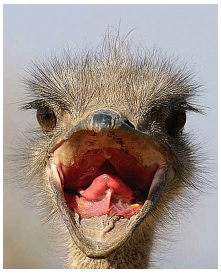
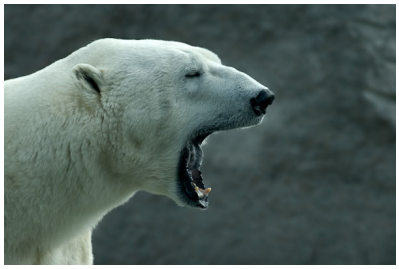
Erlang/OTP

Nachrichtenaustausch
(Message Passing, Shared Nothing)
und Nebenläufigkeit

1

Warum ist Funktionale Programmierung in vieler Munde?



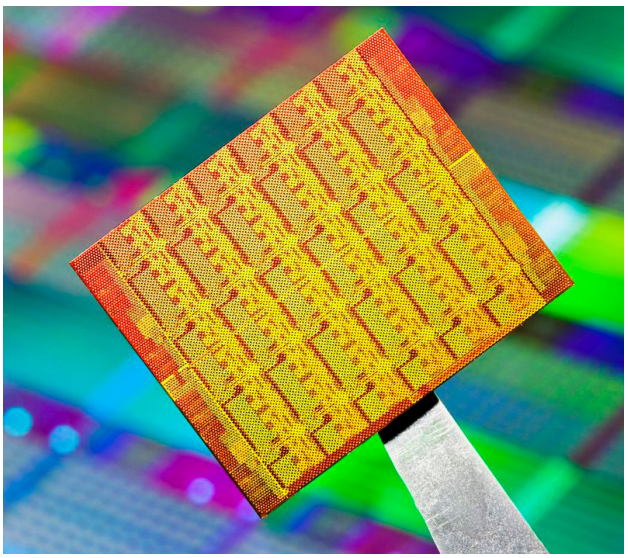


2

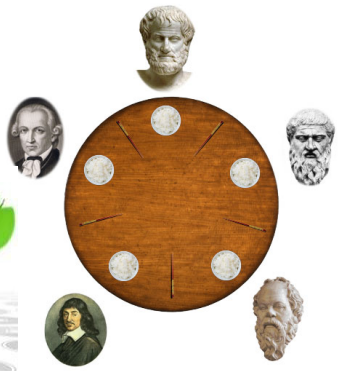
Mercury Computer Systems 6U HDS6601

pair of eight-core
Intel Xeon E5-2648L
processors

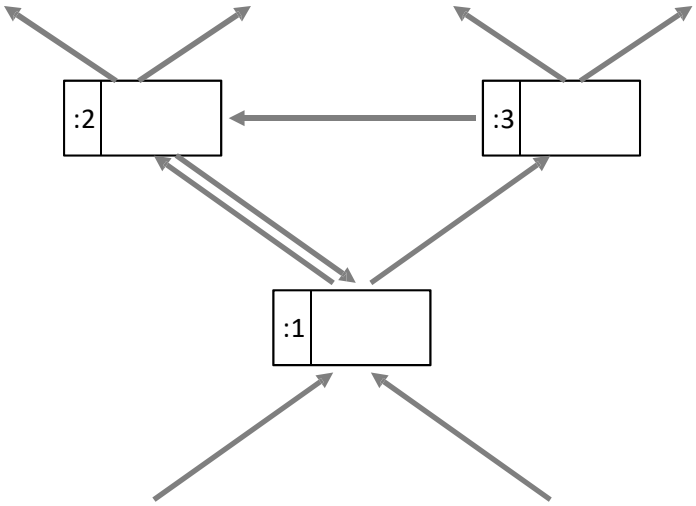
Intel Cloud Computing on a Chip 2009, 48 Cores (Prototyp)



