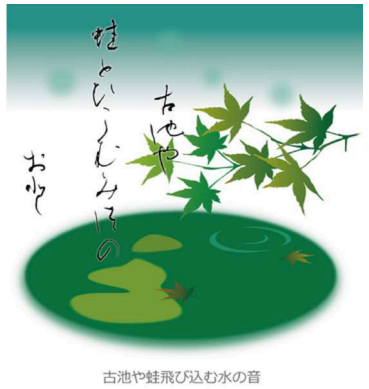
Using Erlang in Blockchain Development

Ulf Wiger

Æternity

Code BEAM STO, Stockholm 1 Jun 2018

What's the Best Language for Poetry?


古池や蛙飛び込む水の音

Kennst du das auch, daß manchesmal
Inmitten einer lauten Lust,
Bei einem Fest, in einem frohen Saal,
Du plötzlich schweigen und hinweggehn mußt?

Dann legst du dich aufs Lager ohne Schlaf
Wie Einer, den ein plötzlich Herzweh traf;
Lust und Gelächter ist verstiebt wie Rauch,
Du weinst, weinst ohne Halt - Kennst du das auch?

Stop all the clocks, cut off the telephone,
Prevent the dog from barking with the juicy bone.
Silence the pianos and, with muffled drum,
Bring out the coffin. Let the mourners come.

Ja nog är det svårt när droppar faller.
Skälvande av ängslan tungt de hänger,
klamrar sig vid kvisten, sväller, glider –
tyngden drar dem neråt, hur de klänger.

Expressing Complex Ideas

- Languages are shaped by culture and experience
- The language shapes the expression of complex ideas
- Universal grammar *(Chomsky 2000)*

Translating Complex Ideas

Furu ike ya
kawazu tobikomu
mizu no oto

Old pond — frogs jumped in — sound of water.

A lonely pond in age-old stillness sleeps . . .
Apart, unstirred by sound or motion . . . till
Suddenly into it a lithe frog leaps.

Into the ancient pond
A frog jumps
Water's sound!

The old pond,
A frog jumps in:
Plop!

http://www.bopsecrets.org/gateway/passages/basho-frog.htm

Opinionated Programming Languages

Instruction-level

```
Start:
    mov   dx,OFFSET okmsg     ; start out optimistically
    fild  [first]          ; load the first number (x)
    fild  [second]           ; and the second (y)
    fdiv  st(1),st         ; perform y/x
    fmulp st(1),st           ; now st(0) = (y/x)*x
    fild  [first]          ; reload y
    fcompp                   ; compare the two
    fnstsw ax                ; put status word into ax
    sahf                  ; load into CPU flags
    jz    short @@NoBug      ; if they're equal, no bug
    mov   dx,OFFSET bugmsg     ; load bad news message...
@@NoBug:
    mov   ah,9               ; print appropriate message
    int   21h             ;
    mov   ah,4ch             ; and exit
    int   21h             ;
```

Math-oriented

```
*     Prints the values of e ** (j * i * pi / 4) for i = 0, 1, 2, ..., 7
*         where j is the imaginary number sqrt(-1)

      PROGRAM CMPLXD
        IMPLICIT COMPLEX(X)
        PARAMETER (PI = 3.141592653589793, XJ = (0, 1))
        DO 1, I = 0, 7
          X = EXP(XJ * I * PI / 4)
          IF (AIMAG(X).LT.0) THEN
            PRINT 2, 'e**(j*', I, '*pi/4) = ', REAL(X), ' - j',-AIMAG(X)
          ELSE
            PRINT 2, 'e**(j*', I, '*pi/4) = ', REAL(X), ' + j', AIMAG(X)
          END IF
    2     FORMAT (A, I1, A, F10.7, A, F9.7)
    1     CONTINUE
        STOP
      END
```

Everything's an object

```
class Circle {          // classname
private:
   double radius;       // Data members (variables)
   string color;
public:
   double getRadius(); // Member functions
   double getArea();
}
```

Concurrent / functional

```
-module(pmap).
-export([f/2]).

f(F, Vals) ->
    Ps = [{V, spawn_monitor(fun() -> exit({ok,F(V)}) end)}
          || V <- Vals],
    [{V, collect(P)} || {V, P} <- Ps].

collect({P, Ref}) ->
    receive
        {'DOWN', Ref, process, P, Reason} ->
            {ok, Res} = Reason,
            Res
    end.
```

The Modern Divide

- Performance vs Productivity
  - High-Level languages—slower but some 10x more productive
    - Erlang, Python, Scala, Haskell, Clojure, ...
  - Low-Level—detailed, low overhead
    - C/C++, linkable
    - Java, etc. non-linkable
- Performance in complex systems is a different beast
  - HL languages may well be faster on some tasks, e.g.
    - Complex memory management
    - Complex concurrency

The Modern Divide (2)

- Concurrency
  - Strong concurrency by design
    - Erlang, Clojure, Haskell, GO (Rust?) ...
  - Concurrency as an afterthought
    - C/C++, Python, (Java), ...
- Fault-tolerance
  - By design
    - Erlang, Akka, Cloud Haskell ...
  - DIY
    - Most of the rest

What About Blockchains?

