Erlang Solutions Ltd.

1000 Year-old Design Patterns

Ulf Wiger
Erlang Solutions Ltd

QCon, San Francisco, 5 November 2010

What this talk is about

- The search for an idea
- A walk down Memory Lane

- No easy recipes

Erlang

Movie Tips

- A few movies will be recommended
- …when they illustrate some aspect of this talk
- Try expensing them as “study of intuitive concurrency design patterns”

From Inception (2010) http://www.imdb.com/title/tt1375666/

Erlang

Human cooperation is naturally concurrent

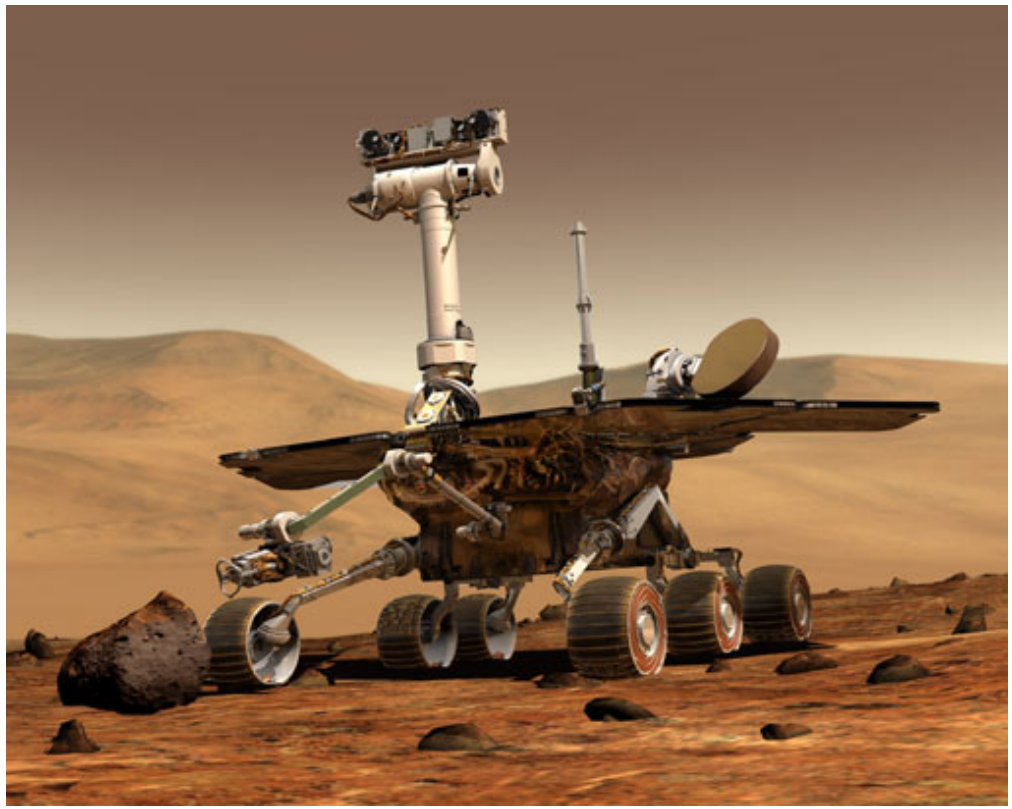
- All sorts of concurrency problems are common knowledge to humans
- Mitigation strategies have been explored for millennia
- Lots of coordination and supervision design patterns

http://www.sassansanei.com/images/fullsize-trafficjam-640x480.jpg

Erlang

A problem…

- Although humans document their algos,
- …they do it for human consumption (S.O.P.s)
- Not for programmers
- Most research into ”human algorithms” is about cognitive modeling (autonomous robots)

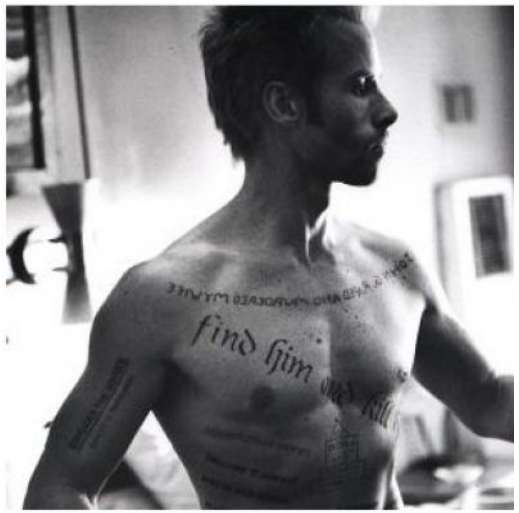


Mars Rover

Erlang

Research into human protocol

- Collect examples of how humans solve cooperation problems
- Go to the movies!
- Observe real-life patterns; consider what could transfer to software systems



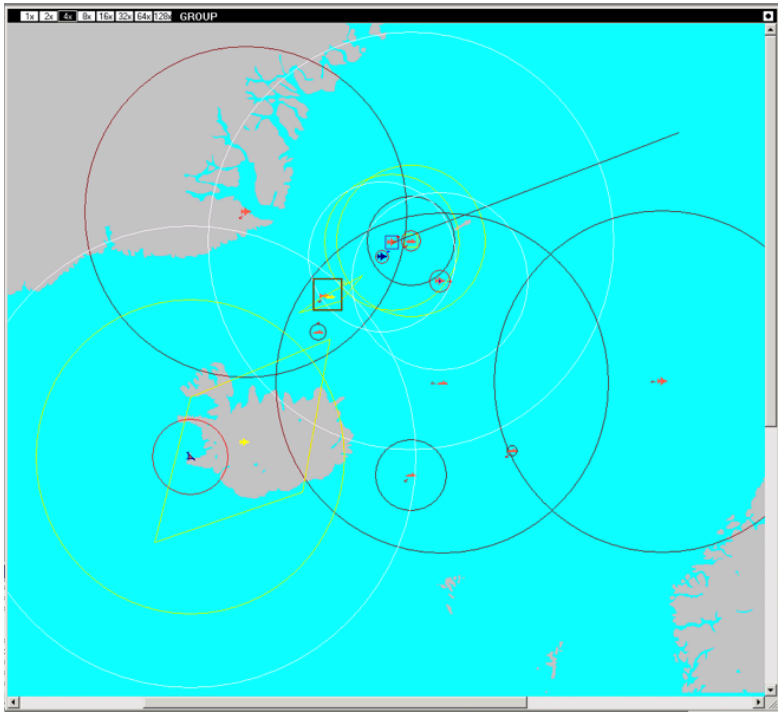
Erlang

AC²SMAN - My formative years

- Alaskan Command & Control System Military Automated Network
  - Built in 4 months by a fighter pilot from Memphis, and some geeks
  - First ever “Overall Outstanding” rating given by NORAD 1989

