Systems that never stop (and Erlang)

Joe Armstrong

How can we get

10 nines reliability?

SIX LAWS

ONE

ISOLATION

ISOLATION

- 10 nines = 99.99999999% availability
- $P(fail) = 10^{-10}$
- If $P(fail \mid one\ computer) = 10^{-3}$ then
  $P(fail \mid four\ computers) = 10^{-12}$
- Fixed

TWO

CONCURRENCY

Concurrency

- World is concurrent
- Need at least TWO computers to make a non-stop sytem
- TWO computer **is** concurrent **and distributed**

“My first message is that concurrency
is best regarded as a program structuring principle”

Structured concurrent programming

– Tony Hoare

Redmond, July 2001

THREE

MUST
DETECT FAILURES

Failure detection

- If you can't detect a failure you can't fix it
- Must work across machine boundaries
  the entire machine might fail
- Implies distributed error handling,
  no shared state,
  asynchronous messaging

FOUR

FAULT IDENTIFICATION

Failure Identification

- Fault detection is not enough - you must no **why** the failure occurred
- Implies that you have sufficient information for post hock debugging

FIVE

LIVE
CODE
UPGRADE

Live code upgrade

- Must upgrade software while it is running
- Want zero down time

SIX

STABLE STORAGE

Stable storage

- Must store stuff forever
- No backup necessary - storage just works
- Implies multiple copies, distribution, ...
- Must keep crash reports

HISTORY

Those who cannot learn from history are doomed to repeat it.

George Santayana

GRAY

As with hardware, the key to software fault-tolerance is to hierarchically decompose large systems into modules, each module being a unit of service and a unit of failure. A failure of a module does not propagate beyond the module.

...

The process achieves fault containment by sharing no state with other processes; its only contact with other processes is via messages carried by a kernel message system

- Jim Gray
- Why do computers stop and what can be done about it
- Technical Report, 85.7 - Tandem Computers,1985

SCHNEIDER

Halt on failure in the event of an error a processor should halt instead of performing a possibly erroneous operation.

Failure status property when a processor fails, other processors in the system must be informed. The reason for failure must be communicated.

Stable Storage Property The storage of a processor should be partitioned into stable storage (which survives a processor crash) and volatile storage which is lost if a processor crashes.

Schneider
ACM Computing Surveys 22(4):229-319, 1990

GRAY

- Fault containment through fail-fast software modules.
- Process-pairs to tolerant hardware and transient software faults.
- Transaction mechanisms to provide data and message integrity.
- Transaction mechanisms combined with process-pairs to ease exception handling and tolerate software fault
- Software modularity through processes and messages.

KAY

Folks --

Just a gentle reminder that I took some pains at the last OOPSLA to try to remind everyone that Smalltalk is not only NOT its syntax or the class library, it is not even about classes. I'm sorry that I long ago coined the term "objects" for this topic because it gets many people to focus on the lesser idea.

The big idea is "messaging" -- that is what the kernal of Smalltalk/Squeak is all about (and it's something that was never quite completed in our Xerox PARC phase)....

http://lists.squeakfoundation.org/pipermail/squeak-dev/1998-October/017019.html

GRAY

Software modularity through processes and messages. As with hardware, the key to software fault-tolerance is to hierarchically decompose large systems into modules, each module being a unit of service and a unit of failure. A failure of a module does not propagate beyond the module.

Fail Fast

The process approach to fault isolation advocates that the process software be fail-fast, it should either function correctly or it should detect the fault, signal failure and stop operating.

Processes are made fail-fast by defensive programming. They check all their inputs, intermediate results and data structures as a matter of course. If any error is detected, they signal a failure and stop. In the terminology of [Cristian], fail-fast software has small fault detection latency.

Gray
Why ...

Fail Early

A fault in a software system can cause one or more errors. The latency time which is the interval between the existence of the fault and the occurrence of the error can be very high, which complicates the backwards analysis of an error ...

