

Concurrency Oriented Programming in Erlang

Joe Armstrong
Distributed Systems Laboratory
Swedish Institute of Computer Science

http://www.sics.se/~joe/talks/112_2002.pdf
joe@sics.se

November, 9, 2002



Agner Krarup Erlang (1878 - 1929)

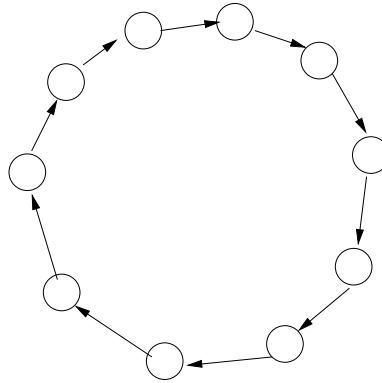
Plan

- Philosophy
- Language
- Wars
- Dollars

Not necessarily in that order.

Challenge 1

Put N processes in a ring:



Send a simple message round the ring M times.

Increase N until the system crashes.

How long did it take to start the ring?

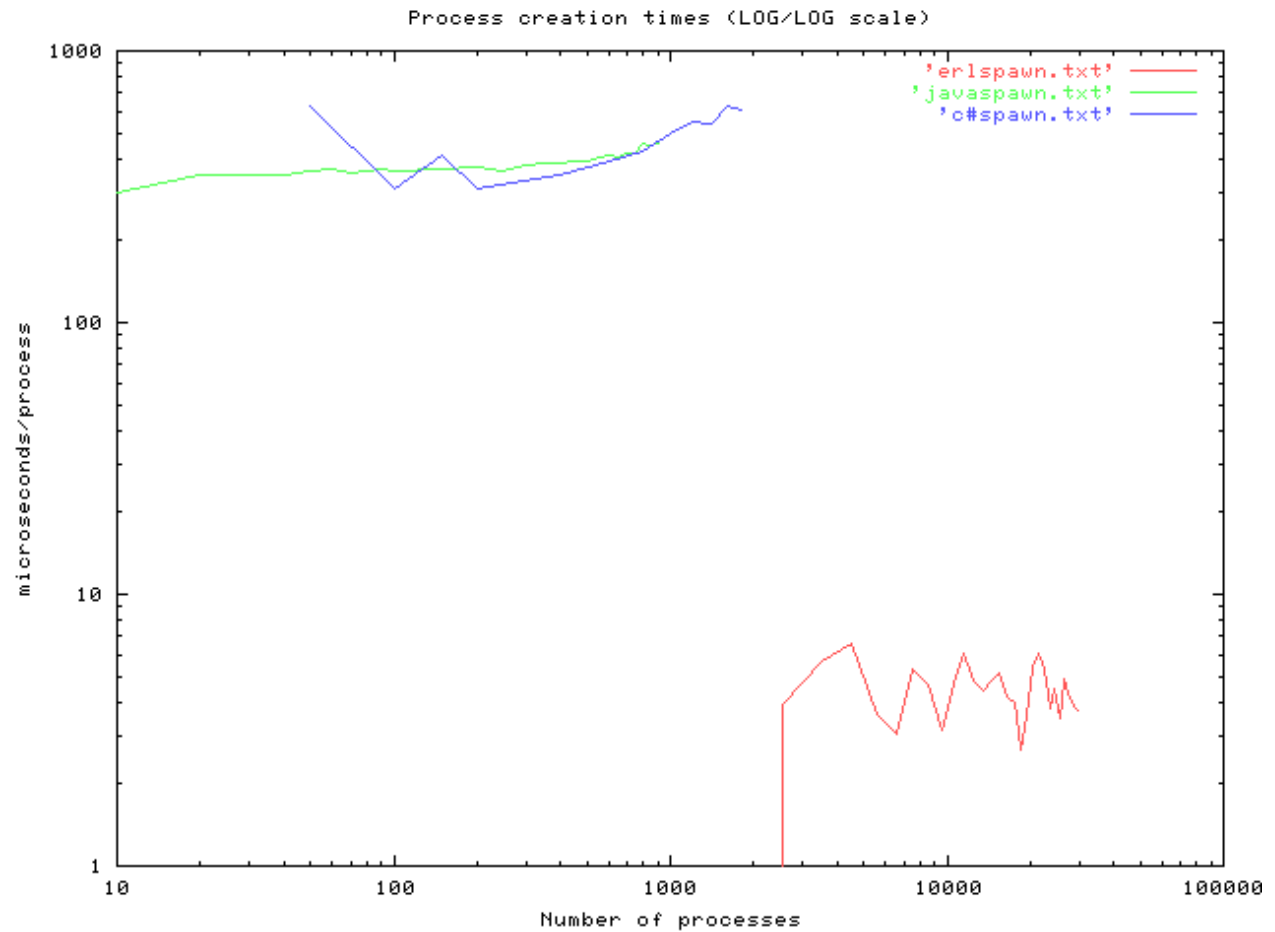
How long did it take to send a message?