Erlang

The C^2 System Design Challenge



- Mission-critical
- Soft real-time
- Inconsistent data input
- Varying operating conditions
- Potentially global scale
- No single point of failure (40+ sites)
- Live, simulation and exercise – sometimes simultaneously

Erlang

The Competition

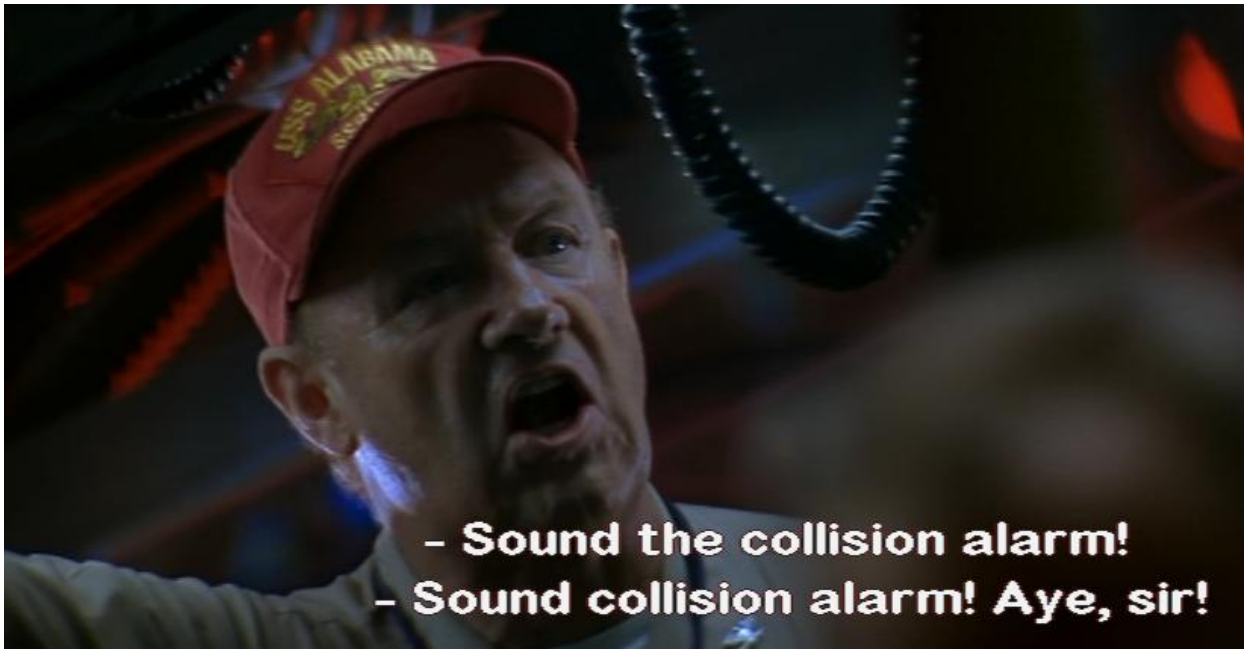
- One project had a $200M/year budget
- Desert Storm C^2 system *installation* took 50K man-hours! (…!!!)
- (in our view) No alternative system came close to competing

- The secret?
- Keep the project small…
- Automate the existing workflow!
  - The Air Force already knew how to do this - manually

Erlang

(Movie Tip)

- Crimson Tide (1995)
- Military command protocol
- Redundancy
- Fail-safes
- Byzantine Generals Problem
- Bully algorithm



http://www.imdb.com/title/tt0112740/

Erlang

AC²SMAN Database issues

- Asynchronous, event-triggered replication
  - Across 40 sites
  - PAMS – Process-Activated Messaging System (later DECMessageQ)
  - PowerHouse 4GL on top of DEC RMS
- No 2-phase commit – no conflicts ”possible”
  - Access control and operational procedure limit what people can do
  - Procedures for assessing multiple conflicting inputs
- Main challenge: full replication over a 19.2Kbps modem line
- Relational databases anno 1989 were simply non-starters