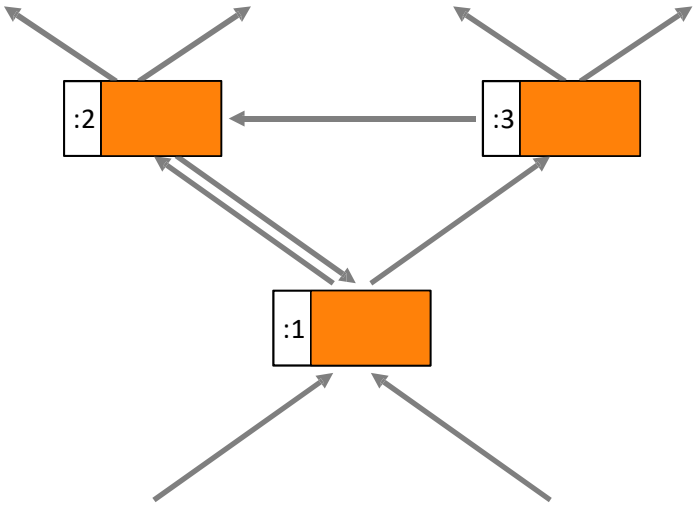
Warum ist Nebenläufigkeit mit Objekten schwierig?



Objekte haben eine Identität

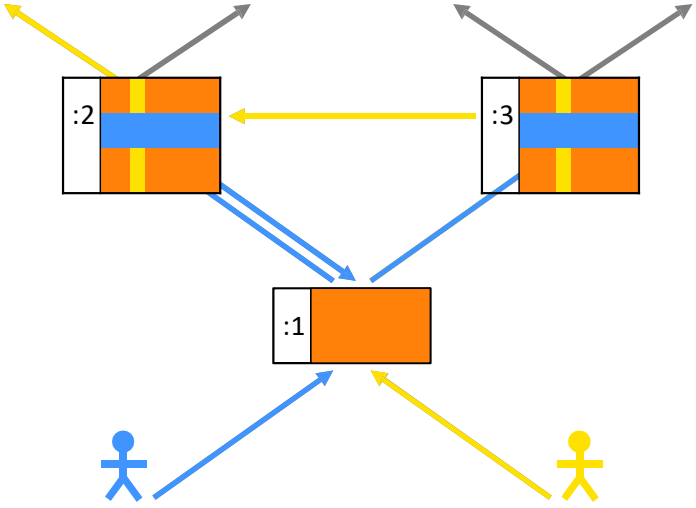


Objekte haben einen (lokalen) Zustand

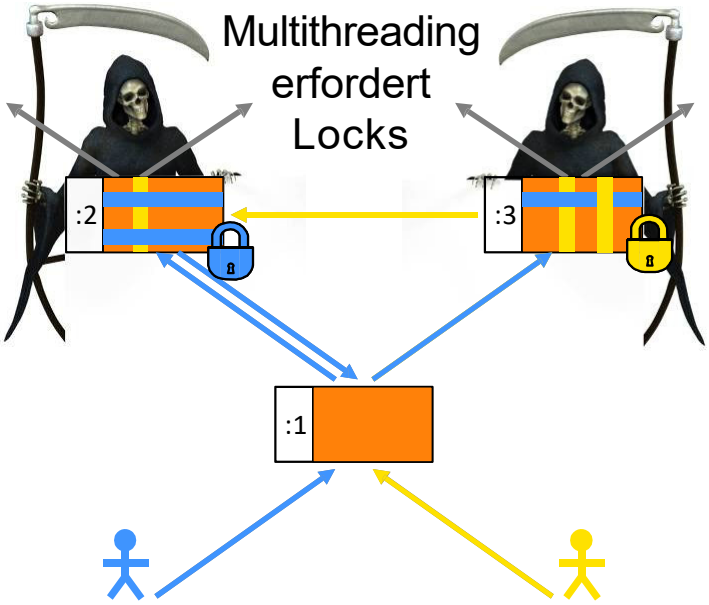


7

Objekte haben Verhalten

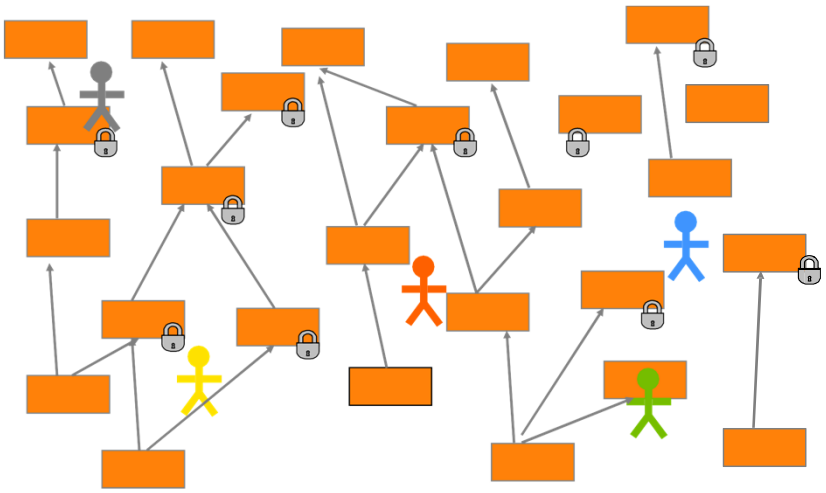


8



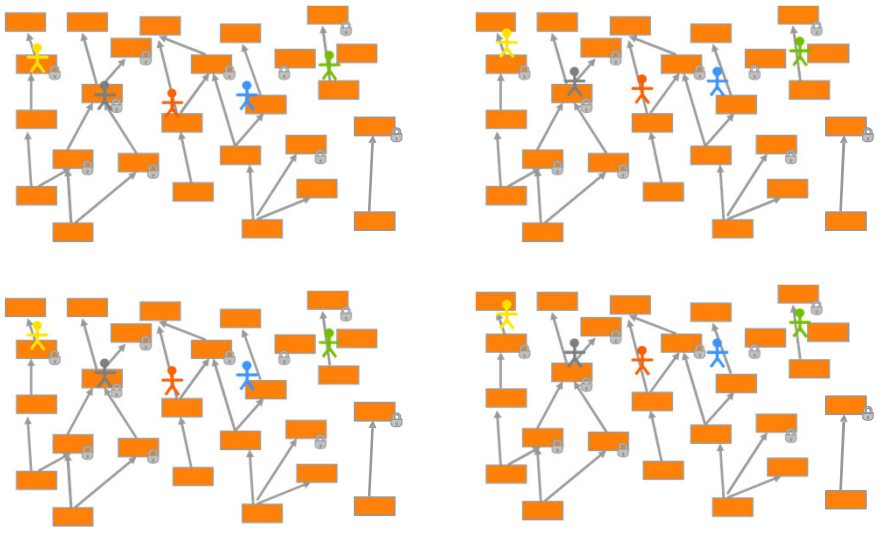
9

Ein hoher Grad an Nebenläufigkeit wird leicht unübersichtlich



10

Ein hoher Grad an Nebenläufigkeit wird leicht unübersichtlich



Wie kann funktionale Programmierung hier helfen?

Was unterscheidet FP von OOP?

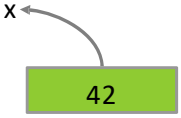
Werte, Variablen, Veränderliche

Was sind imperative Variablen?

- Namen für Adressen von Speicherzellen
- Abstraktionsniveau: Assembler
- Objekte haben eine Identität: Entspricht einer Adresse von Speicherzellen

Was sind funktionale Variablen?

- Namen für Werte: „Sei x beliebig, aber fest“
- Variablen werden an Werte „gebunden“
- Abstraktionsniveau: Mathematik
- Notwendig: Effiziente Abbildung auf Speicherzellen durch Laufzeitsystem



Objekte versus Werte

Funktionen erzeugen aus alten Werte neue!

