel Programming Models

- Message Passing mit Aktoren



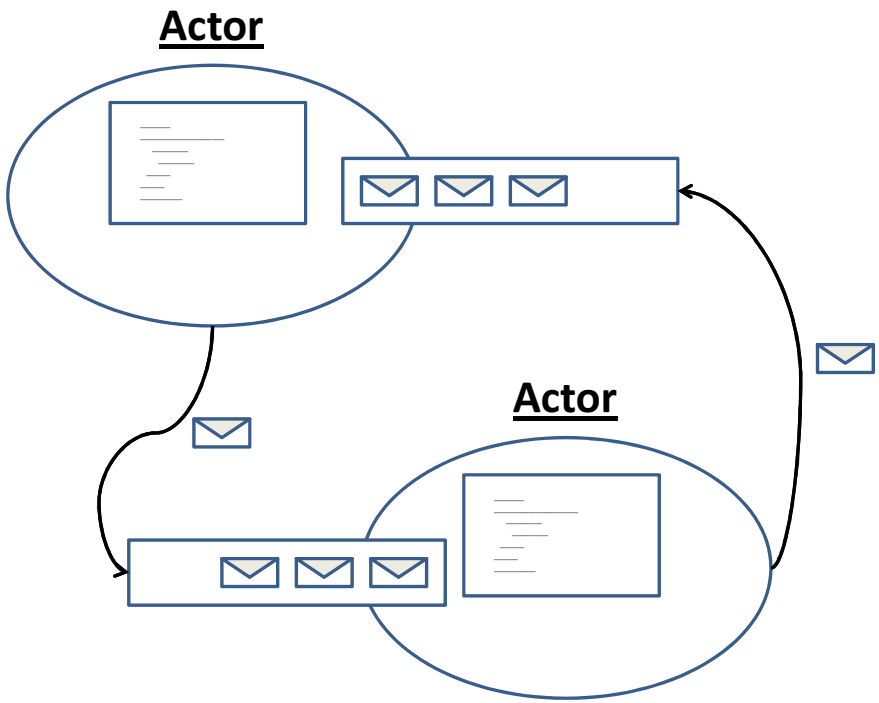
Parallel Programming Models

- Message Passing mit Aktoren
  - jeder hat seinen eigenen Speicher



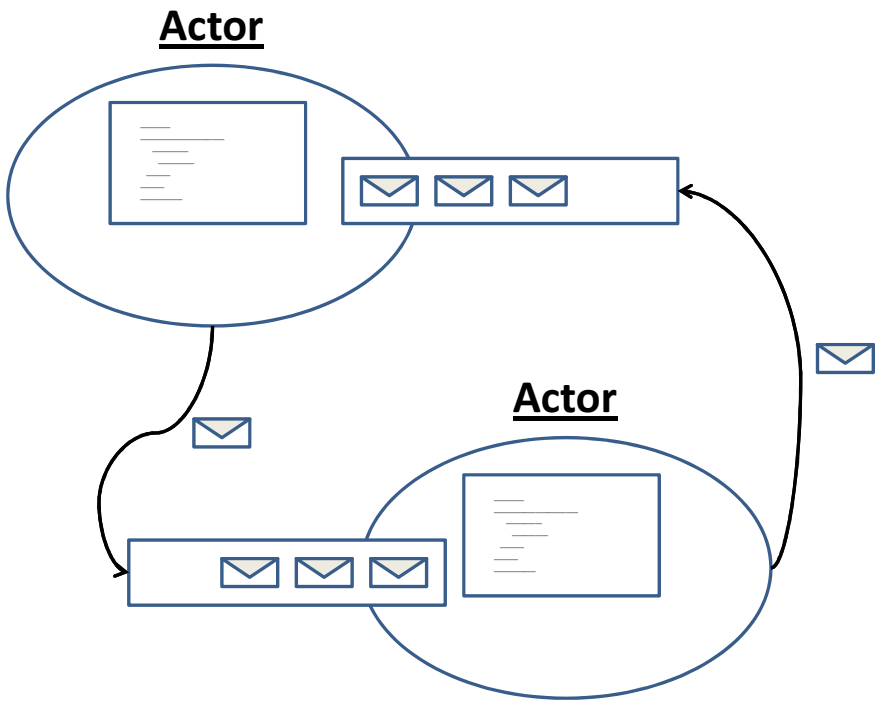
Parallel Programming Models

- Message Passing mit Aktoren
  - jeder hat seinen eigenen Speicher
  - Speicher muss nicht geschützt werden



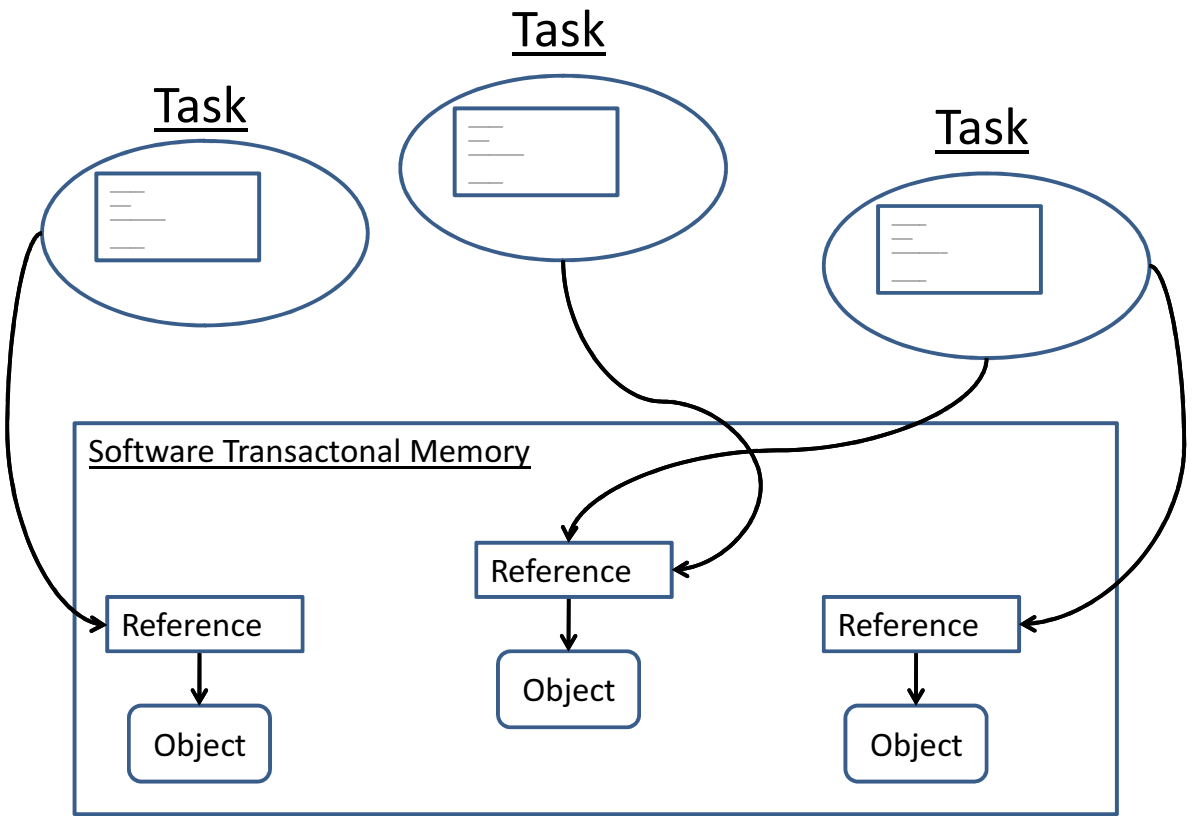
Parallel Programming Models

- Message Passing mit Aktoren
  - jeder hat seinen eigenen Speicher
  - Speicher muss nicht geschützt werden
  - Synchronisation durch das Prinzip