Erlang

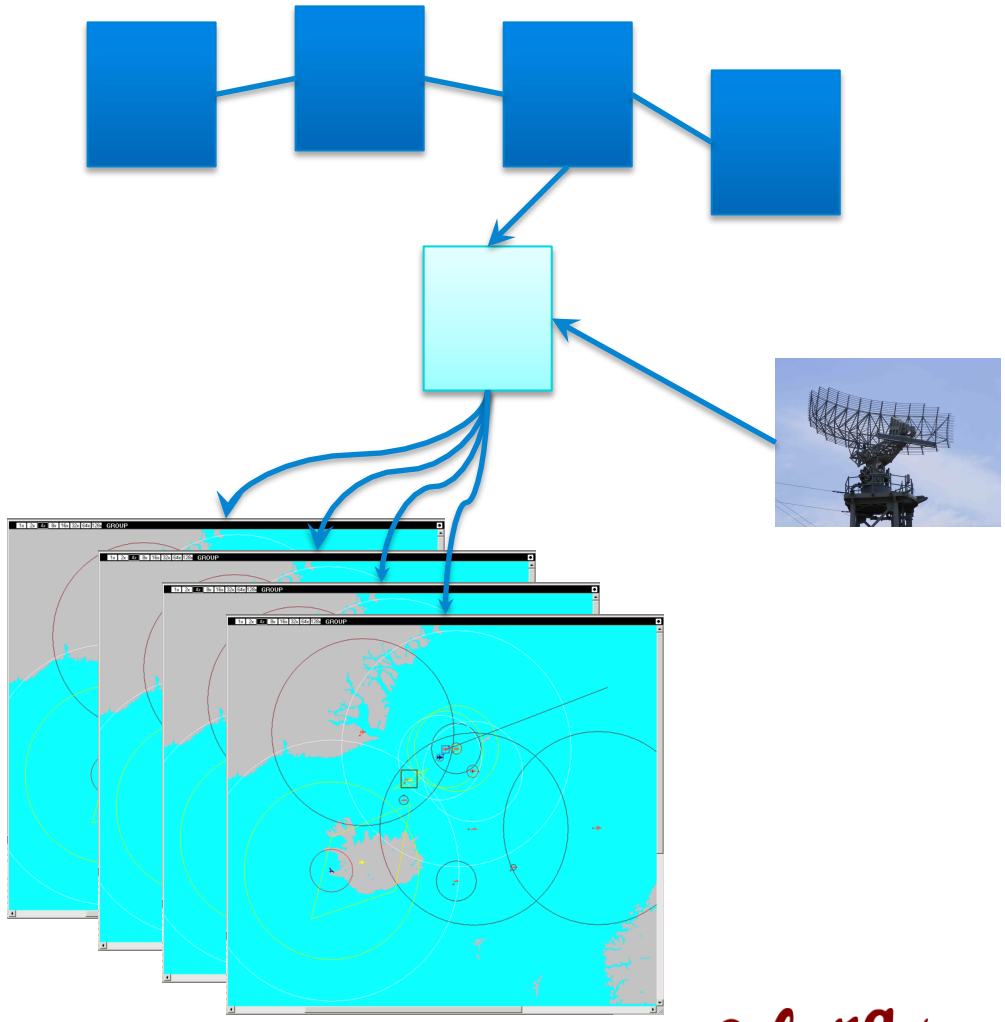
The failed alternative?

- Trying to use early-90s Distributed RDBMS technology
- This was the beginning of the hardships that led to the CAP Theorem
- The problem didn't call for an RDBMS
  - We're automating a workflow that's been around for millennia

Erlang

The Feed Aggregation Problem

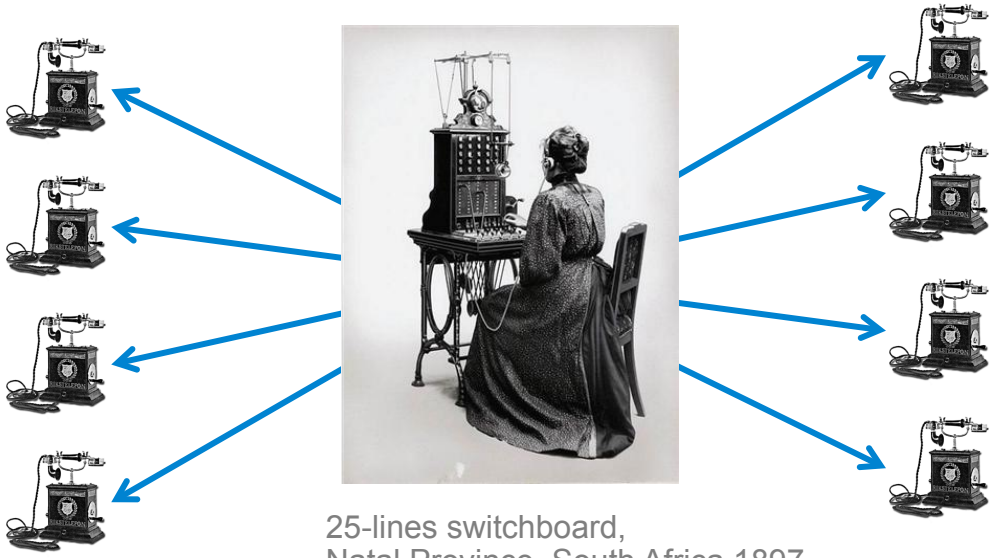
- Real-time subscription feed for tactical map workstations
- Messaging server was a big pile of C++ code
- Single point of failure
- Ran out of memory daily
- (Not due to programmer incompetence)



Erlang

I was Searching for a Solution

- Tons of approaches evaluated
  - CASE Tools, Client-Server middleware, AI middleware…
- Eventually landed in telecoms 1992
  - ”Computers in Telecommunictions” course at KTH, Stockholm
  - Teachers: **B Däcker**, **R Virding**
  - Programming language: **Erlang**
- Erlang seemed to be a perfect fit!



25-lines switchboard,
Natal Province, South Africa 1897
Cross-switchboard calls required human interaction.

Erlang

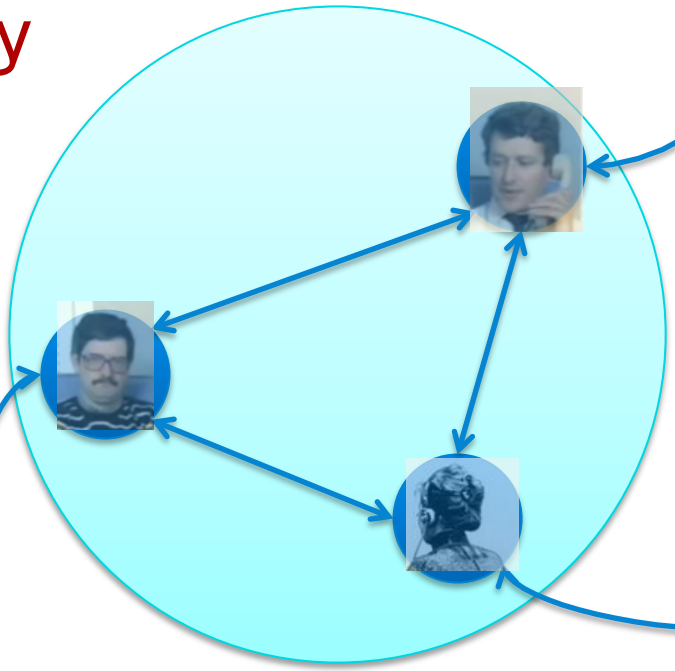
Erlang, Intuitively

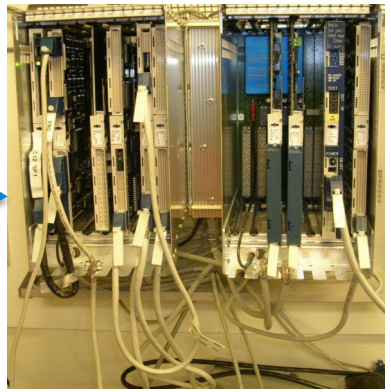
http://video.google.com/videoplay?docid=-5830318882717959520#