For an effective error handling we must detect errors and failures as early as possible

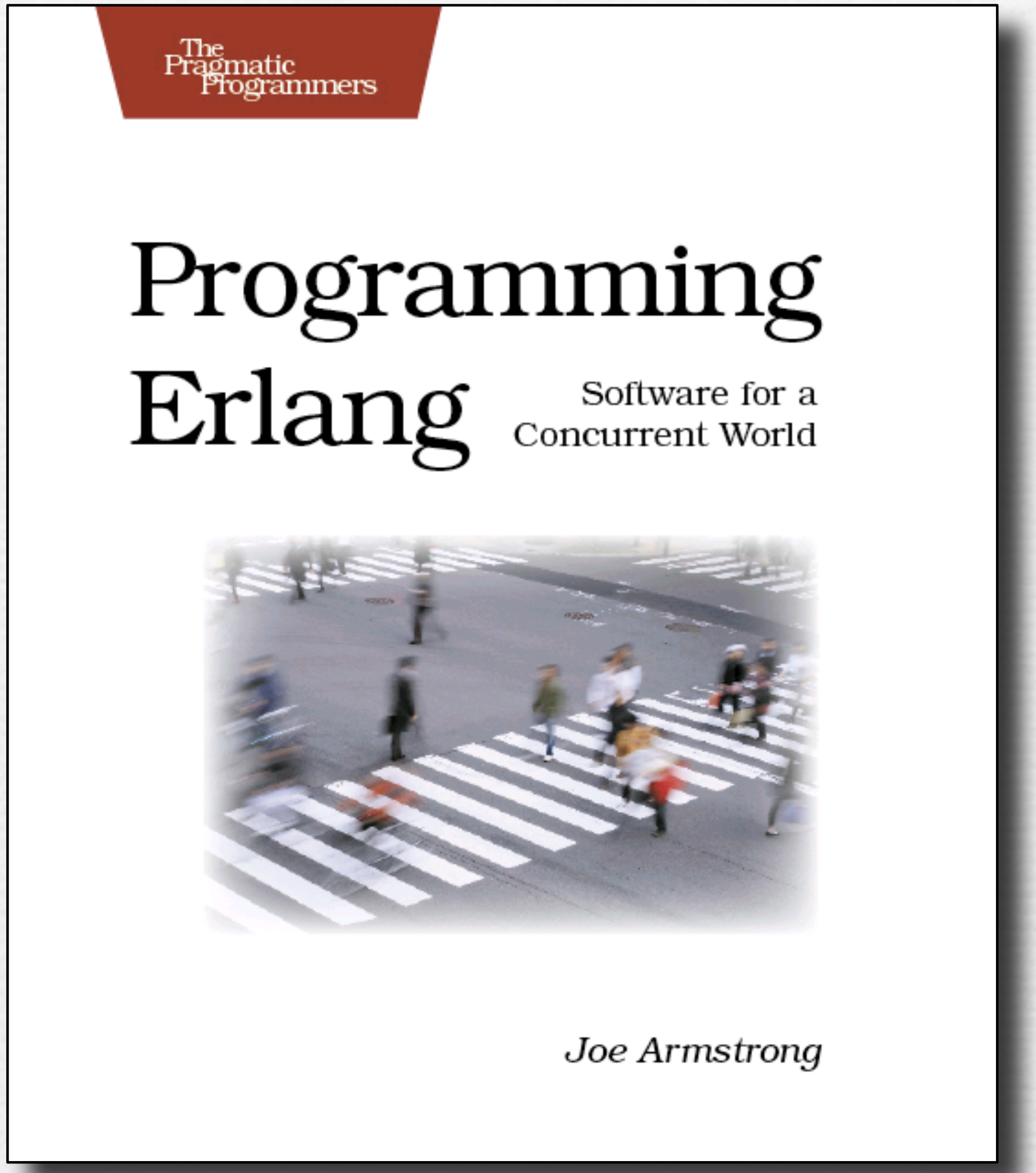
Renzel -
Error Handling for Business Information Systems,
Software Design and Management, GmbH & Co. KG, München, 2003

ARMSTRONG

- Processes are the units of error encapsulation. Errors occurring in a process will not affect other processes in the system. We call this property *strong isolation*.
- Processes do what they are supposed to do or fail as soon as possible.
- Failure and the reason for failure can be detected by remote processes.
- Processes share no state, but communicate by message passing.