When did it crash?

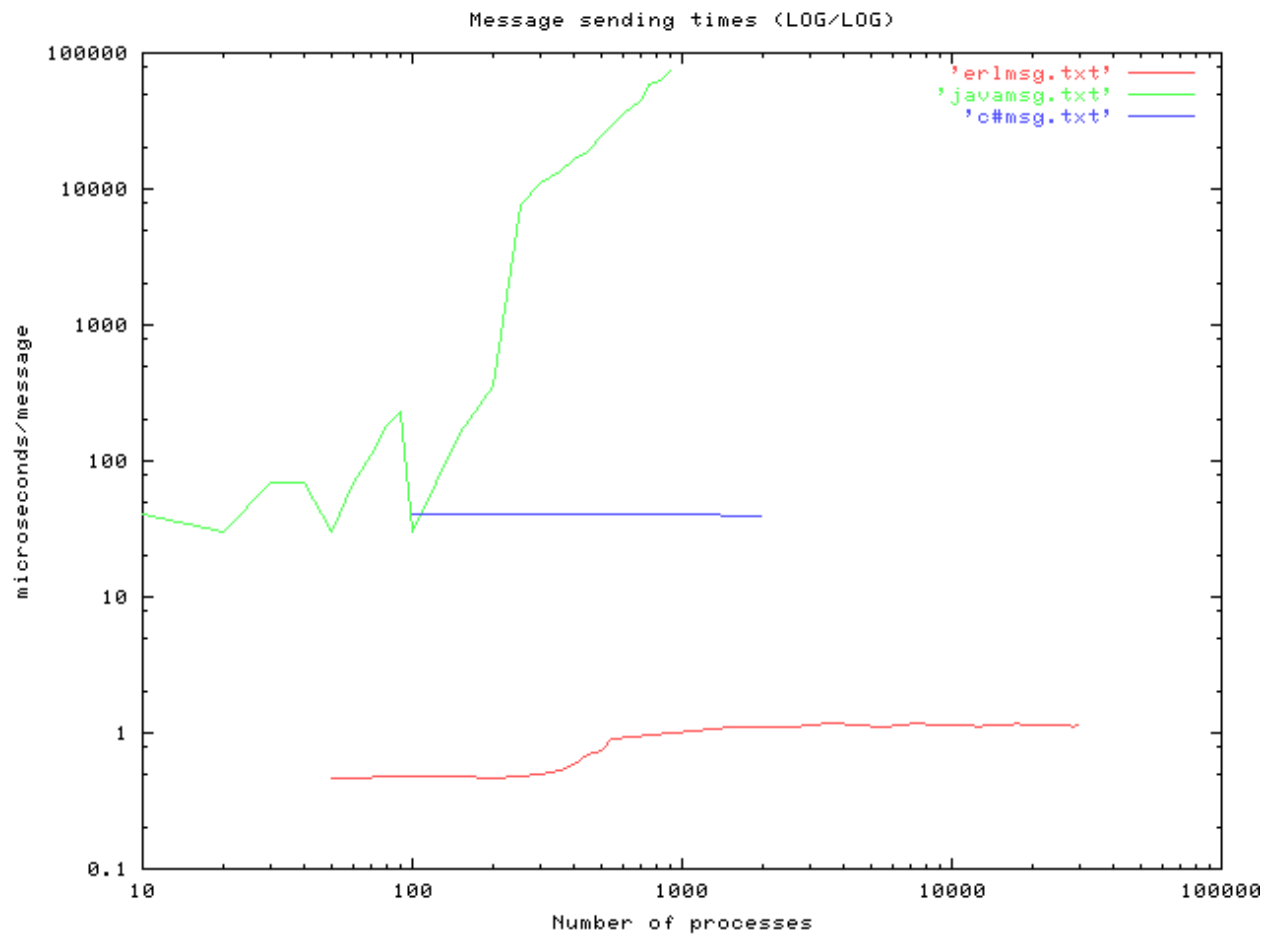
Can you create more processes in your language than the OS allows?