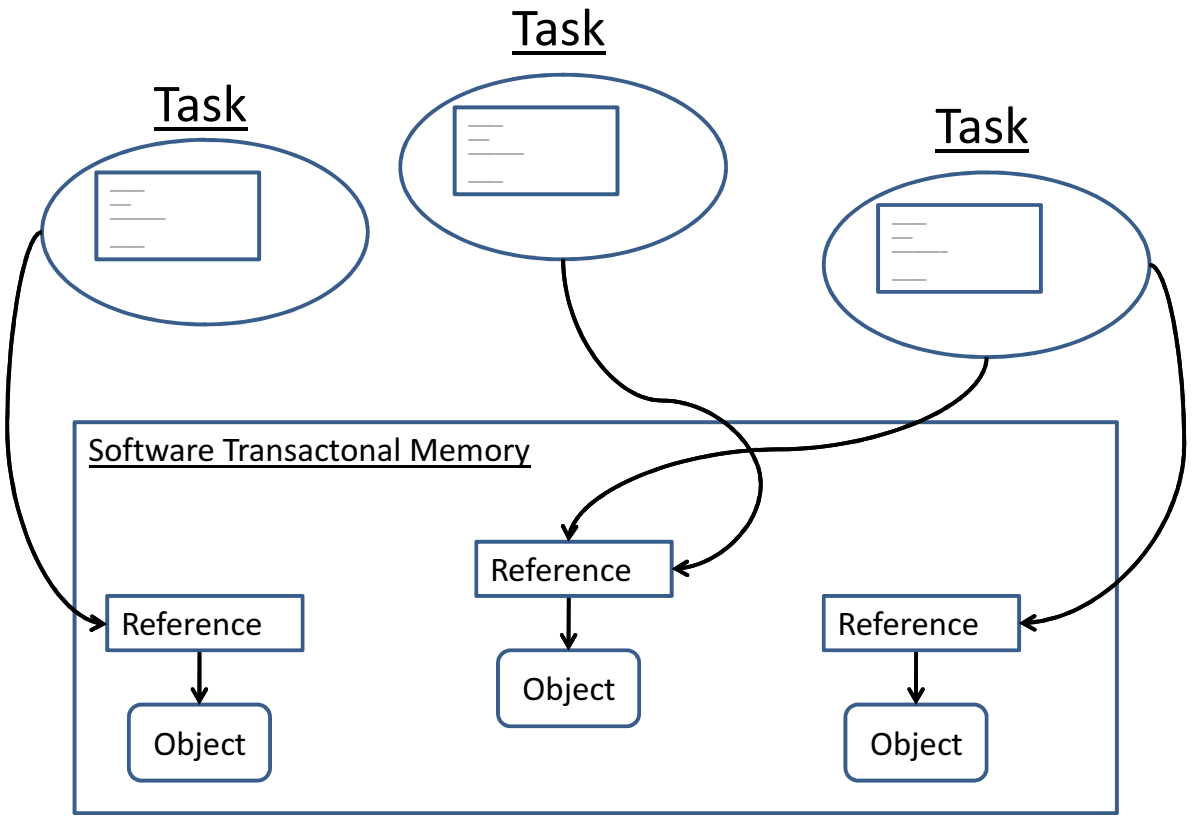
Parallel Programming Models

- Software Transactional Memory



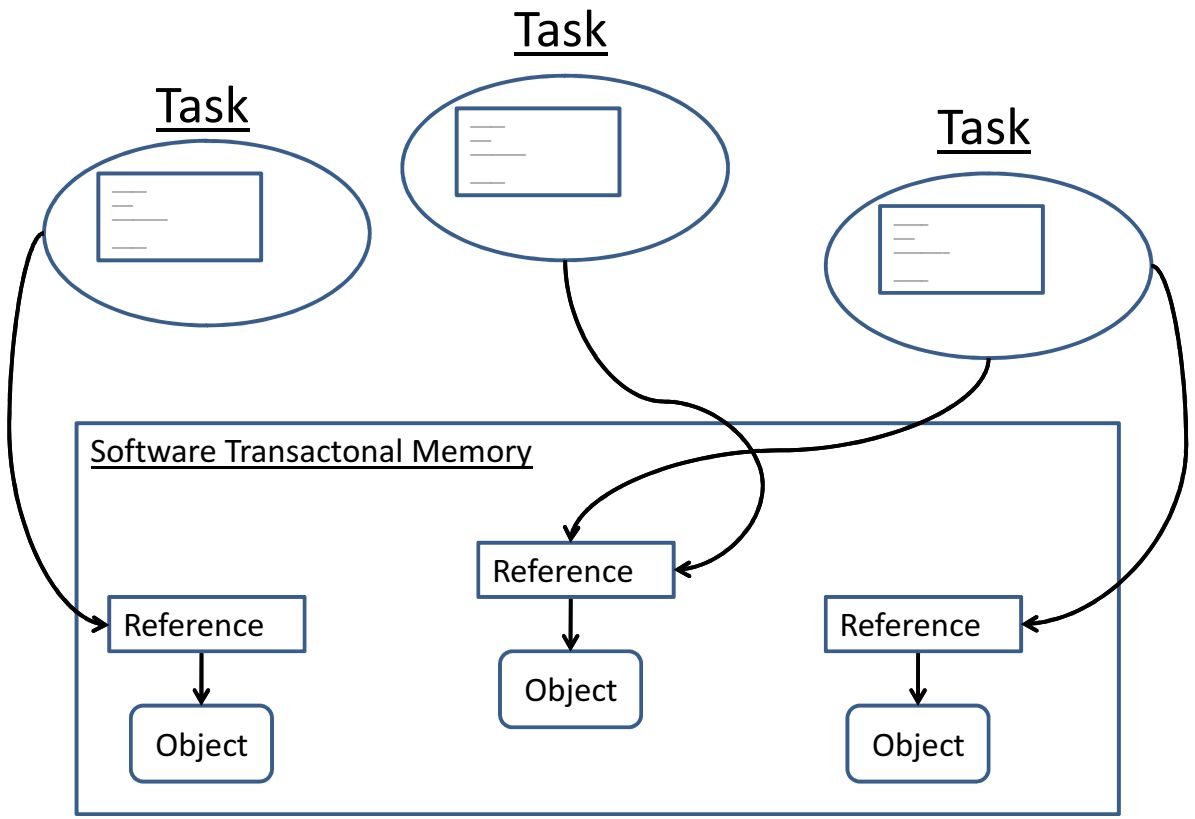
Parallel Programming Models

- Software Transactional Memory
  - ein Speicher für alle



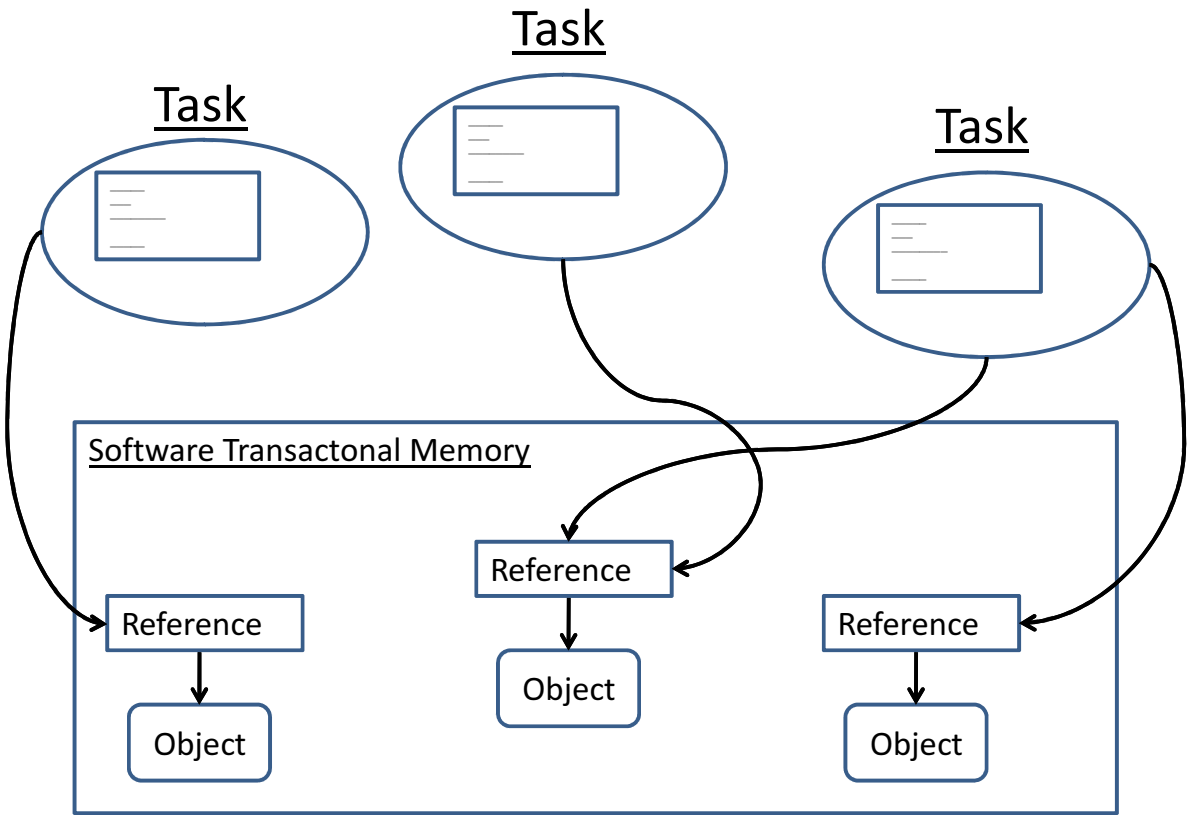
Parallel Programming Models

- Software Transactional Memory
  - ein Speicher für alle
  - Zugriff auf Ressourcen in „transactions“