Armstrong
Making reliable systems in the presence of software errors
PhD Thesis, KTH, 2003

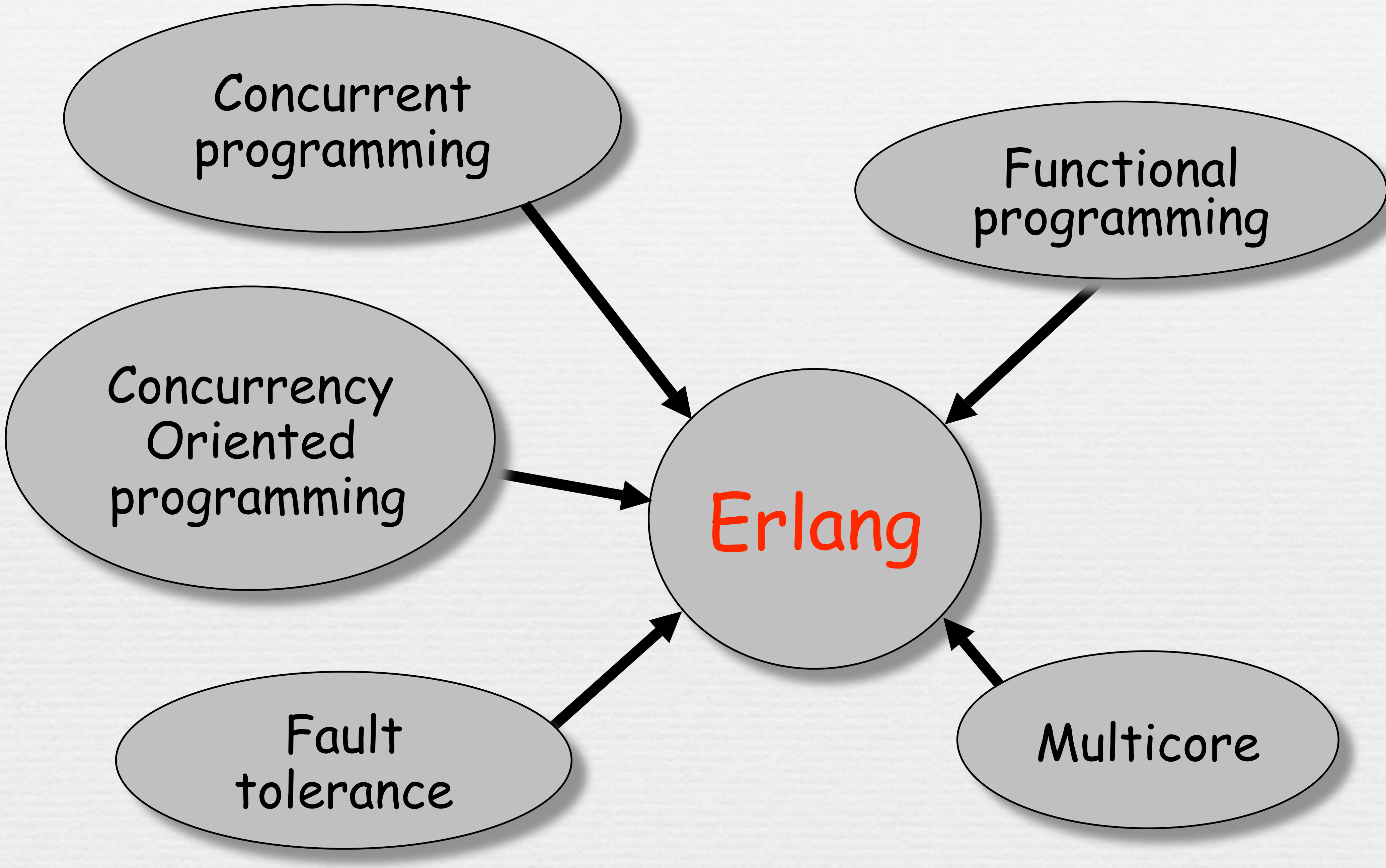
COMMERCIAL BREAK


The Pragmatic Programmers
Programming Erlang
Software for a Concurrent World
Joe Armstrong

Joe's 2'nd theorem

- Whatever Joe starts talking about, He will end up talking about Erlang

Erlang was designed to program fault-tolerant systems


Concurrent programming
Functional programming
Concurrency Oriented programming
Erlang
Fault tolerance
Multicore

