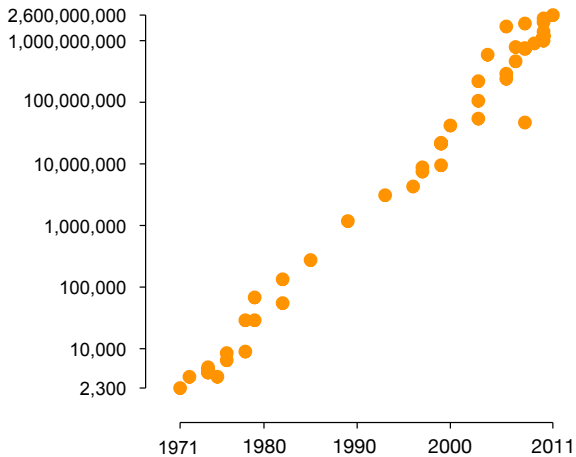
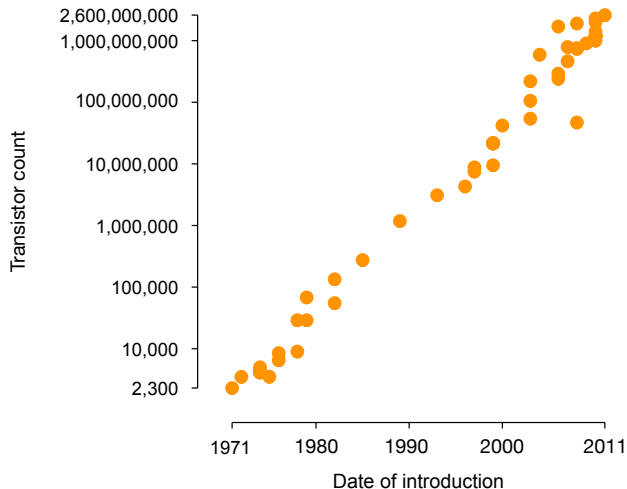


WAS WIRD HIER GEZEIGT?



TRANSISTOR COUNTS 1971-2011 & MOORE'S LAW



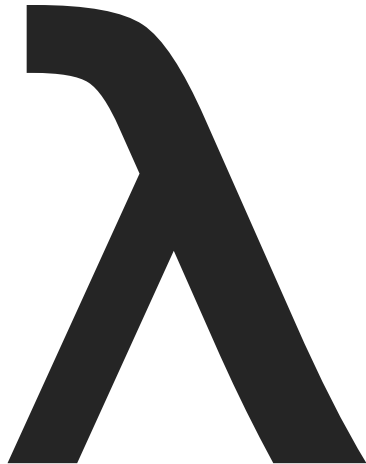
FUNKTIONALE PROGRAMMIERUNG

Nebenläufigkeit & Parallelisierung

Seminar, WS2016

Jan-Philipp Willem

Prof. Dr. Sandro Leuchter
Fakultät für Informatik
Hochschule Mannheim



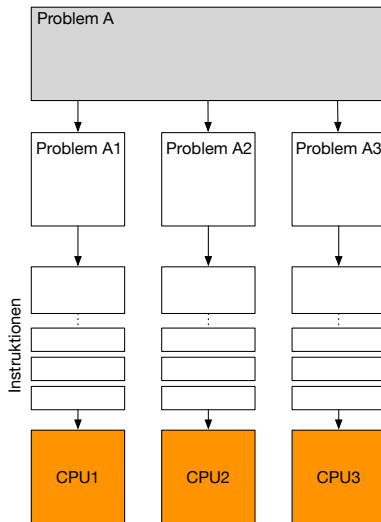
GLIEDERUNG

1. Nebenläufigkeit / Parallelisierung
2. Threads / Locking
3. Functional Paradigm 101
4. Elixir
5. Fazit

NEBENLÄUFIGKEIT /
PARALLELISIERUNG

PARALLEL

- Synonyme:
nebeneinander,
nebenläufig
- Informatik:
parallel \neq nebenläufig!
- „schneller als
sequenzielles Programm,
durch gleichzeitiges
Ausführen von
Anweisungen“
- Multi-Processing