Nicht destruktiv

Funktionale Programmierung = Werteorientierung & Higher-order Funktionen

Er-was?

- Sprache / Laufzeitumgebung / Bibliotheken entwickelt bei *Ericsson* (Joe Armstrong) für die Telekommunikation, benannt nach dem Mathematiker *Agner Krarup Erlang*
- Philosophie: **Concurrency Oriented Programming** für skalierbare, langlebige Systeme; **Nicht** (explizit) objektorientiert
- Erlang/OTP: **O**pen **T**elecom **P**latform ist die Open Source Ausgabe von Erlang

Einfach nur eine weitere Sprache?

- Pattern matching (Erinnerung: Prolog und der Unifikator)
- Optimierung der Endrekursion (Tail call optimization)
- Nebenläufigkeit durch Nachrichtenaustausch (Message-passing concurrency with nothing shared)
- Verteilte Programmierung (Distributed programming)
- „On-the-fly“ Codeaustausch (Hot code update)
- Robustheit: Laufzeitfehler können abgefangen werden

Syntax

- Variablen können nur einmal zugewiesen werden (Erinnerung an Prolog: nicht destruktive Programmierung)
- Variablen beginnen mit Großbuchstaben (Erinnerung an Namenskonvention in Prolog)
- Letzte Auswertung ist Rückgabewert der Funktion

= : Das Pattern Matching

- „Sie können das = Symbol weiterhin verwenden. Ich glaube aber nicht, dass es das bedeutet, was Sie denken das es bedeutet.“
- Ist keine Mutation (Zuweisung) in Erlang
- `LHS = RHS`
  - Wertet `RHS` aus und macht einen Mustervergleich (Pattern Matching) gegen `LHS`.
  - viel mehr Spielraum, um den Ausdruck wahr zu machen

She's got the look

```
-module(math_o_matics).
-export([square/1]).

square(X) ->
             X * X.

cube(X) ->
            square(X) * X.
```

```
math_o_matics:square(42)
```

```
math_o_matics:cube(42)
nicht möglich!
```


Syntax

- **Atome**
  - selbstanzeigende Bezeichner
  - beginnen mit Kleinbuchstaben
  - können auch in einfache Anführungszeichen gesetzt werden

```
atome
dies_ist_ein_atome
'ich bin auch ein Atom'
```

- **Tupel** (Erinnerung: Strukturen in Prolog)
  - Container fester Länge
  - Zerlegung mittels Pattern Matching:

```
Auto = {auto,{honda,civic},{ps, 100}}.
{auto, Typ, Leistung} = Auto.
```

Syntax

- **Listen**
  - Container fester Länge
  - Verwenden Sie `[Kopf|Rest]` Syntax, um das erste Element der Liste (Kopf) und den Rest der Liste zu erhalten

```
List = [1, 2, 3, vier, 5,0]
[Head | Tail] = Liste
[H1, H2 | T2] = Liste
```

- **Zeichenketten**
  - So ähnlich wie in C
  - Zeichenketten (strings) sind nur Listen von (ASCII-)Zahlen
  - Verwendung der doppelten Anführungszeichen

```
Meeting = "Stecker".
Meeting2 = [83,116,101,99107,101,114].
```

```
6> [97,98,99].
"abc"
```


Syntax

- **Stelligkeit**
  - Sie können Funktionen mit dem selben Namen und unterschiedlicher Stelligkeit (als Hilfsfunktionen) verwenden

```
-module(math_o_matics).
-export([sum/1]).
sum(L) -> sum(L, 0).
sum([], N) -> N;
sum([H|T], N) -> sum(T,H+N).
```

- **Module**
  - Logisch zusammenhängender Codeblock
  - Verwende Doppelpunkt (:) um Modulcode anderer Module zu verwenden
  - Verwende `-import` um den Doppelpunkt zu vermeiden

```
io:format("Using the module io~n").
```