- Few parts are performance critical (today)
  - Mainly Proof of Work, hashing, signatures
  - Treat as an external service or BIFs (potentially specific hardware)
- Lots of networking
- Moving target
  - Algorithms/features still evolving

How Does Erlang Help?

- Loosely coupled components
  - Simplifies parallel development
  - Simplifies reuse
  - Flexible evolution
- Concurrency Done Right
  - Protocol aspects isolated from program logic
  - Easy to change/evolve protocols
  - Networking scalability not a big concern
  - (we're not using Distributed Erlang)
  - Complex state machine support (more later)

How Does Erlang Help? (2)

- Functional Programming
  - Simplifies testing
  - Code, once correct, tends to *stay* correct
  - Reduces surprising side-effects
  - Powerful for blockchain state management
- Carrier-Class Product Mentality
  - Stellar backward compatibility
  - Rock-solid VM
  - No "dependency hell"
  - Basically 'attack-proof' networking support

Challenges?

- Few other blockchain projects use Erlang
  - Fewer opportunities for direct reuse
  - Then again, re-writing/porting aids understanding ;-)

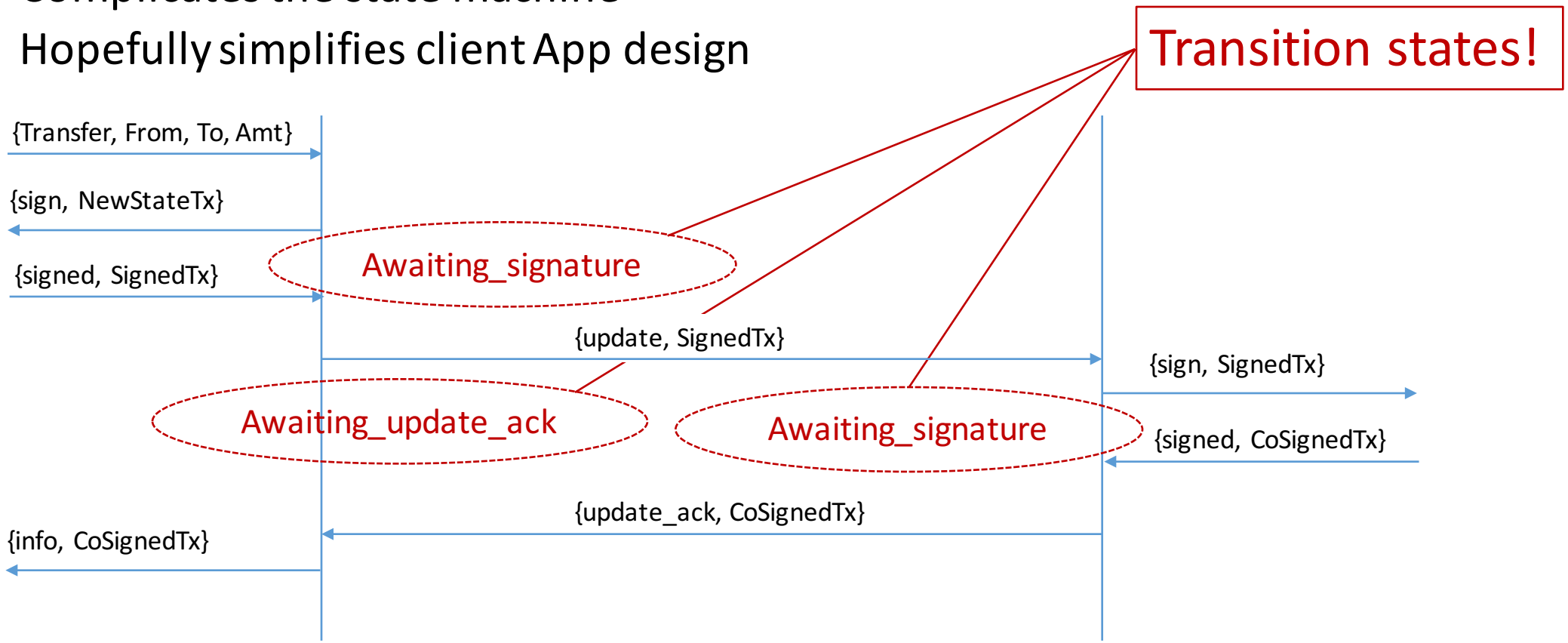
- Doesn't run on iOS or Android
  - Not necessarily much of a disadvantage