Is process creation in your language faster than process creation in the OS?

Process creation times



Message passing times



They forgot concurrency

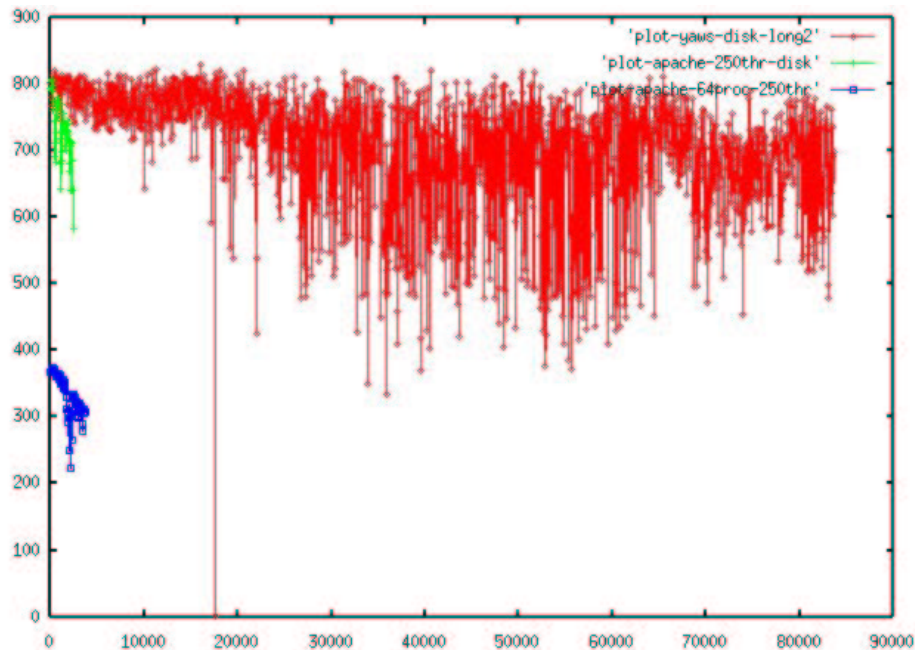
In languages like `<noname>` they forgot about concurrency. It either wasn't designed in from the beginning or else it was added on as an afterthought.

This doesn't matter for sequential programs.

If your problem is essentially concurrent then this is a fatal mistake.

But when you build something like a ...

Web Server



- Red = yaws (Yet another web server, in Erlang, on NFS)
- Green = apache (local disk)
- Blue = Apache (NFS)
- Yaws throughput = 800 KBytes/sec up to 80,000 disturbing processes)
- Apache misbehaves and crashes at about 4000 processes
- Details: <http://yaws.hyber.org>
<http://www.sics.se/~joe/apachevsyaws.html>

Philosophy

Concurrency Oriented Programming

- Processes are totally independent - *imagine they run on different machines*
- Process semantics = No sharing of data = Copy-everything message passing. Sharing = inefficient (can't go parallel) + complicated (mutexes, locks, ..)
- Each process has an unforgeable name
- If you know the name of a process you can send it a message
- Message passing is "send and pray" you send the message and pray it gets there
- You can monitor a remote process

Pragmatics

A language is a COPL if:

- Process are truly independent
- No penalty for massive parallelism
- No unavoidable penalty for distribution
- Concurrent behavior of program same on all OSs
- Can deal with failure

Why is COP Nice?

- The world is parallel
- The world is distributed
- Things fail
- Our brains intuitively understand parallelism (think driving a car)
- To program a real-world application we *observe* the concurrency patterns = no guesswork (only observation, and getting the granularity right)
- Our programs are *automatically* scalable, have *automatic fault tolerance* (if the program works at all on a uni-processor it will work in a distributed network)
- Make more powerful by adding more processors

What is Erlang/OTP?

