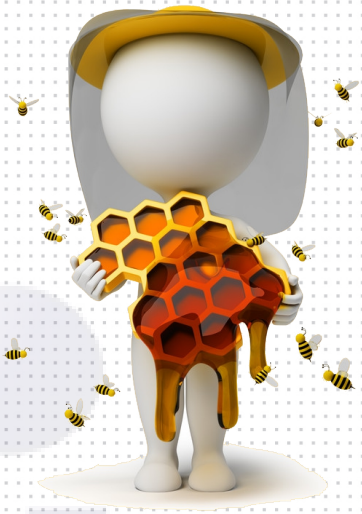
GDC 12 ONLINE

Why … Erlang?

Henning Diedrich
CEO Eonblast



EONBLAST

Your Host

Henning Diedrich

- Founder, CEO Eonblast
- CTO Freshworks
- CTO, Producer at Newtracks
- Team Lead, Producer at Bigpoint
- OS Maintainer Emysql, Erlvolt

Acknowledgements

Thank You!

Joe Armstrong	
Robert Virding	*Erlang Solutions*
Ulf Wiger	*Feuerland Labs*
Felix Geisendörfer	*Transloadit*

… for vetting and improving these slides in various stages.
All errors and omissions are, of course, mine.

Why Erlang?

1. Why Care About It?
2. Who Uses It?
3. What for?
4. Is It for Me?
5. How It Looks
6. Getting Started!

Erlang may be to Java what Java was to C++

Erlang may be to Java what Java was to C++

C++ – pointers = Java

Java – deadlocks = Erlang

Erlang may be to Java what Java was to C++

„Drop Slides 4 and 5“

Joe Armstrong

Erlang may be to Java what Java was to C++

Erlang is a lot more ...

Who Uses It?

You are Using It

„You probably use systems based on Erlang/OTP every day without knowing it.“

Mike Williams

Erlang Game Servers

Zynga: FarmVille via membase, Activision Blizzard: Call of Duty, Bigpoint: Battle Star Galactica, Wooga: Magic Land

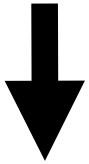
Distributed DBs using Erlang

Membase, riak, BigCouch

Handling state: secure, fast and distributed.

EA contributed Emysql

RUPTURE



http://eonblast.github.com/Emysql

EONBLAST

The Erlang Poster Child

Klarna AB

- Financial Services for E-Commerce
- 600 Employees, $38M revenue
- 12,000 e-commerce stores
- 30 seconds downtime in 3 years
- Investment by Sequoia Capital

Sequoia Capital

1975 Atari
1978 Apple
1982 Electronic Arts
1987 Cisco
1993 Nvidia
1995 Yahoo!
1999 Google
1999 Paypal

2000 Rackspace
2003 LinkedIn
2005 YouTube
2007 Dropbox
2009 Unity 3D
2010 Klarna
2012 Instagram

Why Use It?

Why Erlang?

Business Perspective

- Reduce Costs
- Improve Retention
- Shorten Time To Market

Why Erlang?

Production Perspective

- High Productivity
- Low Hardware Requirements
- More Robust Servers

Why Erlang?

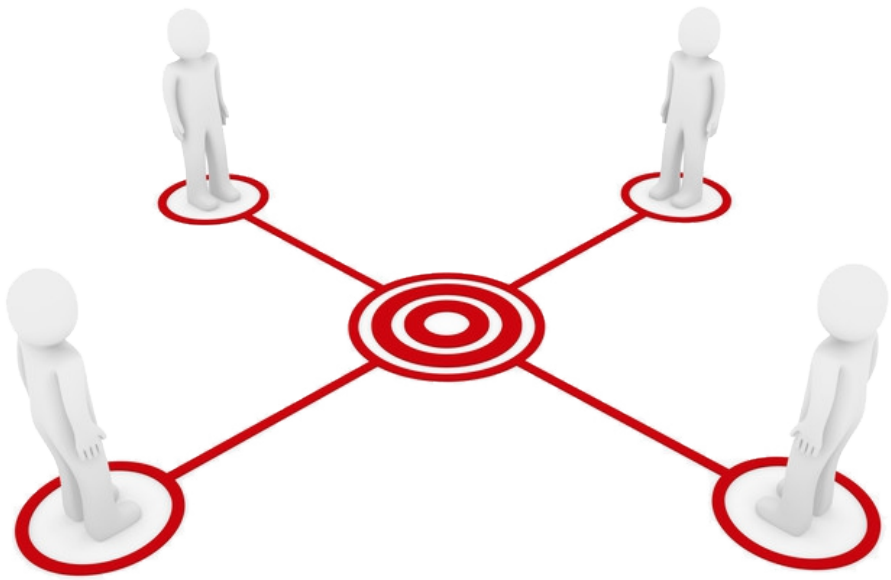
Design Perspective

- More Complex Designs
- Profitable On Small Markets
- Less Mainstreaming Pressure

When Use It?

Sweet Spots

- Stateful Servers with High Throughput
- Cluster Distribution Layers
- Chats*



* Chats are a bitch. The Facebook Chat was written in Erlang.

Why Is It Good At These Things?

Origins

ERICSSON

PLEX

- Ericsson makes billions with telecom switches
- They used PLEX, an all proprietary software
- PLEX delivers, but has bad productivity

Origins

- The 80's: Ericsson Computer Science Lab

  Joe Armstrong, Robert Virding, Mike Williams

  „What aspects of computer languages make it easier to program telecom systems?"

Origins

Mission

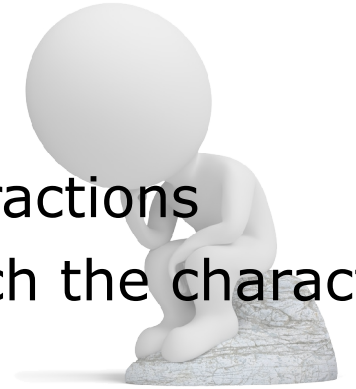
- Keep features, but invent a more productive PLEX.

Approach

- Programmed a small telephone exchange (MD110) in
  Prolog, CHILL, Ada, Concurrent Euclid,
  Rules Based Systems, AI Systems, Functional Langs

Conclusion

- Many good abstractions
- None could match the characteristics of PLEX



Origins

PLEX

- **Safe pointers**
- Ability to change size of arrays etc **without memory leaks**
- Fine grained **massive concurrency**
- Ability to develop software in **independent "blocks"**
- Ability to **change code at runtime** without stopping
- **Advanced tracing** ability at runtime
- **Restart Mechanisms** to recover software & hardware failure

Erlang was Built For

- Reliability
- Maintenance
- Distribution
- Productivity

EONBLAST

Features Achieved

- Productive
- Reliable
- Fast
- Scalable
- Great to Maintain

… how?

The Magic

- **Microprocesses**
- **Pattern Matching***
- **Immutable Variables**

* Not your familiar Regex string matching

EONBLAST

What Is That?

Thinking Erlang

- The Actor Model
- Thinking Parallel
- Thinking Functional
- Thinking Processes
- Let It Crash!

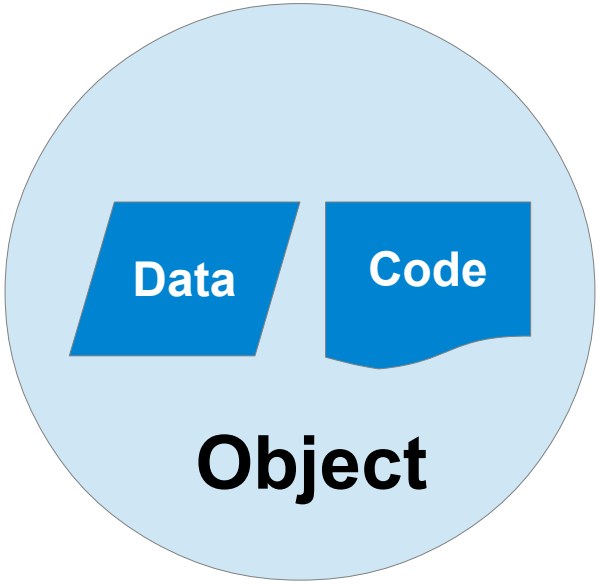
Actor Model vs. OO

The Actor Model

Carl Hewitt 1973

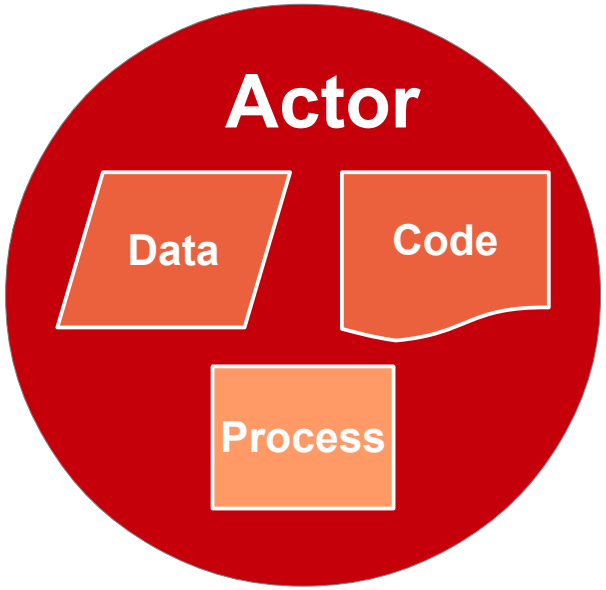
- Behavior
- State
- Parallel
- Asynchronous Messages
- Mailboxes
- No Shared State

Object Oriented



- Data + Code
- Encapsulation
- Inheritance
- Polymorphy
- Late Binding

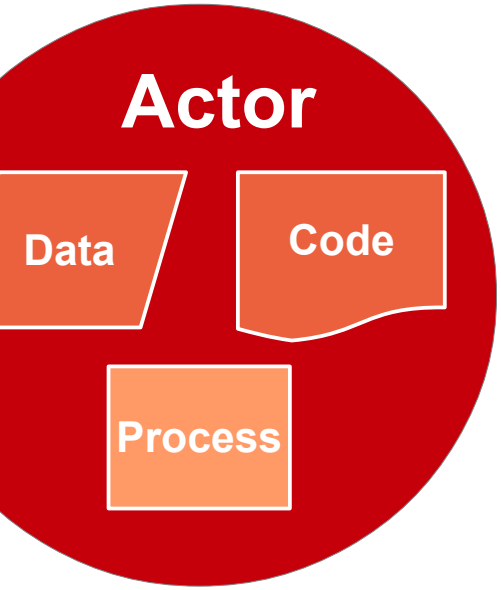
Actor Model

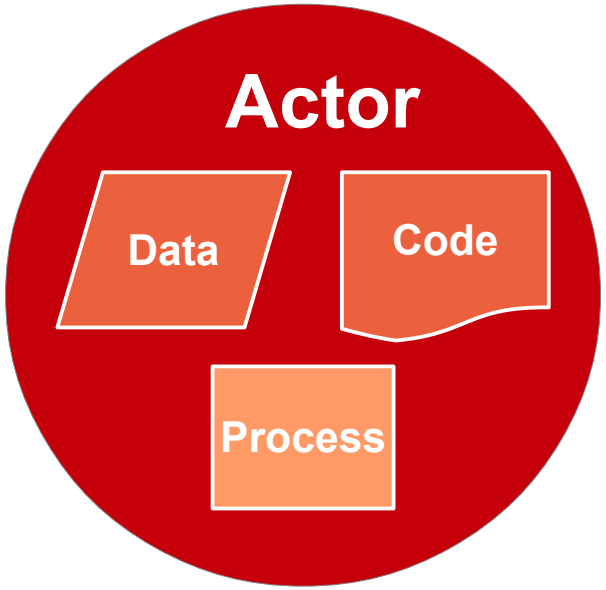


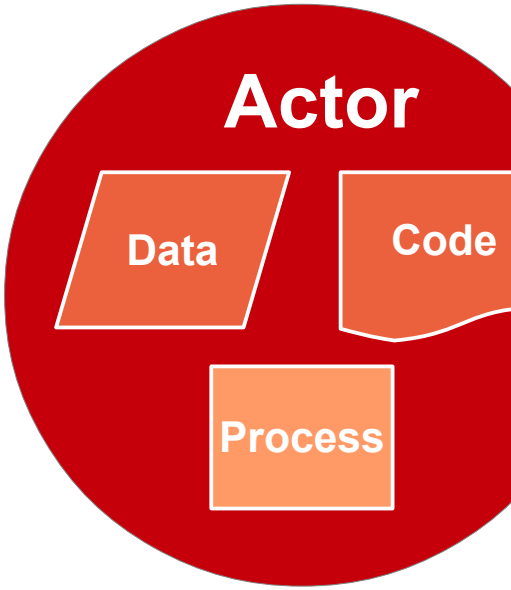
- Data + Code **+ Process**
- Self-Contained Machines
- Stronger Encapsulation
- Less Inheritance
- Type Inference
- Hot Code Upgrades

EONBLAST

Actor Model



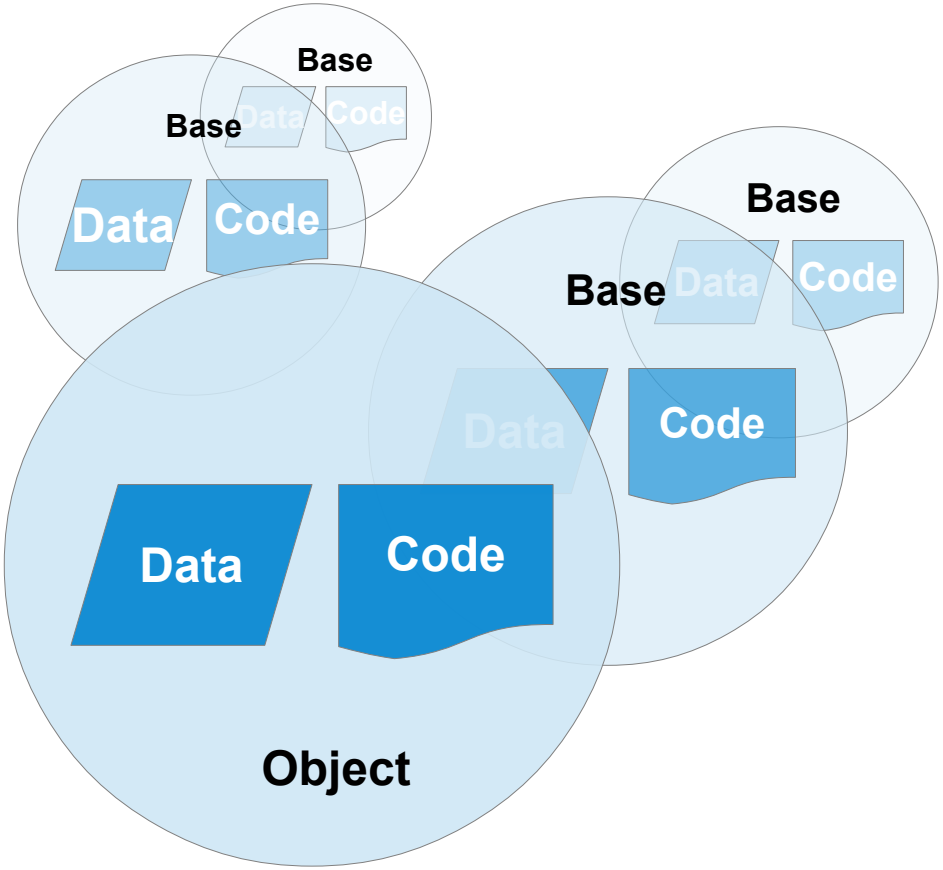




- Data + Code **+ Process**
- Self-Contained Machines
- Stronger Encapsulation
- Less Inheritance
- Type Inference
- Hot Code Upgrades

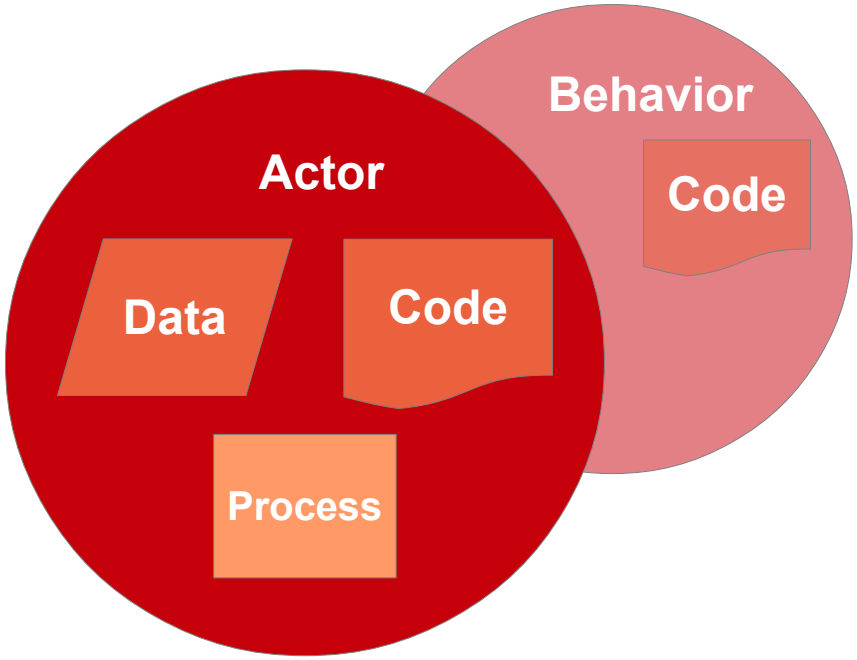
EONBLAST

OO Inheritance



- Inheritance of Class
- Multi-Level, Multi-Branch
- Overloading

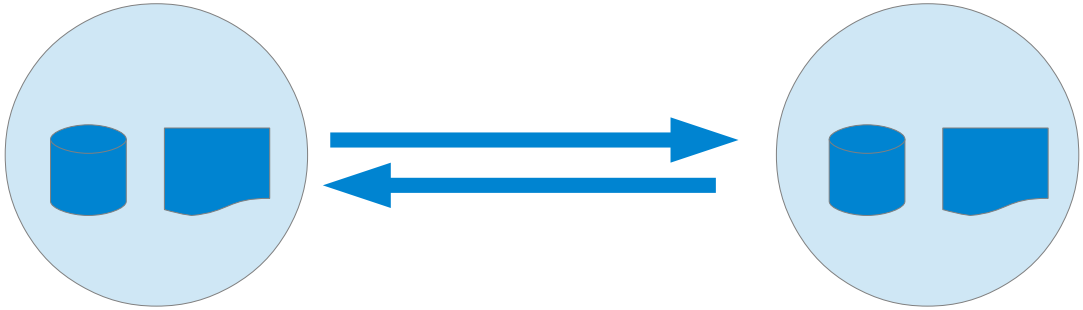
Erlang Behavior



- Inheritance of Behavior only.
- Usually only one level deep.
- Usually one of the standard OTP behaviors:
  Generic Server, Event, State Machine, Supervisor.