NEBENLÄUFIG

- concurrent (engl.)
- „Systeme, welche zur gleichen Zeit mehrere **Aufgaben** haben“
- muss nicht zwangsläufig parallel sein
- Multi-Tasking

typischer Fat-Client

„auf Benutzereingaben reagieren“

„Benutzeroberfläche zeichnen“

•

•

•

•

„Request/Response mit Server“

ROB PIKE - 'CONCURRENCY IS NOT PARALLELISM' (1)

- „Concurrency is about dealing with lots of things at once.“
- „Parallelism is about doing lots of things at once.“
- „Concurrency is about structure, parallelism is about execution.“

ROB PIKE - 'CONCURRENCY IS NOT PARALLELISM' (2)



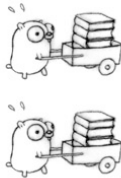
→ sequenziell

ROB PIKE - 'CONCURRENCY IS NOT PARALLELISM' (3)



→ mehrere Cores, jedoch keine Optimierung

ROB PIKE - 'CONCURRENCY IS NOT PARALLELISM' (4)



→ parallel

ROB PIKE - 'CONCURRENCY IS NOT PARALLELISM' (5)

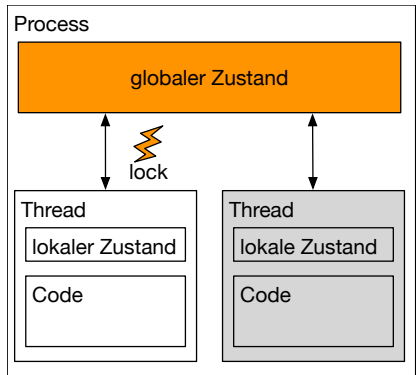


→ concurrent

THREADS / LOCKING

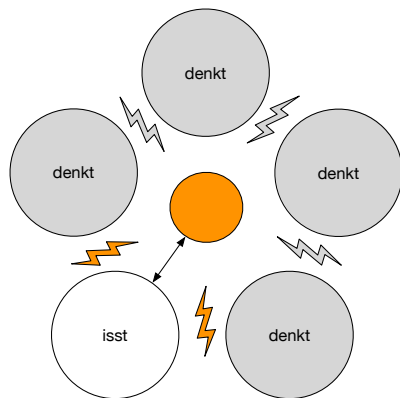
WAS IST EIN THREAD?

→ Shared Memory



WAS IST EIN LOCK?

→ Dining philosophers problem



HÄUFIGE BUGS

Race-Condition

→ ++

→ ++

Deadlock

→ -

→ -

FAZIT: THREADS PROGRAMMING

→ „State is Evil“

FUNCTIONAL PARADIGM 101

FUNCTIONAL PARADIGM 101

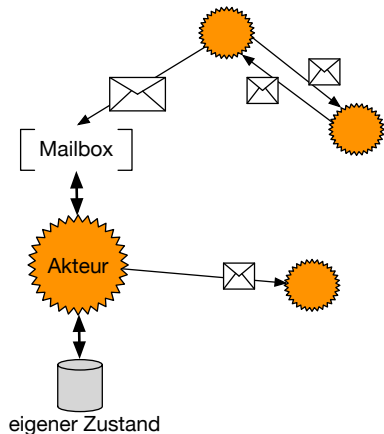
- reine Funktionale Sprachen
- immutable // mutable
- no side-effects
- deterministic
- data-in <-> data-out
- functions as first-class citizens
- lamdas

ELIXIR

- moderne Variante von Erlang (1987, Ericsson)
- Beam-VM
- Fault-Tolerant
- „Let it crash“
- Supervision-Trees
- Shared & Distributed Memory
- Open Telecom Platform (OTP)