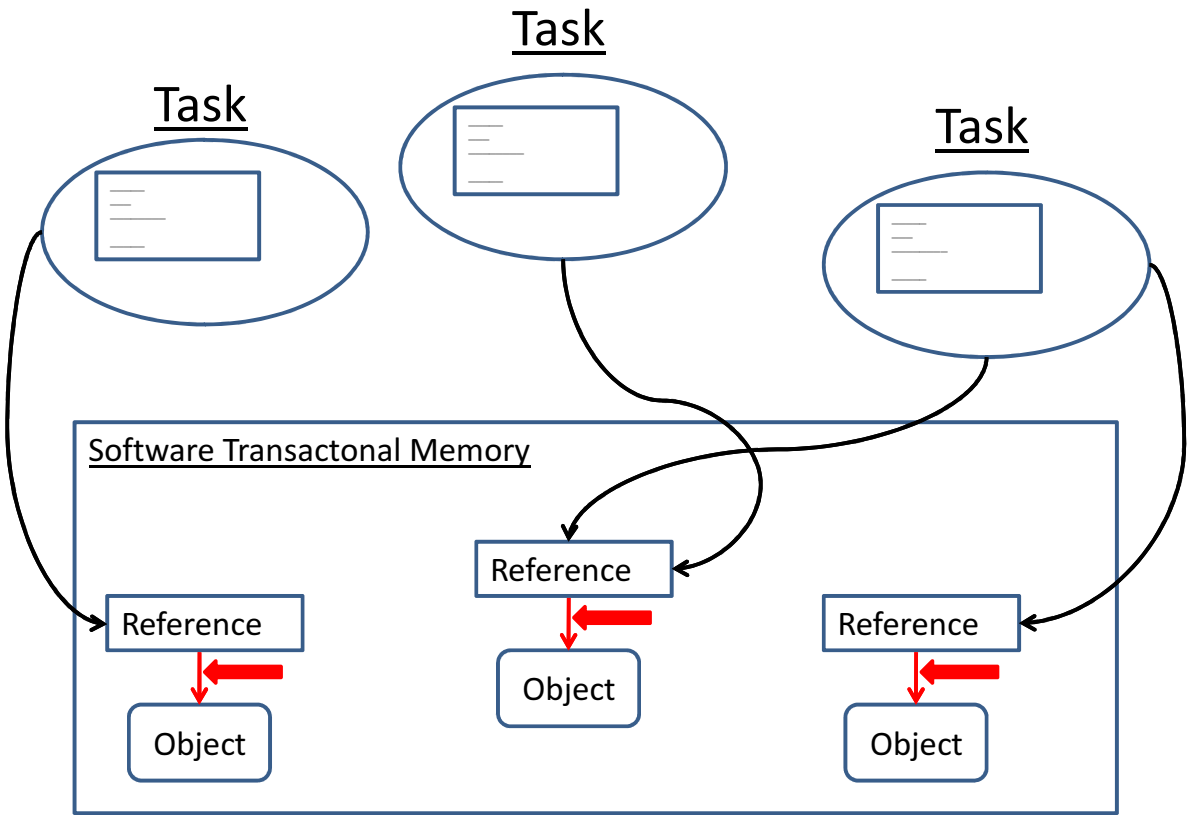
Parallel Programming Models

- Software Transactional Memory
  - ein Speicher für alle
  - Zugriff auf Ressourcen in „transactions“
  - automatische Synchronisation



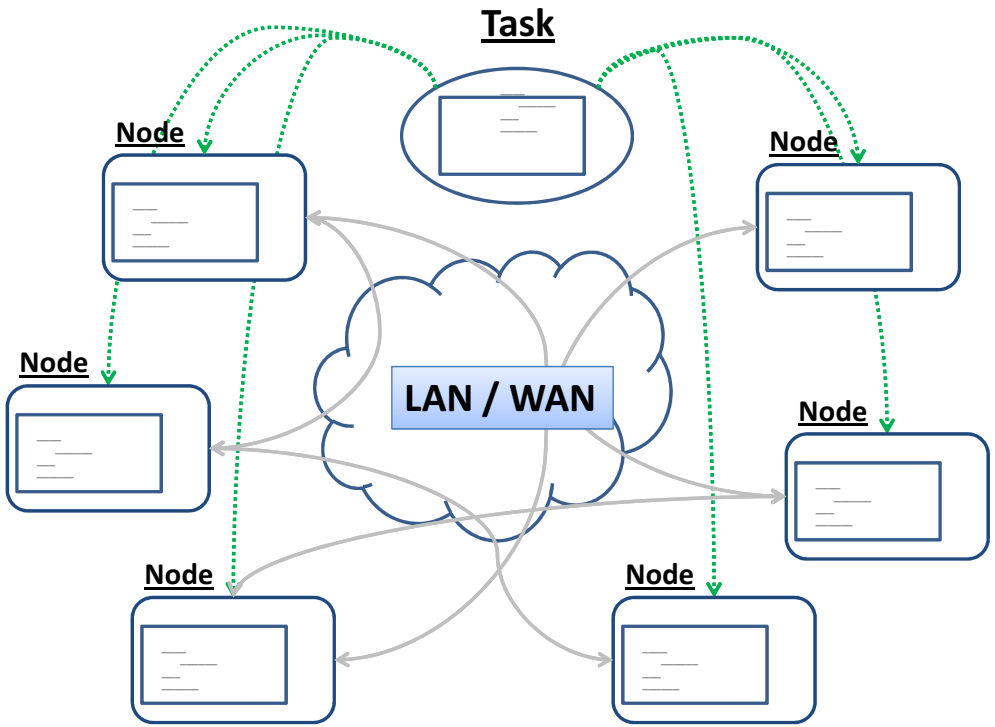
Parallel Programming Models

- Software Transactional Memory
  - ein Speicher für alle
  - Zugriff auf Ressourcen in „transactions“
  - automatische Synchronisation