OO Methods: Synchronous Calls

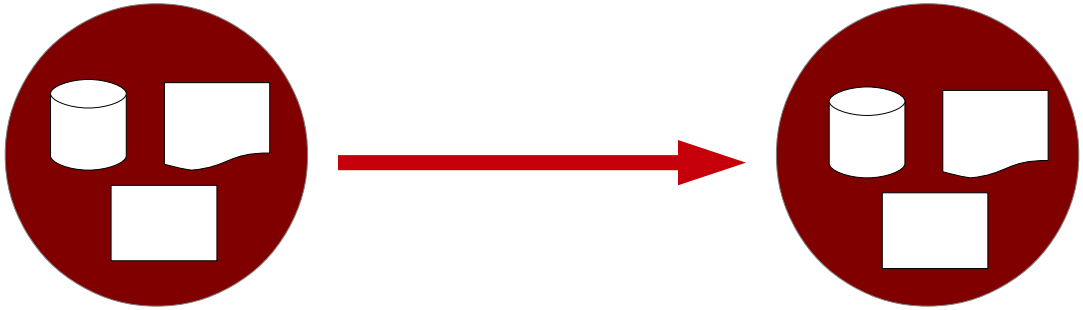
```
o.method(a)
```



- OO “method calls” are simply **synchronous** function calls.
- Not really the OO “messages” once promised.
- OO fails itself where building on Algol.

Actors: Asynchronous Messages

Pid ! Msg



- Message dispatch is one-way, truly **asynchronous**.
- **Not** function calls but something in their own right.
- Clean break from the FP paradigm.

Actor Model: Benefits

- More true to the real world
- Better suited for parallel hardware
- Better suited for distributed architectures
- Scaling garbage collection (sic!)
- Less Magic

Thinking Processes

Thinking Processes

- **What** should be a Process?

„Easy!“

Joe Armstrong

Thinking Processes

- Three Elevators
- Ten Floors
- How many processes?

Thinking Processes

Thirteen!

„It's so obvious!" - Joe Armstrong

- elevators hold state
- floors hold state
- All live separate lives
- All don't share state
- Elevators and floors interact independently

The Algorithm courtesy Joe:

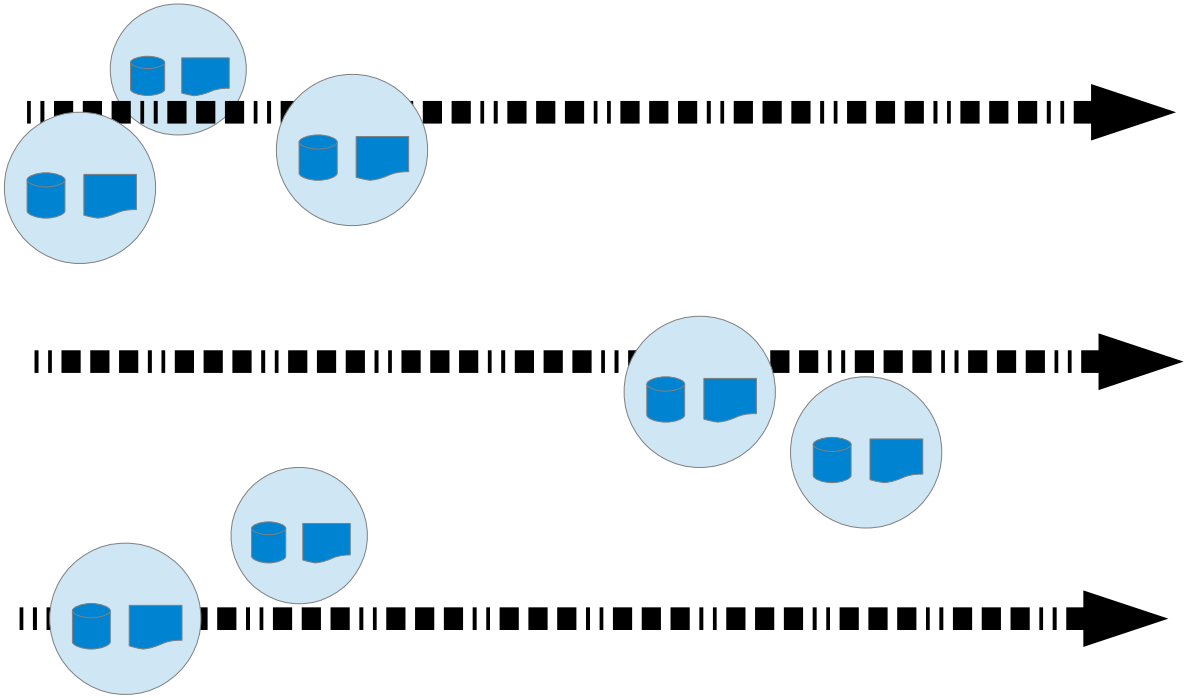
1. *each floor has it's own **stop list***
2. *when you press the "up" button on floor K you **broadcast** to all lifts "I want to go up, how long will it take to get to me?"*
3. *each lift computes this independentally and*
4. *sends the result to floor K.*
5. *Floor K waits for 3 messages then*
6. *Chooses the minimum*
7. *then sends a message to this list "add me to your stop list."*

Thinking Processes

Processes

- Don’t share State
- Communicate Asynchronously
- Are Very Cheap to create and keep
- Monitor Each Other
- Provide Contention Handling
- Constitute the Error Handling Atom

Objects share Threads

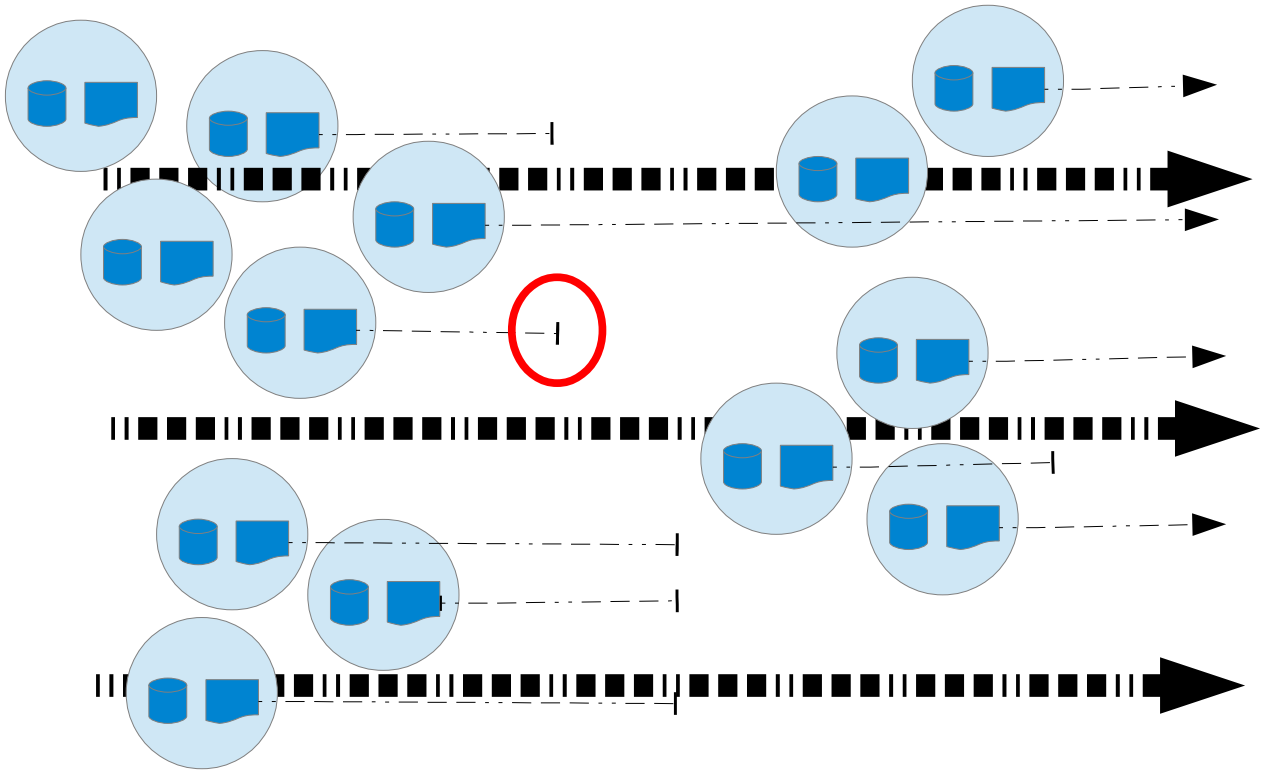


- Multiple objects share threads.
- Objects can be accessed across threads.
- Threads - and objects - share state.

Actors *are* Processes

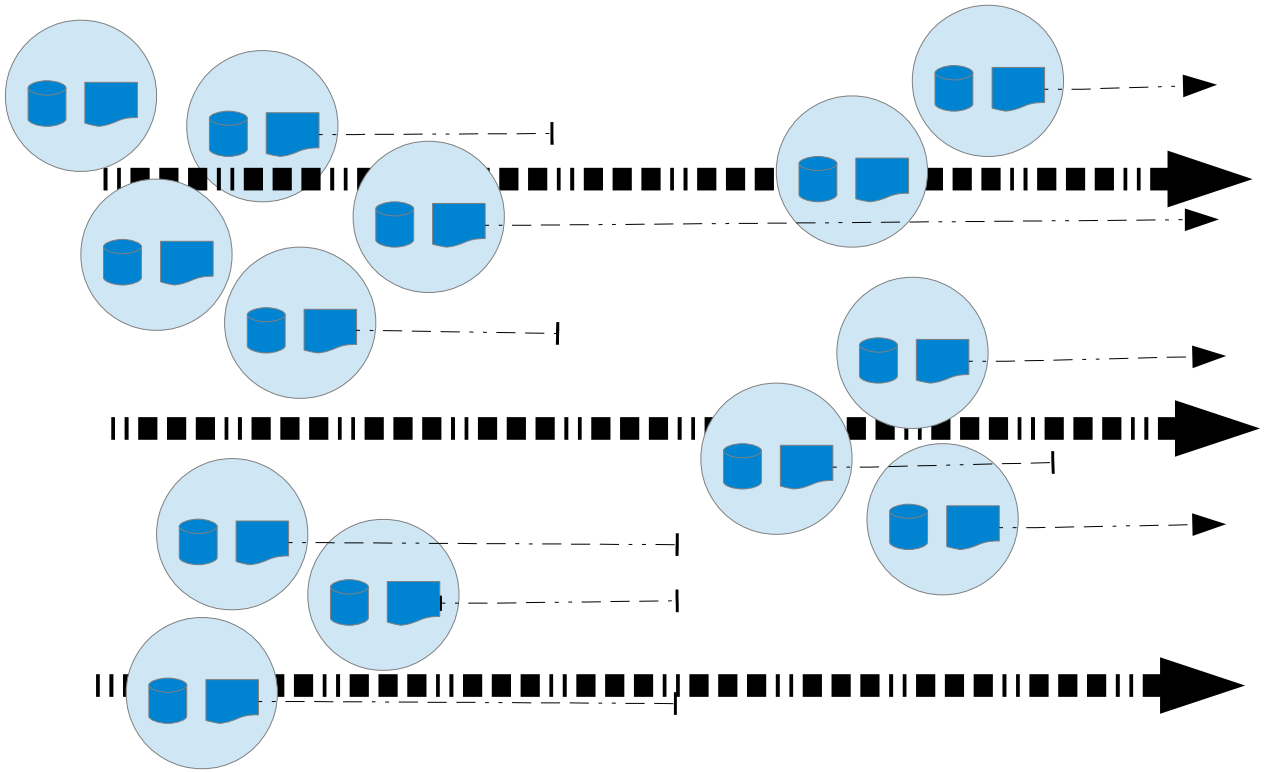
- State, code and process form a unity: the actor.
- Like processes, actors do **not** share state.
- In fact, like humans. Who mostly work quite well.

