goto; copenhagen

GOTO Copenhagen 2019
Conference Nov. 18 - 20

Building a blockchain in Erlang

Ulf Wiger, Aeternity

Follow us @gotocph

goto; copenhagen

GOTO Copenhagen 2019
Conference Nov. 18 - 20

Click ‘Rate Session’ to rate session and ask questions.

Follow us @gotocph

Ulf Wiger

- 1989-95 Command & Control, Disaster Response, Alaska, USA
- Erlang since 1992
- 1996-2008 Telecoms, Ericsson
- 2008-11 CTO Erlang Solutions
- 2011-present Entrepreneur, freelance consultant
- 2017-present Aeternity Blockchain Core Team

- (Also: professional opera singer)

https://github.com/uwiger

- Gproc (registry)
- Jobs (load regulation)
- Exometer (metrics)
- Locks (deadlock detection)
- Unsplit (netsplit resolver)

OTP Contribs

- XMErl 😬
- Application start phases
- Mnesia majority flag
- Mnesia backend & index plugins

Show of Hands

How many of you have programmed in Erlang?

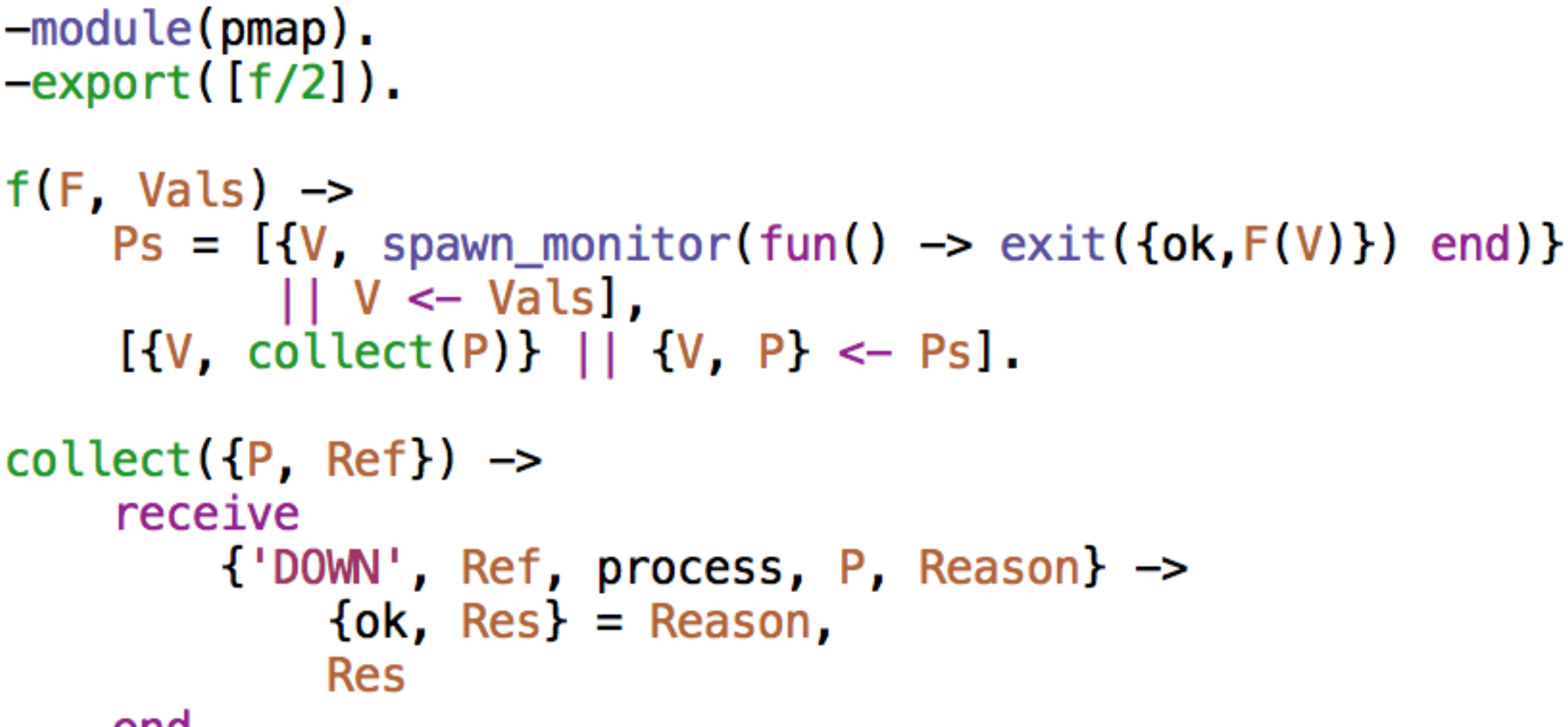
Show of Hands

How many of you are familiar with blockchains?