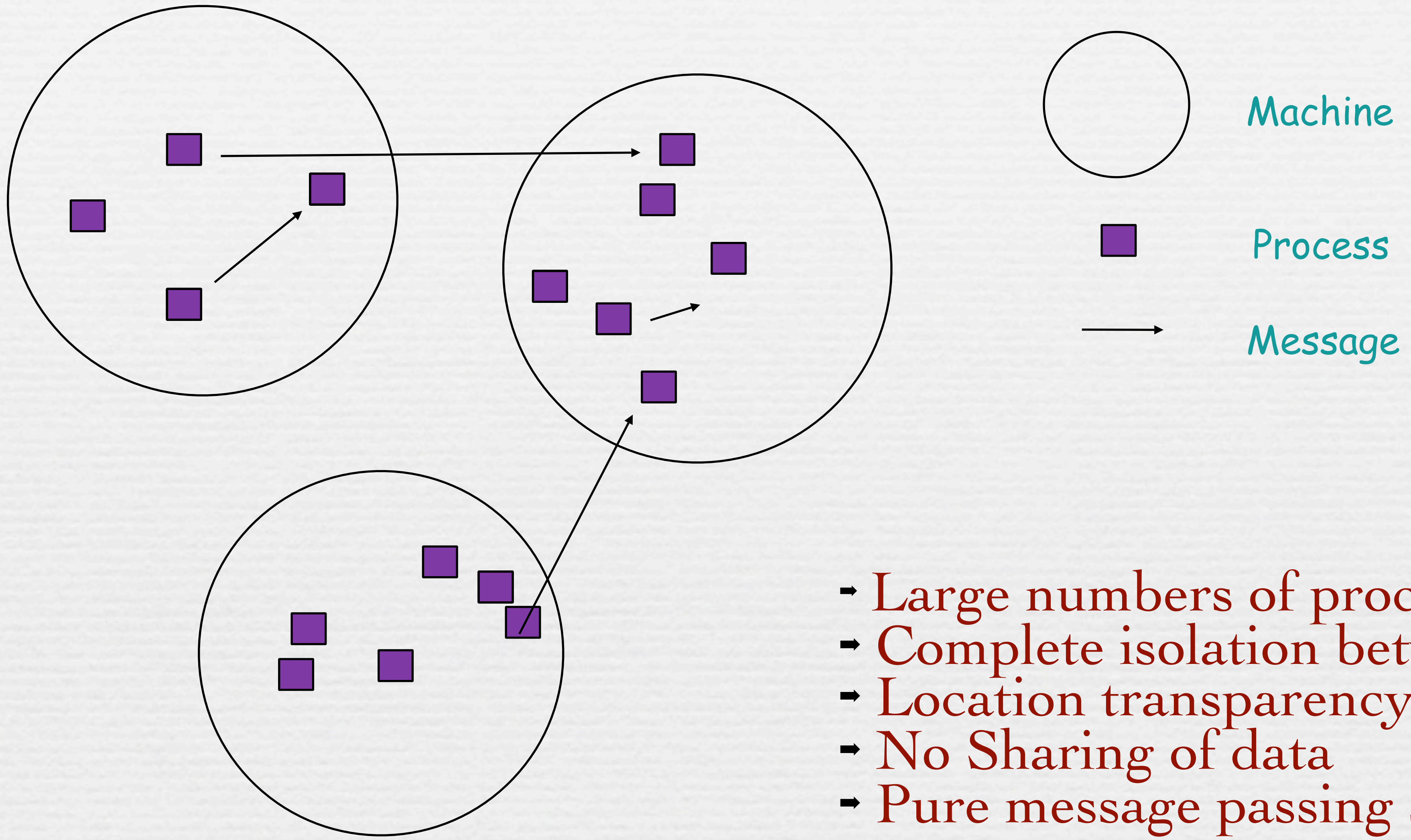
Erlang

- Very light-weight processes
- Very fast message passing
- Total separation between processes
- Automatic marshalling/demarshalling
- Fast sequential code
- Strict functional code
- Dynamic typing
- Transparent distribution
- Compose sequential AND concurrent code

Properties

- No sharing
- Hot code replacement
- Pure message passing
- No locks
- Lots of computers (= fault tolerant scalable ...)