Objects and Threads



Lifetime & Destruction

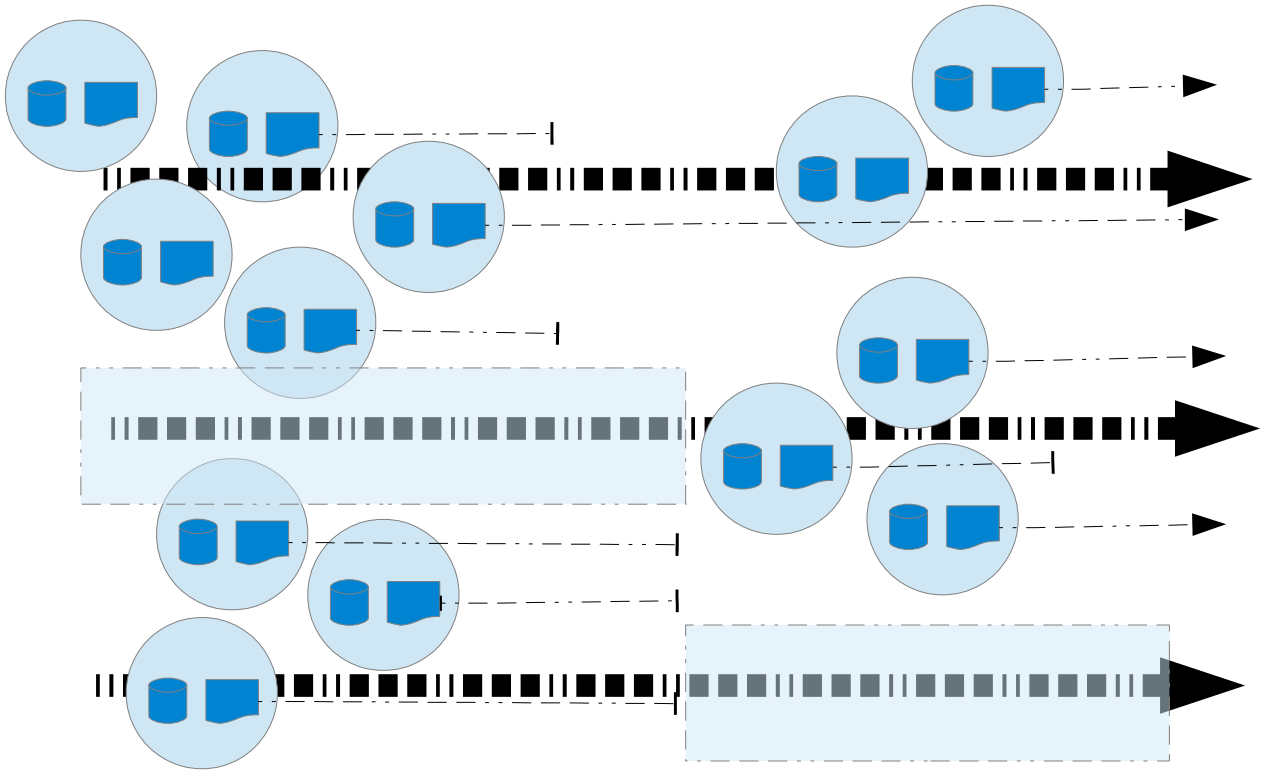
Objects and Threads



C C++ C# Java JavaScript Node Lua Python

EONBLAST

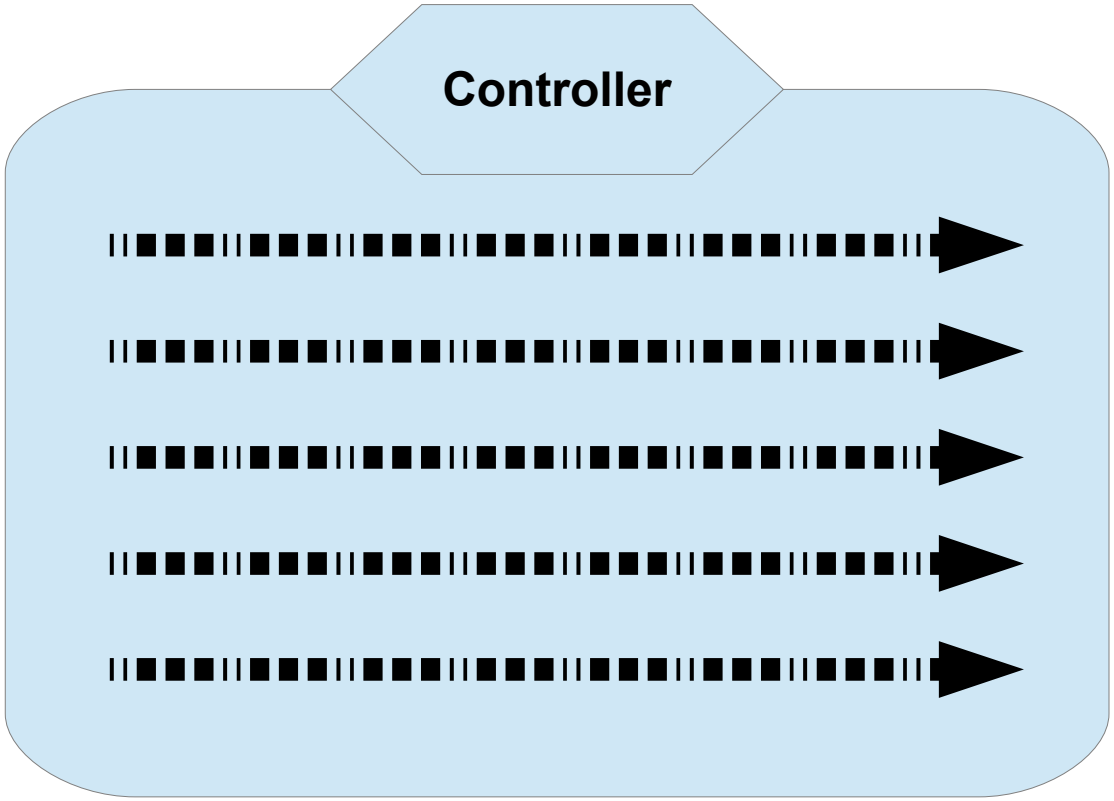
Objects and Threads



Idle Threads

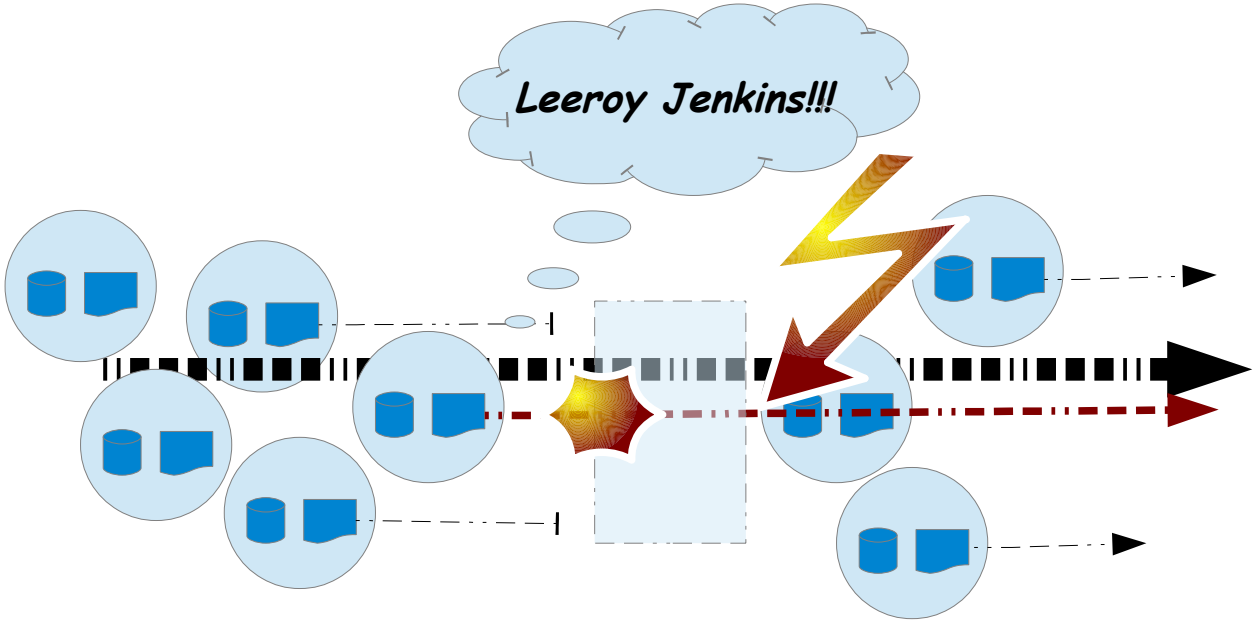
EONBLAST

Objects and Threads



Thread Pooling for Recycling

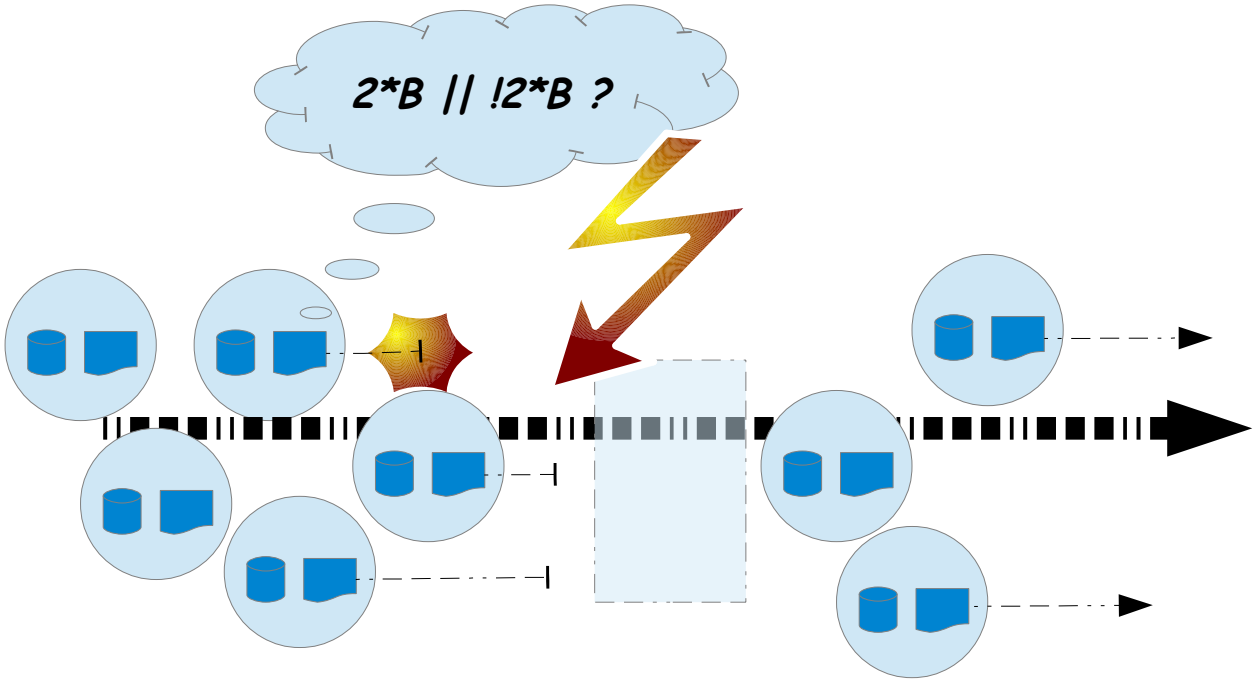
Objects and Threads



Unwanted surviving objects

EONBLAST

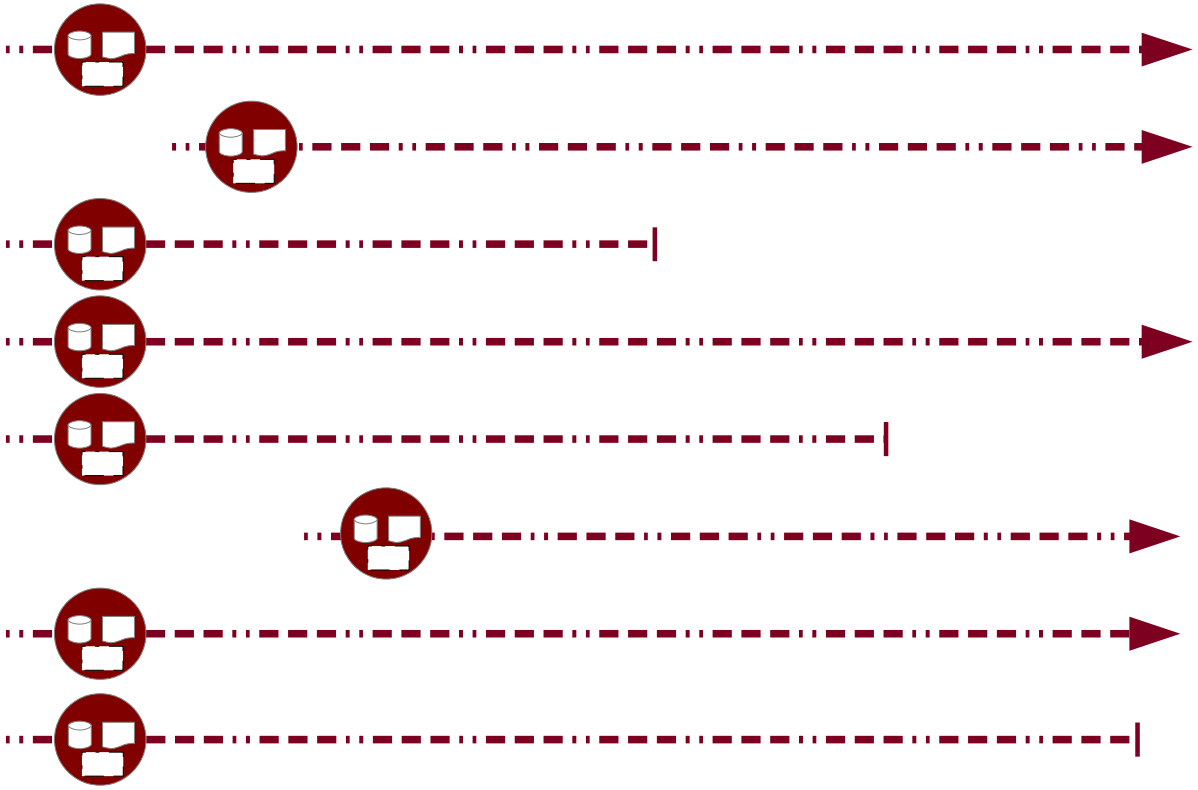
Objects and Threads



Prematurely destroyed objects

EONBLAST

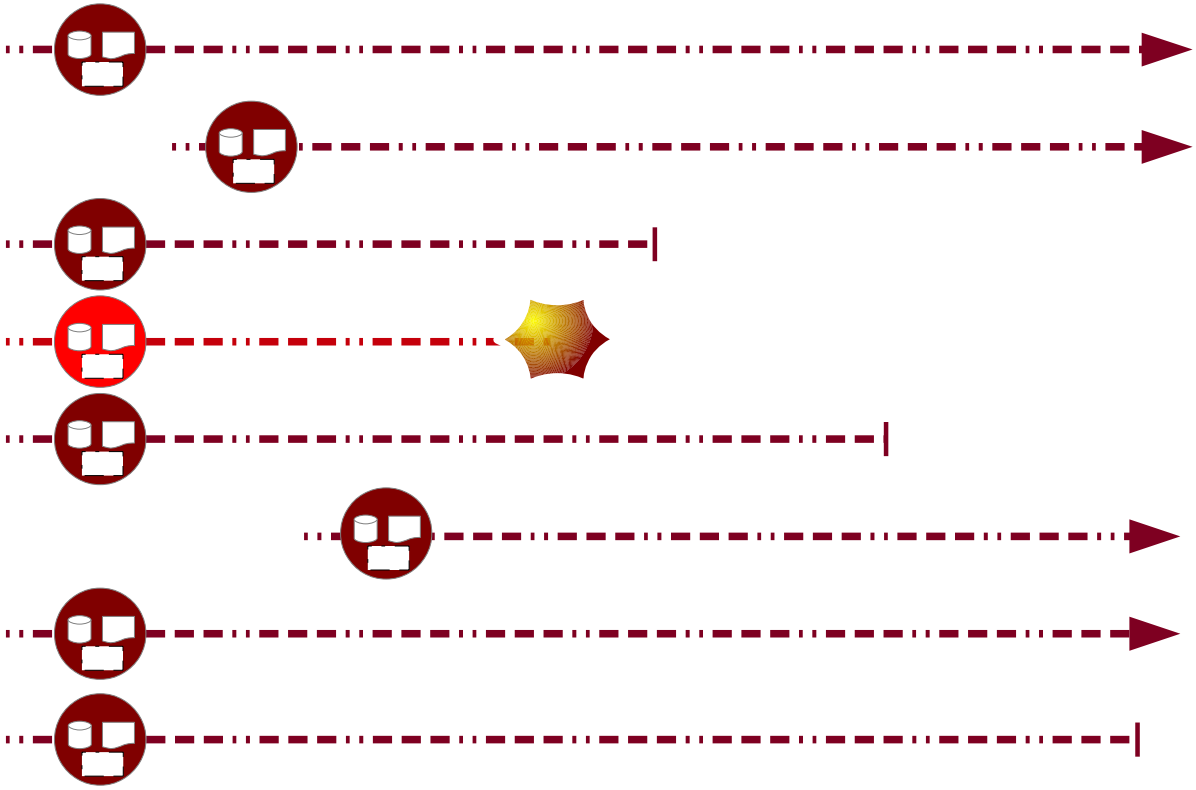
Erlang Processes



Erlang Actors: State + Code + Process

EONBLAST

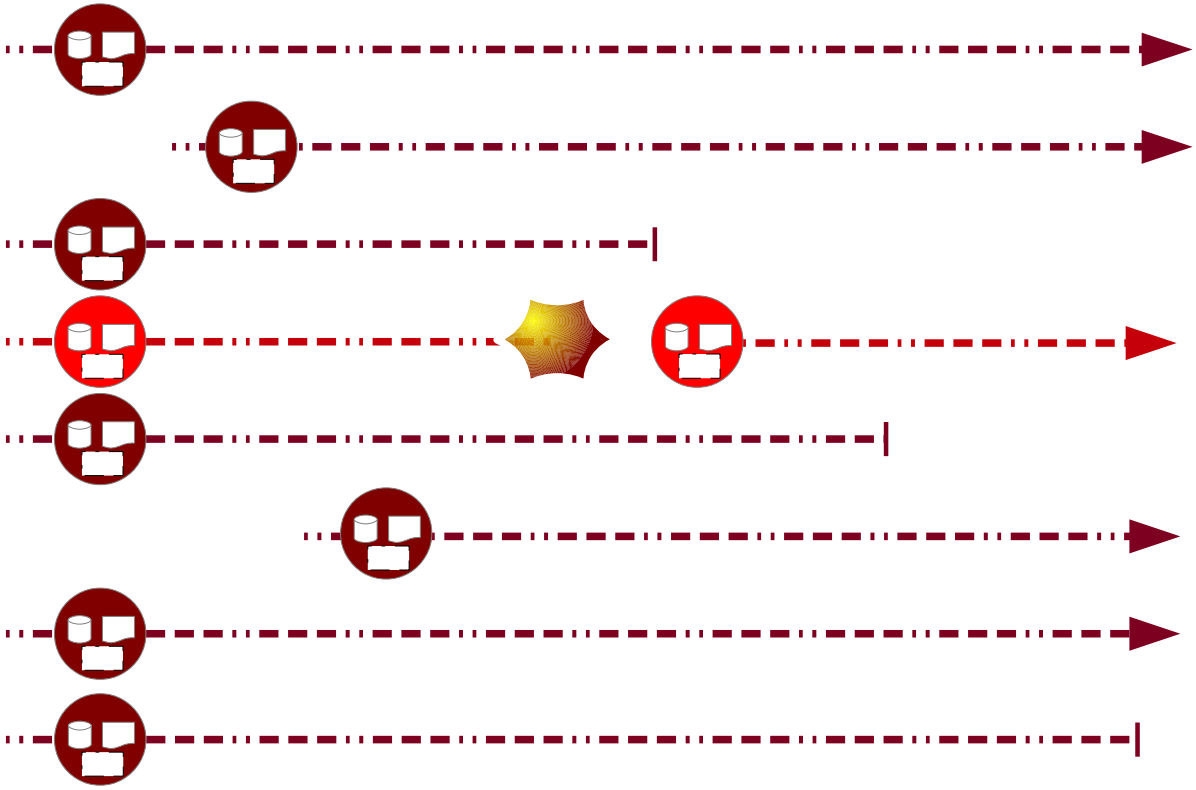
Erlang Processes



One dies.