- Erlang = a new(ish) programming language
- OTP = a set of libraries (making an Erlang application OS)
- New way of developing distributed fault-tolerant applications (OTP)
- Battle tested in Ericsson and Nortel products. AXD301, GPRS, SSL accelerator...
- Biggest COPL used to earn money in the world ...
- Developed and maintained by a very small team ... Björn, Klacke, Robert, Mike,...
- A tool for writing reliable applications

Erlang

- Functional/single assignment
- Light weight processes
- Asynchronous message passing (send and pray)
- OS independent (true)
- Special error handling primitives
- Lists, tuples, binaries
- Dynamic typing
- Soft real-time GC
- Transparent distribution

Erlang in 11 minutes

One minute per example.

- Sequential Erlang in 5 examples
- Concurrent Erlang 2 examples
- Distributed Erlang 1 example
- Fault-tolerant Erlang in 2 examples
- Bit syntax in 1 example

Sequential Erlang in 5 examples

1 - Factorial

```
-module(math).  
-export([fac/1]).  
  
fac(N) when N > 0 -> N * fac(N-1);  
fac(0)             -> 1.  
  
> math:fac(25).  
15511210043330985984000000
```

2 - Binary Tree

```
lookup(Key, {Key, Val, _, _}) ->  
  {ok, Val};  
lookup(Key, {Key1, Val, S, B}) when Key < Key1 ->  
  lookup(Key, S);  
lookup(Key, {Key1, Val, S, B}) ->  
  lookup(Key, B);  
lookup(Key, nil) ->  
  not_found.
```

3 - Append

```
append([H|T], L) -> [H|append(T, L)];  
append([], L) -> L.
```

4 - Sort

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

5 - Adder

```
> Adder = fun(N) -> fun(X) -> X + N end end.  
#Fun  
> G = Adder(10).  
#Fun  
> G(5).  
15
```

Concurrent Erlang in 2 examples

6 - Spawn

```
Pid = spawn(fun() -> loop(0) end)
```

7 - Send and receive

```
Pid ! Message,  
.....  
  
receive  
  Message1 ->  
    Actions1;  
  Message2 ->  
    Actions2;  
  ...  
  after Time ->  
    TimeOutActions  
end
```


Distributed Erlang in 1 example

8 - Distribution

```
...  
Pid = spawn(Fun@Node)  
...  
alive(Node)  
...  
not_alive(Node)
```

Fault tolerant Erlang in 2 examples

9 - Catch/throw

```
...
case (catch foo(A, B)) of
  {abnormal_case1, Y} ->
    ...
    {'EXIT', Opps} ->
      ...
      Val ->
        ...
end,
...

foo(A, B) ->
  ...
  throw({abnormal_case1, ...})
```

10 - Monitor a process

```
...
process_flag(trap_exit, true),
Pid = spawn_link(fun() -> ... end),
receive
  {'EXIT', Pid, Why} ->
    ...
end
```

Bit syntax in 1 example

11 - Parse IP Datagram

Dgram is bound to the consecutive bytes of an IP datagram of IP protocol version 4. We extract the header and the data of the datagram:

```
-define(IP_VERSION, 4).
-define(IP_MIN_HDR_LEN, 5).
```

```
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, Body/binary>> when
    HLen >= 5, 4*HLen =< DgramSize ->
      OptsLen = 4*(HLen - ?IP_MIN_HDR_LEN),
      <<Opts:OptsLen/binary,Data/binary>> = Body,
    ...
end.
```

Behaviours

A universal *Client - Server* with hot code swapping :-)

```
server(Fun, Data) ->
  receive
    {new_fun, Fun1} ->
      server(Fun1, Data);
    {rpc, From, ReplyAs, Q} ->
      {Reply, Data1} = Fun(Q, Data),
      From ! {ReplyAs, Reply},
      server(Fun, Data1)
  end.
```

```
rpc(A, B) ->
  Tag = new_ref(),
  A ! {rpc, self(), Tag, B},
  receive
    {Tag, Val} -> Val
  end
```

Programming Patterns

Common concurrency patterns:



Cast

```
A ! B
```

Event

```
receive A -> A end
```

Call (RPC)