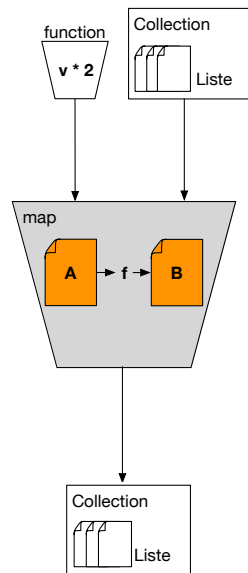
OTP / ACTOR-MODEL

- unabhängige Akteure
- Message-Passing
- FIFO-Verhalten von **Mailboxes**
- Locks werden nicht gebraucht
- Alternativen:
 - Akka (Java/Scala)
 - Akka.NET
 - Pykka (Python)
 - CAF (C++)
 - Celluloid (Ruby)



LIST-PROCESSING IN ELIXIR: MAP (1)

- Traversieren der Elemente
- Nutzen von transform-function
- Ergebnis von gleichem oder verschiedenen Collection-Typ



LIST-PROCESSING IN ELIXIR: MAP (2)

```
→ iex> Enum.map [1, 2, 3], fn x -> x + 1 end  
    [2, 3, 4]
```


LIST-PROCESSING IN ELIXIR: MAP (2)

```
→ iex> Enum.map [1, 2, 3], fn x -> x + 1 end  
[2, 3, 4]
```

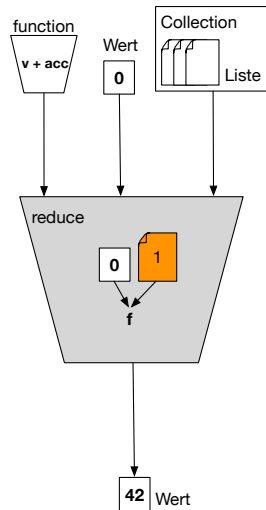
```
→ iex> Enum.map [1, 2, 3], &(&1 * &1)  
[1, 4, 9]
```

LIST-PROCESSING IN ELIXIR: MAP (2)

```
→ iex> Enum.map [1, 2, 3], fn x -> x + 1 end  
  [2, 3, 4]  
→ iex> Enum.map [1, 2, 3], &(&1 * &1)  
  [1, 4, 9]  
→ iex> defmodule Math do  
  ...> def multWithKey({k, v}), do: k * v  
  ...> end  
  ...  
iex> list = Enum.with_index([1, 2, 3])  
  [{1, 0}, {2, 1}, {3, 2}]  
iex> Enum.map list, &Math.multWithKey/1  
  [0, 2, 6]
```

LIST-PROCESSING IN ELIXIR: REDUCE (1)

- Traversieren der Elemente
- Nutzen von aggregate-function und Startwert
- Mitführen von accumulator
- „Reduzieren“ der Elemente auf einzelnen gemeinsamen Wert
- auch `foldr`, `foldl` genannt

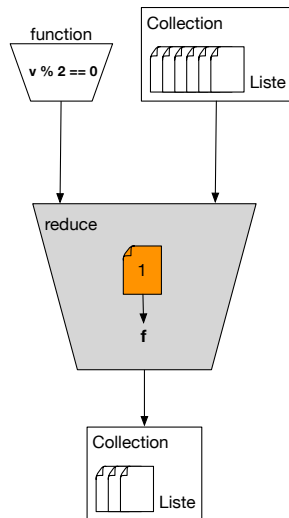


LIST-PROCESSING IN ELIXIR: REDUCE (2)

```
→ iex> List.foldl [1, 2, 3, 4], 0, &(&1 + &2)  
10
```

LIST-PROCESSING IN ELIXIR: FILTER (1)

- Traversieren der Elemente
- Nutzen von predicate-function
- Teilmenge bilden, welche Elemente enthält, welche Anforderung erfüllen



LIST-PROCESSING IN ELIXIR: FILTER (2)

ELIXIR-STREAMS

FAZIT

FAZIT: FUNCTIONAL PROGRAMMING

Vorteile

- composeability of behaviors
- Art und Abfolge der Anweisung kann genau und deklarativ bestimmt werden

Nachteile

- Punkt 1
- Punkt 2

FUNCTIONAL STYLE IN IMPERATIVE LANGUAGES