EONBLAST

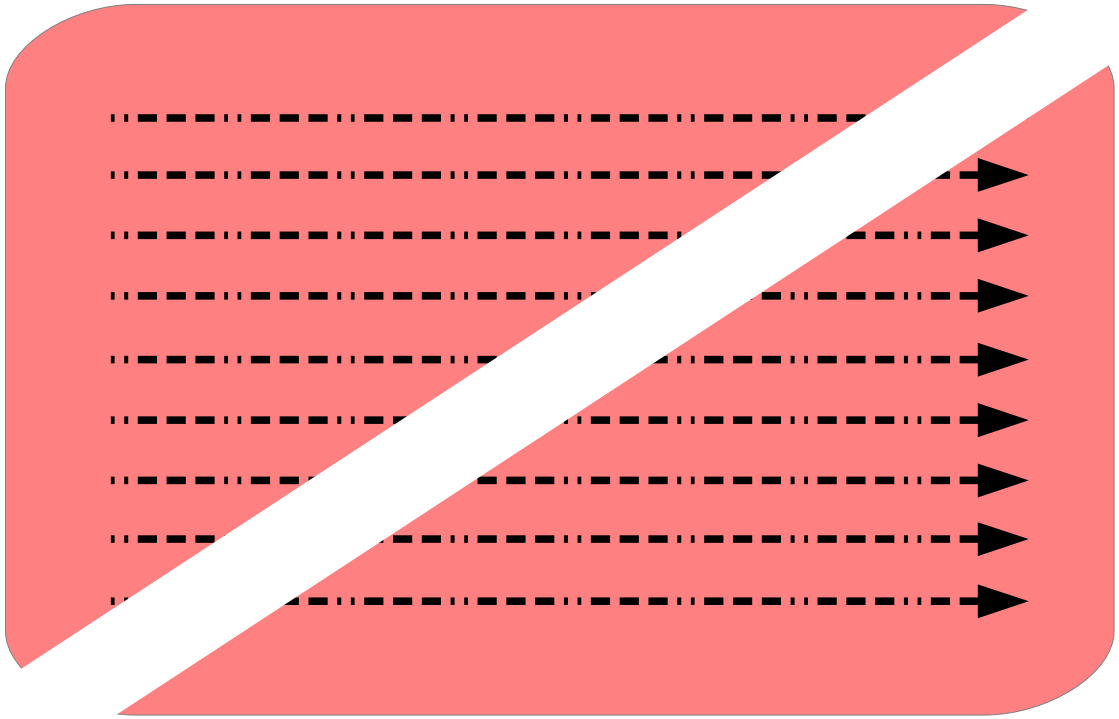
Erlang Processes



The Erlang way: the process is restarted.

EONBLAST

Processes are Cheap



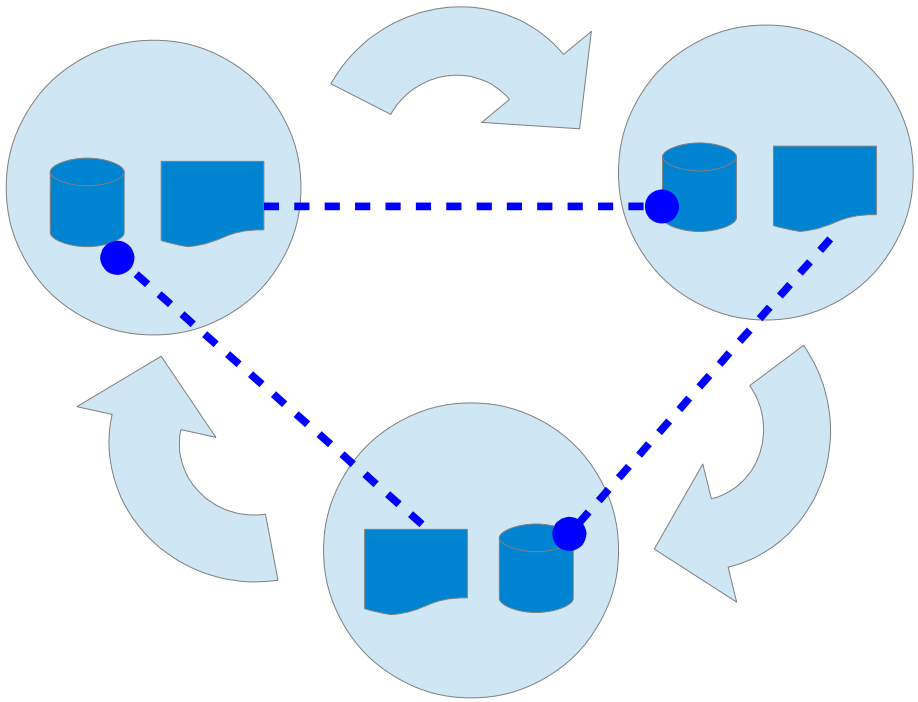
→ No Process Pooling in Erlang

Processes are Cheap

Have *millions* of them.

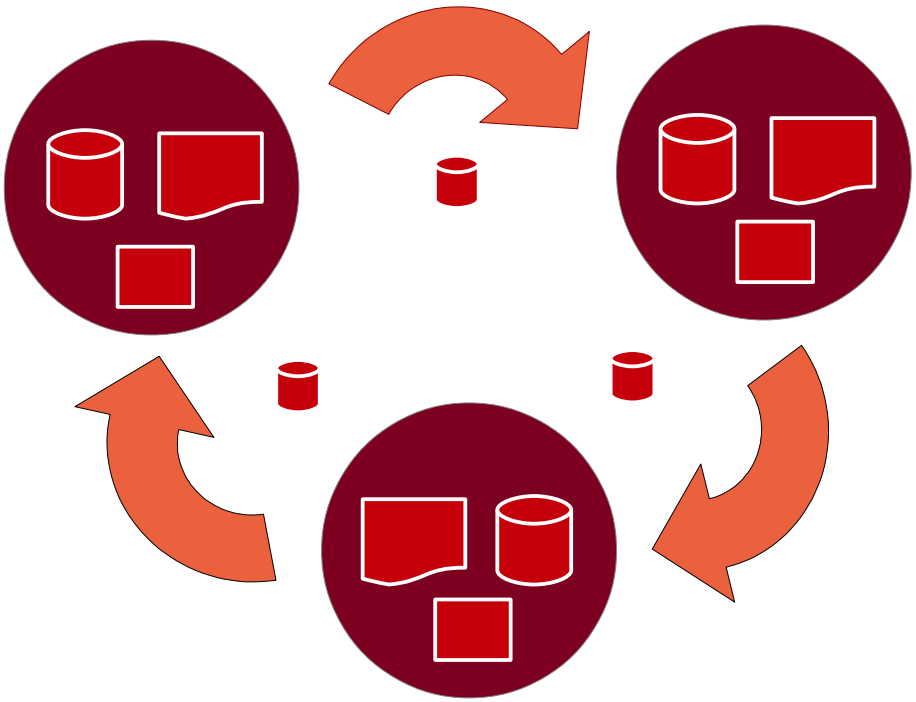
Locks and Deadlocks

Objects **share** State



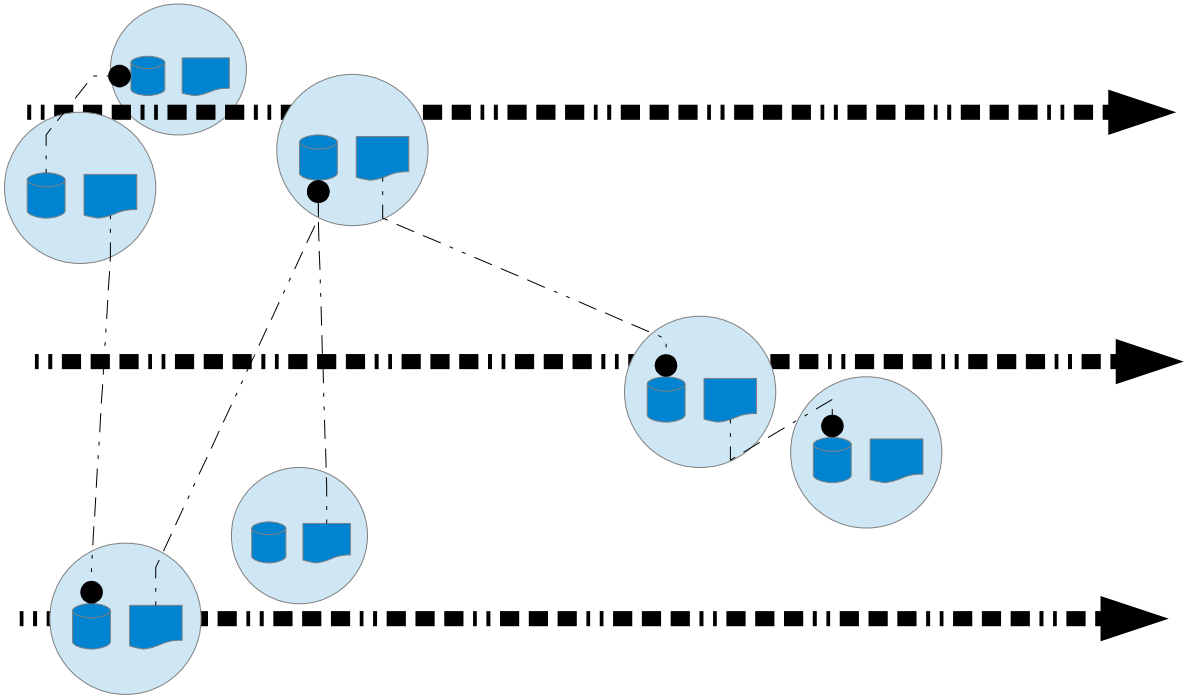
- State can be contested.
- Locks invite **deadlocks**.
- Truly parallel architectures increase **fringe case** race conditions.

Actors message **Copies**



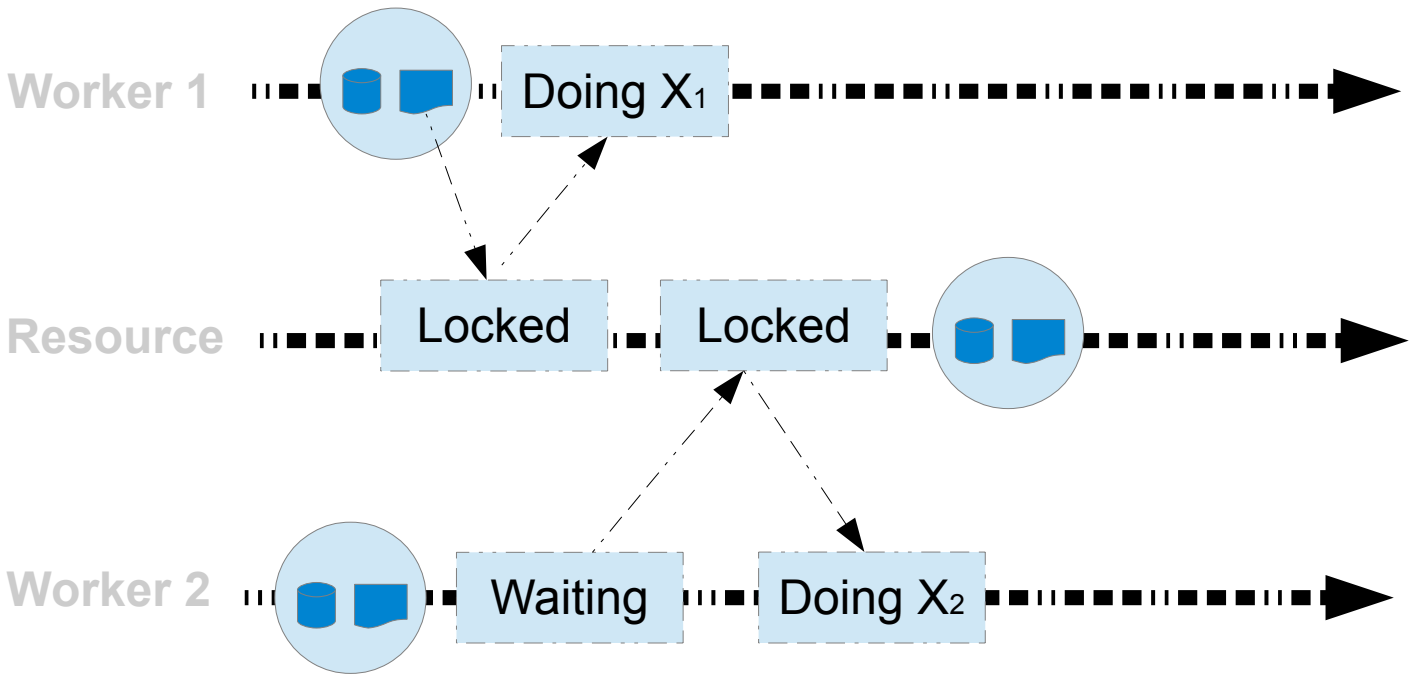
- Messages can only communicate via copies of state.
- Eliminates most race conditions.
- (But references and locks do exist for global lists.)

Objects reference State



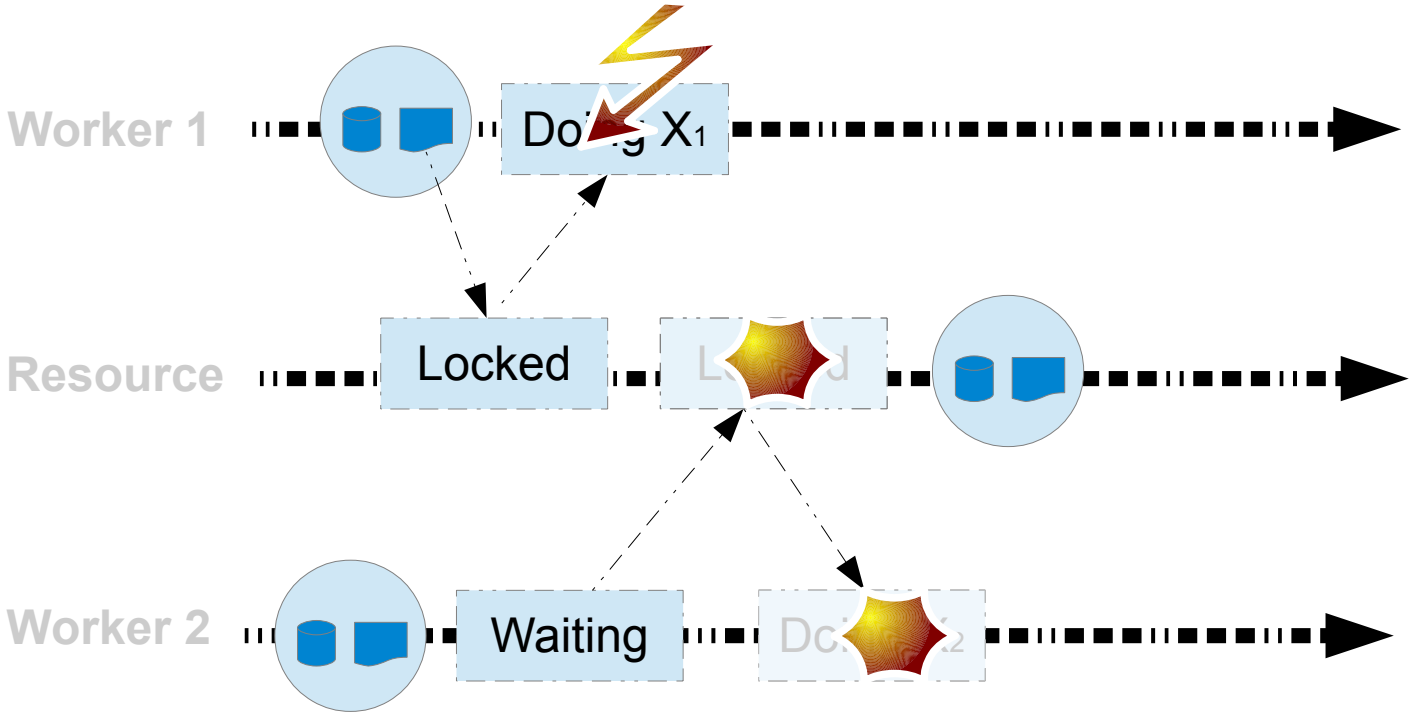
- Multiple objects share threads.
- Objects can be accessed across threads.
- Threads - and objects - share state.

Objects need Locks



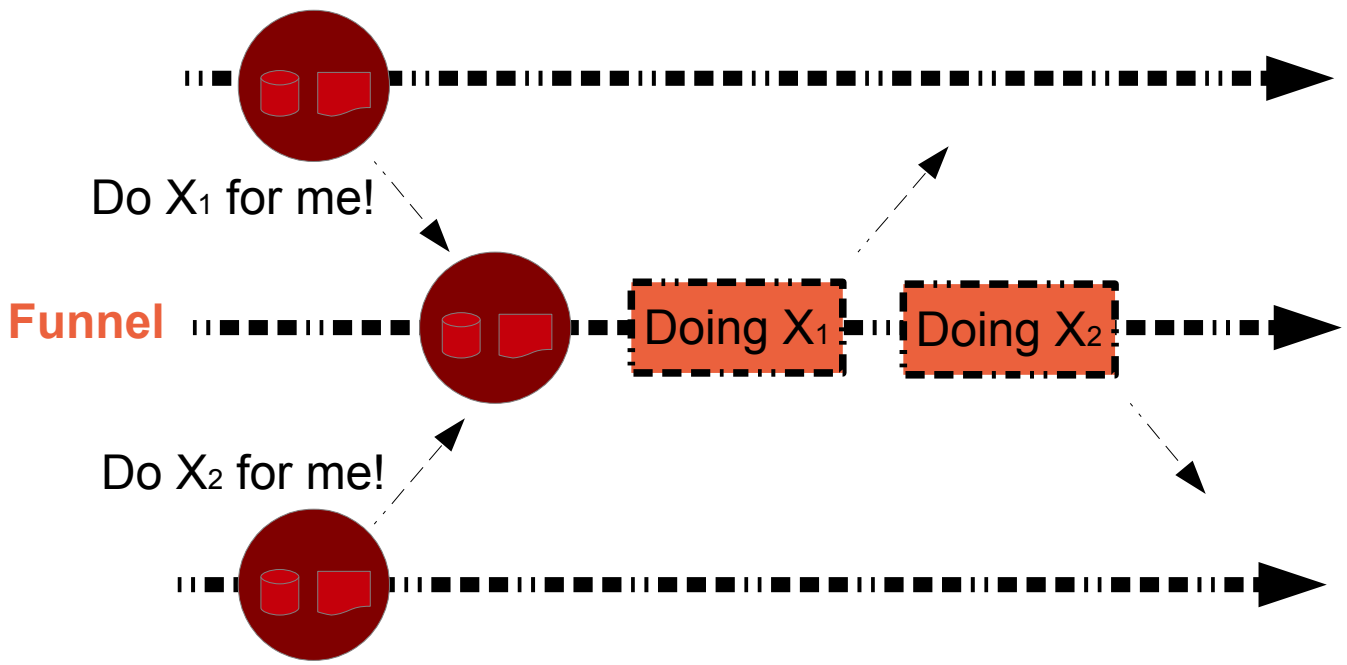
- System design is disrupted by explicit locks.
- Overly cautious locking slows things down.
- Forgotten locks create errors that show under load.