Erlang

Erlang, Intuitively

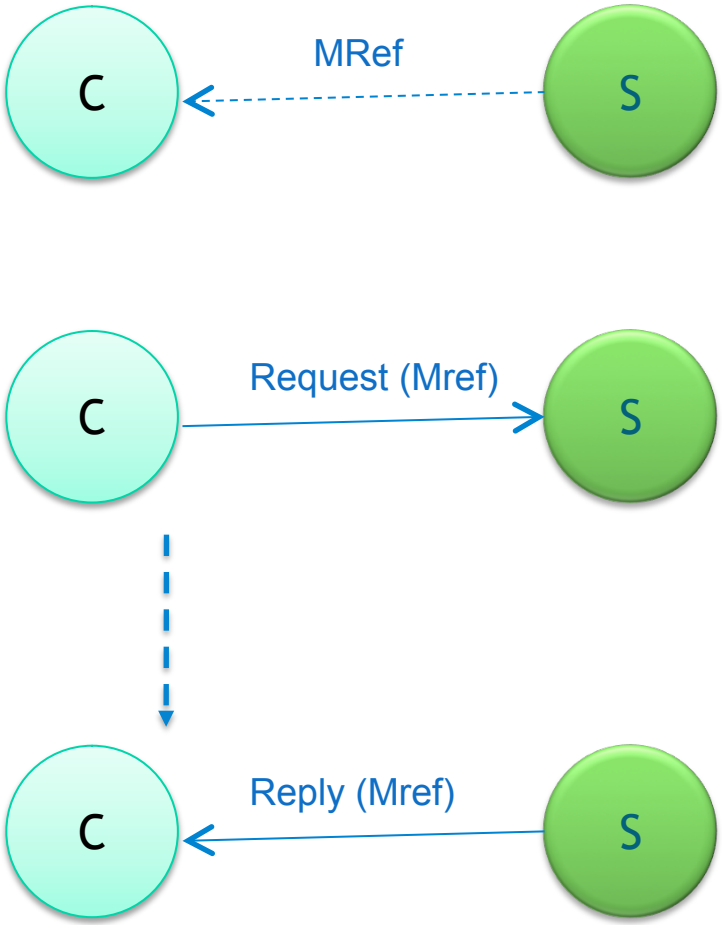
- One concurrent process for each naturally concurrent activity