State channels in Erlang

- Purpose: Establish "off-chain" channels for fast and cheap transactions
  - On-chain activity only when opening and closing channel
  - Funds locked into the channel can be transferred in co-signed transactions "for free"
  - "Trust but verify" off-chain,
    Mutual close or dispute resolution on-chain

Ónen i-Estel Edain. Ú-chebin Estel anim

- Design decision: SC daemon with a simplified WebSocket API
  - Complicates the state machine
  - Hopefully simplifies client App design

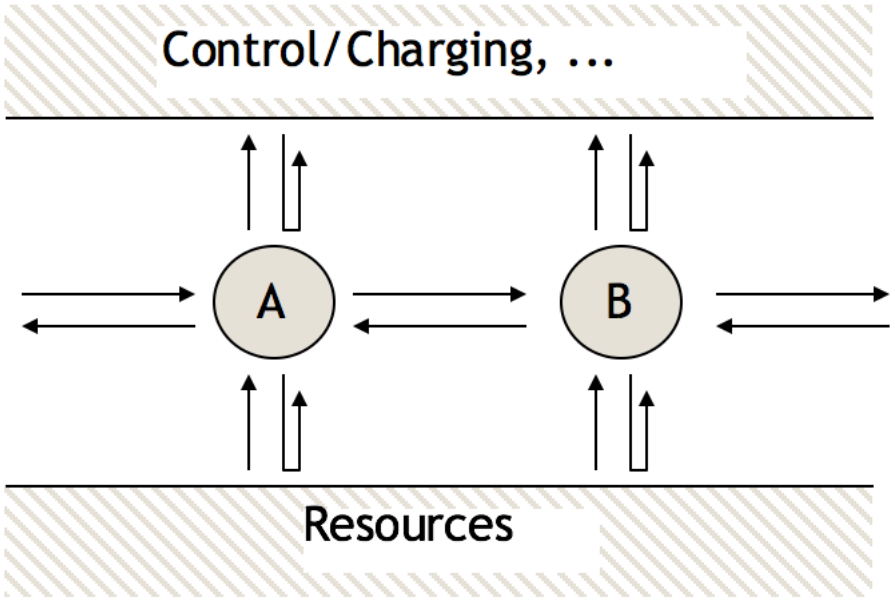


Avoid Death by Accidental Complexity

- https://www.infoq.com/presentations/Death-by-Accidental-Complexity
  (2010 talk, based on Structured Network Programming EUC 2005)

- Must avoid having to handle all possible orderings of incoming messages
- Otherwise, complexity explosion in transition states

Telecom "Half-Call" model



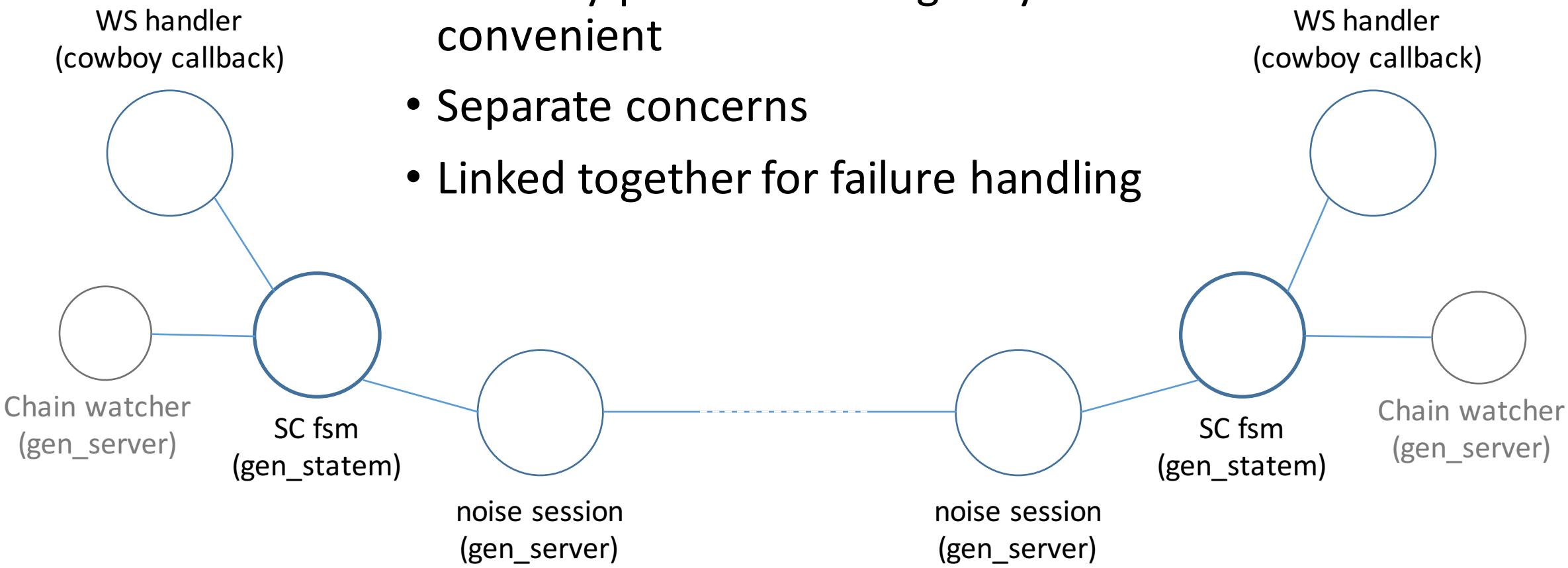
A = originating side
B = terminating side