Syntax

- Der „Spaß“ (fun) bei **Funktional**
  - Anonyme (Lambda) Funktionen, definiert durch Clauses
  - Funktionen statt fixem Ablaufplan, Rekursionen statt Schleifen
  - higher-order programming

```
Square = fun(X) -> X * X end.
Cube = fun(X) -> Square(X) * X end.
```

- **List Comprehensions**
  - Nimmt einen Ausdruck und eine Reihe von Auswahlkriterien und gibt (basierend auf einer Liste) eine neue Liste zurück
  - ähnlich der mathematischen Mengenschreibweise:

$[X \mid\mid Q_1, Q_2, \ldots, Q_n]$

```
qsort([]) -> [];
qsort([Pivot|T]) -> qsort([X || X <- T, X < Pivot])
                    ++ [Pivot] ++
                    qsort([X || X <- T, X >= Pivot]).
```


Syntax

- **Guards**
  - Einfache Tests für ein pattern matching
  - Führen zu konsistenterem und lesbarerem Code

```
max(X, Y) when X > Y -> X;
   max(X, Y) -> Y.
```

- Biting the **bits**
  - Syntax für das Extrahieren/ Verpacken von Bits
  - Sehr praktisch für binäre Protokolle (IPv4, MPEG, etc.)

```
<<?IP_VERSION:4,
  HLen:4, SrvcType:8, TotLen:16,
  ID:16, Flgs:3, FragOff:13,
  TTL:8, Proto:8, HdrChkSum:16,
  SrcIP:32, DestIP:32, RestDgram/binary>> = Message
```

Beispiel: löschen des letzten Elementes

```
%% Löscht das letzte Element einer Liste
%  Beispielaufruf: Erg = delete_last([a,b,c])
%
delete_last(List) -> delete_last(List,[]).
%
delete_last([_H|[]],NewList) -> NewList;
  delete_last([H|T],NewList) -> delete_last(T,NewList++[H]).
```


Variablen

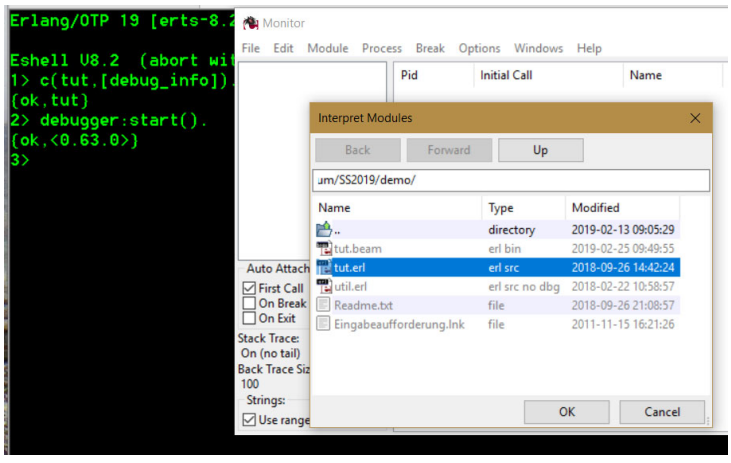
```
Eshell V8.2  (abort with ^G)
1> PID = demoRemote:start(server).
<0.58.0>
2> List = [1,2,3,4,5,6,7,8,9,0].
[1,2,3,4,5,6,7,8,9,0]
3> Test = {daten,PID,List}.
{daten,<0.58.0>,[1,2,3,4,5,6,7,8,9,0]}
4> b().
List = [1,2,3,4,5,6,7,8,9,0]
PID = <0.58.0>
Test = {daten,<0.58.0>,[1,2,3,4,5,6,7,8,9,0]}
ok
5> f(PID).
ok
6> b().
List = [1,2,3,4,5,6,7,8,9,0]
Test = {daten,<0.58.0>,[1,2,3,4,5,6,7,8,9,0]}
ok
7> f().
ok
8> server ! kill.
Received kill...
kill
9>
```