Erlang

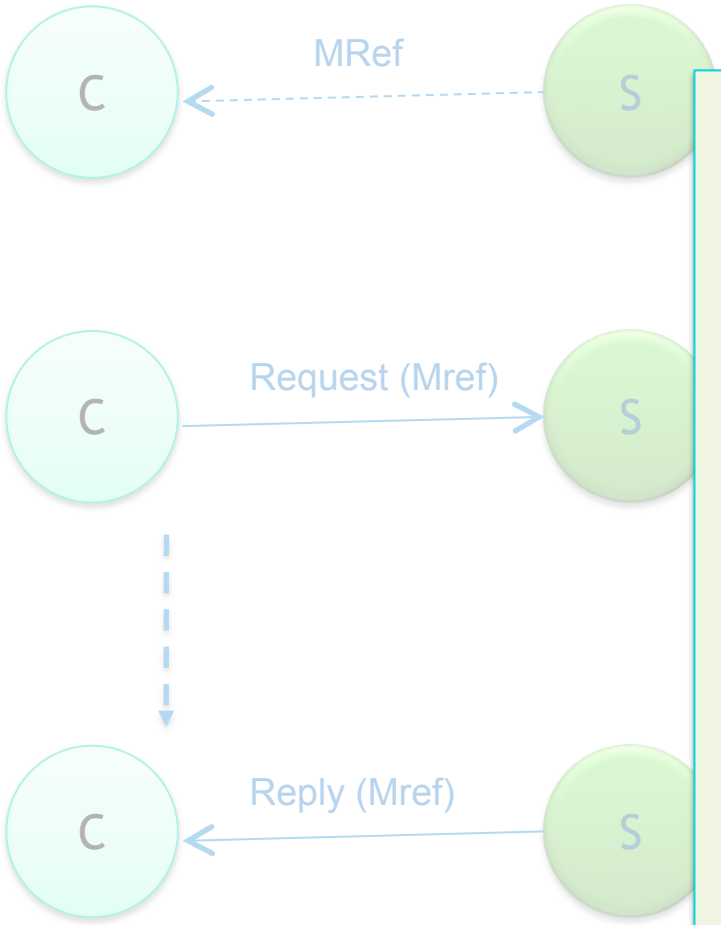
Client-server in Erlang



1. Client monitors server
2. Client sends a request
3. (Blocks while waiting)
4. Server sends reply

Erlang

Client-server in Erlang

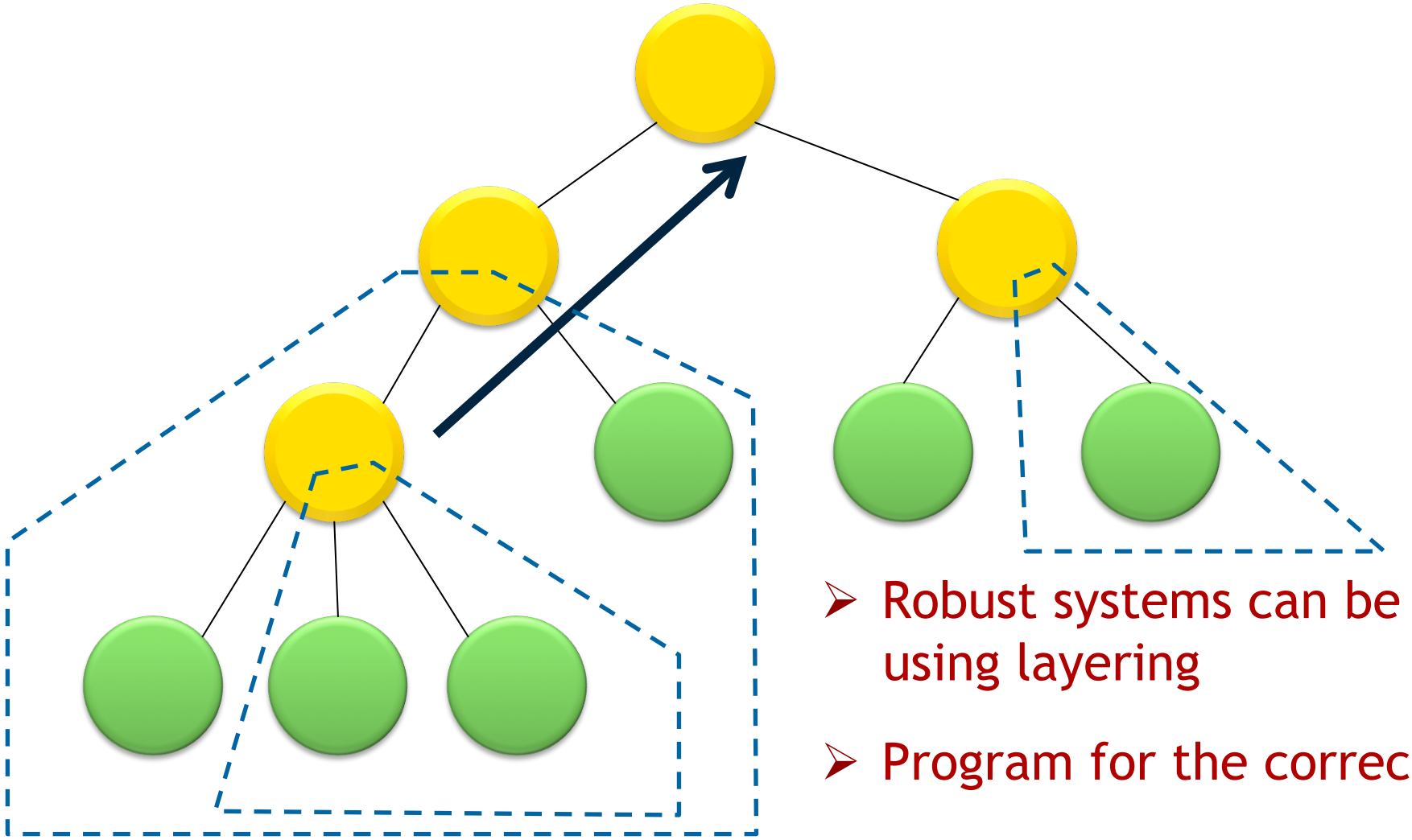



```
call(S, Request, Timeout) ->
    Mref = monitor(process, S),
    S ! {call, Mref, Request},
    awaiting_reply(Mref, Timeout).

awaiting_reply(Mref, Timeout) ->
    receive
        {Mref, Reply} ->
            Reply;
        {'DOWN', Mref, _, _, Reason} ->
            error(Reason)
    after Timeout ->
        error(timeout)
    end.
```


Erlang

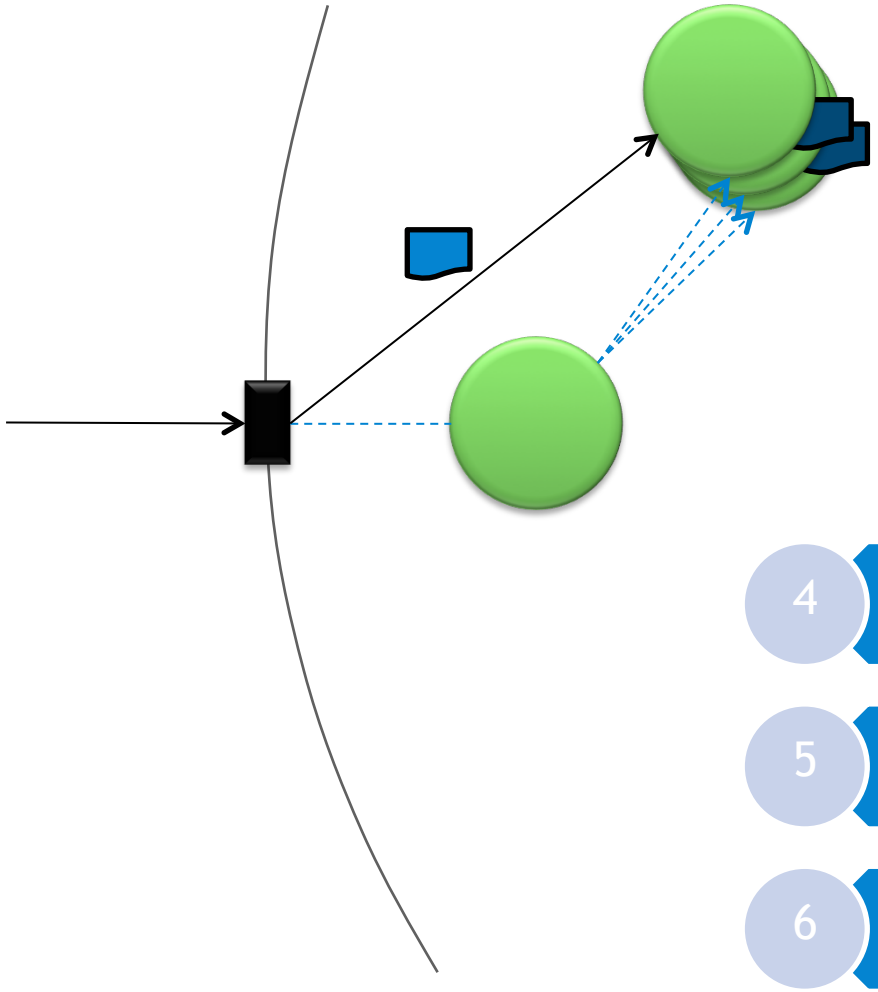
Supervisors – Out-of-Band Error Handling