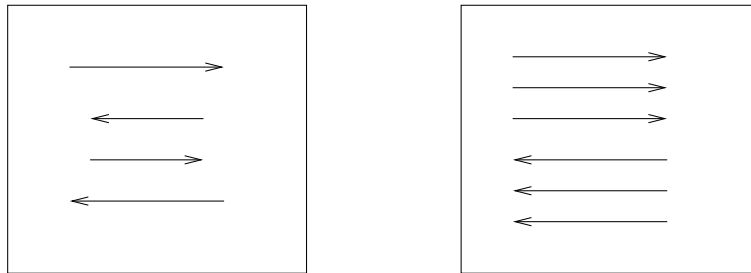
```
A ! {self(), B},
receive
  {A, Reply} ->
    Reply
end
```

Callback

```
receive
  {From, A} ->
    From ! F(A)
end
```

Challenge 2

Can we easily program tricky concurrency patterns?



Callback within RPC

```
A ! {Tag, X}, g(A, Tag).
```

```
g(A, Tag) ->  
  receive  
    {Tag, Val} -> Val;  
    {A, X} ->  
      A ! F(X),  
      go(A, Tag)  
  end.
```

Parallel RPC

```
par_rpc([A,B,C], M)
```

False encapsulation

Q: Should we hide the message passing of an RPC inside a function stub?

A: No - can't do parallel RPC's

How do we *implement* parallel RPCs?

```
par_rpc(Ps, M) ->
  Self = self(),
  Tags = map(fun(I) ->
    Tag = make_ref(),
    spawn(fun() ->
      Val = rpc(I, M),
      Self ! {Tag, Val}
    end),
    Tag
  end, Ps),
  yield(Tags).

yield([]) ->
  [];
yield([H|T]) ->
  Val1 = receive {H, Val} -> Val end,
  [Val1|yield(T)].
```


Is Erlang an COPL?

Yes - ish.

Fine for LAN behind a firewall. Various security policies need implementing for use in a WAN.

Wars

- Erlang is secret (sssh
- Erlang is slow ...
- Erlang banned in Ericsson (for new products) - it wasn't C++ :-)
- Erlang escapes (Open source)
- Erlang infects Nortel
- Erlang still used in Ericsson (despite ban)
- Erlang controls the world (or at least the BT networks)