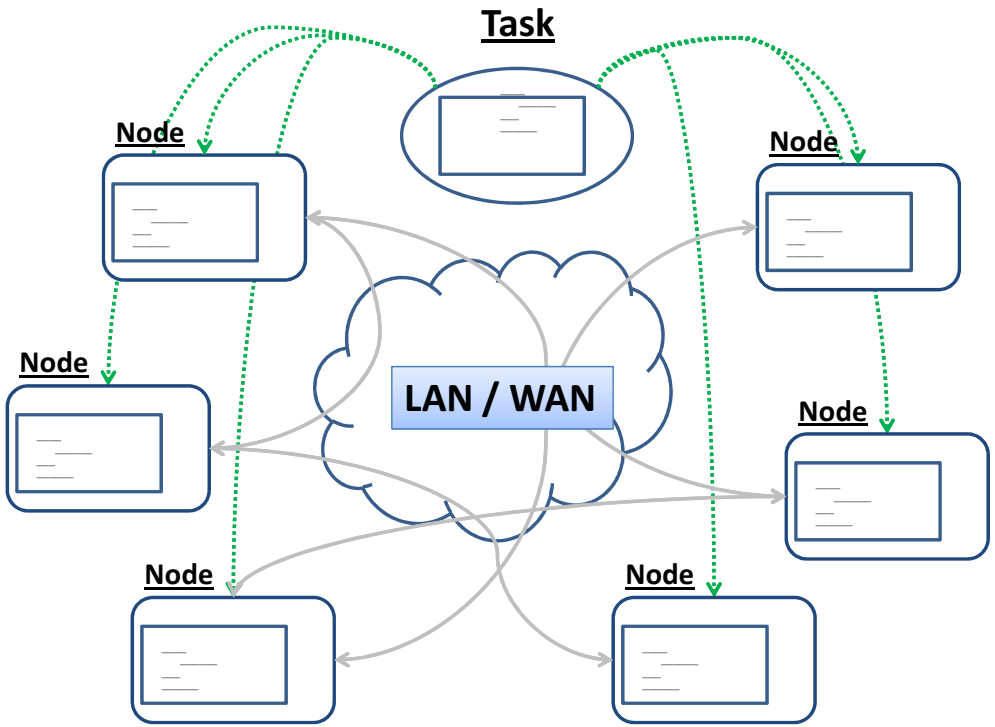
Parallel Programming Models

- Grid Computing



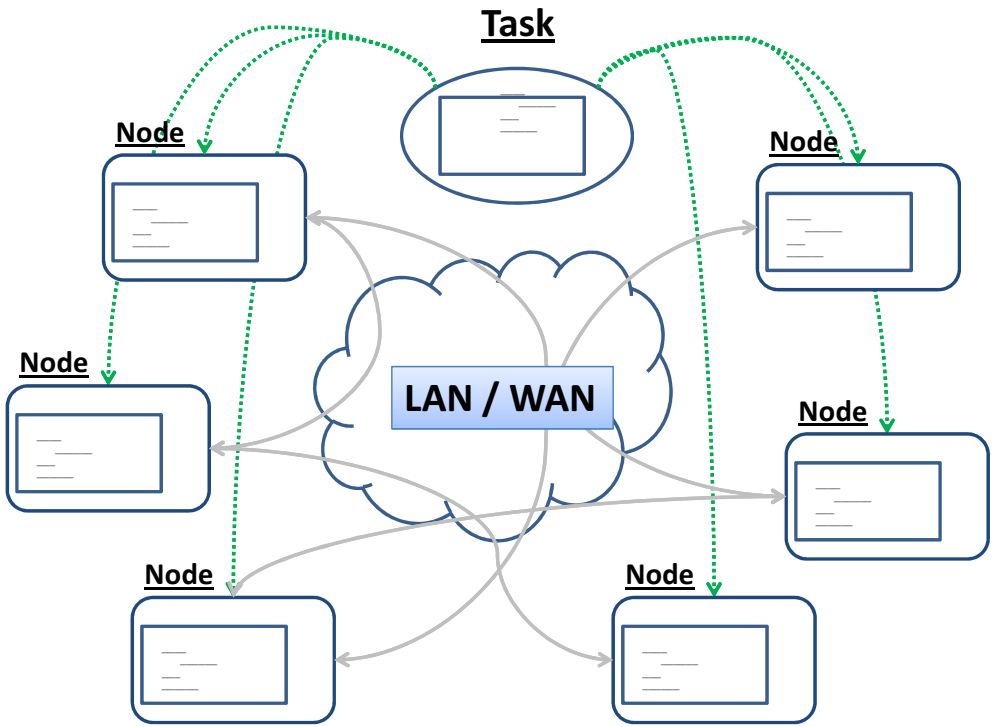
Parallel Programming Models

- Grid Computing
  - verteilter Speicher für Daten



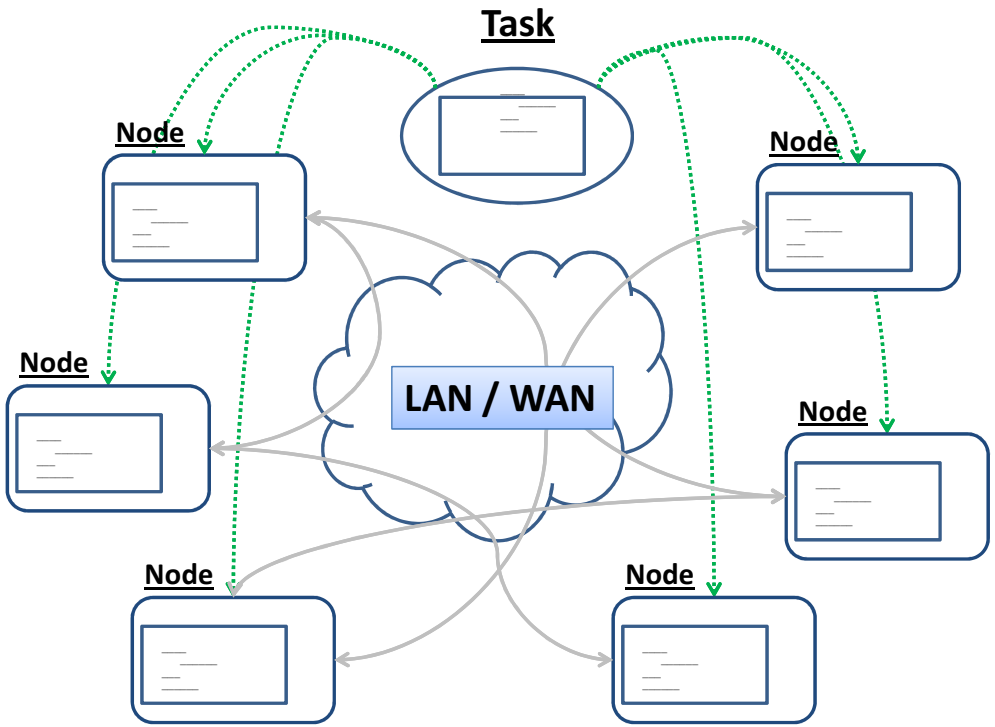
Parallel Programming Models

- Grid Computing
  - verteilter Speicher für Daten
  - meist SIMD (single instruction, multiple data) inklusive Verteilung der Daten



Parallel Programming Models

- Grid Computing
  - verteilter Speicher für Daten
  - meist SIMD (single instruction, multiple data) inklusive Verteilung der Daten
  - asynchrone Synchronisation (später genauer)

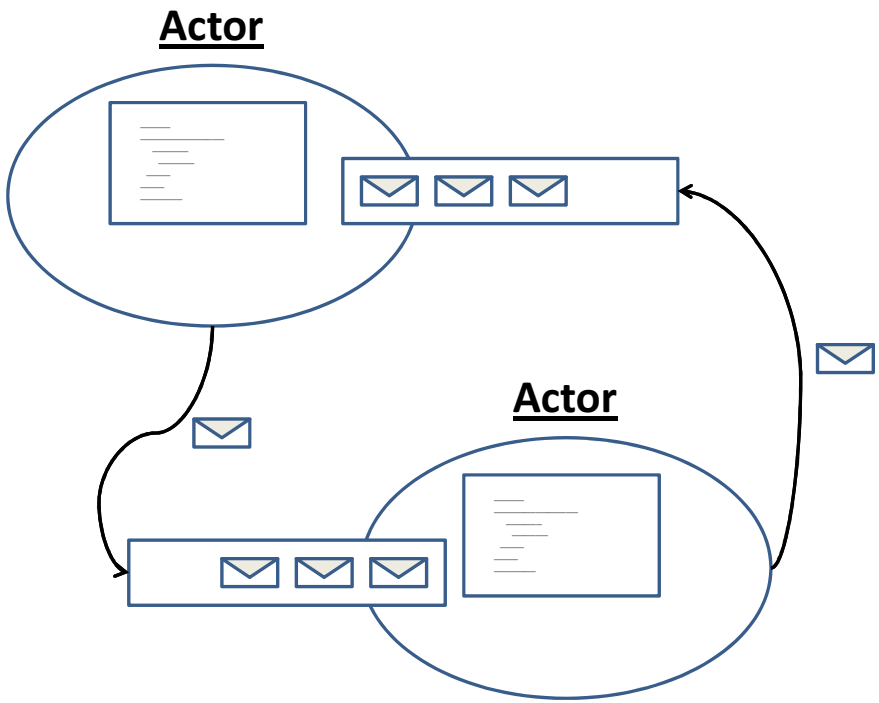


Projekte

- Actors (Scala)
  Lokales Prinzip auf Netzwerk übertragbar
- STM.NET (C#)
  Lokales Prinzip mit (bei Clojure maximalem) Automatismus
- Googles MapReduce (C++)
  Verteiltes Prinzip mit maximalem Automatismus

Actors

Scala Actors



Scala Actors - Merkmale

- „Light-weight“ Prozesse (Erlang Style) [6]

Scala Actors - Merkmale

- „Light-weight“ Prozesse (Erlang Style) [6]
- (A)synchrones Message Passing

Scala Actors - Merkmale

- „Light-weight“ Prozesse (Erlang Style) [6]
- (A)synchrones Message Passing
- Pattern matching [7]

Scala Actors - Merkmale

- „Light-weight“ Prozesse (Erlang Style) [6]
- (A)synchrones Message Passing
- Pattern matching [7]
- keine „transactions“

Scala Actors - Merkmale

- „Light-weight“ Prozesse (Erlang Style) [6]
- (A)synchrones Message Passing
- Pattern matching [7]
- keine „transactions“
- nicht Netzwerkübergreifend

Scala Actors - Funktionsweise

```
0  import scala.actors.Actor
   import scala.actors.Actor._

   class Ping(count: int, pong: Actor) extends Actor {
     def act() {

5      pong ! Ping

         receive {
           case Pong =>

10

15

20       }

     }
   }
```