- Robust systems can be built using layering
- Program for the correct case

Erlang

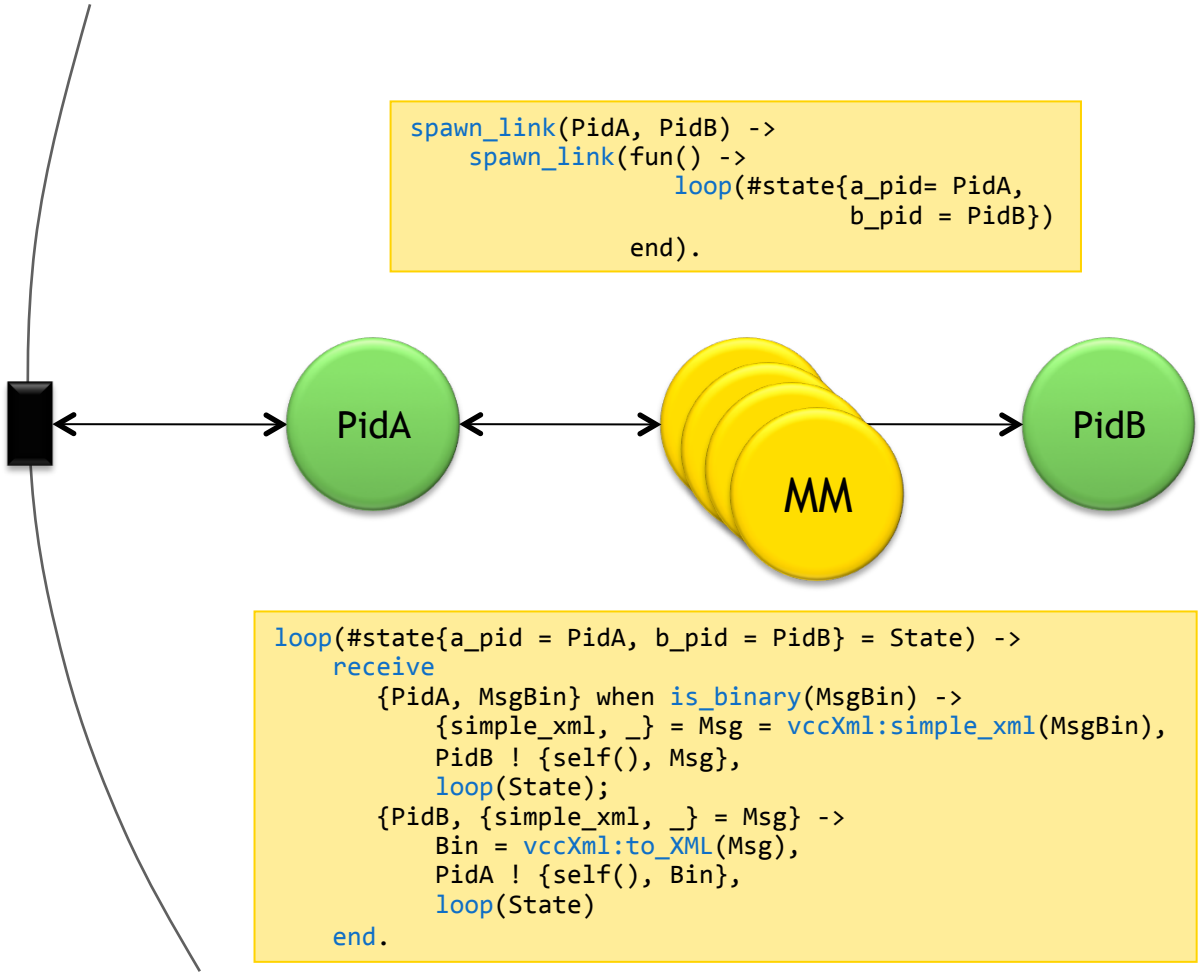
Handling sockets in Erlang



1. Static process opens listen socket
2. Spawns an acceptor process
3. Acceptor receives incoming
4. Acks back to socket owner
5. New acceptor is spawned
6. Replies sent directly to socket

Erlang

Middle-man Processes




```
spawn_link(PidA, PidB) ->
    spawn_link(fun() ->
                   loop(#state{a_pid= PidA,
                               b_pid = PidB})
               end).
```

```
loop(#state{a_pid = PidA, b_pid = PidB} = State) ->
    receive
       {PidA, MsgBin} when is_binary(MsgBin) ->
           {simple_xml, _} = Msg = vccXml:simple_xml(MsgBin),
           PidB ! {self(), Msg},
           loop(State);
       {PidB, {simple_xml, _} = Msg} ->
           Bin = vccXml:to_XML(Msg),
           PidA ! {self(), Bin},
           loop(State)
    end.
```