modul_info

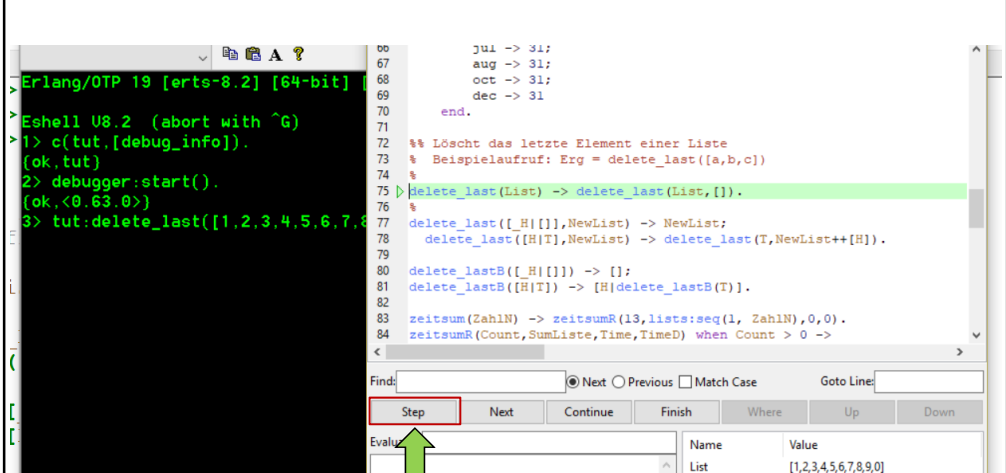
```
2> tut:module_info().
[{module,tut},
 {exports,[{sum,1},
           {sum,2},
           {sumD,1},
           {convert,2},
           {convert_length,1},
           {convert_lengthD,1},
           {test_if,2},
           {month_length,2},
           {delete_last,1},
           {delete_last,2},
           {delete_lastB,1},
           {zeitsum,1},
           {zeitsumR,4},
           {float_to_int,1},
           {powerset,1},
           {generate,1},
           {generate,3},
           {selectionS,1},
           {getmin,1},
           {getmin,2},
           {module_info,0},
           {module_info,1}]},
 {attributes,[{vsn,[66954540935715351249330336919156369097]}]},
 {compile,[{options,[]},
           {version,"7.0.3"},
```


Debugger

1> c(tut,[debug_info]).



Debugger



Aufgaben

Schreien Sie folgende Funktionen:

1. **mylist:del_element(P, E, L)**, löscht Element E aus der nicht typisierten Liste L an relativer Position P und gibt die neue Liste R zurück. P bezeichnet die Position: e erstes, l letztes, a alle Vorkommen von E.
2. **mylist:diffList(L1, L2)**, die aus den beiden Listen L1 und L2 die Differenzenliste L3 erzeugt (in L3 sind alle Elemente aus L1, die nicht in L2 sind).
3. **mylist:eo_count(L)**, zählt rekursiv die Anzahl der in der Länge geraden bzw. ungeraden Listen in der Liste L inklusive dieser Listen selbst, also alle möglichen Listen! Eine leere Liste wird als Liste mit gerader Länge angesehen. In der Liste können Elemente vorkommen, die keine Liste sind, d.h. L ist nicht typisiert. Rückgabewert ist ein Tupel mit {<Anzahl gerader Listen>,< Anzahl ungerader Listen>}.

Wie geht es weiter?

1. Informieren
explorieren
z.B. Artikel lesen

2. Planen
z.B. Arbeitsschritte festlegen

3. Durchführen
experimentieren
z.B. erste Programme schreiben

4. Kontrollieren
z.B. testen und besprechen

5. Bewerten
akkommodieren
z.B. Kriterien prüfen

6. Präsentieren
anwenden
z.B. im Praktikum

Erlang/OTP
Programm

z.B. Aufgabe 1