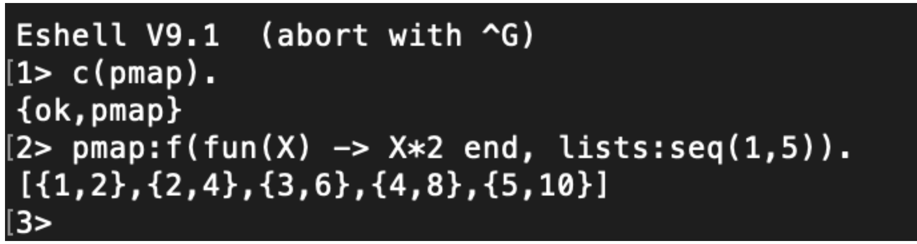
Erlang Primer

- Functional (mostly)
- Dynamically typed
- Garbage-collected
- Concurrent
- Fault-tolerant
- Pesky punctuation

- Opinionated



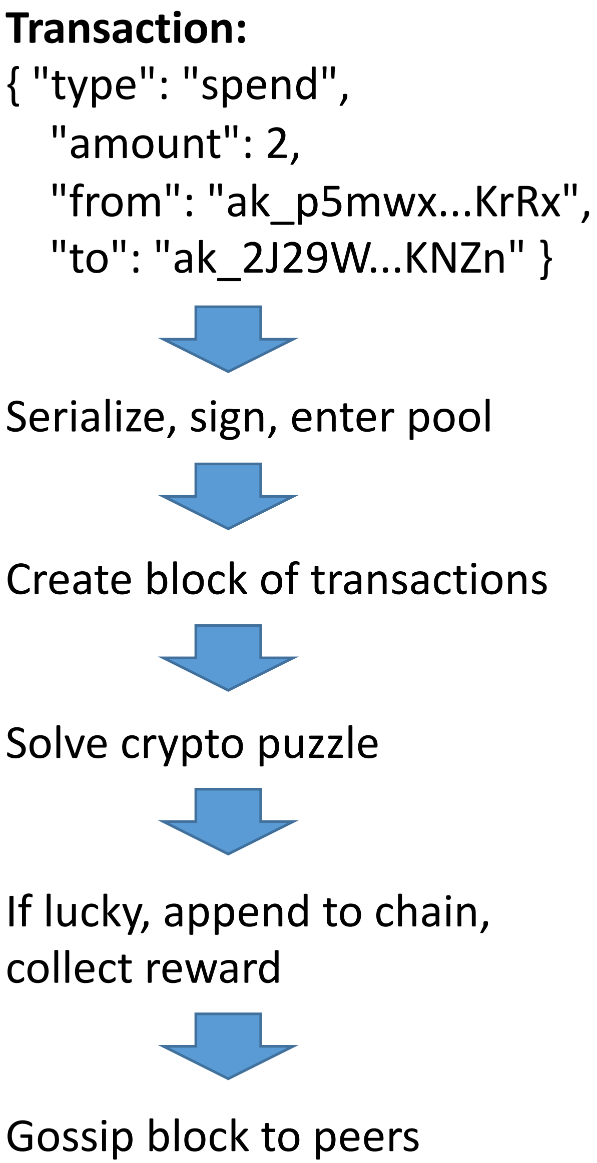
```
-module(pmap).
-export([f/2]).

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.
```



```
Eshell V9.1  (abort with ^G)
[1> c(pmap).
{ok,pmap}
[2> pmap:f(fun(X) -> X*2 end, lists:seq(1,5)).
[{1,2},{2,4},{3,6},{4,8},{5,10}]
[3>
```

Blockchain Primer

- World's slowest append-only DB tech
- **No-trust**
- Peer-to-peer
- Heavy reliance on crypto proofs



Transaction:

```
{ "type": "spend",
  "amount": 2,
  "from": "ak_p5mwx...KrRx",
  "to": "ak_2J29W...KNZn" }
```