Crashed Locks Stall



- Locks can need cross-thread error handling.
- Stalling and time outs aggravate load.

Processes are Transactional



Obviously:

- One actor is one process and so, cannot “race itself”.
- Mandating a job kind to an actor creates a transactional funnel.
- Only one such job will ever be executing at any one time.

Couldn't I just ...

… be disciplined? And program like this in Java?

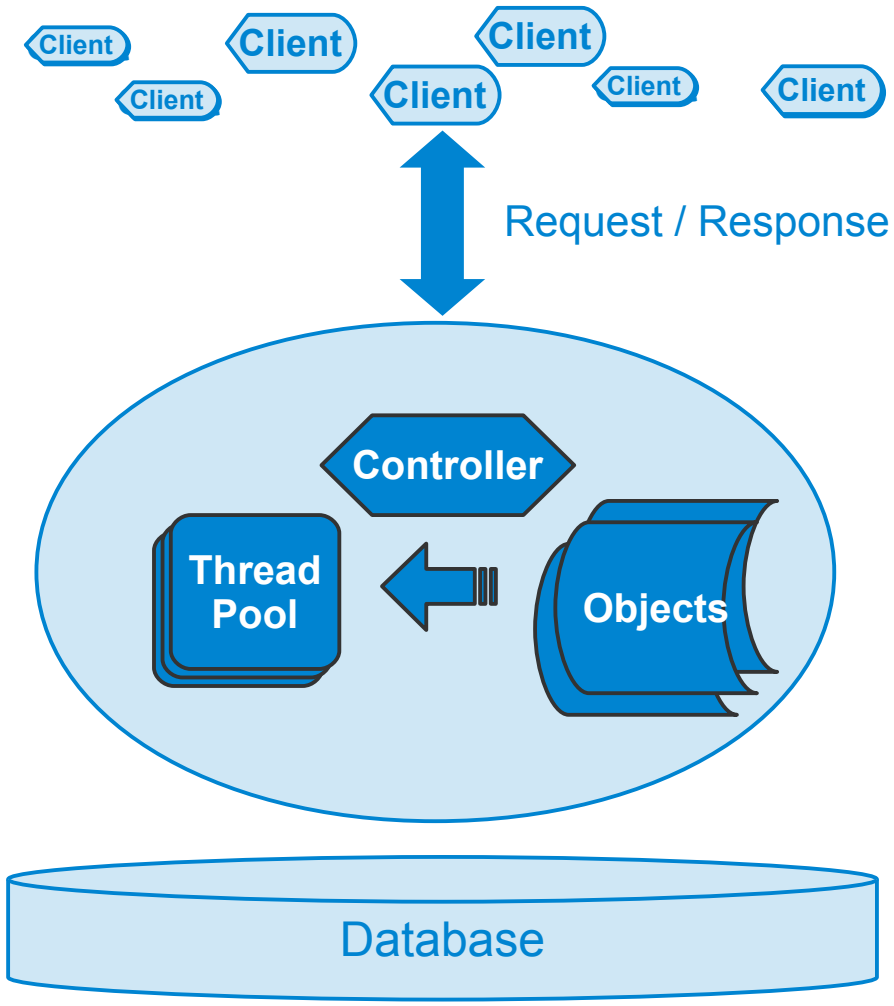
:-) Almost, yes, plus some extensions.
:-) Like, you can avoid null pointers in C by discipline.
:-) And Conwell's Game of Life is Turing Complete.

:-(So realistically, not at all.
:-(Erlang encourages the right way.
:-(Erlang performs better at what it is made for.
:-(Erlang/OTP is made for servers.

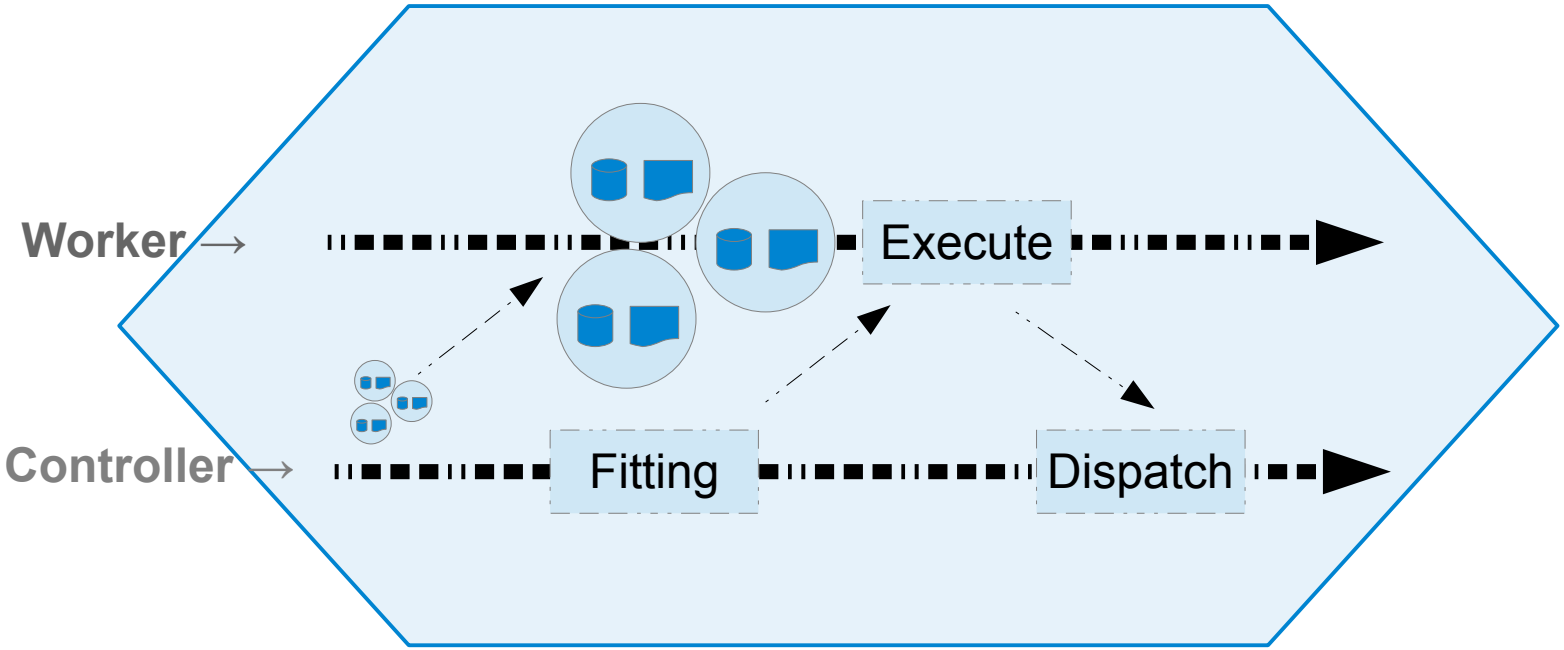
→ **you will be faster learning and using Erlang.**

Server Architecture

OO Server Architecture

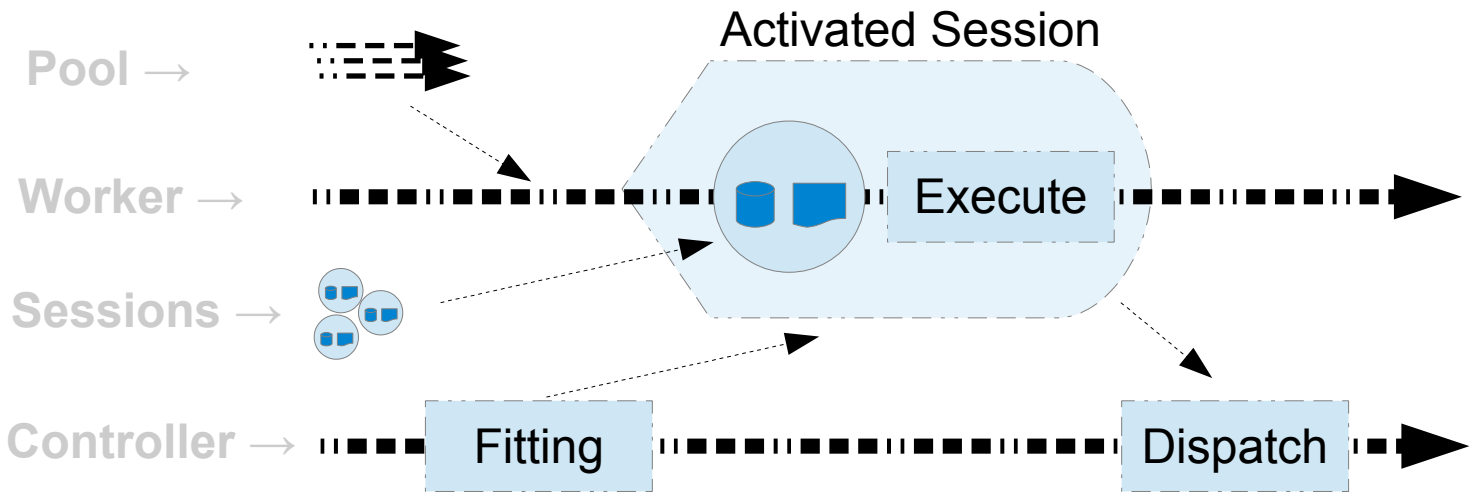


Fitting Recycled Threads



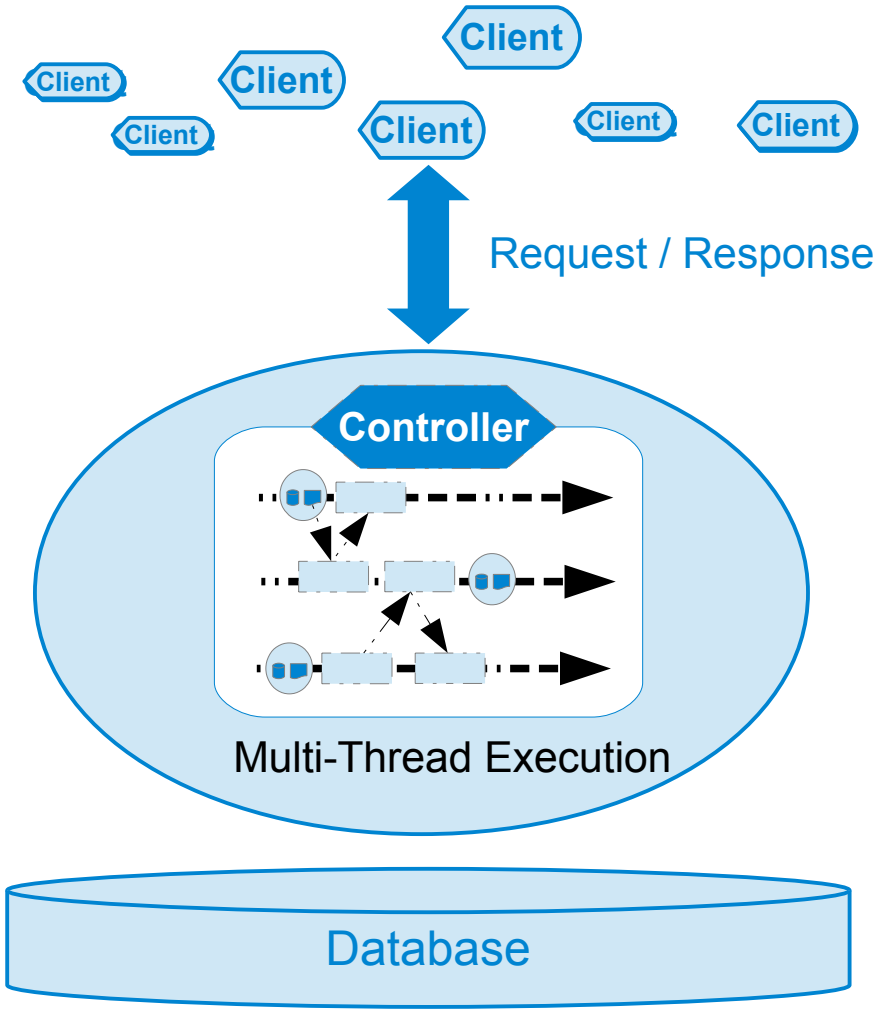
- One thread fitting per single request.
- Pooling owed to heavy footprint of system threads.
- Cracks traumatically under pressure.

Fitting Recycled Threads

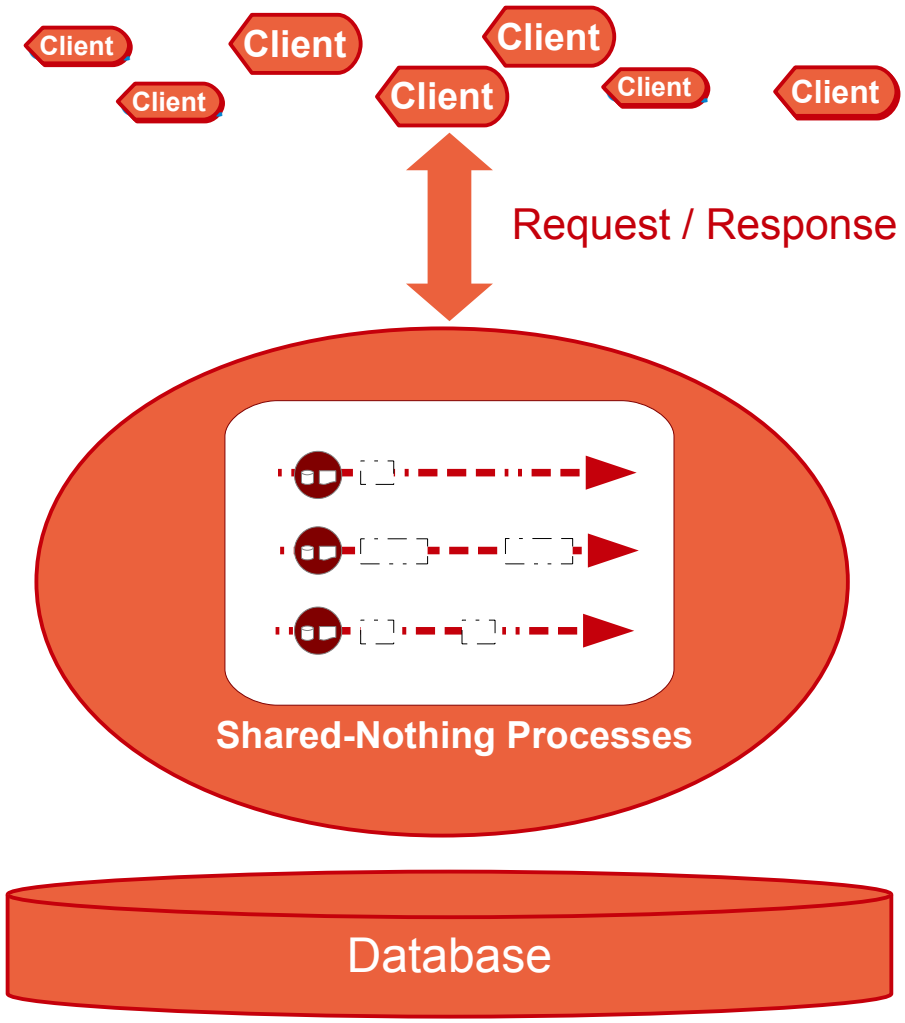


- One thread fitting per single request.
- Pooling owed to heavy footprint of system threads.
- Cracks traumatically under pressure.