- Functional programming (no side effects)

What is COP?



- Large numbers of processes
- Complete isolation between processes
- Location transparency
- No Sharing of data
- Pure message passing systems

Thread Safety

Erlang programs are automatically thread safe if they don't use an external resource.

Functional

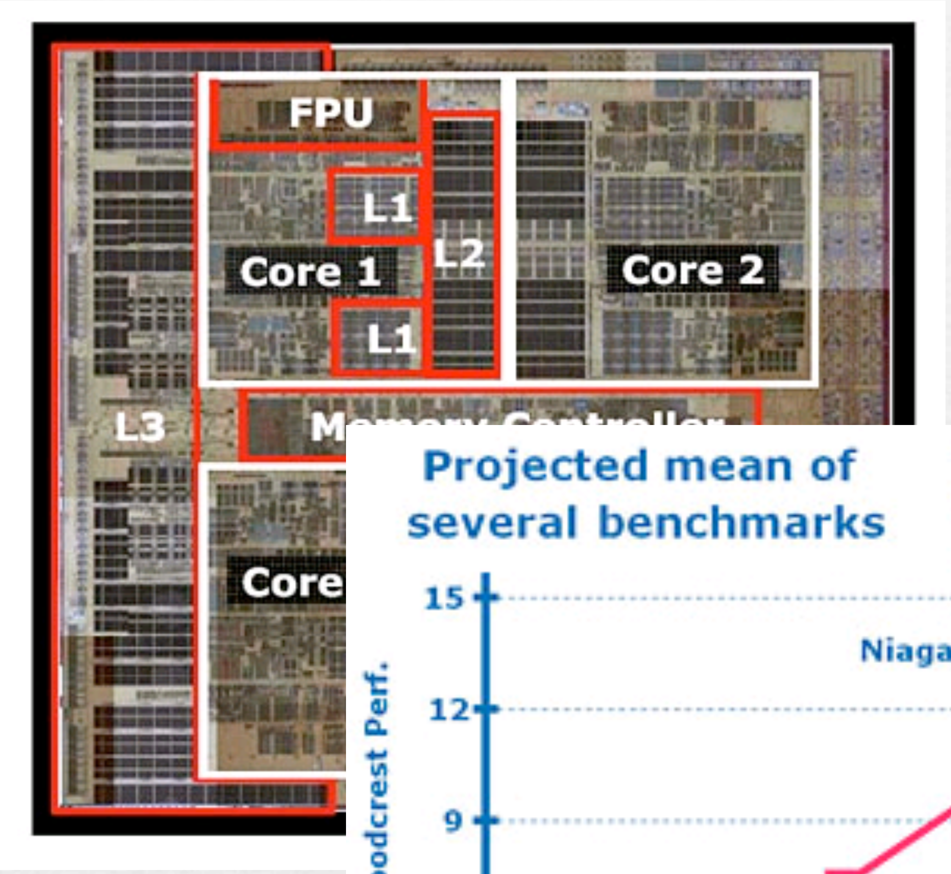
If you call the
same function twice with
the same arguments
it should return the same value