Serialize, sign, enter pool

Create block of transactions

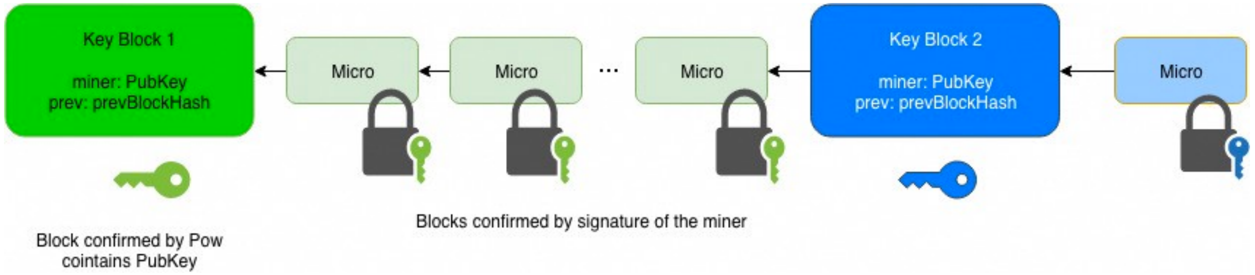
Solve crypto puzzle

If lucky, append to chain,
collect reward

Gossip block to peers

The Æternity Blockchain

- Standard Proof-of-Work model (Cuckoo Cycle)
- Bitcoin-NG consensus



- New Smart Contract Language (Sophia)
- Interesting use cases as first-class objects
    - State Channels
    - Oracles (ports to the outside world)
    - Naming System
    - Generalized Accounts (pluggable authentication methods)

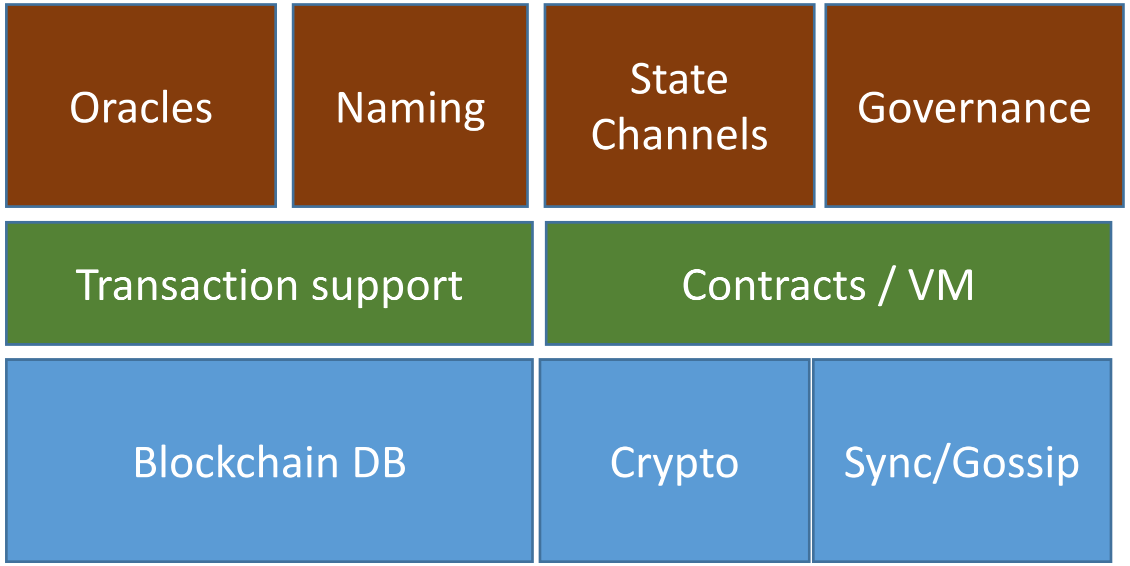
https://medium.com/aeternity-crypto-foundation/aeternity-bitcoin-ng-the-way-it-was-meant-to-be-df7bb1d65a4b

Performance aspects of 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



Oracles

Naming

State Channels

Governance

Transaction support

Contracts / VM

Blockchain DB

Crypto

Sync/Gossip

How Does Erlang Help? (2)

- 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? (3)

- Functional Programming
  - Simplifies testing
  - Code, once correct, tends to *stay* correct
  - Reduces surprising side-effects
  - Powerful for blockchain state management

- Erlang doesn't enforce purity
  - Pragmatism + culture
  - Ubiquitous design patterns, manifested as 'behaviors'

How Does Erlang Help? (4)

- Carrier-Class Product Mentality
  - Stellar backward compatibility
  - Rock-solid VM
  - No "dependency hell"
  - Basically 'attack-proof' networking support
  - Community culture

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
    - ... Except regarding State Channels

Æternity 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,
  - Jesse (JSON-Schema validation)
  - IDNA
  - enacl, sha3
  - gproc, jobs, lager, poolboy, ...