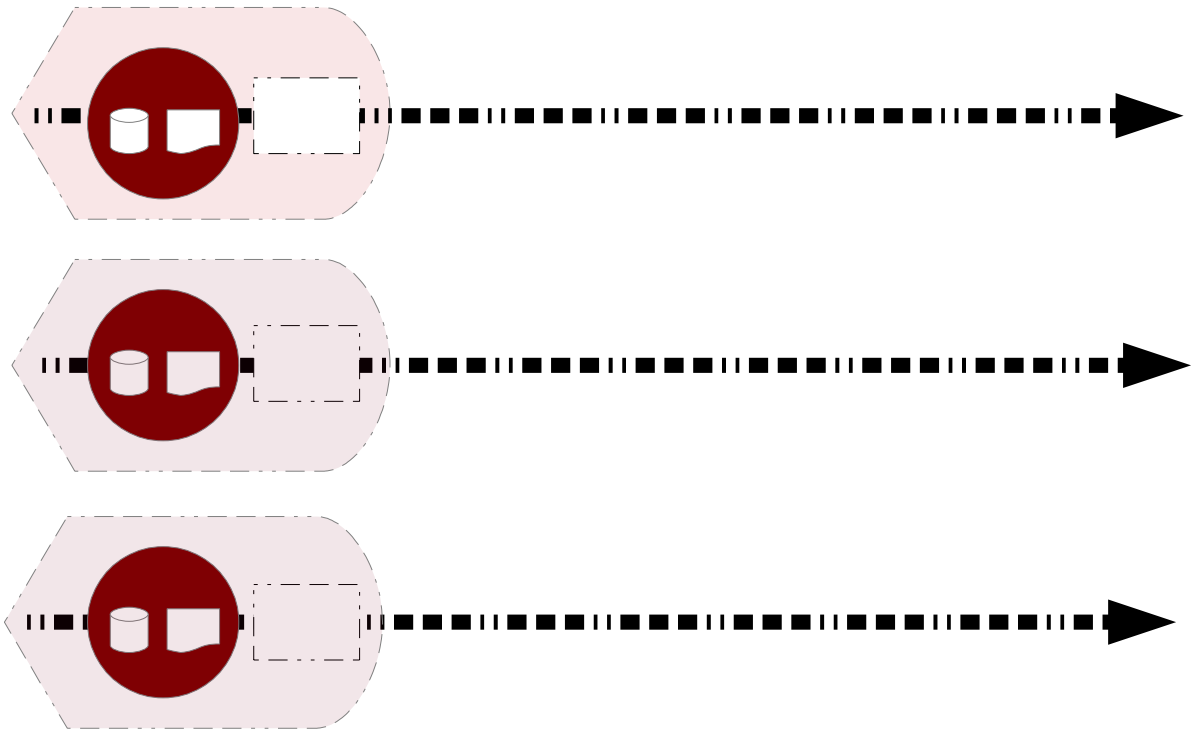
OO Server Architecture



Erlang Server Architecture



Erlang: One Process per Session



- **Natural congruence** of requirements and system.
- Thread management way simpler.
- Enabled by light-weight processes.

Sessions & Processes

Sessions and Processes correlate.

- VM schedules & spreads across Cores
- Asynchronous Messages + Mailboxes
- Shared-Nothing: Messages are Copies
- Individual Memory Management & GC
- Strong Built-In Monitoring Features

Sessions & Processes

One Player Session per Process

+ Immutable State

= Transactional Behavior

Hello CloudDB!

Sessions & Processes

1 Session per Process

+ VM is Process-Aware

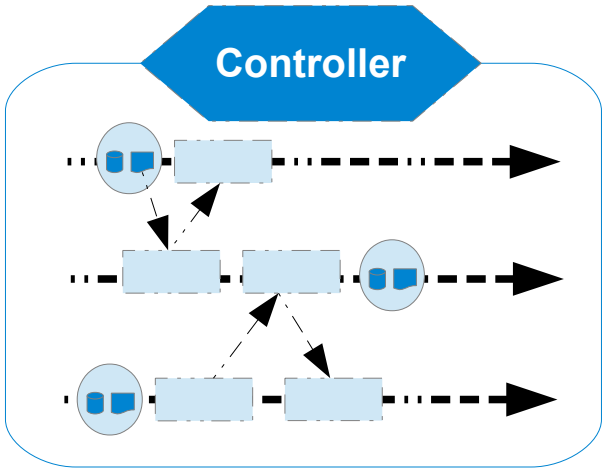
= VM is Session-Aware

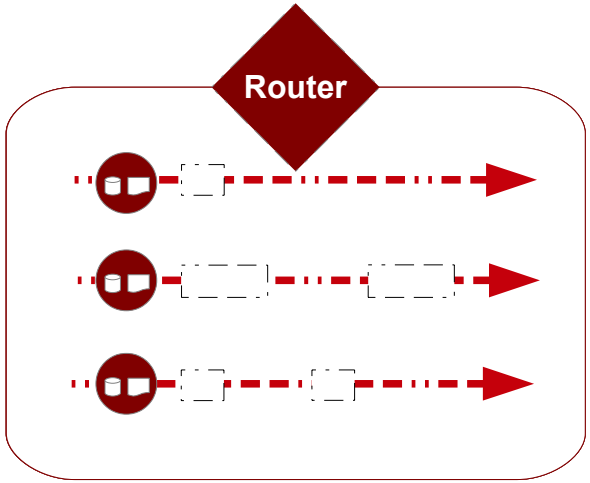
→ Process Stats = **session stats**

→ Per Process GC = **per session *sweep***

Courtesy Wooga: http://www.slideshare.net/wooga/erlang-the-big-switch-in-social-games

OO vs Erlang Architecture





Thinking Parallel

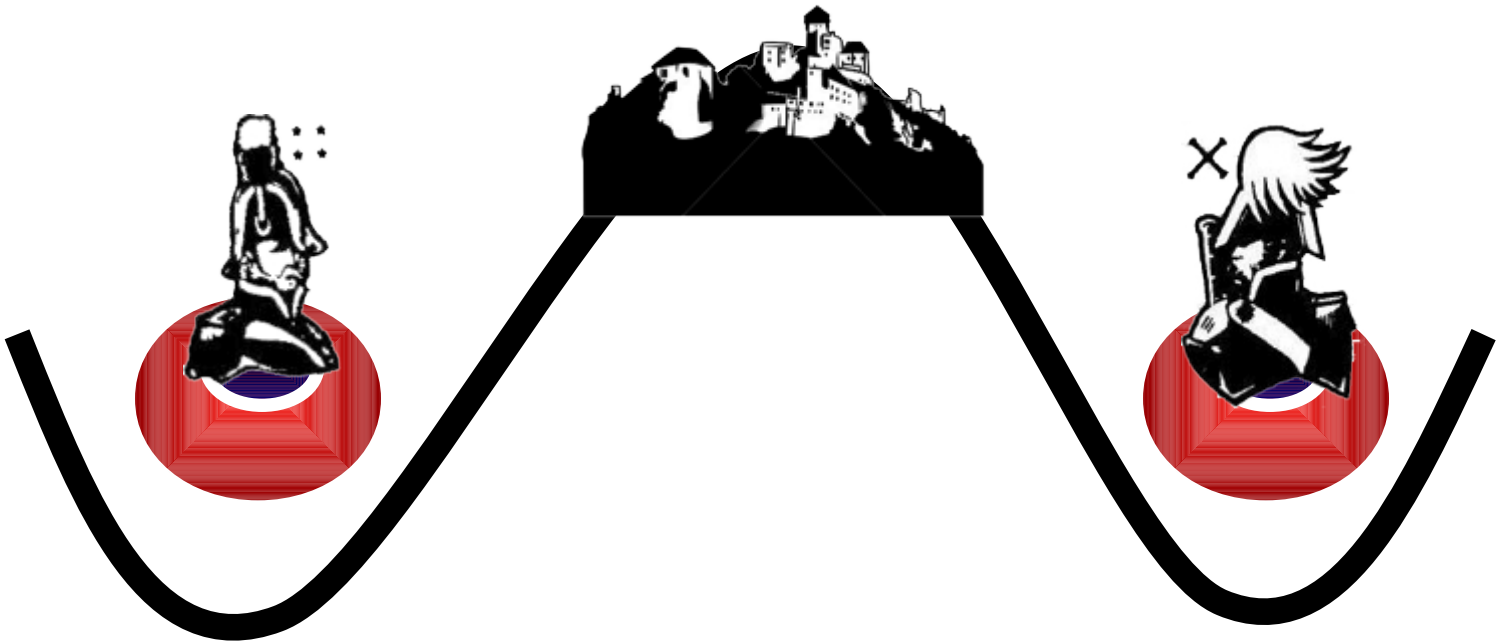
„It's not easy."

Robert Virding

Thinking Parallel

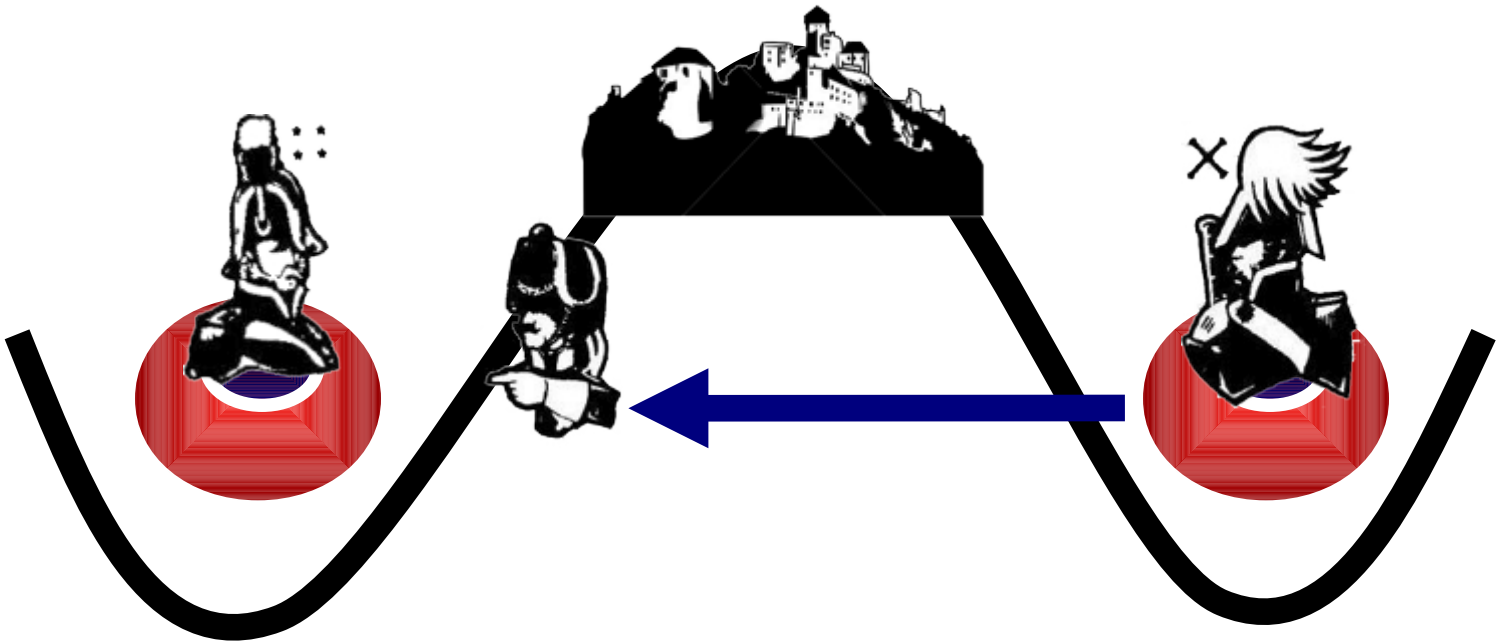
- The Generals' Problem
- Lamport Clocks
- No Guarantees

Generals' Problem



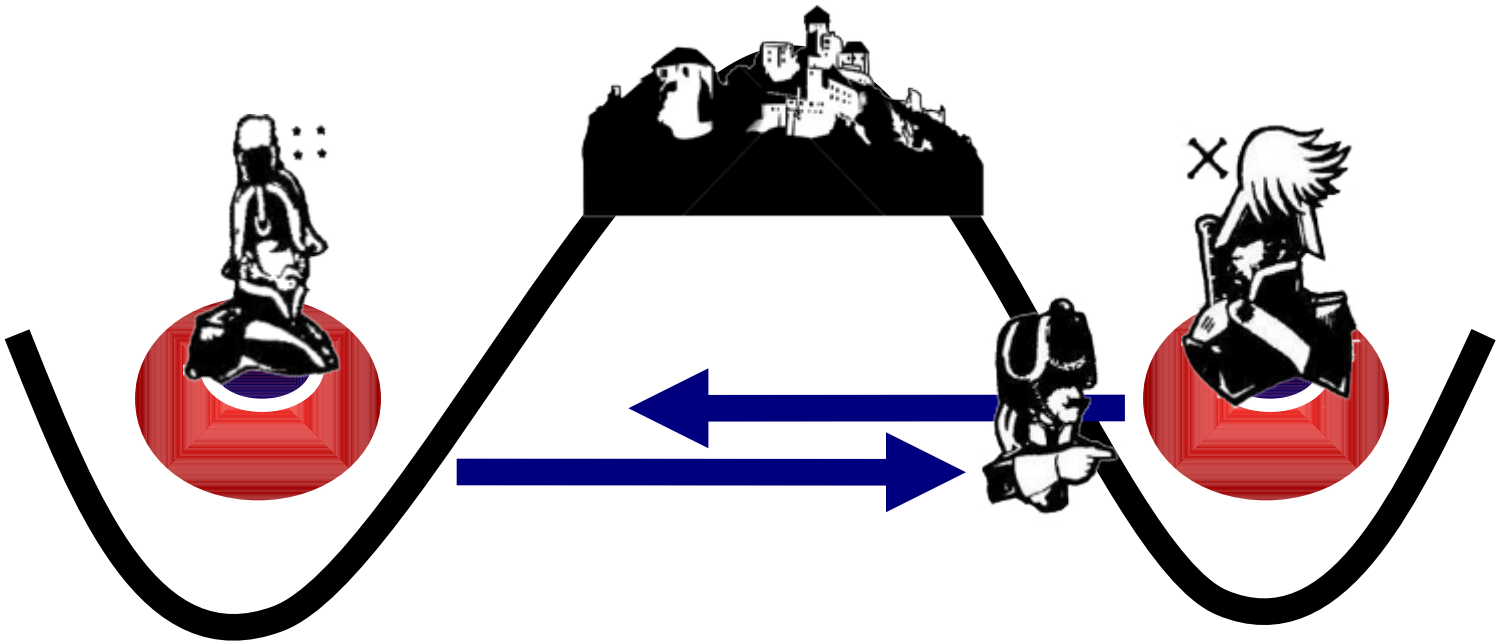
Two generals must agree on a time to attack.

Generals' Problem



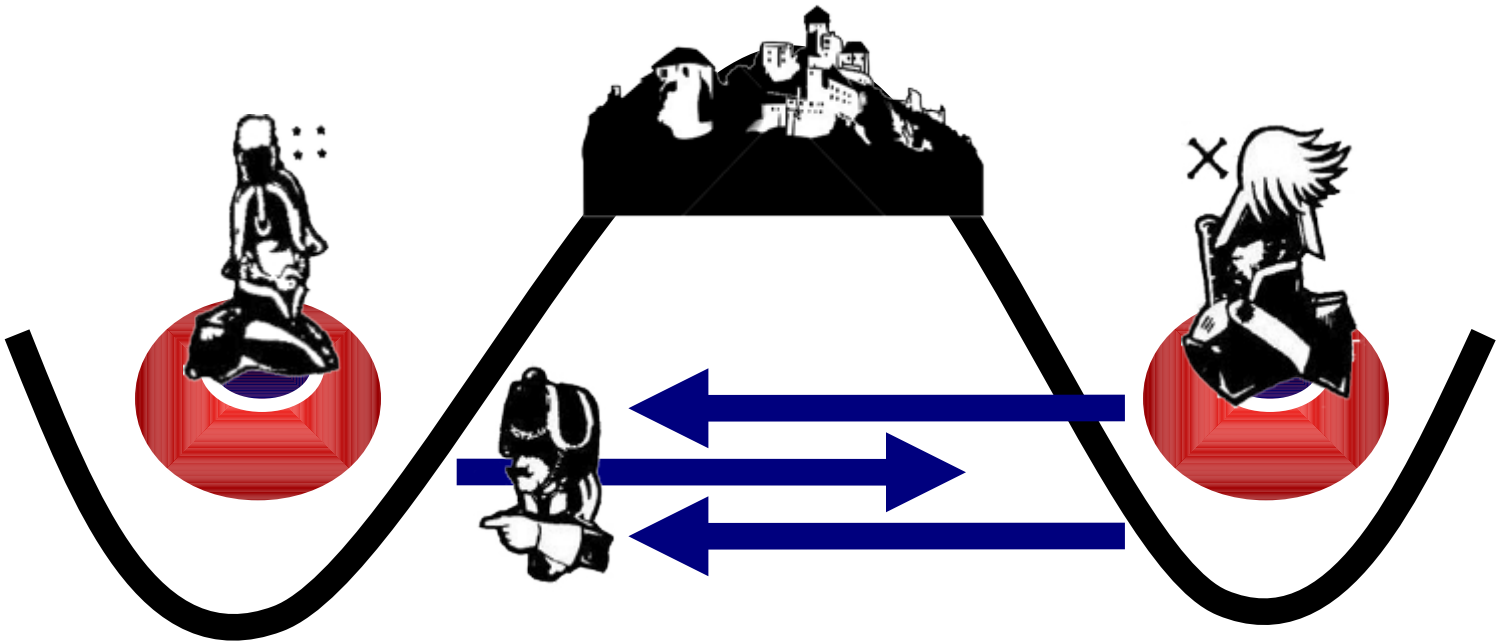
One sends a messenger.

Generals' Problem



The other acknowledges.