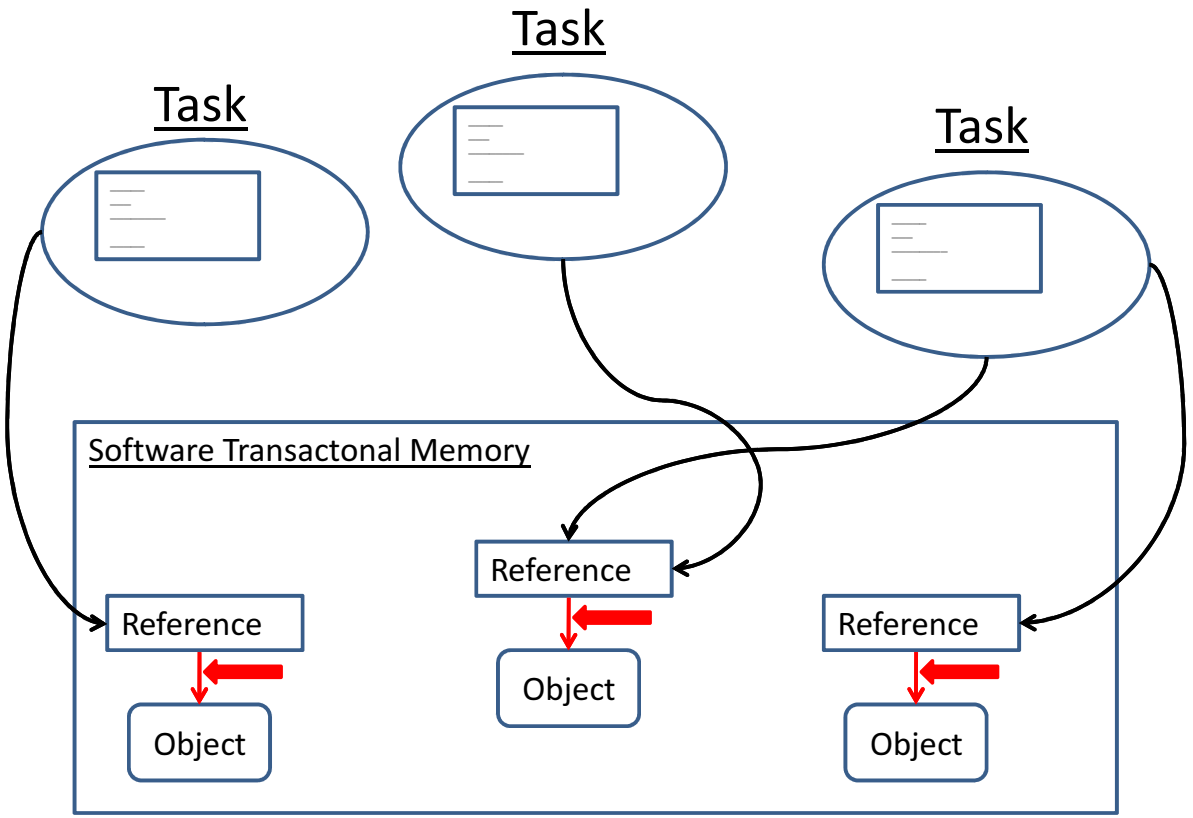
Scala Actors - Funktionsweise

```
0  import scala.actors.Actor
   import scala.actors.Actor._

   class Ping(count: int, pong: Actor) extends Actor {
     def act() {
5      var pingsLeft = count - 1
       pong ! Ping
       while (true) {
         receive {
           case Pong =>
10           if (pingsLeft % 1000 == 0)
               Console.println("Ping: pong")
             if (pingsLeft > 0) {
               pong ! Ping
               pingsLeft -= 1
15           } else {
               Console.println("Ping: stop")
               pong ! Stop
               exit()
             }
20       }
       }
     }
   }
```

Software Transactional Memory

STM.NET



STM.NET - Merkmale

- Teil der Common Language Runtime (CLR) von Microsoft

STM.NET - Merkmale

- Teil der Common Language Runtime (CLR) von Microsoft
- Führt den „Atomic.Do(() =>...)“ Codeblock ein

STM.NET - Merkmale

- Teil der Common Language Runtime (CLR) von Microsoft
- Führt den „Atomic.Do(() =>...)“ Codeblock ein
  - Transaktionen können geschachtelt werden (innere Lock-Scopes werden augeblendet)

STM.NET - Merkmale

- Teil der Common Language Runtime (CLR) von Microsoft
- Führt den „Atomic.Do(() =>...)“ Codeblock ein
  - Transaktionen können geschachtelt werden (innere Lock-Scopes werden augeblendet)
  - Bei Fehlern „partial-rollbacks“

STM.NET - Merkmale

- Teil der Common Language Runtime (CLR) von Microsoft
- Führt den „Atomic.Do(() =>...)“ Codeblock ein
  - Transaktionen können geschachtelt werden (innere Lock-Scopes werden augeblendet)
  - Bei Fehlern „partial-rollbacks“
- „Optimistic Locking“

STM.NET - Merkmale

- Teil der Common Language Runtime (CLR) von Microsoft
- Führt den „Atomic.Do(() =>...)“ Codeblock ein
  - Transaktionen können geschachtelt werden (innere Lock-Scopes werden augeblendet)
  - Bei Fehlern „partial-rollbacks“
- „Optimistic Locking“
- „retry“

STM.NET - Merkmale

- Teil der Common Language Runtime (CLR) von Microsoft
- Führt den „Atomic.Do(() =>...)“ Codeblock ein
  - Transaktionen können geschachtelt werden (innere Lock-Scopes werden augeblendet)
  - Bei Fehlern „partial-rollbacks“
- „Optimistic Locking“
- „retry“
- blockiert Zugriff auf Shared Variables komplett

STM.NET - Merkmale

- Teil der Common Language Runtime (CLR) von Microsoft
- Führt den „Atomic.Do(() =>...)“ Codeblock ein
  - Transaktionen können geschachtelt werden (innere Lock-Scopes werden augeblendet)
  - Bei Fehlern „partial-rollbacks“
- „Optimistic Locking“
- „retry“
- blockiert Zugriff auf Shared Variables komplett
- verbietet falschen Zugriff nicht [5]

STM.NET - Funktionsweise

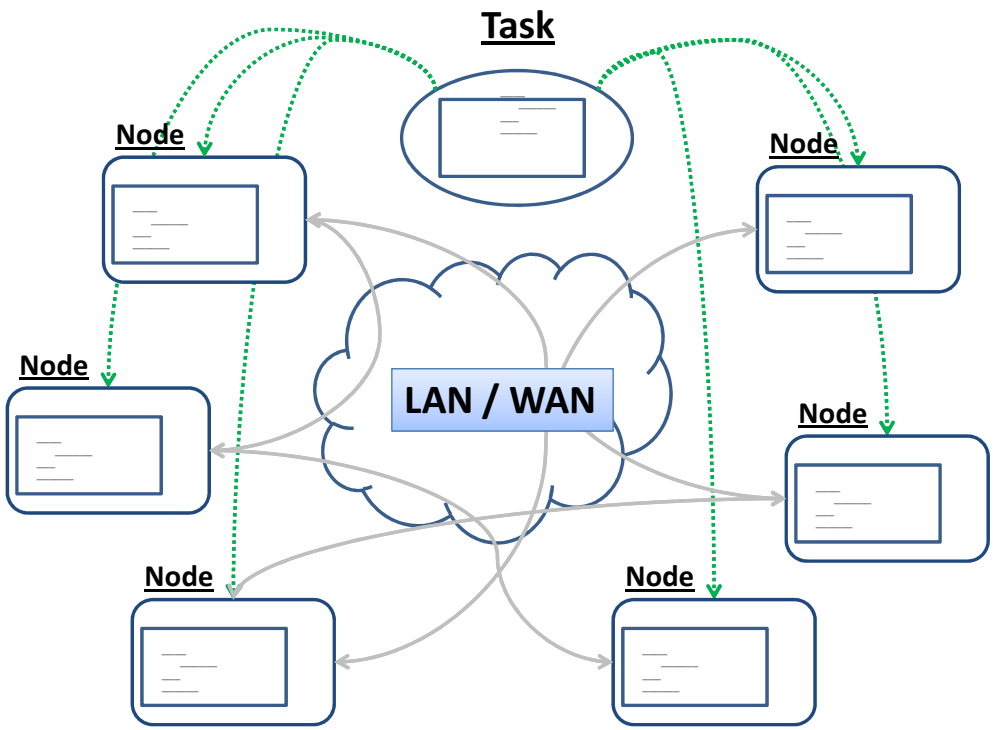
```
0   using System.TransactionalMemory;
    ...
    public static void Transfer(BankAccount from, BankAccount to,
                        int amount, int transferFee) {
      Atomic.Do(() => {
5
      })
    }
10  ...
```

STM.NET - Funktionsweise

```
0  using System.TransactionalMemory;
   ...
   public static void Transfer(BankAccount from, BankAccount to,
                               int amount, int transferFee) {
     Atomic.Do(() => {
5      from.ModifyBalance( - ( amount + transferFee ) );
       to.ModifyBalance(amount);
       bankAccount.GiveUsMoney(transferFee);
     })
   }
10 ...
```

Grid Computing

Googles MapReduce



Googles MapReduce - Merkmale

- Verteiltes System

Googles MapReduce - Merkmale

- Verteiltes System
- skaliert auf mehrere Tausend PCs (2004 avg. 157 / 2004 max. 1800)

Googles MapReduce - Merkmale

- Verteiltes System
- skaliert auf mehrere Tausend PCs (2004 avg. 157 / 2004 max. 1800)
- Fehlertolerant

Googles MapReduce - Merkmale

- Verteiltes System
- skaliert auf mehrere Tausend PCs (2004 avg. 157 / 2004 max. 1800)
- Fehlertolerant
- Master synchronisiert

Googles MapReduce - Merkmale

- Verteiltes System
- skaliert auf mehrere Tausend PCs (2004 avg. 157 / 2004 max. 1800)
- Fehlertolerant
- Master synchronisiert
- Einfache API [4]

Googles MapReduce - Map & Reduce

- Eine Beispiel Map Funktion:
  Wird auf jedes Element einer Collection angewendet.

