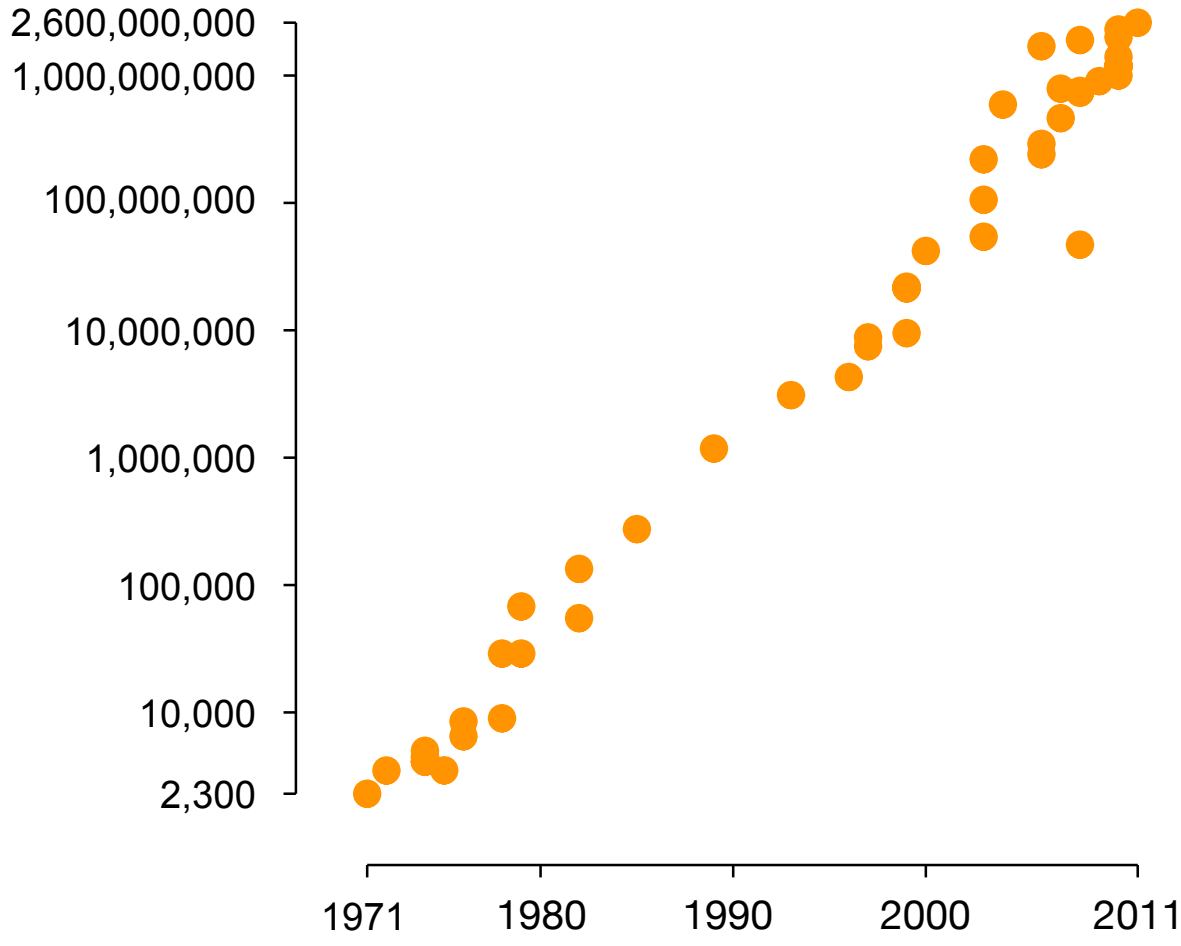