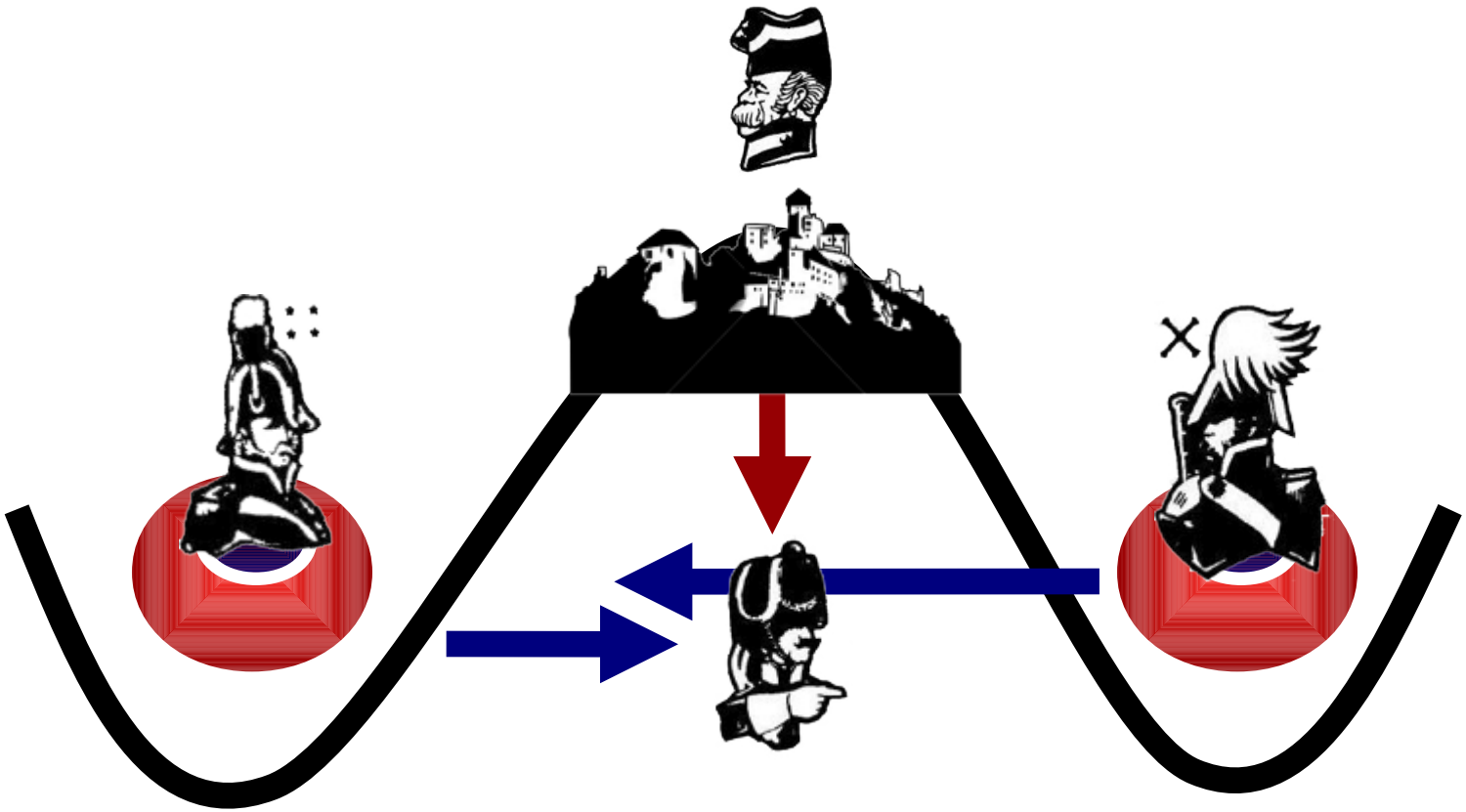
Generals' Problem



ACK the ACK. Etc.

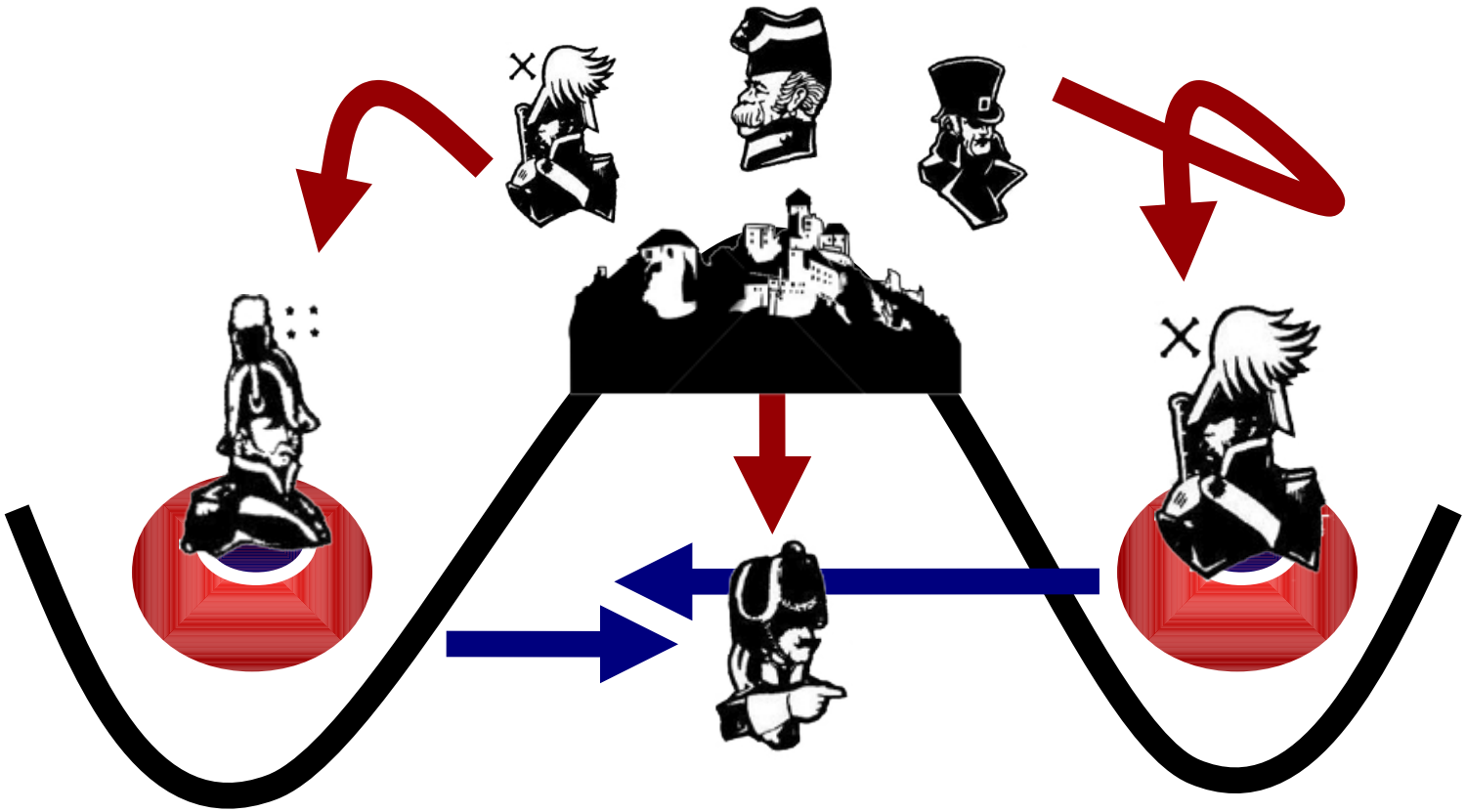
EONBLAST

Generals' Problem



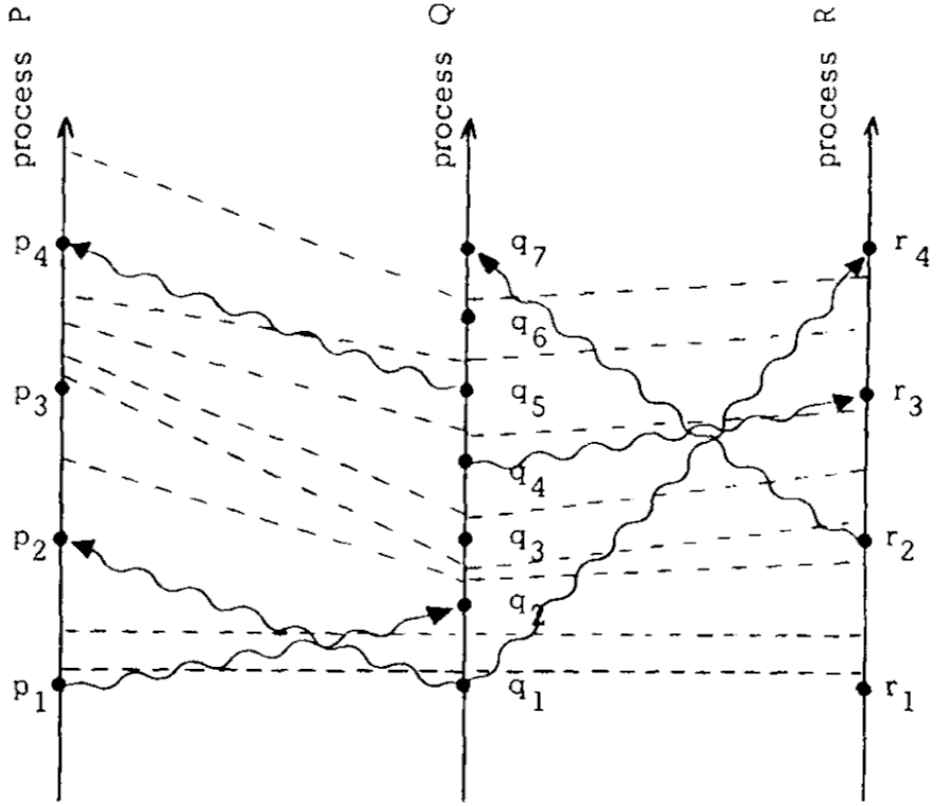
The messenger may get lost.

Byzantine Generals



The generals, actually, too.

Lamport Clocks



Order matters more than time.

Source: Lamport http://research.microsoft.com/users/lamport/pubs/time-clocks.pdf

Thinking Parallel

- Erlang makes it easy
- Some things have no clean solution
- Some things have complicated solutions

Thinking Functional

Thinking Functional

Small Functions

+ Immutable Variables

→ *Don't assign variables: return results!*

Complete State in Plain Sight

→ *Awful for updates in place.*

→ *Awsome for debugging & maintenance.*

Side Effects

Erlang is *not* side-effect free at all.

- Messages between Processes
- Terminal Output
- Logging
- Global Registry
- Database Access

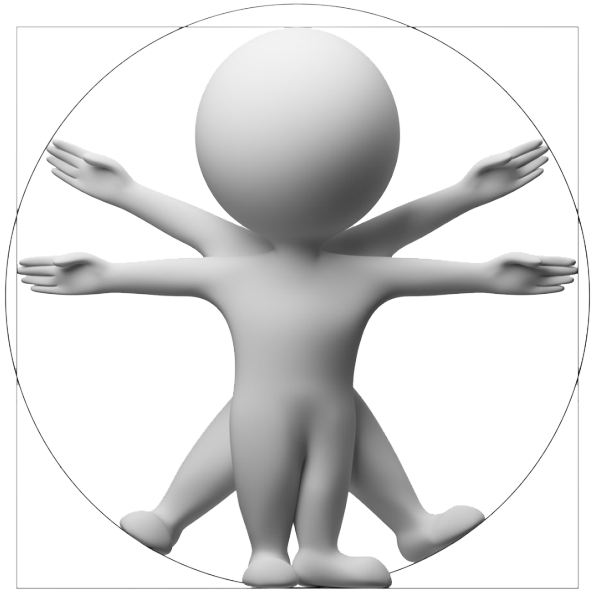
Let It **Crash!**

Let It Crash!

- No Defense Code
- On Error, restart Entire Process
- Built-In Process Supervision & Restart
- Missing Branches, Matches cause Crash

→ Shorter, Cleaner Code
→ Faster Implementation
→ More Robust: handles A*ll E*rrors

What Does That **look** Like?



Hello, World!

```
io:format("Hello, World!").
```

The Optics

- Alien 60ies-Looking Prolog Heir
- Variables start on Capitals
- Very Short Functions
- No Type Declarations
- Statements end on Commas, Semicolons, Dots, Arrows, Nothing
- Pattern Matched Function Heads
- A Church of Short Variables Names exists

Declarative

Fibonacci looks like a Math explanation of it.

```
fib(0) -> 0;
fib(1) -> 1;
fib(N) when N>1 -> fib(N-1) + fib(N-2).
```

Pattern Matching

Function heads matching **0**, **1** or **anything**.

```
fib(0) -> 0;
fib(1) -> 1;
fib(N) when N>1 -> fib(N-1) + fib(N-2).
```

The Syntax

- Small
- Easy
- Stable

- Declarative
- Started out as Prolog
- Inspired by Prolog and ML
- Obvious State, Implicit Thread

Compiling & Executing

```
$ erlc hello.erl
$ erl -s hello
```

Hello, World! Full Module

```
-module(hello).
-export([start/0]).

start() ->
    io:format("Hello, World!~n").
```


EONBLAST

Creating a Process

```
Pid = spawn(mod, func, [A, B, C]).
```

Creating a Process

```
Pid = spawn(mod, func, [A, B, C]).
```

New Process' ID

Code Module

Start Function

Parameters to the function.

EONBLAST

Sending a Message

```
Pid ! Msg.
```

Sending a Message

```
Pid ! Msg.
```

Process ID Message

EONBLAST

Receive a Message

```
receive
    Msg -> Msg
end.
```

Receive a Message

```
receive
    Msg -> Msg
end.
```

Assign Name

Return Value of this block

EONBLAST

Hello, World! The Erlang Way

```
-module(hello).
-export([start/0, loop/0]).

start() ->
  Pid = spawn(hello, loop, []),
  Pid ! hello.

loop() ->
   receive
      hello ->
         io:format("Hello, World!~n"),
         loop()
   end.
```

From Edward Garson's Blog at http://egarson.blogspot.de/2008/03/real-erlang-hello-world.html

Start

```
-module(hello).
-export([start/0, loop/0]).

start() ->
  Pid = spawn(hello, loop, []),
  Pid ! hello.

loop() ->
   receive
      hello ->
         io:format("Hello, World!~n"),
         loop()
   end.
```

From Edward Garson's Blog at http://egarson.blogspot.de/2008/03/real-erlang-hello-world.html

EONBLAST

Output

```
-module(hello).
-export([start/0, loop/0]).

start() ->
  Pid = spawn(hello, loop, []),
  Pid ! hello.

loop() ->
   receive
      hello ->
         io:format("Hello, World!~n"),
         loop()
   end.
```

From Edward Garson's Blog at http://egarson.blogspot.de/2008/03/real-erlang-hello-world.html

Process Spawning

```
-module(hello).
-export([start/0, loop/0]).

start() ->
  Pid = spawn(hello, loop, []),
  Pid ! hello.
```

```
loop() ->
   receive
      hello ->
         io:format("Hello, World!~n"),
         loop()
   end.
```

New Process

From Edward Garson's Blog at http://egarson.blogspot.de/2008/03/real-erlang-hello-world.html

EONBLAST

Blocking Receive

```
-module(hello).
-export([start/0, loop/0]).

start() ->
  Pid = spawn(hello, loop, []),
  Pid ! hello.
```

```
loop() ->
   receive
      hello ->
         io:format("Hello, World!~n"),
         loop()
   end.
```

From Edward Garson's Blog at http://egarson.blogspot.de/2008/03/real-erlang-hello-world.html

EONBLAST

Message Passing

```
-module(hello).
-export([start/0, loop/0]).

start() ->
  Pid = spawn(hello, loop, []),
  Pid ! hello.

loop() ->
   receive
      hello ->
         io:format("Hello, World!~n"),
         loop()
   end.
```

From Edward Garson's Blog at http://egarson.blogspot.de/2008/03/real-erlang-hello-world.html

EONBLAST

Pattern Matching

```
-module(hello).
-export([start/0, loop/0]).

start() ->
  Pid = spawn(hello, loop, []),
  Pid ! hello.
```

```
loop() ->
   receive
      hello ->
         io:format("Hello, World!~n"),
         loop()
   end.
```

From Edward Garson's Blog at http://egarson.blogspot.de/2008/03/real-erlang-hello-world.html

EONBLAST

Atoms

```
-module(hello).
-export([start/0, loop/0]).

start() ->
  Pid = spawn(hello, loop, []),
  Pid ! hello.
```

```
loop() ->
   receive
      hello ->
         io:format("Hello, World!~n"),
         loop()
   end.
```

From Edward Garson's Blog at http://egarson.blogspot.de/2008/03/real-erlang-hello-world.html

EONBLAST

Tail Recursion

```
-module(hello).
-export([start/0, loop/0]).

start() ->
  Pid = spawn(hello, loop, []),
  Pid ! hello.

loop() ->
   receive
      hello ->
         io:format("Hello, World!~n"),
         loop()
   end.
```

From Edward Garson's Blog at http://egarson.blogspot.de/2008/03/real-erlang-hello-world.html

EONBLAST

Dots

```
-module(hello).
-export([start/0, loop/0]).

start() ->
  Pid = spawn(hello, loop, []),
  Pid ! hello.

loop() ->
   receive
      hello ->
         io:format("Hello, World!~n"),
         loop()
   end.
```

From Edward Garson's Blog at http://egarson.blogspot.de/2008/03/real-erlang-hello-world.html

EONBLAST

Commas

```
-module(hello).
-export([start/0, loop/0]).

start() ->
  Pid = spawn(hello, loop, []),
  Pid ! hello.

loop() ->
   receive
      hello ->
         io:format("Hello, World!~n"),
         loop()
   end.
```

From Edward Garson's Blog at http://egarson.blogspot.de/2008/03/real-erlang-hello-world.html

End

```
-module(hello).
-export([start/0, loop/0]).

start() ->
  Pid = spawn(hello, loop, []),
  Pid ! hello.

loop() ->
   receive
      hello ->
         io:format("Hello, World!~n"),
         loop()
   end.
```

From Edward Garson's Blog at http://egarson.blogspot.de/2008/03/real-erlang-hello-world.html