State Machine programming in Erlang

- Old-school: textbook Erlang
  - Simple and beautiful
  - No automatic support for systems (OTP) functionality
  - `plain_fsm` – a cludgy way of getting both
- Old behavior: `gen_fsm`
  - Supports OTP functionality
  - Doesn't handle FSM complexity (no selective receive)
- New behavior: `gen_statem`
  - Supports OTP functionality
  - Supports selective receive

Erlang pays off—FSM programming in practice

- As many processes as logically convenient
- Separate concerns
- Linked together for failure handling



Transition state handling in `gen_statem`

```
awaiting_signature(cast, {?SIGNED, ?WDRAW_CREATED, SignedTx},
                   #data{latest = {sign, ?WDRAW_CREATED, HSCTx}} = D) ->
    NewSignedTx = aetx_sign:add_signatures(
                    HSCTx, aetx_sign:signatures(SignedTx)),
    D1 = send_withdraw_signed_msg(NewSignedTx, D),
    {ok, D2} = start_min_depth_watcher(?WATCH_WDRAW, NewSignedTx, D1),
    next_state(awaiting_locked, D2);
```

Pattern-match asserting that we got the event we were waiting for

Valid events, but should not be handled here

Invalid events (for now) handled by default

```
awaiting_locked(cast, {?MIN_DEPTH_ACHIEVED, ChainId, ?WATCH_WDRAW, TxHash},
                #data{on_chain_id = ChainId,
                      latest = {watch, ?WATCH_WDRAW, TxHash, SignedTx}} = D) ->
    report(info, own_withdraw_locked, D),
    next_state(
      wdraw_signed, send_withdraw_locked_msg(
                      TxHash,
                      D#data{latest = {withdraw, SignedTx}}));
awaiting_locked(cast, {?FND_LOCKED, _Msg}, D) ->
    postpone(D);
awaiting_locked(cast, {?DEP_LOCKED, _Msg}, D) ->
    postpone(D);
awaiting_locked(cast, {?WDRAW_LOCKED, _Msg}, D) ->
    postpone(D);
awaiting_locked(cast, {?DISCONNECT, _Msg}, D) ->
    close(disconnect, D);
awaiting_locked(timeout, awaiting_locked = T, D) ->
    close({timeout, T}, D).
```

In summary

- Not always easy to say *why* a language is initially chosen
- Languages (esp. *opinionated* ones) shape your thinking
- Erlang well suited to blockchain development
  - Brilliant for state channel programming!
- The `gen_statem` behavior is an excellent addition to OTP

Æternity epoch Dependencies

- OTP components used
  - Mnesia (DBMS)
  - ssl, inets, asn1 (comms)
  - runtime_tools (tracing)
- Æternity core apps
  - Core svcs, mining, chain, txs, ...
  - HTTP-, Websocket API, Gossip
  - Smart Contracts, AEVM
  - Naming Service
  - Oracles
- External components
  - Cuckoo cycle (C++, own wrapper)
  - RocksDb (mnesia backend)
  - Exometer (metrics)
  - Cowboy (web server)
  - Jsx, yamerl, base58, msgpack
  - Jesse (JSON-Schema validation)
  - IDNA
  - enacl, sha3
  - gproc, jobs, lager, poolboy, ...

Build and Test

- Rebar3 for build (works so-so)
- EUnit, Common Test for test automation
- Dialyzer type analysis
- Quviq QuickCheck models

- Python-based acceptance test suite