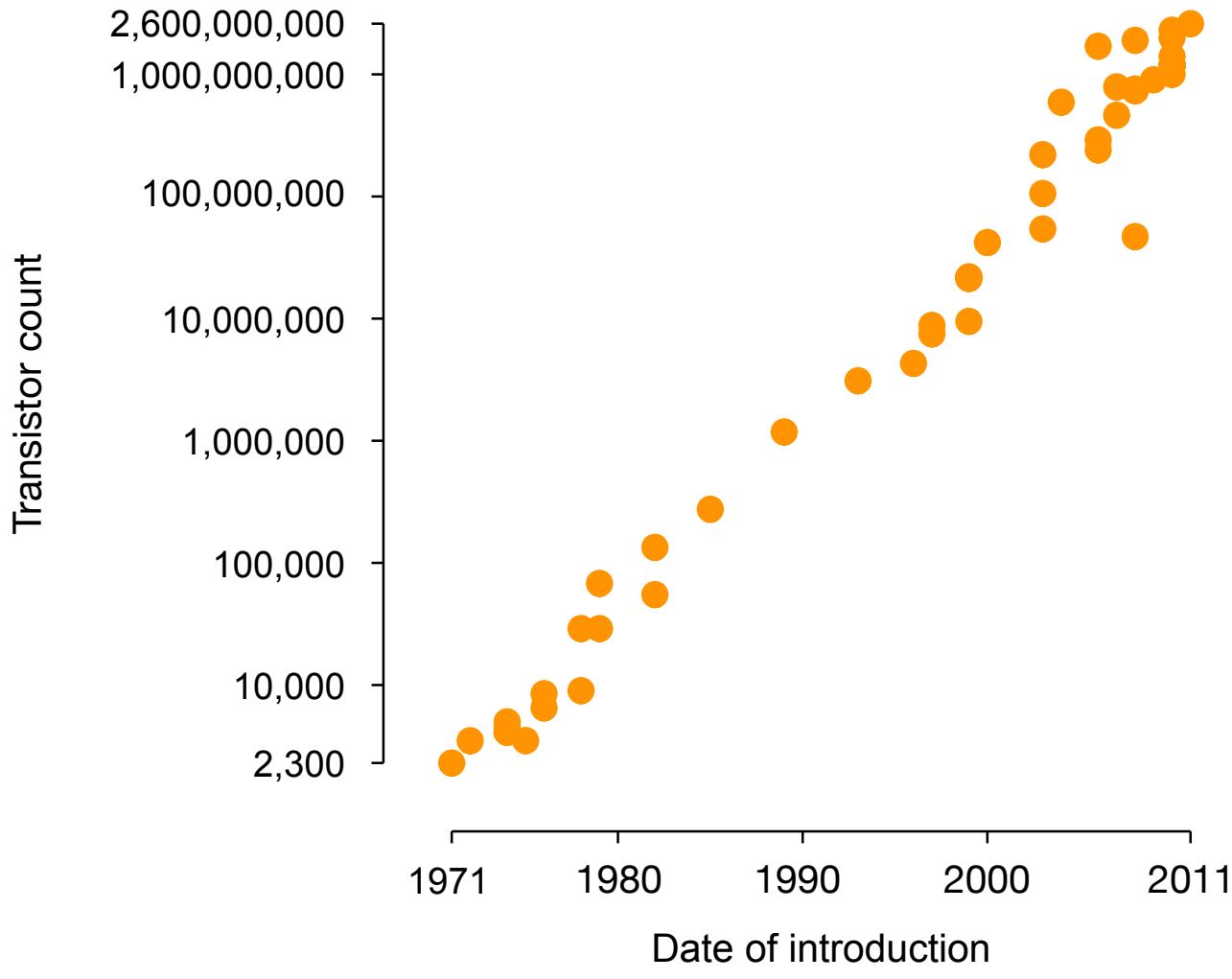
WAS WIRD HIER GEZEIGT?