EONBLAST

Arrows

```
-module(hello).
-export([start/0, loop/0]).

start() ->
  Pid = spawn(hello, loop, []),
  Pid ! hello.

loop() ->
   receive
      hello ->
         io:format("Hello, World!~n"),
         loop()
   end.
```

From Edward Garson's Blog at http://egarson.blogspot.de/2008/03/real-erlang-hello-world.html

EONBLAST

Arrows

```
-module(hello).
-export([start/0, loop/0]).

start() ->
  Pid = spawn(hello, loop, []),
  Pid ! hello.

loop() ->
   receive
      hello ->
         io:format("Hello, World!~n"),
         loop()
   end.
```

From Edward Garson's Blog at http://egarson.blogspot.de/2008/03/real-erlang-hello-world.html

EONBLAST

Nothing

```
-module(hello).
-export([start/0, loop/0]).

start() ->
  Pid = spawn(hello, loop, []),
  Pid ! hello.

loop() ->
   receive
      hello ->
         io:format("Hello, World!~n"),
         loop()
   end.
```

From Edward Garson's Blog at http://egarson.blogspot.de/2008/03/real-erlang-hello-world.html

EONBLAST

Modules mix Processes

Calling Processes

```
-module(hello).
-export([start/0, loop/0]).

start() ->
  Pid = spawn(hello, loop, []),
  Pid ! hello.
```

Module's Own Process

```
loop() ->
   receive
      hello ->
         io:format("Hello, World!~n"),
         loop()
   end.
```

From Edward Garson's Blog at http://egarson.blogspot.de/2008/03/real-erlang-hello-world.html

EONBLAST

Modules mix Processes

```
-module(hello).
-export([start/0, say/1, loop/0]).

start() ->
  spawn(hello, loop, []).

say(Pid) ->
  Pid ! hello.

loop() ->
   receive
      hello ->
         io:format("Hello, World!~n"),
         loop()
   end.
```

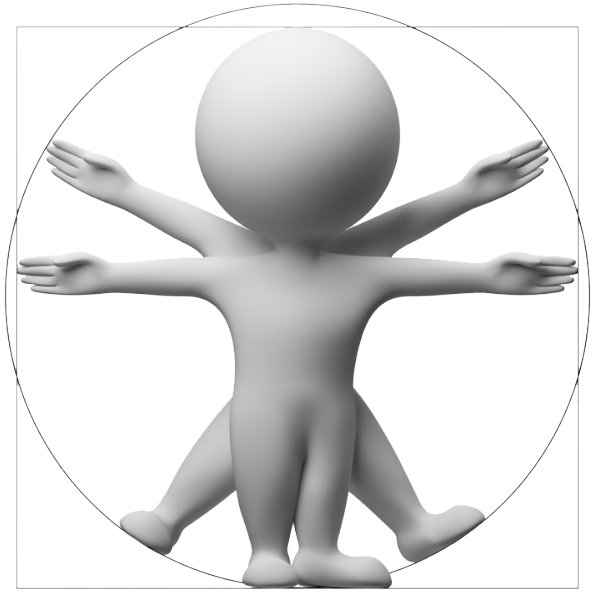
Calling Processes

Module's Own Process

From Edward Garson's Blog at http://egarson.blogspot.de/2008/03/real-erlang-hello-world.html

EONBLAST

Immutable Variables



Immutable Variables

Can't assign a second time:

A = A + 1.

A = 1, A = 2.

Immutable Variables

It has to be:

B = A + 1.

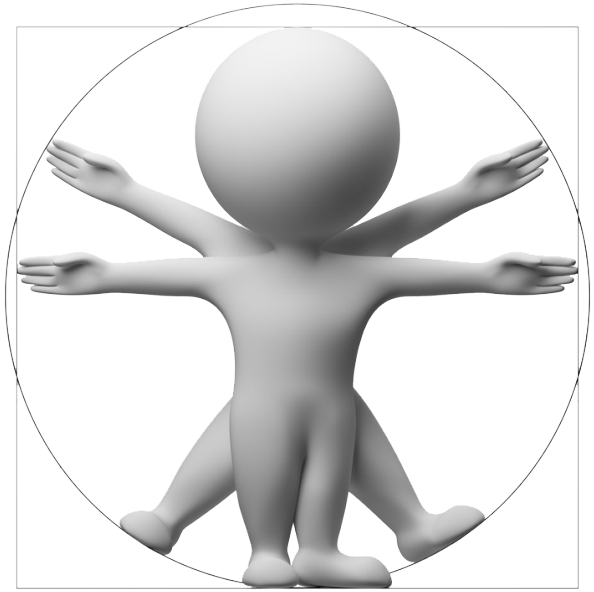
A = 1, B = 2.

Immutable Variables

- Prevent Coding Errors
- Provide Transactional Semantic
- Allow for Pattern Matching Syntax
- Can be a Nuisance

```
S1 = dosomething(S),
S2 = dosomemore(S1)
...
```

Pattern Matching



Pattern Matching

This can mean **two** things:

```
A = func().
```

The meaning depends on wheter **A** is already assigned.

Pattern Matching

The common, mixed case:

```
{ok, A} = func().
```

ok is an assertion **AND**
A is being assigned.

Pattern Matching

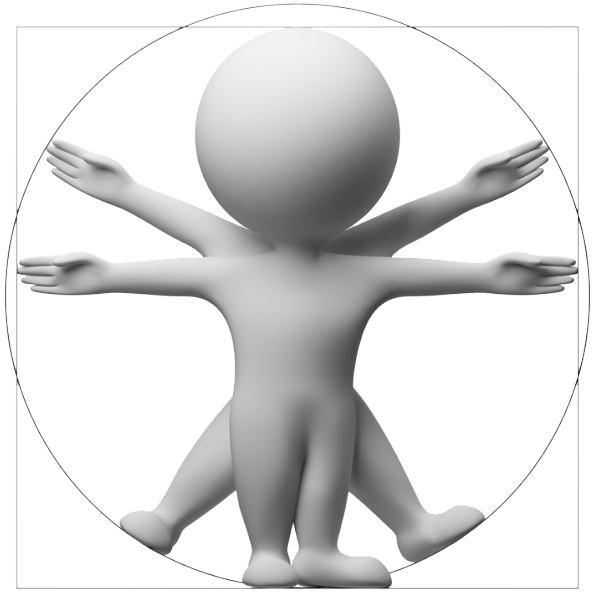
The common, mixed case:

```
{ok, A} = func().
```

„This makes it hard to remodel Erlang syntax into a more C-like syntax.“

Robert Virding

Erlang Compared



Erlang vs. Stackless Python

- **Truly parallel VM**
- Stackless has a GIL
  thus in reality works sequential
  only its paradigm is parallel
- Pattern Matching
- Immutable Variables

Erlang vs. C

- More productive
- More concise
- More reliable
- Much slower for Number Crunching
- Microprocesses
- Pattern Matching
- Immutable Variables

Erlang vs. C++

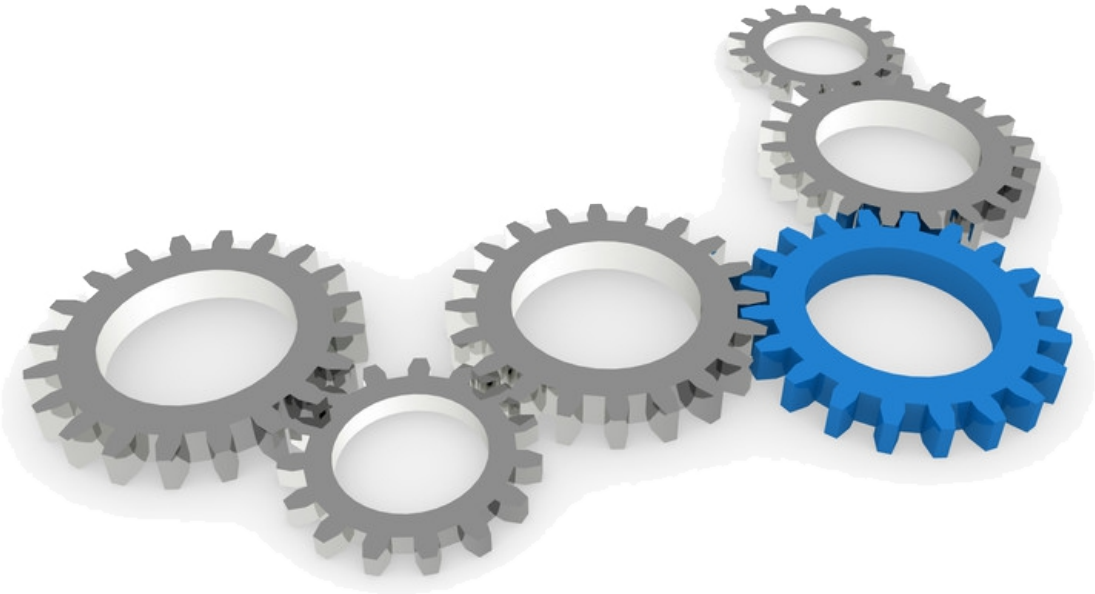
- Virtual Machine
- Actors Model
- Less Magic
- More Safety
- Much Slower for Number Crunching
- Microprocesses
- Pattern Matching
- Immutable Variables

Erlang vs. Lua

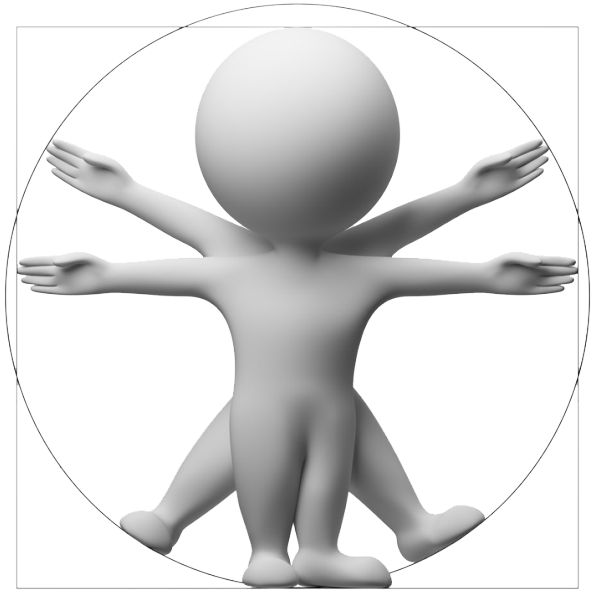
- **Made to Scale**
- Single Paradigm
- Less Magic
- Much Bigger Footprint
- Microprocesses
- Pattern Matching
- Immutable Variables

Great Matches

- C
- Redis
- MySQL
- VoltDB
- AWS S3
- EC2
- Ubuntu



Productivity



Productivity & Maintainability

"Erlang systems have

4 – 10 x less code

than C / C++ / Java systems"

Ulf Wiger

Productivity & Maintainability

Shorter programs:

- Faster to develop
- Fewer errors
- Easier to maintain

Erlang LOCs show the same error frequency as C / C++ / Java code.

Scientific Proof

The Motorola Study of 2002 – 2006

- Motorala UK Labs
- Heriot-Watt University
- EPSRC UK Govt Project

http://www.slideshare.net/JanHenryNystrom/productivity-gains-in-erlang

Scientific Proof

"High Level Techniques for Distributed Telecoms Software"

Looking at

- Robustness
- Configurability
- Productivity
- Maintainability

Scientific Proof

"Erlang shows …

- 2x higher throughput
- 3x better latency
- 3 - 7x shorter code

… than the equivalent C++ implementation."

Scientific Proof

Reasons

- Lightweight process management
- Code only the successful case – saves 27%
- Automatic memory management – saves 11%
- High-level communications – saves 23%
- Telecom design pattern libraries (suit games)

Scientific Proof

Overload & Hardware Failure

- C++ "fails catastrophically"
- Erlang
  - Never completely fails
  - Recovers automatically after load drops

Challenges

The Warts

- Untidy Standard Libs
- Egregious String Handling
- „Records Suck“
- Cryptic Error Messages
- Lack of Advanced Tutorials & Books
- No-One Seems To Know All of OTP
- Hard To Find Developers

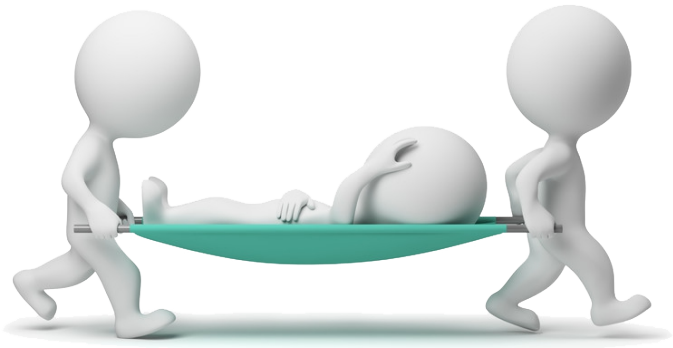
Strings

- Are* Lists. Or Binaries.
- Slow*
- Clumsy*
- Error prone*
- Cure announced for next release (R16)

Erlang was not built for text processing.

*according to Joe Armstrong these are myths. „Erlang has no strings. The only sane way to handle UTF-8 is lists of integers."

Records

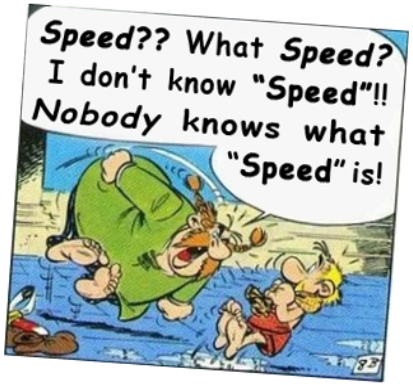


- "Records Suck"
- Verbose
- Error Prone
- Pure syntactic sugar over tupels.
- Cure announced for next release (R16)

"Records were added as a hack."

Raw Number Crunching Speed

- Erlang is fast
- Except at Shoot Outs
- Benchmark your **real** problem
- Use C NIFs to outsource crunching

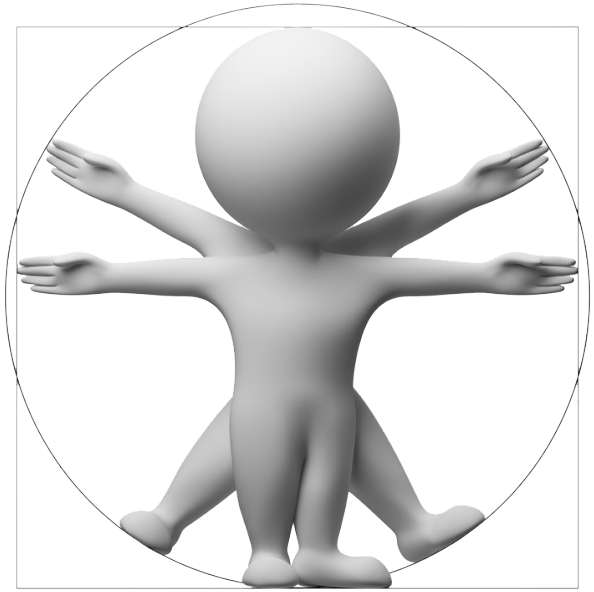


Perceived reaction after asking about benchmarks on the Erlang mailing list.

Hiring

- Can Be Difficult
- Roll Your Own Programmers
  - Good Programmers Are Interested
  - High Productivity Can Be Reached Fast
  - Excellent Workshops can be booked

Getting **Started!**



Learning Erlang

- Scour Post-Mortems
- Download and Install from *erlang.org*
- Fred's site *Learn You Some Erlang!*
- Joe's book *Programming Erlang*
- IRC *#erlounge*
- Erlang Mailing List
- Local *Erlounge* Meetings
- Erlang Factories & User Conferences

Expected Timeframe

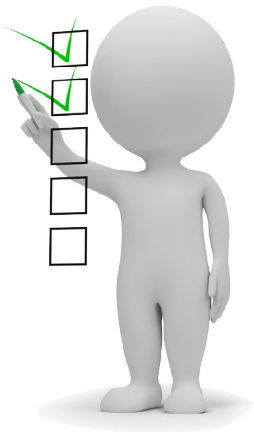
Ballparks

- Language – 2 Weeks
- OTP – 3 Months
- First Product – ½ Year
- Thinking Erlang – 2 Years

Business View

Makes Sense For

- New Projects
- Rewrites



Starting Out

- Find a Senior Erlang Developer
- Or Hire Erlang Solutions
- You Will Train Your Own Developers

Pitching It In-House

A Waste of Time
Joe Armstrong

Just tell your boss that Erlang is used for banking applications.
Mike Williams

- Most Often a Top-Down Thing
- Demonstrate the Productivity: prototype a demo solution to a real problem.
- Cite Facebook, Zynga, Blizzard, Wooga

EONBLAST

References

Web
http://www.erlang.org/
http://learnyousomeerlang.com/
http://www.erlang.org/static/getting_started_quickly.html

List
http://erlang.org/mailman/listinfo/erlang-questions
http://groups.google.com/group/erlang-programming

Books
http://pragprog.com/book/jaerlang/programming-erlang
http://shop.oreilly.com/product/9780596518189.do

References
http://erldocs.com/
http://www.erlang.org/doc/

Post Mortems
http://www.facebook.com/note.php?note_id=14218138919
http://www.slideshare.net/wooga/erlang-the-big-switch-in-social-games

Your Talk Evaluation

Please check your Email **now** and give your **evaluation** of this talk to the GDC.

Any questions, feedback now or later, please email me at hd*eonblast.com

EONBLAST

Questions

- Email: hdiedrich*eonblast.com
- Twitter: @hdiedrich
- IRC: #erlounge
- List: erlang-questions@erlang.org