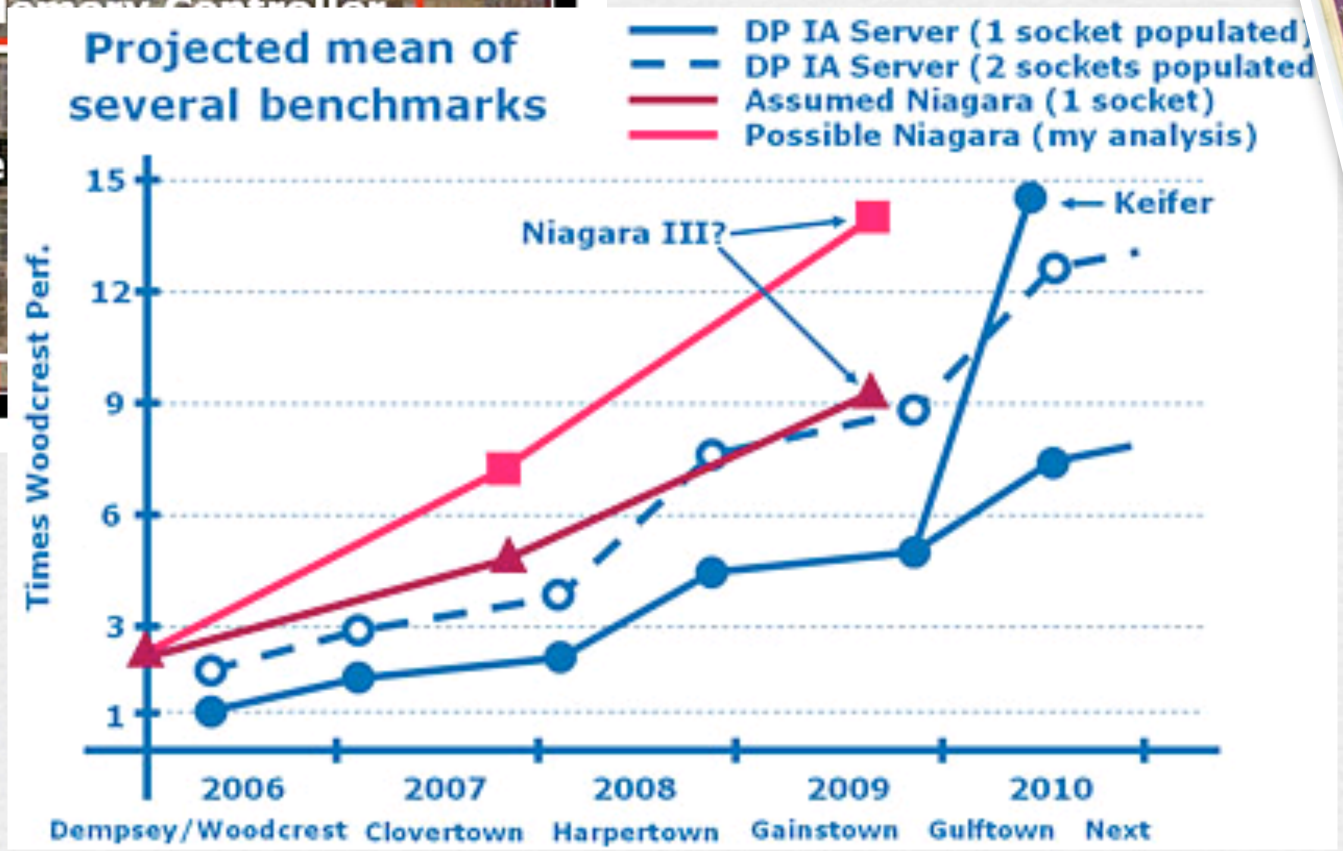
"jolly good"
Joe Armstrong

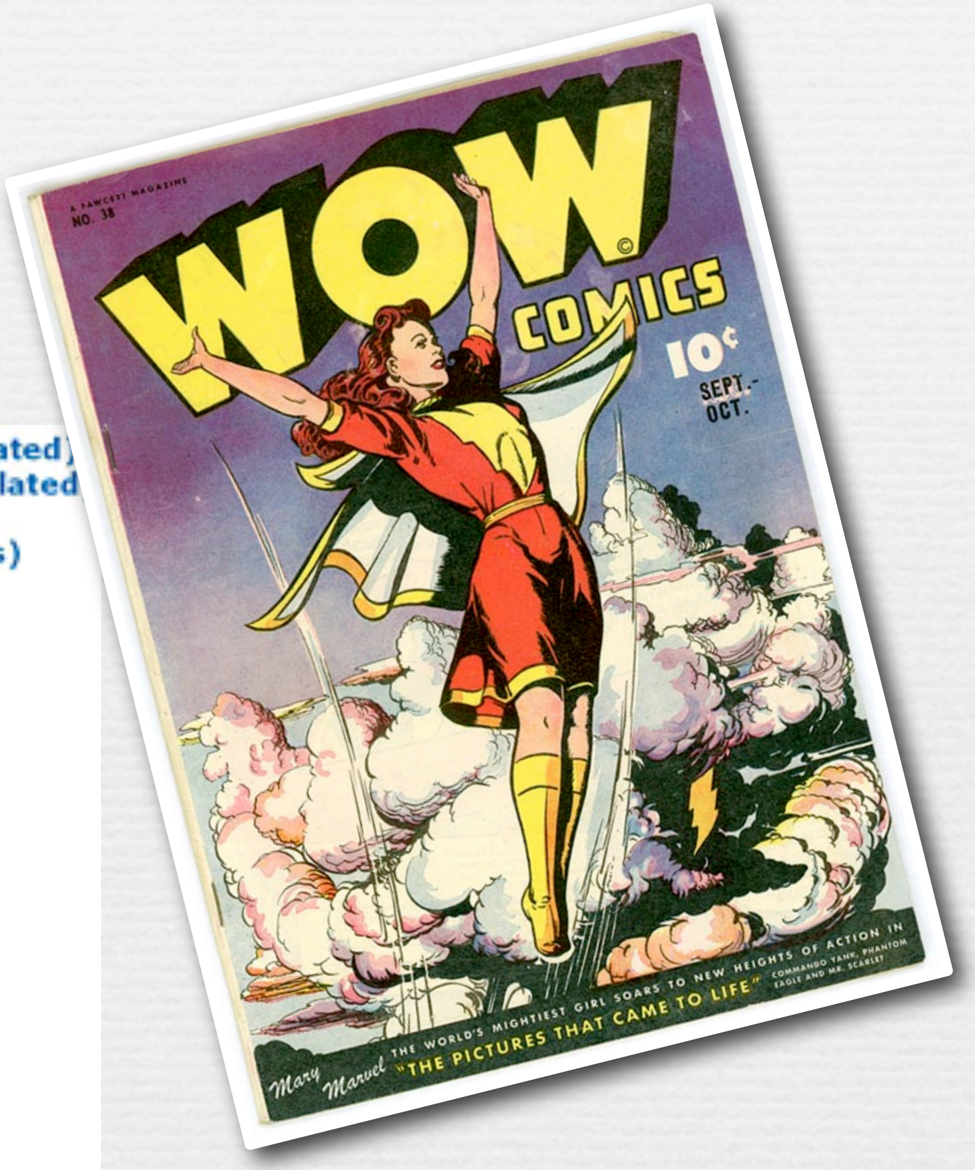
No Mutable State

- Mutable state needs locks
- No mutable state = no locks = programmers bliss

Multicore ready







The rise of the cores

- 2 cores won't hurt you
- 4 cores will hurt a little
- 8 cores will hurt a bit
- 16 will start hurting
- 32 cores will hurt a lot (2009)
- ...
- 1 M cores ouch (2019)
- (complete paradigm shift)