TRANSISTOR COUNTS 1971-2011 & MOORE'S LAW



GEAR & GADGETS —

Intel Core i7-7700K Kaby Lake review: Is the desktop CPU dead?

With identical performance to Skylake, Intel brings desktop performance to a standstill.

MARK WALTON - 3/1/2017, 18:00

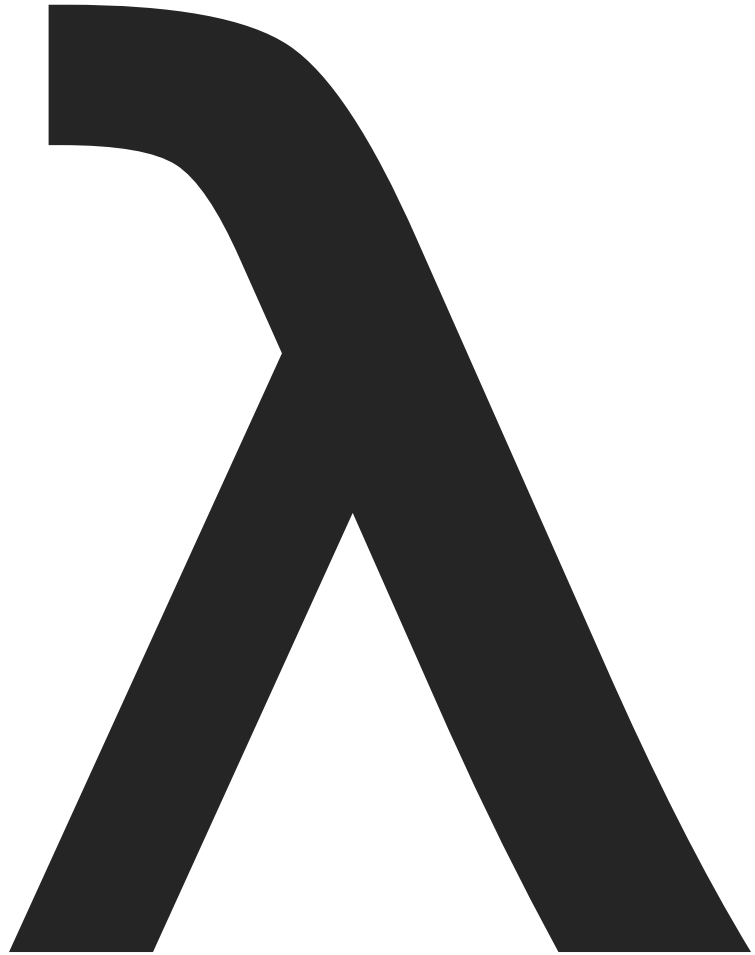
ARSTECHNICA UK - 03.01.2017 (2)

„With low-power laptops and all-in-ones continuing to outsell desktops—and with high-end workloads like video editing, 3D animation, and machine learning increasingly being offloaded to GPUs – perhaps it was inevitable that Intel would stop caring so much about its high-end consumer CPUs.“