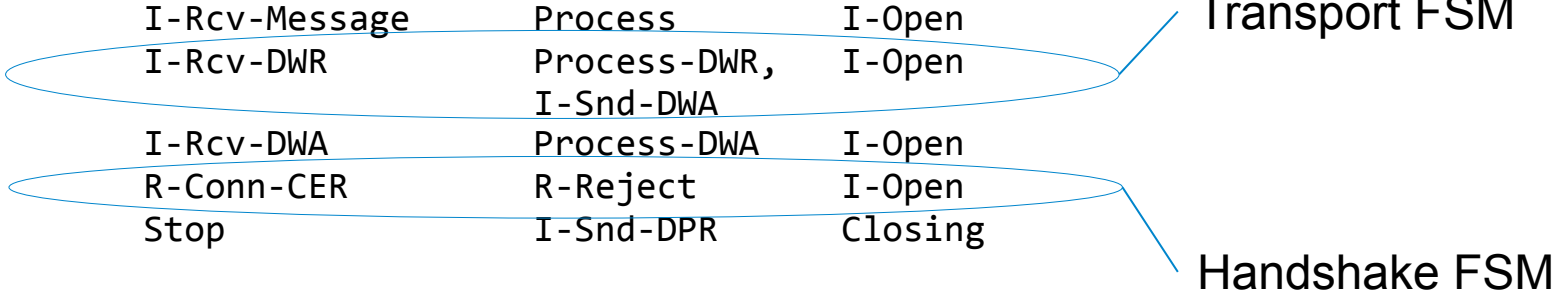
```
await_negotiation(State) ->
    receive
        {From,
         {simple_xml,
          [{"offer", Attrs, Content}]}} ->
            HisOffer =
                inspect_offer(Attrs, Content),
            Offer = calc_offer(HisOffer, State),
            From ! {self(), Offer};
        …
    end.
```

- Practical because of light-weight concurrency
- Normalizes messages
- Main process can pattern-match on messages
- Keeps the main logic clear

Language Model Affects our Thinking

Example: RFC 3588 – DIAMETER Base Protocol

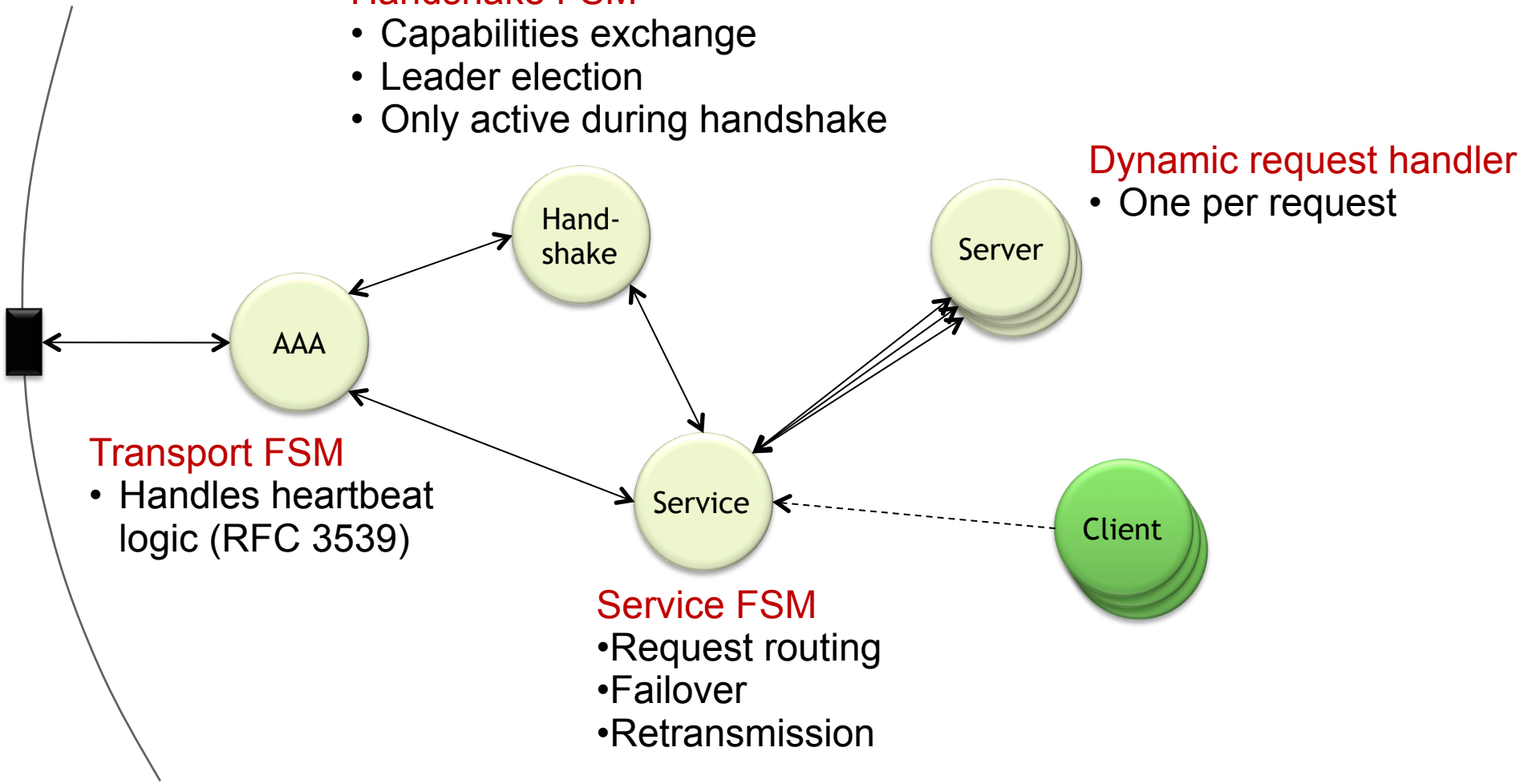
```
state            event                action           next state
-------------------------------------------------------------------
...
I-Open           Send-Message         I-Snd-Message    I-Open
                 I-Rcv-Message        Process          I-Open
                 I-Rcv-DWR            Process-DWR,     I-Open
                                      I-Snd-DWA
                 I-Rcv-DWA            Process-DWA      I-Open
                 R-Conn-CER           R-Reject         I-Open
                 Stop                 I-Snd-DPR        Closing
...
```



- Three state machines described as one
- Implies a single-threaded event loop
- Introduces accidental complexity

Erlang

Use processes to separate concerns



Erlang

Ericsson – The Mythical Project

- I joined Ericsson 1996 to work with Erlang
- A very large project had just been canceled
  - A well-publicized failure
- Distributed real-time, fault-tolerant complex systems in C++

Erlang

Why did it crash?