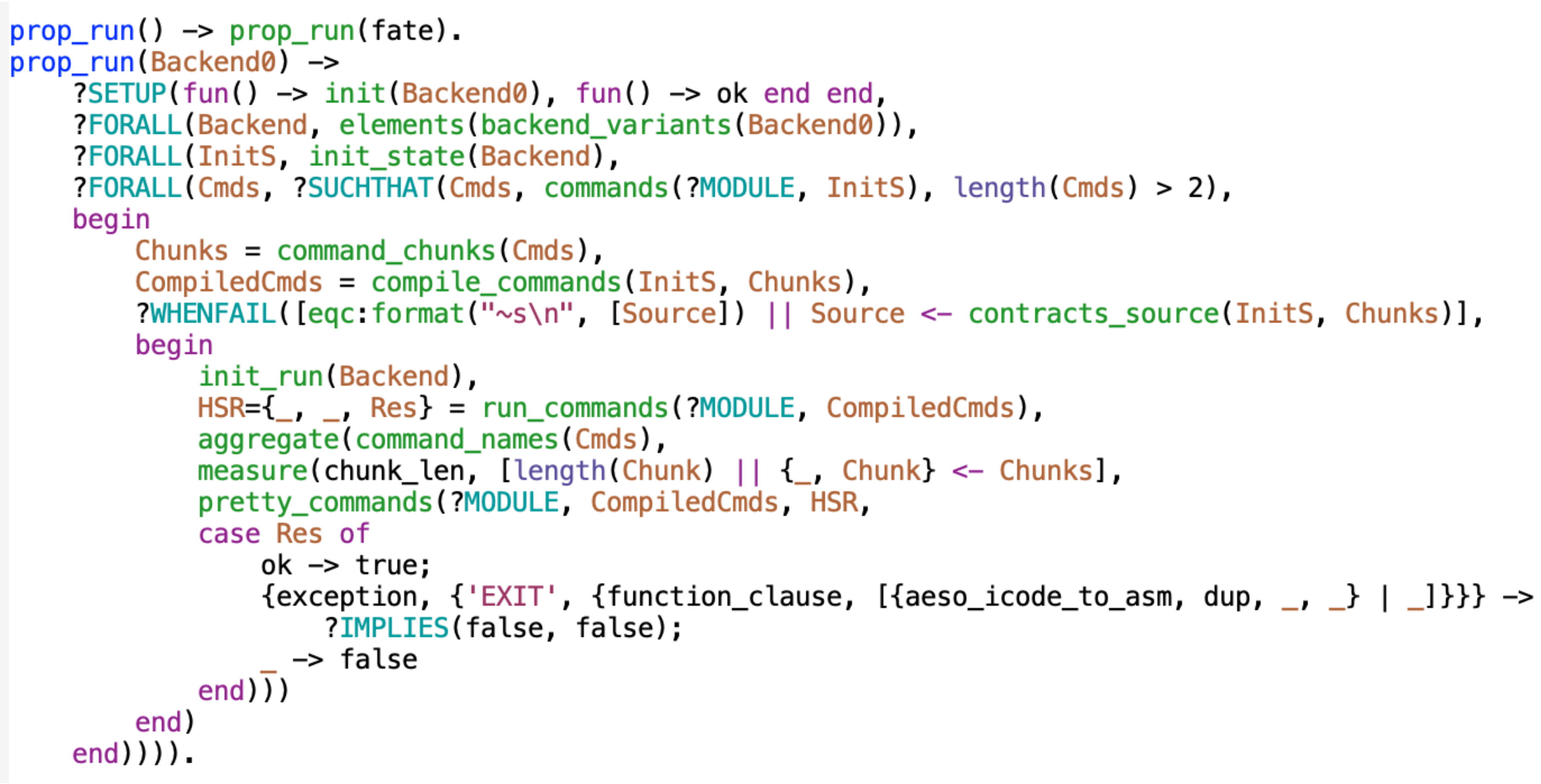
Build and Test

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

- Python-based acceptance test suite

QuickCheck – Testing on steroids

- Controlled random test case generation



```
prop_run() -> prop_run(fate).
prop_run(Backend0) ->
    ?SETUP(fun() -> init(Backend0), fun() -> ok end end,
    ?FORALL(Backend, elements(backend_variants(Backend0)),
    ?FORALL(InitS, init_state(Backend),
    ?FORALL(Cmds, ?SUCHTHAT(Cmds, commands(?MODULE, InitS), length(Cmds) > 2),
    begin
        Chunks = command_chunks(Cmds),
        CompiledCmds = compile_commands(InitS, Chunks),
        ?WHENFAIL([eqc:format("~s\n", [Source]) || Source <- contracts_source(InitS, Chunks)],
        begin
            init_run(Backend),
            HSR={_, _, Res} = run_commands(?MODULE, CompiledCmds),
            aggregate(command_names(Cmds),
            measure(chunk_len, [length(Chunk) || {_, Chunk} <- Chunks],
            pretty_commands(?MODULE, CompiledCmds, HSR,
            case Res of
                ok -> true;
                {exception, {'EXIT', {function_clause, [{aeso_icode_to_asm, dup, _, _} | _]}}} ->
                    ?IMPLIES(false, false);
                _ -> false
            end)))
        end)
    end)))).
```