FUNKTIONALE PROGRAMIERUNG

Nebenläufigkeit & Parallelisierung

Seminar, WS2016



Jan-Philipp Willem

Prof. Dr. Sandro Leuchter
Fakultät für Informatik
Hochschule Mannheim

GLIEDERUNG

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

NEBENLÄUFIGKEIT / PARALLELISIERUNG

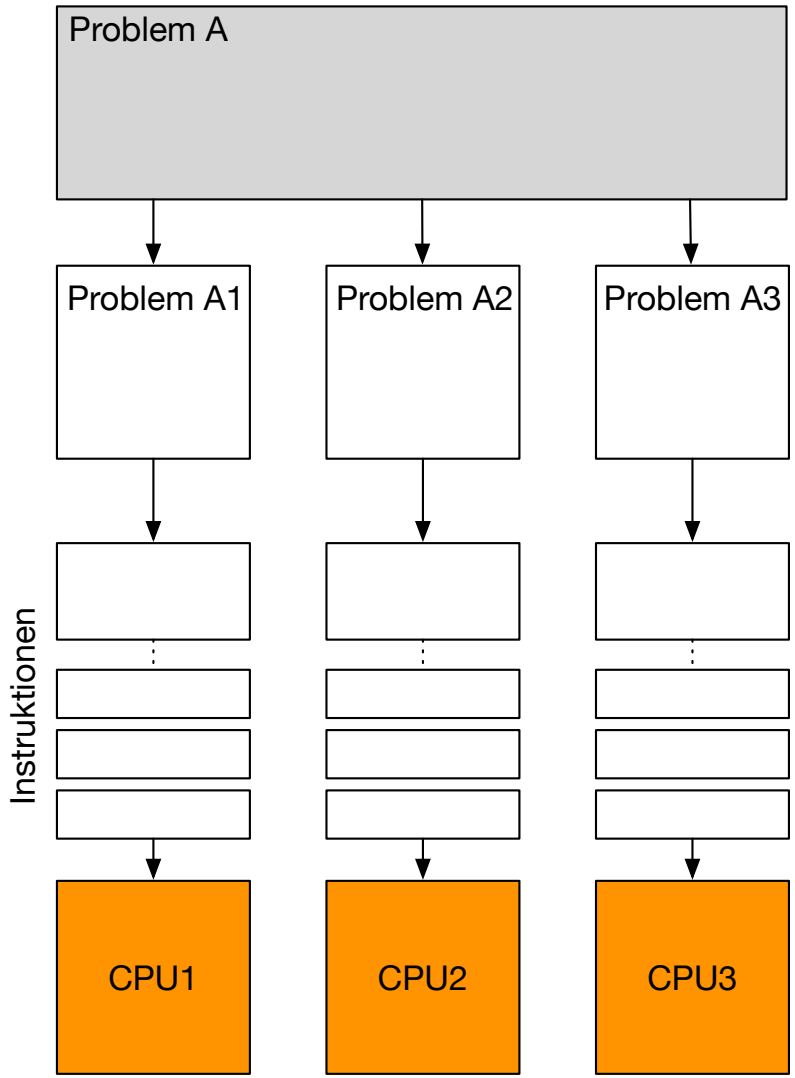
PARALLEL

→ Synonyme:
nebeneinander,
nebenläufig

→ Informatik:
parallel ≠ nebenläufig!

→ „schneller als
sequenzielles Programm,
durch gleichzeitiges
Ausführen von
Anweisungen“

→ Multi-Processing



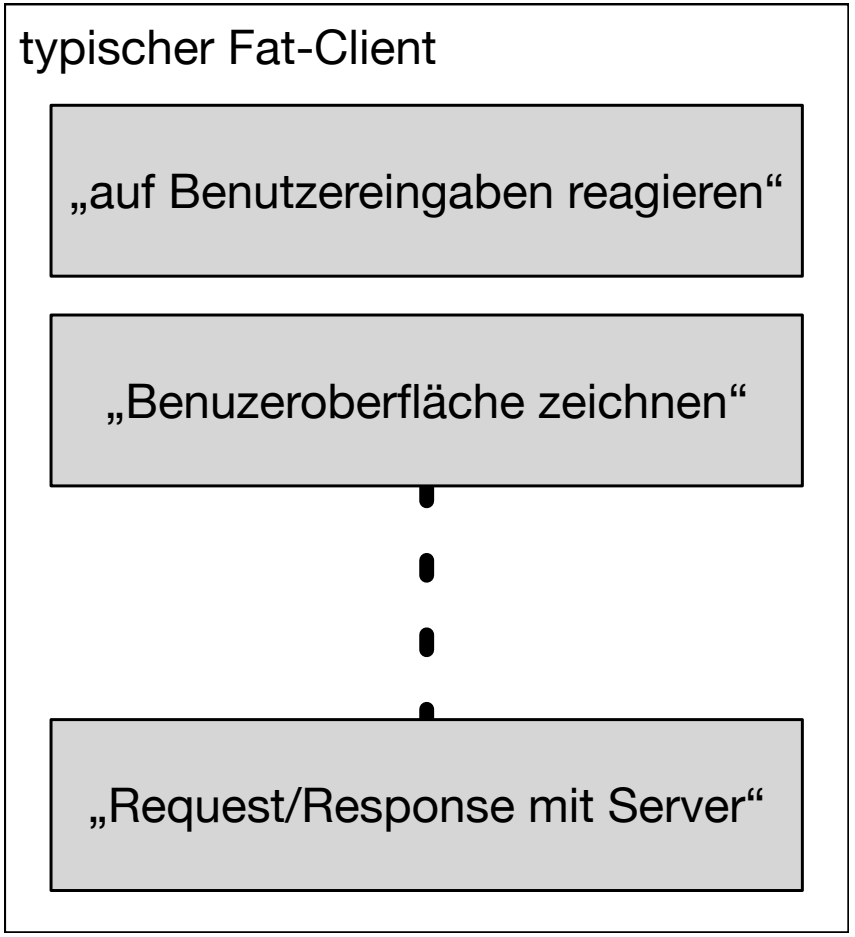
NEBENLÄUFIG

→ concurrent (engl.)