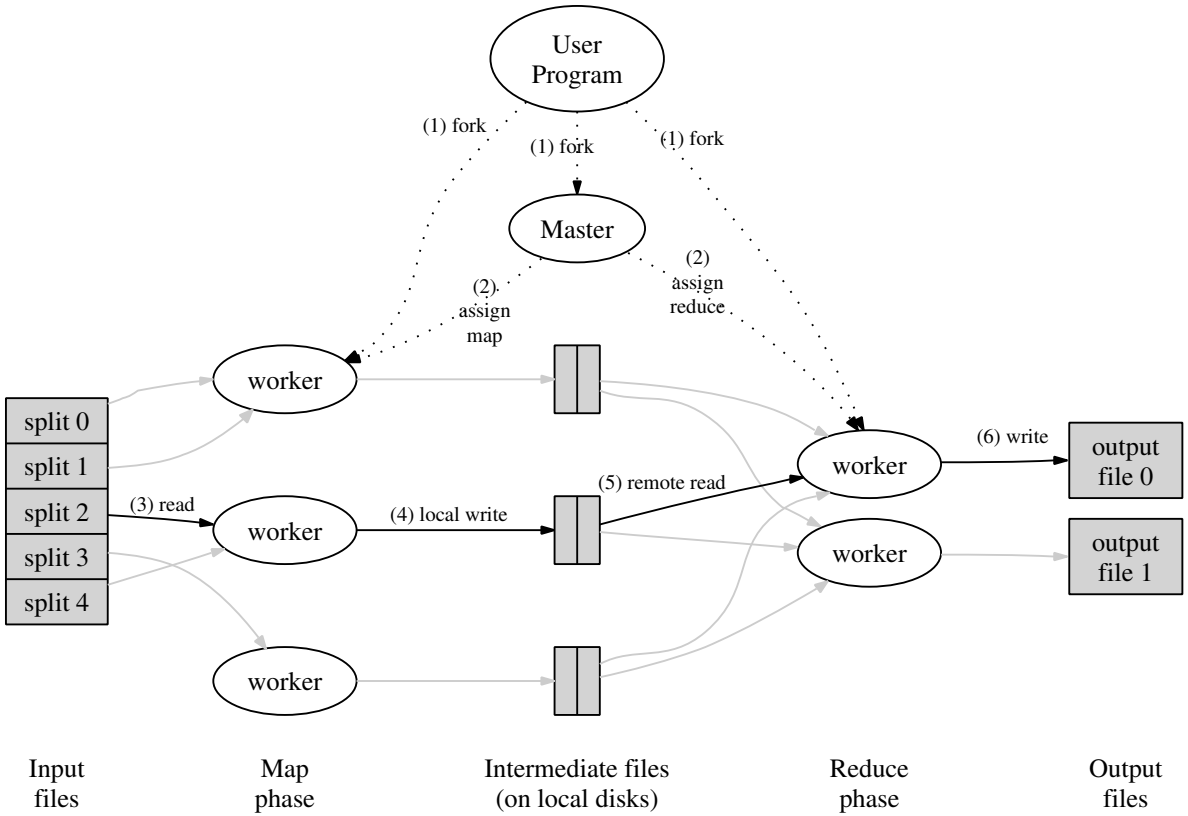
```
0   map( AddOne, <array> );
    -> array[0] = AddOne(array[0]) ;
    -> array[1] = AddOne(array[1]) ;
    ...
```

- Eine Beispiel Reduce Funktion:
  Wird benutzt um eine Collection auf ein Ergebnis zu reduzieren.

```
0   reduce( AddXY, 0, <array> );
    -> tmp = AddXY(0, array[0]);
    -> tmp = AddXY(tmp, array[1]);
    ...
```

