Karlsruher Institut für Technologie

# Seminar: Sprachen für Parallelprogrammierung (SS 2010)

Nils Adermann

IPD Snelting, Lehrstuhl Programmierparadigmen

# Ericsson Computer Science Lab 1980's

*to provide a better way of programming telephony applications*

Joe Armstrong, Erfinder von Erlang

Abbildung: Ericsson MD110 Telephone Exchange

# Erlang: Ein Prolog Meta-Interpreter

Karlsruher Institut für Technologie

Prolog:

- kurze, prägnante Notation
- keine Nebenläufigkeit

1986 ensteht Erlang als Prolog mit Prozessen

Entwicklung experimenteller Software mit Erlang:

- Sprache entfernt sich von Prolog
- Einflüsse aus EriPascal und Ada
- Aber: zu langsam für echte Projekte

# Die Sprache Erlang

- referenziell transparent
  Imperative Anweisung resultiert in Fehler:

```
X = X + 1.
```

- deklarativ funktional: „Was?” - nicht „Wie?”

```
-module(math)
-export([fac/1])

fac(N) when N > 0 -> N * fac(N - 1);
fac(0)            -> 1.
```

- nicht Seiteneffektfrei
- dynamisch typisiert

# Erlang: Die Fakultätsfunktion

```
-module(math)
-export([fac/1])
```

`math` Modul exportiert `fac` Funktion mit Arität 1.

```
fac(N) when N > 0 -> N * fac(N - 1);
```

Partielle rekursive Funktionsdefinition mit Guard (`N > 0`).

```
fac(0)                -> 1.
```

Rest der Funktionsdefinion mit Pattern Matching (`0`).

# Pattern Matching: Atome, Tupel & Listen

```
Point = {point, 1, 2}.
{point, X, Y} = Point.
X = 1. Y = 2.

L1 = [apple, 5, 3].
[Head|Tail] = L1.
Head = apple.
Tail = [5, 3].
Tail ++ [pear] = [5, 3, pear].
```

List Comprehensions:

```
[f(X)||X <- L, cond(x), ...]
[8,10] = [X*2||X <- [1,2,3,4,5], X > 3].
```

# Bit Syntax: IPv4 Datagram Header entpacken

Karlsruher Institut für Technologie

```
DgramSize = size(Dgram)
case Dgram of
  <<?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>> when
      HLen >= 5, 4*HLen =< DgramSize ->
        OptsLen = 4*(HLen - ?IP_MIN_HDR_LEN),
        <<Opts:OptsLen/binary,Data/binary>> =
            RestDgram,
        ...
```

# Erlang: Der Rest

Karlsruher Institut für Technologie

- Records
- Strings
- Case & If
- Funs
- ...

# Erlang bei Ericsson bis 1998

- Compiler (erste JAM, dann BEAM) und bessere VM
- Open Telecom Platform: Erlang Laufzeitbibliothek
- AXD 301 - Ericsson flagship ATM switch
  - 2,6 Mio. Zeilen Erlang
  - linear skalierbar: 10 - 160 Gbit/s
- Erkenntnis: Erlang für Projekte in großen Maßstäben geeignet

Ericsson verbannt Erlang um Kosten zu sparen

# Open Source Erlang

- HiPE (High Performance Erlang) kompiliert Erlang in nativen Maschinencode
- Bluetail AB

  *we had spent the last ten years designing and building fault-tolerant telecoms devices, we turned our attention to Internet devices*

  Joe Armstrong

# Warum Nebenläufigkeit?

Karlsruher Institut für Technologie

Telekommunikationssysteme müssen hoch verfügbar sein, d.h.

- Fehlertoleranz
- Robustheit
- Ausfallfreie Wartbarkeit

Erlangs Lösung:

- Redundante isolierte Prozesse für jede System-Komponente
- Kommunikation nur mittels Message Passing
- Hot Code Upgrade

*Shared-Nothing Architektur*

# Erlang Prozesse

- Prozesse brauchen wenig Speicher
- > 100k Prozesse können in < 10ms erstellt werden
- Wegen Isolation kein Locking nötig
- Isolation ermöglicht transparent verteilte Systeme (Erlang Nodes)
- Prozess-Links erlauben einem Prozess einen anderen zu überwachen und auf Fehler zu reagieren
- Jeder Prozess hat eine „Mailbox” für Kommunikation

# Concurrency Oriented Programming

Karlsruher Institut für Technologie

- Neuer Prozess: `Pid = spawn(Fun)` Mit Link: `Pid = spawn_link(Fun)`
- Nachricht senden: `Pid ! Message`
- Nachricht empfangen (Pattern Matching): `receive ... end`

# Concurrency Oriented Programming

Karlsruher Institut für Technologie

```
area_server() -> receive
        {rectangle, W, H} ->
            io:format("Rect: ~p~n", [W*H]),
            area_server();
        {circle, R} ->
            io:format("Circ: ~p~n", [3.141*R*R]),
            area_server()
    end.
Pid = spawn(fun area_server/0).
Pid ! {rectangle, 6, 10}.
   Rect: 60
```

# Error Handling

- Exceptions: `throw(no_such_element)`
  `try ... catch ... after ... end`
- Errors: `erlang:error(badarith)`
- Exits: `exit(unexpected_result)`

```
1> catch throw(whoa).
whoa
2> catch exit(die).
{'EXIT',die}
3> catch 1/0.
{'EXIT',{badarith,[{erlang,'/',[1,0]},
                   ......
```

# Demo

*Demo: Concurrency*

# Erlang auf Multicore CPUs

*In most other programming communities, the challenge of the multicore CPU is to answer the question, "How can I parallelize my program?" Erlang programmers do not ask such questions; their programs are already parallel.*

Joe Armstrong

# Wo wird Erlang eingesetzt?








und viele mehr ...

```
last_slide() ->
    receive
        {Person, question} ->
            Person ! answer(question),
            last_slide();
    after 30 ->
        true
    end.
```