→ „Systeme, welche zur gleichen Zeit mehrere **Aufgaben** haben“

→ muss nicht zwangsläufig parallel sein

→ Multi-Tasking



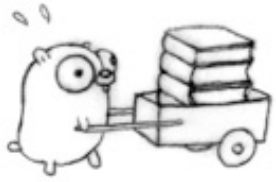
ROB PIKE - „CONCURRENCY IS NOT PARALLELISM“ (1)

→ „Concurrency is about dealing with lots of things at once.“

→ „Parallelism is about doing lots of things at once.“

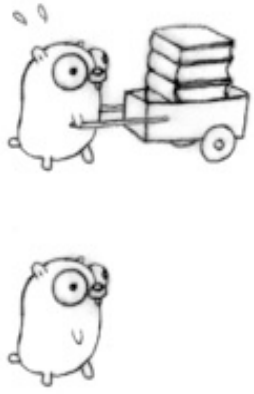
→ „Concurrency is about structure, parallelism is about execution.“

ROB PIKE - „CONCURRENCY IS NOT PARALLELISM“ (2)



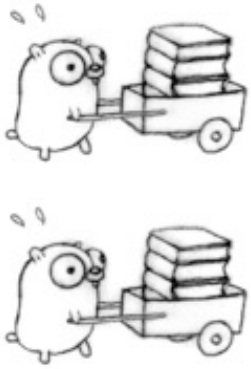
→ sequenziell

ROB PIKE - „CONCURRENCY IS NOT PARALLELISM“ (3)



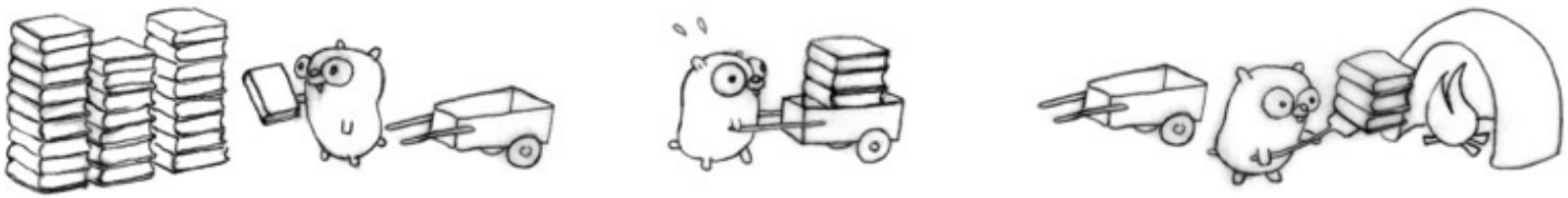
→ mehrere Cores, jedoch keine Optimierung

ROB PIKE - „CONCURRENCY IS NOT PARALLELISM“ (4)



→ parallel

ROB PIKE - „CONCURRENCY IS NOT PARALLELISM“ (5)