- No obvious single culprit
  - Discussions about what went wrong dragged on for years
- Obviously, the size of the project was a problem
  - But why so large?
- OO mania, featuritis, hubris?
- My thought: failure to contain the problem

Erlang

AXD301 – The Pickup Project

- 200 people put into one building
- Mission: Build a product within 2 years
  - Something in the ATM domain with Telecom Characteristics
- Much leeway was given
- Erlang/OTP chosen as key implementation technology
- Result: A product *was* delivered in 2 years
  - Eventually returned Wireline Division to profit

Erlang

Pragmatic thinking

- Shell shocked from previous project
- Fall back on what's known to work
- Straight and simple took us pretty far
  - Up to 16x16 = 256 interconnected boards
  - Up to 32 control plane processors
  - Up to 500k simultaneous phone calls
  - > 99.999% consistent uptime
    - **(including maintenance & upgrades)**

Erlang

Abstractions for non-determinism

- We were building complex distributed message-passing systems
- Key challenge: contain the non-determinism!
- Prevent explosion of the state-event matrix
- This had been identified by Ericsson already in the late 70s…

Erlang

What's the Secret Sauce?

- We weren't smarter, more experienced
- We used an unproven technology
  - Beta-tested the first version of the OTP middleware
- Yet, we outperformed other comparable projects
- What did the trick?
  - Immutability?
  - Functional programming?
  - Concurrency model?
  - Nothing?

Erlang

Outsiders about Erlang

- Non-programmers in our projects liked Erlang
- They understood the abstractions and design patterns

Some similar projects

- In one (mature) UML/C++ project, 10% of all bugs were related to unexpected order of events
- Inadequate methods for abstracting away accindental ordering

Erlang

Programs modeling ”human protocols”

- Must have their own thread of control
- Communicate with messages
- A sense of time
- Adapt to changes/problems
- Control order of input processing

Erlang

Sanity check

- When assessing a concurrency pattern in software, try to imagine what it would correspond to in real-life, enacted by humans

Erlang

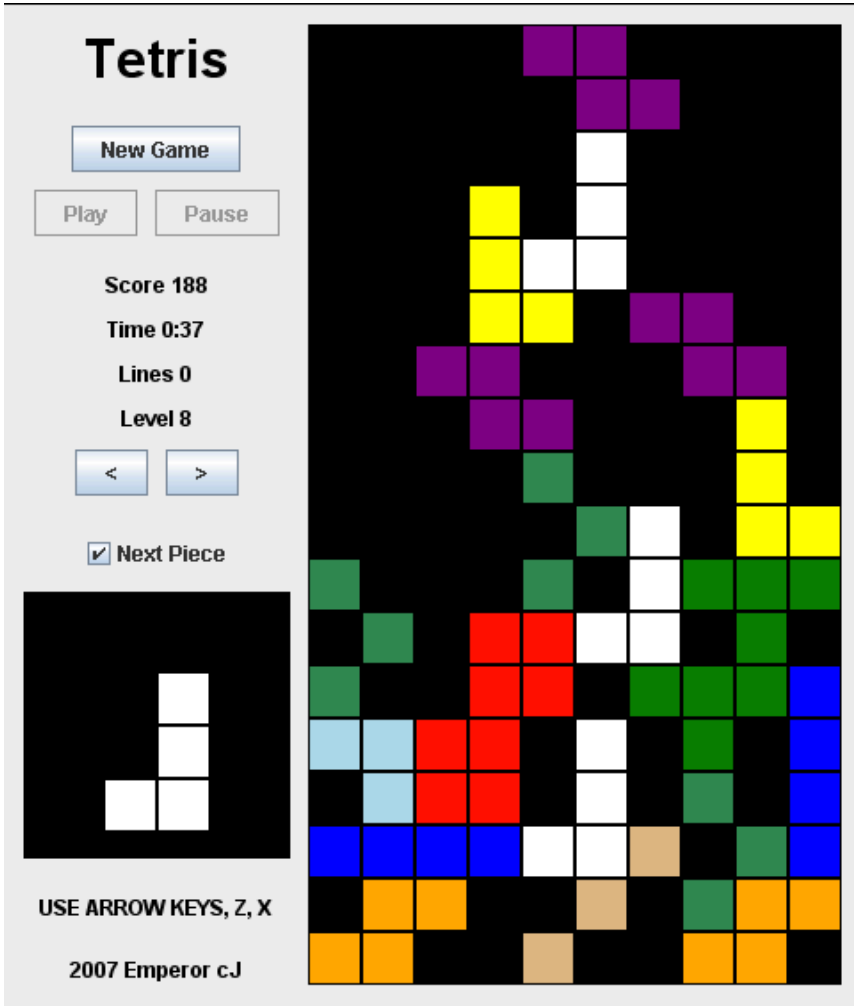
Tetris Management

- The age-old classic has coined a new time management method
- The idea: learn how to keep the pile small

Erlang

Tetris Management

- Used in a derogatory sense at a major software development project
- As in ”reactive management without a plan”
- Basically, don’t let your project become a tetris game



Erlang

A different kind of puzzle

- What if your problem more resembles this?
- Would you attack this problem with a tetris approach?

http://www.worldslargestpuzzle.com/hof-008.html

Erlang

Event Handling Strategies

- Twist and place the next piece – before it lands
- In cheat mode, you get to peek at the next one
- Otherwise, hope for the best

- Search for a specific piece
- Put away pieces that don't fit
- Keep at it until fitting piece found

Erlang

Event Handling in Software

- FIFO, run-to-completion event handling
- Not allowed to block
- Fine, as long as the pieces fit…

- Blocking, selective receive
- Wait until the next *desired* piece arrives
- Buffer unknown pieces

Erlang

(Movie Tip)

- Memento (2000)
- Human FIFO, run-to-completion event handling
- Storing context for future reference

Memento (2000) http://www.imdb.com/title/tt0209144/

Erlang

In conclusion

- Our mental models greatly influence how we attack software problems
- Our real-life experience is full of useful patterns for concurrency
- Actor-style programming is a pretty good fit for modeling such patterns

Wall-E (2008) http://www.imdb.com/title/tt0910970/

Erlang

Questions?