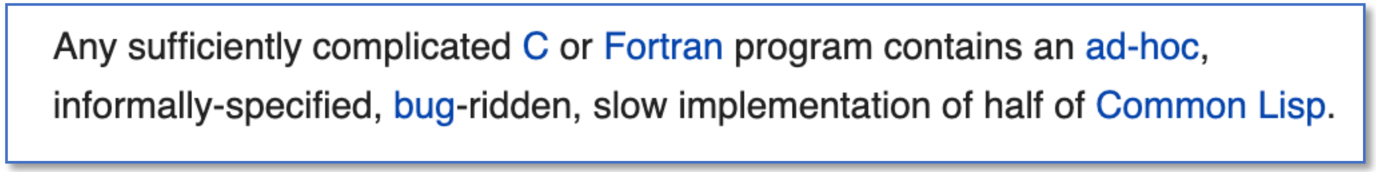
https://github.com/Quviq/epoch-eqc

Fast æternity Transaction Engine (FATE)

- Virtual machine for the Sophia contract language
- Implemented in Erlang (!)
- 1st VM (AEVM) a version of the Ethereum VM
  - Typical low-level byte-code VM
- FATE is a **high-level** byte-code VM
  - 90% reduction in byte code size

But high-level languages are slooow!

- For complex problems, this is not always true
- Greenspun's Tenth Rule



Any sufficiently complicated C or Fortran program contains an ad-hoc, informally-specified, bug-ridden, slow implementation of half of Common Lisp.

- A VM in Erlang will do poorly at low-level evaluation
- But lots of things are already there
  - Isolation
  - Memory management + GC
  - Efficient data structures
- If you're already using Erlang, it makes sense

https://en.wikipedia.org/wiki/Greenspun%27s_tenth_rule

State channels in Erlang

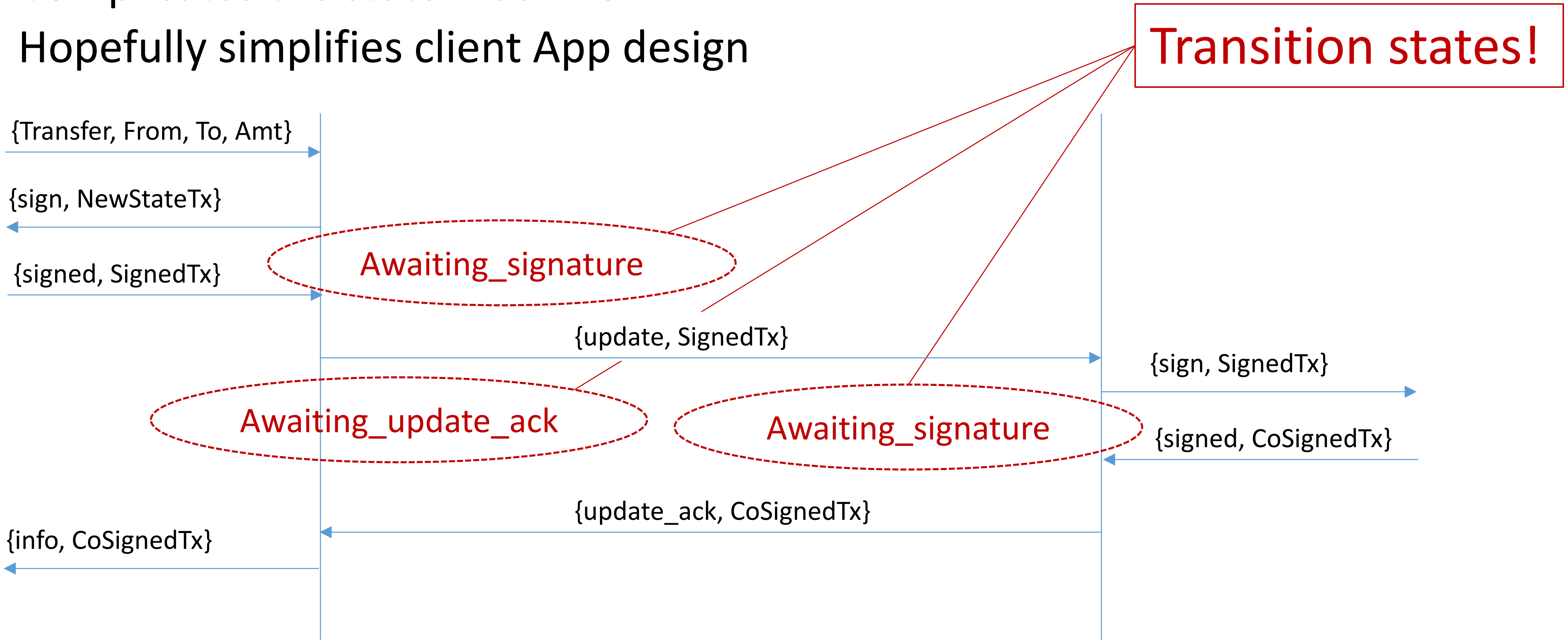
- Purpose: Establish "off-chain" channels for fast and cheap transactions
    - On-chain activity only when opening and closing channel
    - Funds locked into the channel can be transferred in co-signed transactions "for free"
    - "Trust but verify" off-chain,
      Mutual close or dispute resolution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