→ concurrent

FUNCTIONAL PARADIGM 101

FUNCTIONAL PARADIGM 101

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

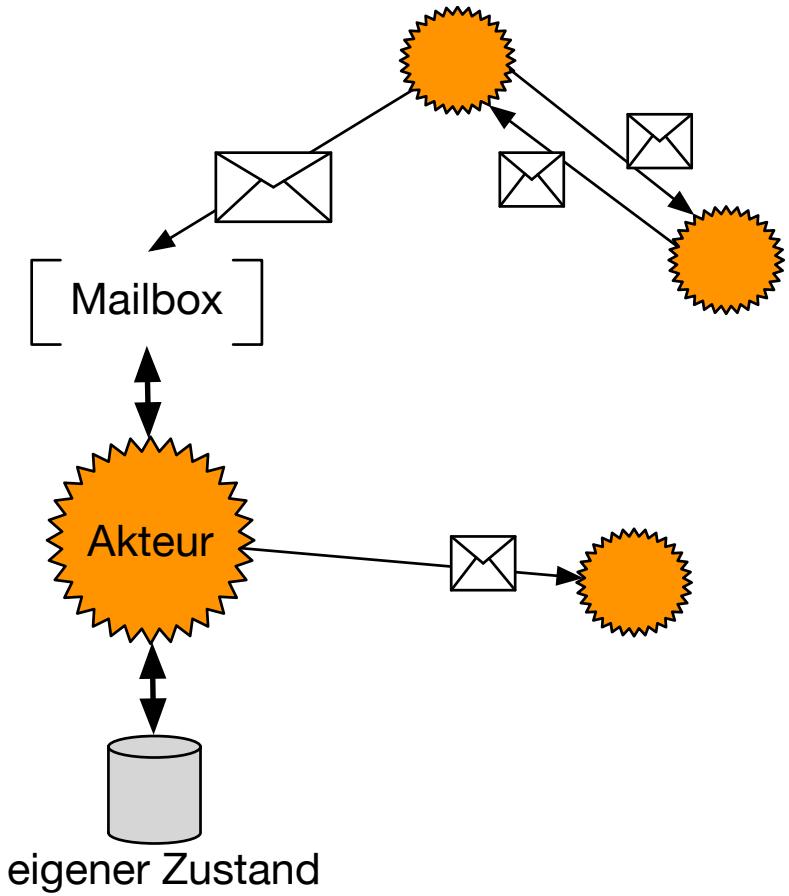
ELIXIR

ELIXIR

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

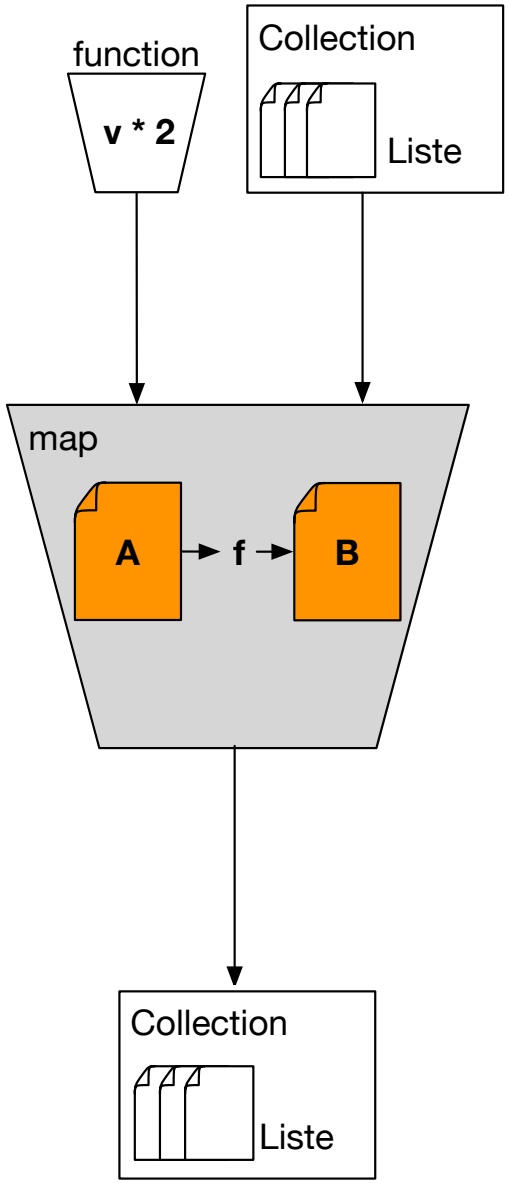
OTP / ACTOR-MODEL

- → 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]