- 1997 1 Tflop = 850 KW
- 2007 1 Tflop = 24 W (factor 35,000)
- 2017 1 Tflop = ?

LAWS

ISOLATION CONCURRENCY

```
Pid = spawn(.....)
Pid = spawn(Node, ....)

Pid ! Message
```

```
receive
     Pattern1 -> Actions1;
     Pattern2 -> Actions2;
     ...
end
```

FAULT IDENTIFICATION

```
link(Pid),
     receive
          {Pid, 'EXIT', Why} ->
                ...
     end
```

LIVE CODE UPGRADE

- Can upgrade code while its running
- Existing processes continue to use original code, new processes run new code - no mixups of namespaces
- Sophisticated roll-forward, roll-back, roll-back-on-error functions in OTP libraries
- Properly designed systems can be rolled-forward and back with no loss of service. Not easy, but possible

STABLE STORAGE

- Performed in libraries

```
mnesia:transaction(
    fun() ->
            Val = mnesia:read(Key),
             mnesia:write({Key,Val}),
             ...
    end)
```

Projects

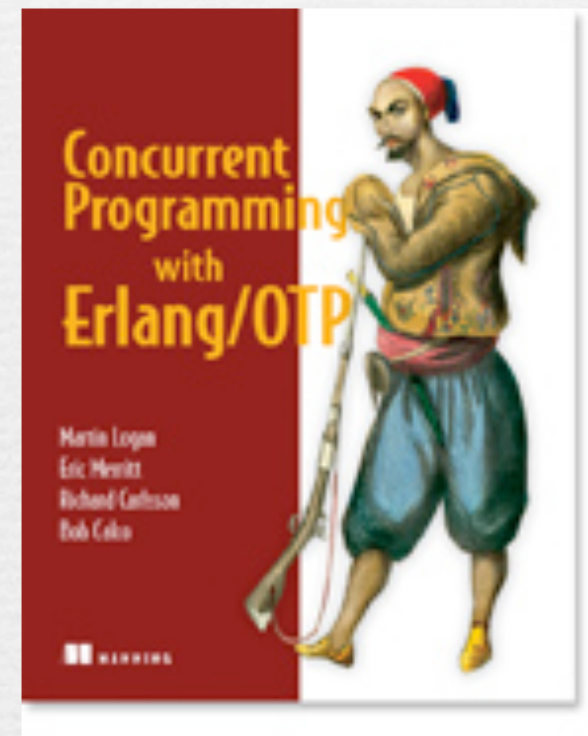
- CouchDB
- Amazon SimpleDB
- Mochiweb (facebook chat)
- Scalaris
- Nitrogren
- Ejabberd (xmpp)
- Rabbit MQ (amqp)
- ....

Companies

- Ericsson
- Amazon
- Tail-f
- Kreditor
- Synapse
- ...

Books


The Pragmatic Programmers
Programming Erlang
Software for a Concurrent World
Joe Armstrong
Code You Can Believe In
Real World Haskell
Bryan O'Sullivan, John Goerzen & Don Stewart
Rough Cuts
CouchDB
J. Chris Anderson, Noah Slater & Jan Lehnardt
O'REILLY
A Concurrent Approach to Software Development
Erlang
Programming
Francesco Cesarini & Simon Thompson
O'REILLY


Concurrent Programming with Erlang/OTP
Martin Logan
Eric Merritt
Richard Carlsson
Bob Calco

THE END