State Channels: Surprisingly complex

- No-trust means everything must be verified
- Be prepared for malicious counterpart
- On-chain dispute protocols
- Channel may be subverted on-chain
- Off-chain contracts may refer to objects on-chain
- Chain may 'fork' – essentially a roll-back
- Normal comms error scenarios

Ónen i-Estel Edain. Ú-chebin Estel anim

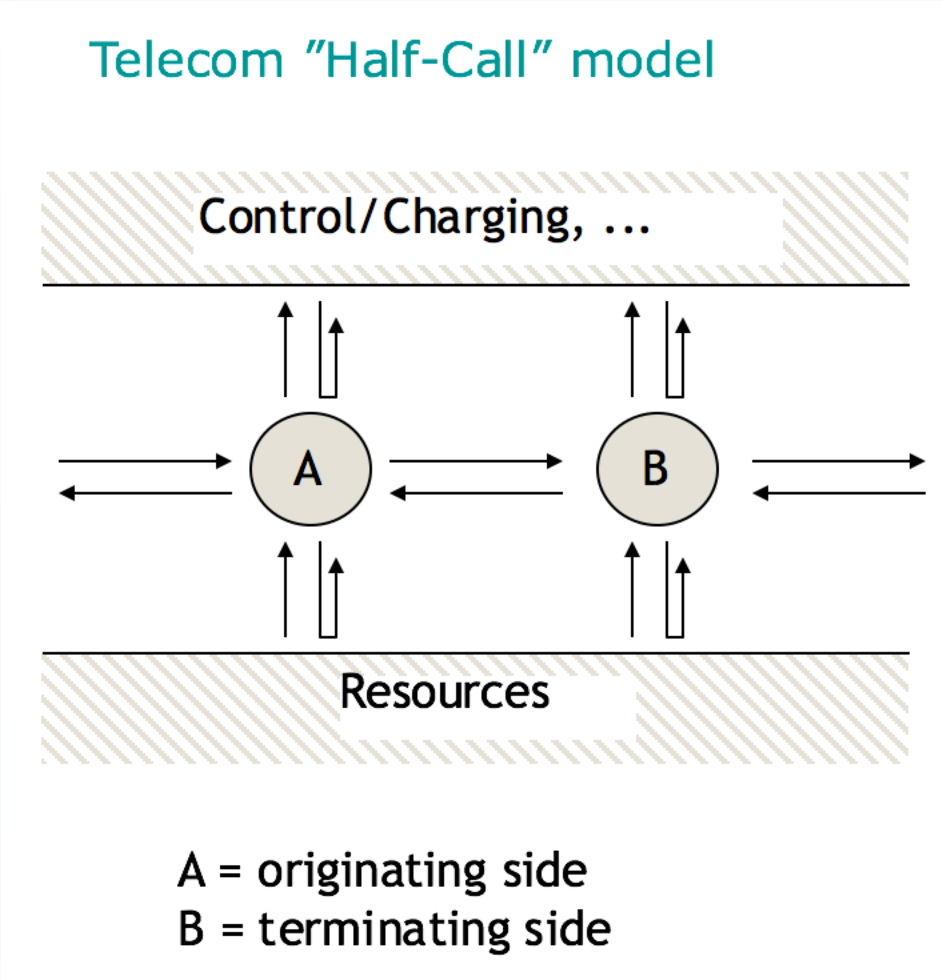
- Design decision: SC daemon with a simplified WebSocket API
  - Complicates the state machine
  - Hopefully simplifies client App design