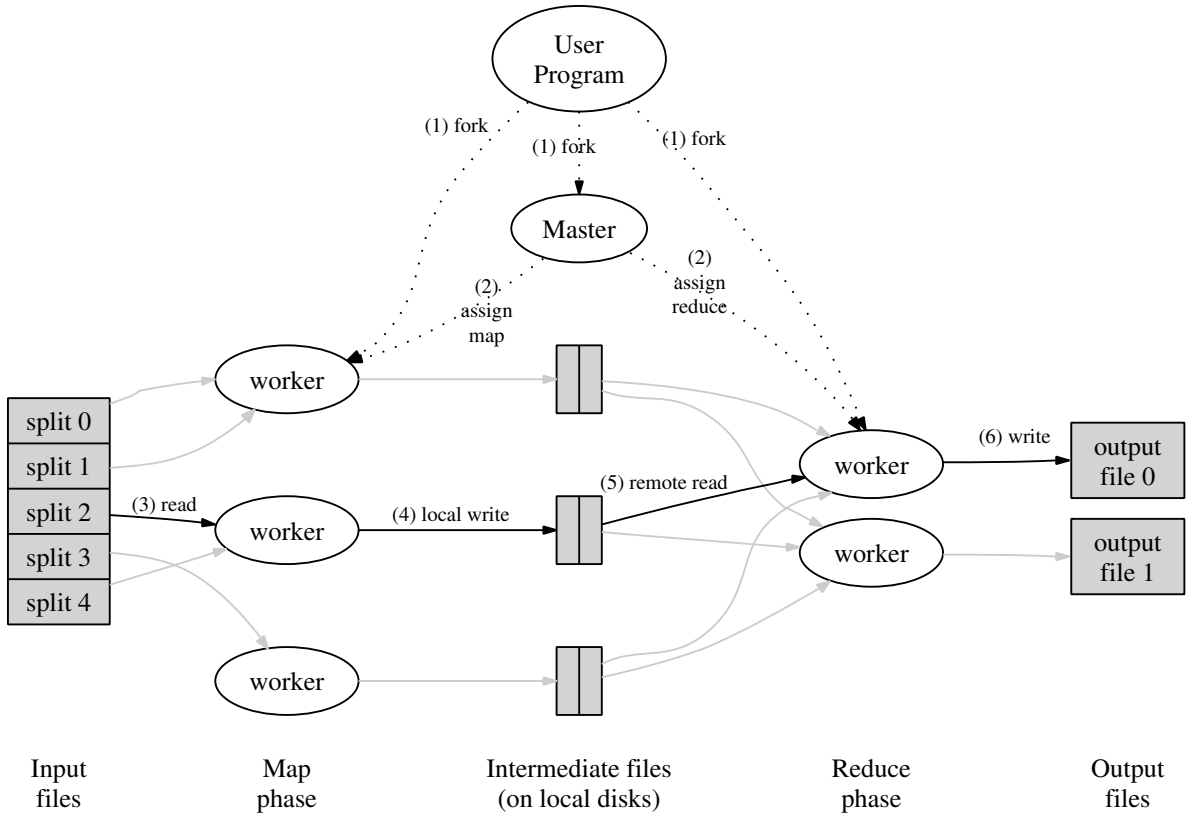
Googles MapReduce - Execution Overview

- Teilt die Eingabe auf und startet sich auf anderen Workern



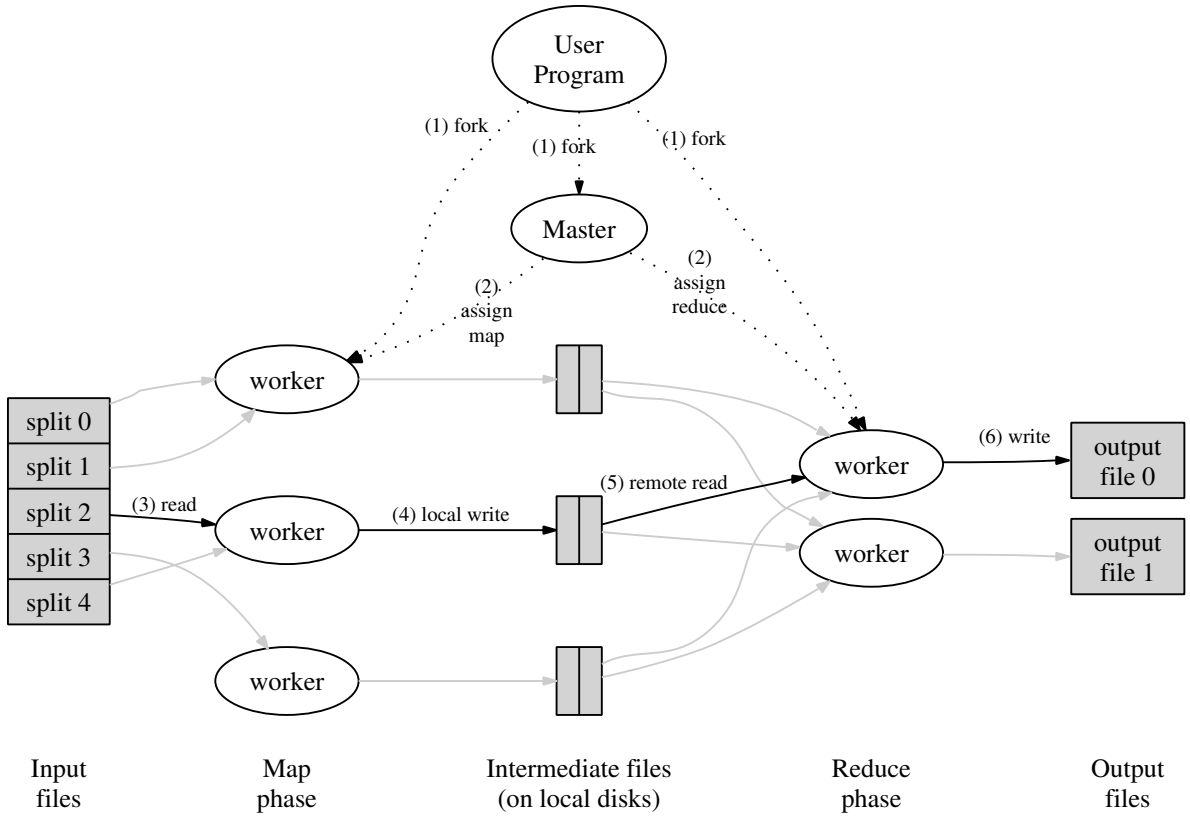
Googles MapReduce - Execution Overview

- Teilt die Eingabe auf und startet sich auf anderen Workern
- Master verteilt die Tasks auf untätige Worker



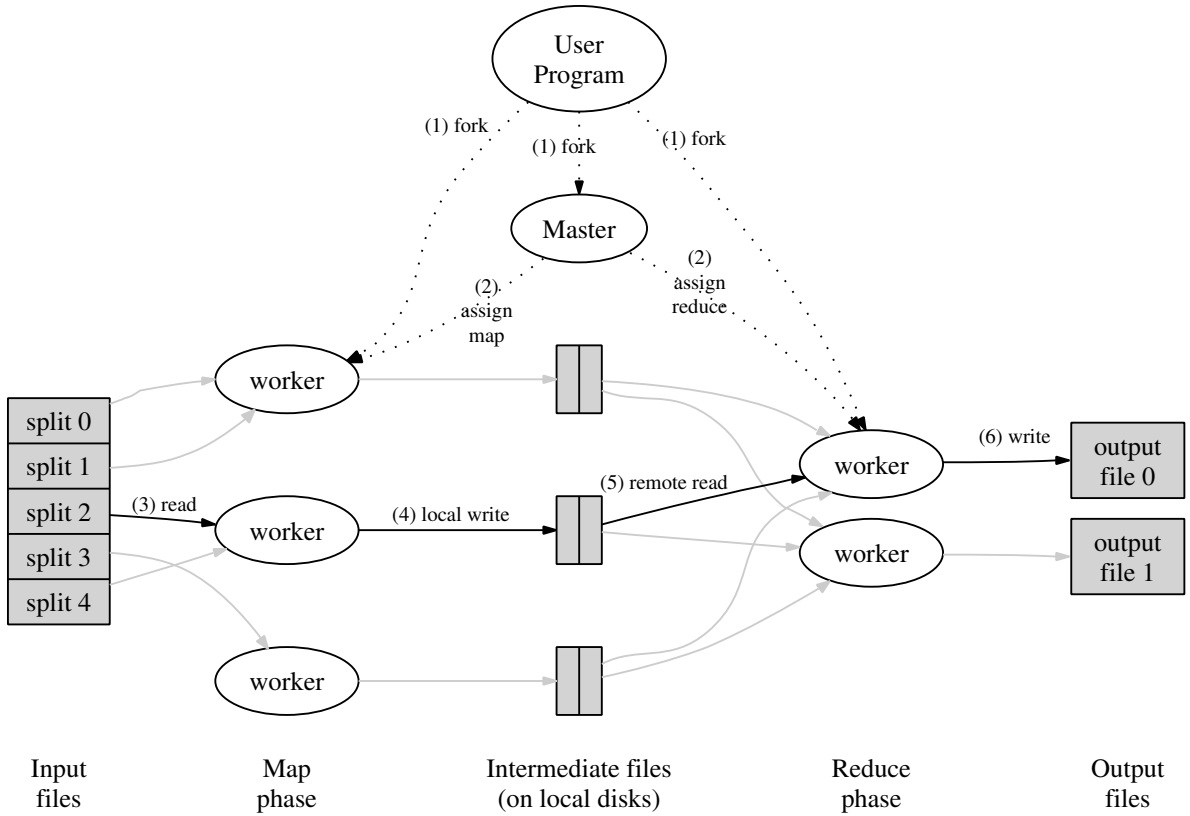
Googles MapReduce - Execution Overview

- Teilt die Eingabe auf und startet sich auf anderen Workern
- Master verteilt die Tasks auf untätige Worker
- Liest die Eingabe und „mapped“, Ergebnisse im RAM



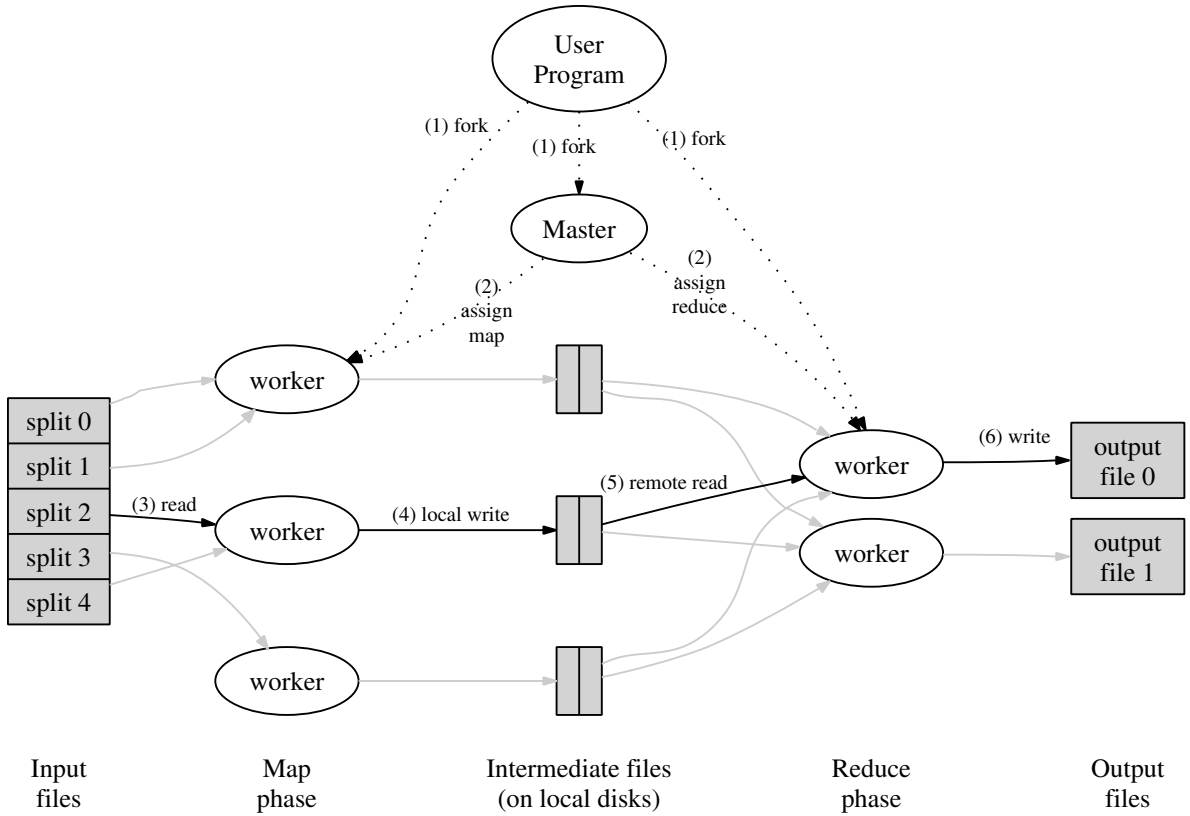
Googles MapReduce - Execution Overview

- Teilt die Eingabe auf und startet sich auf anderen Workern
- Master verteilt die Tasks auf untätige Worker
- Liest die Eingabe und „mapped“, Ergebnisse im RAM
- Periodisches schreiben der Key/Value Pairs, Adressen an den Master