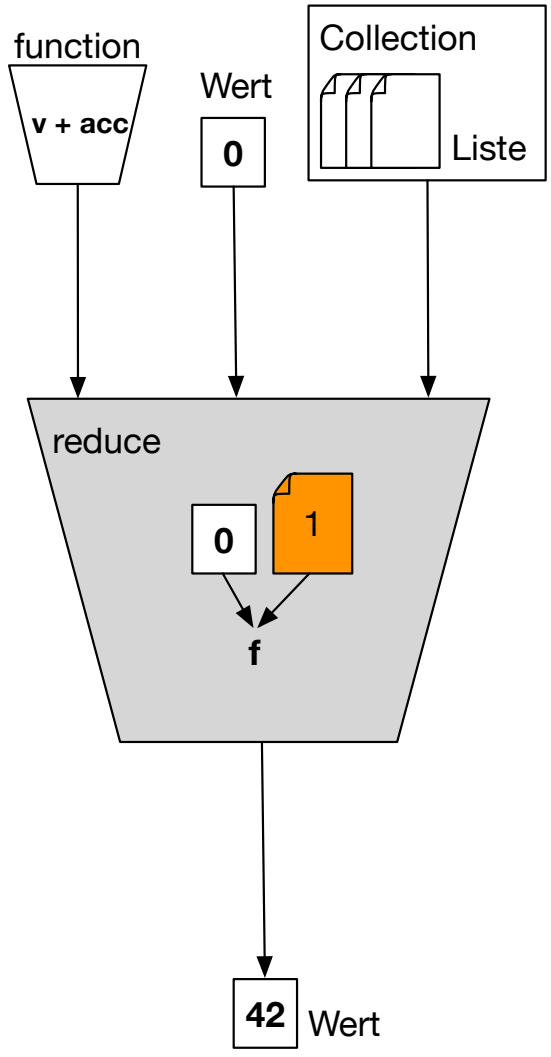
→ `iex> Enum.map [1, 2, 3], &(&1 * &1)`
[1, 4, 9]

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)

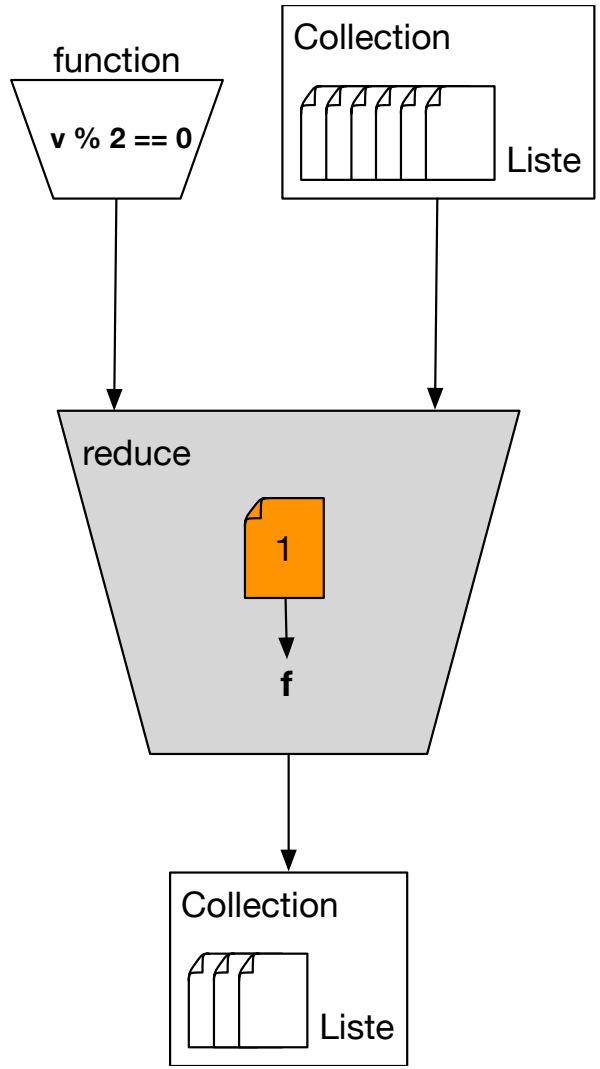
```
→ iex> List.foldl [1, 2, 3, 4], 0, &(&1 + &2)
  10
```

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