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

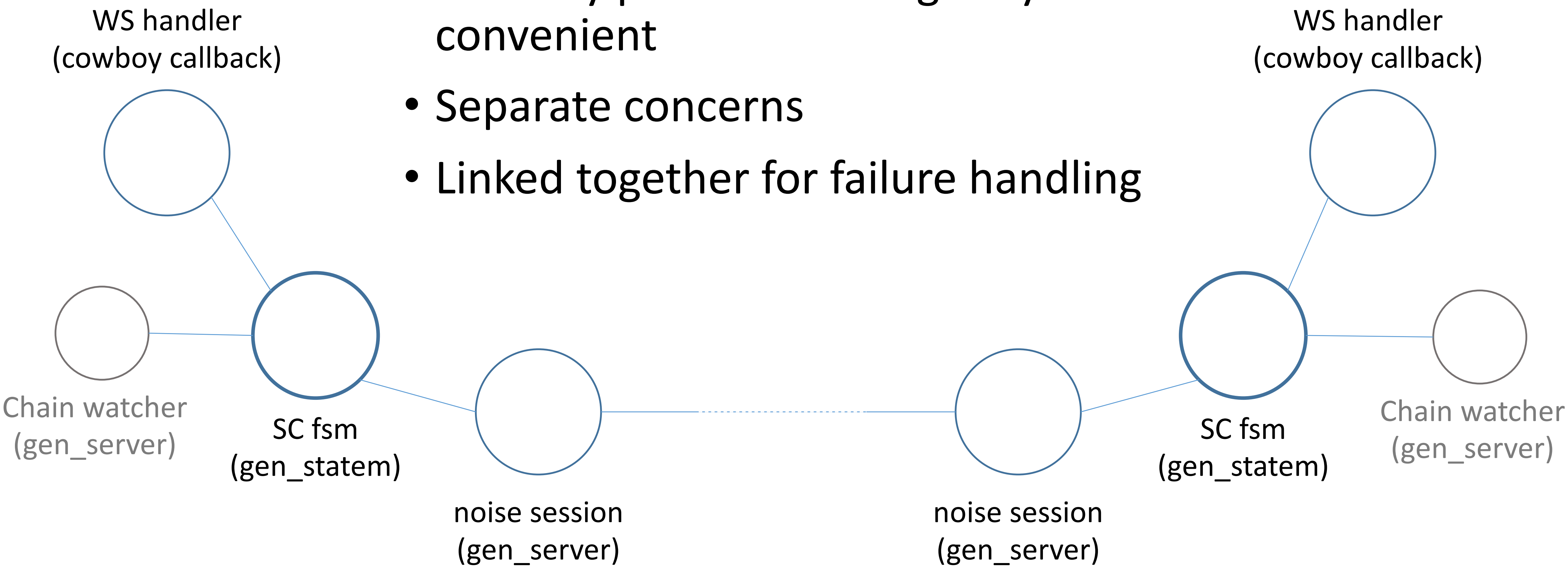


Telecom "Half-Call" model

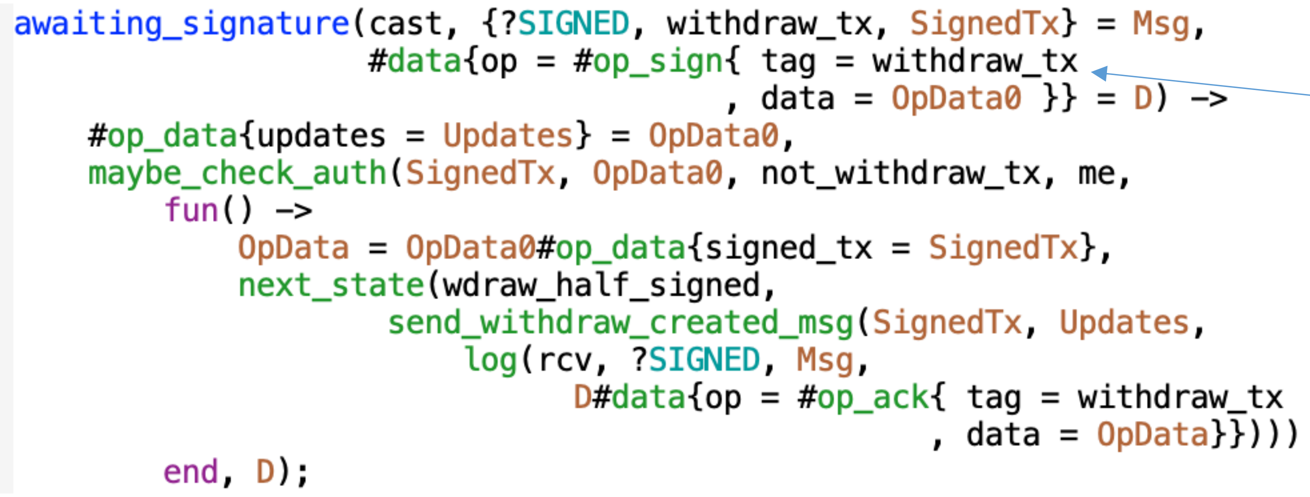
A = originating side
B = terminating side