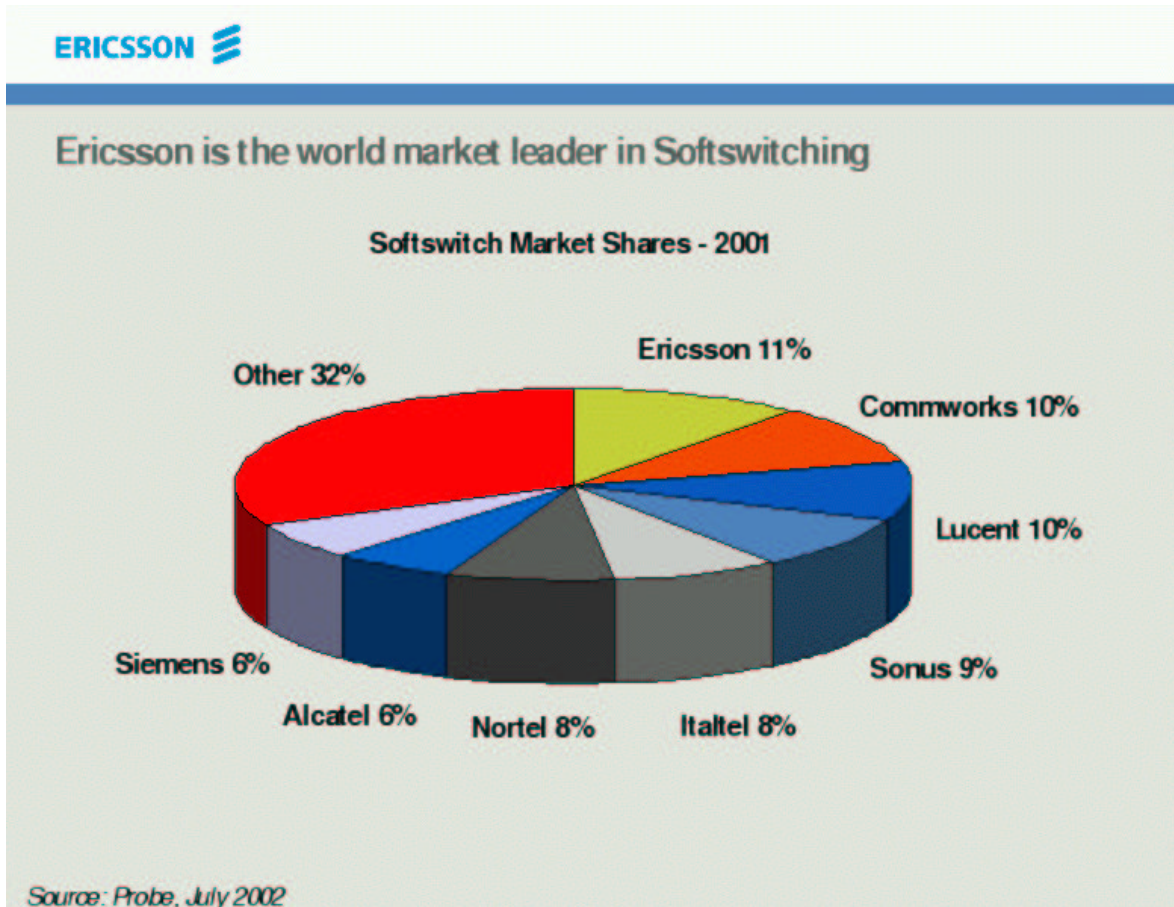
Dollars

- AXD301
- GPRS
- Nortel SSL accelerator
- Bluetail - *very successful Swedish IT startup*


AXD301

- ATM switch (Media gateway=AXD, Soft switch (ENGINE) =AXD+AXE)
- 11% of world market = Market leader
- 99.99999999% reliability (9 nines) (31 ms. year!)
- 30-40 million calls per week
- World's largest telephony over ATM network
- 1.7 million lines of Erlang

AXD Market share



9 nines reliability





ENGINE Integral proves outstanding capacity capability....

BT, UK (ENGINE Integral in Hybrid configuration)


During the very popular "Pop Idol" TV show in February 2002, the Ericsson server in Ilford was on two occasions exposed to 3.4 and 2.8 MBHCA. The dynamic congestion control kicked in and limited the throughput to the configured 1.4 MBHCA. The situation was handled smoothly and did not cause any problems with the server.

In September 2002, 14 nodes has been deployed out of planned 23 nodes by the end of 2002. BT are handling about 30-40 Million calls per node and week. In total BT has switched circa 7 Billion calls to date using VoATM Technology and ATM Transport (September 2002).




Source: Ericsson, BT

High performance

ERICSSON 

BT, UK chooses Ericsson and ENGINE Integral for migration of it's transit telephony network to the world's largest Telephony over ATM network

Situation: Business Drivers	Solution	
 <ul style="list-style-type: none"> ? Existing transit circuit-switched network needed modernization ? Rapid traffic growth from new and existing services ? Increase capacity and reduce cost through evolution to new multi-service communication system capable of carrying all telephony, data and multi-media services 	<ul style="list-style-type: none"> ? Partnership ? ENGINE Integral in hybrid configuration for >50% of BT transit network - 23 nodes across UK ? Management system ? Live cut-over from NB switches 	
	<th data-bbox="917 961 1339 1010">Result</th>	Result
	<ul style="list-style-type: none"> ? 14 nodes carrying live traffic September 2002 out of planned 23 before end of 2002 (according to time plan) ? 99,9999999% availability ? 30-40 Million calls per week & node ? World's largest Telephony over ATM network ? Best Supplier of the year, 2000 	

Alteon SSL Accelerator



Nortel networks press release 28 August, 2002

In the security arena, Infonetics Research credited Nortel Networks with the #1 position in the dedicated Secure Sockets Layer (SSL) appliance market with 48 percent market share in the first half of 2002, improving on the 41 percent market share reported for first half of 2001.

Total market c. 42M\$ (2002) growing to 60\$M 2005 (Infonetics research)

GPRS

- 45% of world market
- 79 commercial contracts
- 76% of code is in Erlang

Recap

- The world is concurrent
- Things in the world don't share data
- Things communicate with messages
- Things fail

Model this in a language

Now you must ...

Make it easy for your users to program concurrent, distributed fault-tolerant applications.