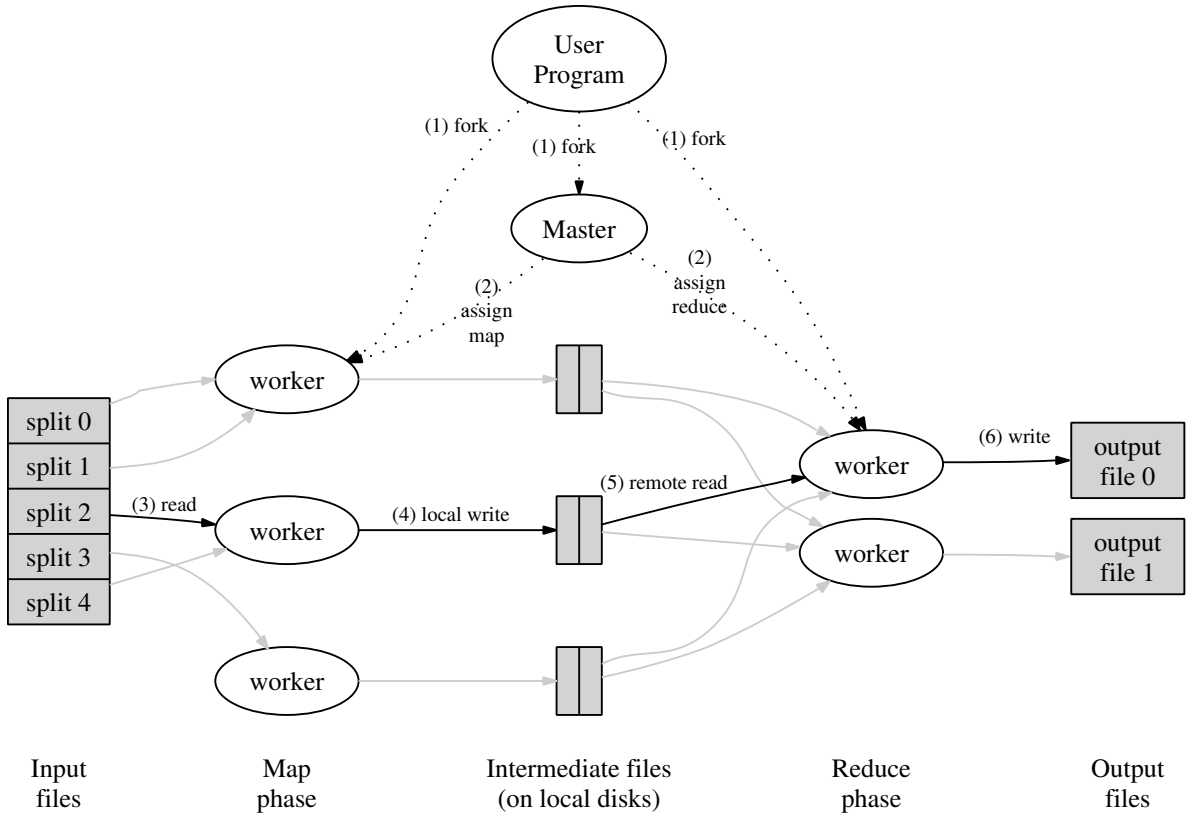
Googles MapReduce - Execution Overview

- Lesen über RPC vom „Mapper“, sortieren der Ergebnisse



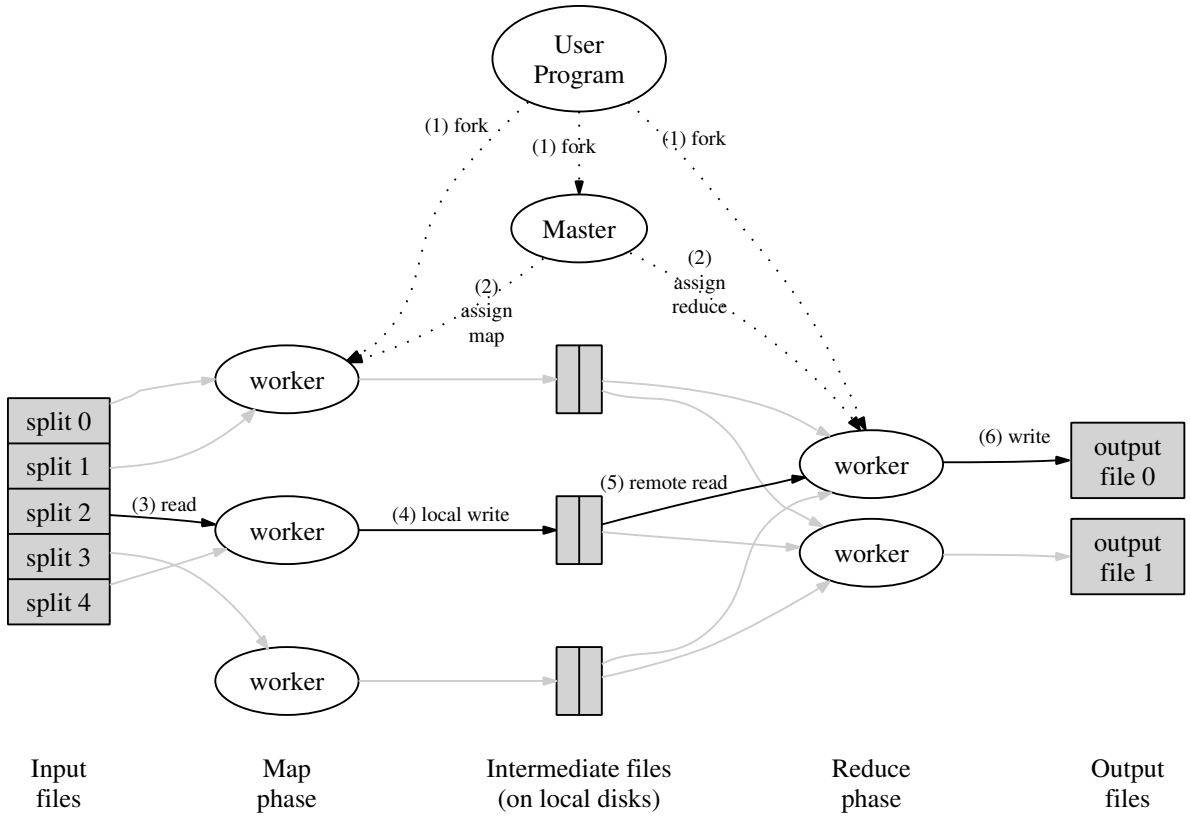
Googles MapReduce - Execution Overview

- Lesen über RPC vom „Mapper“, sortieren der Ergebnisse
- für jeden Key ein „reduce“



Googles MapReduce - Execution Overview

- Lesen über RPC vom „Mapper“, sortieren der Ergebnisse
- für jeden Key ein „reduce“
- Zurück zum Userprogramm wenn MapReduce fertig ist



Zusammenfassung und Ausblick

- Parallel Computing Models

Zusammenfassung und Ausblick

- Parallel Computing Models
- Related Work

Zusammenfassung und Ausblick

- Parallel Computing Models
- Related Work
- Ausblick

Zusammenfassung und Ausblick

- Parallel Computing Models
- Related Work
- Ausblick
  - iFlat-Einsatz im Intelligenzmittelpunkt

Zusammenfassung und Ausblick

- Parallel Computing Models
- Related Work
- Ausblick
  - iFlat-Einsatz im Intelligenzmittelpunkt
  - Semantic Interpretation

Zusammenfassung und Ausblick

- Parallel Computing Models
- Related Work
- Ausblick
  - iFlat-Einsatz im Intelligenzmittelpunkt
  - Semantic Interpretation
  - Wofür könnt ihr es brauchen?

Ende

Vielen Dank für die Aufmerksamkeit!
Fragen?

Quellen

[1] Barney, Blaise: *Parallel Computation Models*. Lawrence Livermore National Laboratory. 2010. – URL https://computing.llnl.gov/tutorials/parallel_comp/#Models. – [Online; Stand 22. Mai 2010]

[2] Bauke, Heiko ; Mertens, Stephan: *Cluster Computing*. 2006. – ISBN 3-540-42299-4

[3] Clojure.org: *Clojure — Language Home*. 2009. – URL http://www.clojure.org. – [Online; Stand 28. Oktober 2009]

[4] Dean, Jeffrey ; Ghemawat, Sanjay: *MapReduce: Simplifed Data Processing on Large Clusters*. Google, Labs. 2010. – URL http://labs.google.com/papers/mapreduce.html. – [Online; Stand 23. Mai 2010]

[5] Microsoft: *STM Programmer´s Guide*. Microsoft. 2010. – URL http://download.microsoft.com/download/9/5/6/9560741A-EEEC-4C02-822C-BB0AEE860E31/STM_User