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`



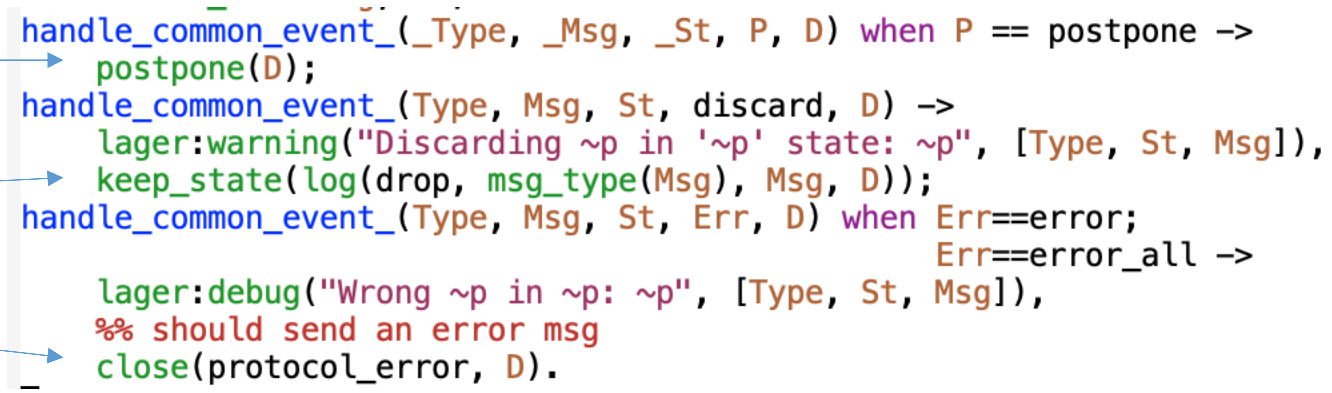
```erlang
awaiting_signature(cast, {?SIGNED, withdraw_tx, SignedTx} = Msg,
                   #data{op = #op_sign{ tag = withdraw_tx
                                      , data = OpData0 }} = D) ->
    #op_data{updates = Updates} = OpData0,
    maybe_check_auth(SignedTx, OpData0, not_withdraw_tx, me,
        fun() ->
            OpData = OpData0#op_data{signed_tx = SignedTx},
            next_state(wdraw_half_signed,
                send_withdraw_created_msg(SignedTx, Updates,
                    log(rcv, ?SIGNED, Msg,
                        D#data{op = #op_ack{ tag = withdraw_tx
                                           , data = OpData}})))
        end, D);
```

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

Valid events, but should not be handled here

Unknown or stray events, safe to discard

Protocol violations



```erlang
handle_common_event_(_Type, _Msg, _St, P, D) when P == postpone ->
    postpone(D);
handle_common_event_(Type, Msg, St, discard, D) ->
    lager:warning("Discarding ~p in '~p' state: ~p", [Type, St, Msg]),
    keep_state(log(drop, msg_type(Msg), Msg, D));
handle_common_event_(Type, Msg, St, Err, D) when Err==error;
                                                  Err==error_all ->
    lager:debug("Wrong ~p in ~p: ~p", [Type, St, Msg]),
    %% should send an error msg
    close(protocol_error, D).
```

In summary

- Blockchain tech is a moving target
- Loosely coupled components
- Correctness is key
- A few performance-critical components written in C
- Erlang well suited to blockchain development
  - Brilliant for state channel programming!

goto; copenhagen

GOTO Copenhagen 2019
Conference Nov. 18 - 20

Please

Remember to rate this session

Thank you!

Follow us @gotocph

goto; copenhagen

GOTO Copenhagen 2019
Conference Nov. 18 - 20

Did you **remember to rate** the previous session ?

Follow us @gotocph