Erlang in the Browser

Fredrik Svahn

Outline

- What is “Erlang in the browser”?
- Why?
- Behind the scenes
- Writing your own emulator - experiences
- Current status & Future plans

What is it?

- An Erlang virtual machine in javascript
- A subset of the functionality in BEAM
- Based on reverse engineering
- Understands the BEAM file format
- Executes BEAM opcodes like the normal Erlang VM
- API for javascript functions and accessing the DOM
- Can connect/communicate with a “normal” node
- Example Erlang in browser demo on github

Why?

- To learn and make it easier for others to learn
- Javascript is everywhere
- Running Erlang programs without installing
- Test the limits of javascript in browsers

- Because it seemed like a funny challenge...

Behind the scenes

- The beam file loader
- The emulator main loop
- Built in functions, BIF:s
- Distribution

The BEAM file loader

- Loads individual beam files or tar files
- Uses sourced in zlib compression library
- Uses Typed Arrays with fallback to strings
- Loading beam files is time consuming in older browsers – optimize?
- Cannot do hot code loading (at the moment)

The Emulator main loop

- Large loop over a switch statement, process yields when runs out of reductions
- Handles some 120 opcodes
- Most opcodes 3-4 lines javascript
- Mostly quite simple to reverse engineer
- Some opcodes seem to be obsolete?

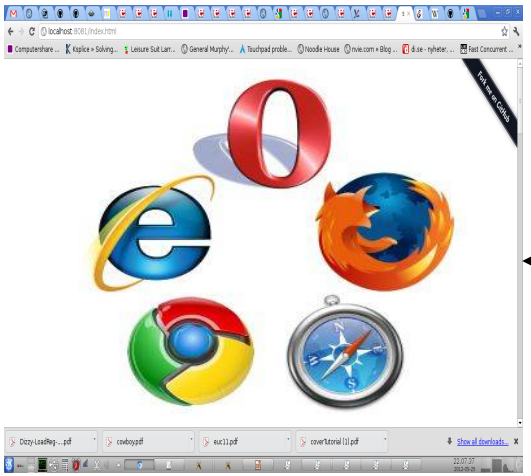
Built in functions – BIF:s

- Lots of work – mostly validating function arguments
- Some BIF stubs/hacks needed to run a full system
- Some BIF:s will never work due to browser limitations

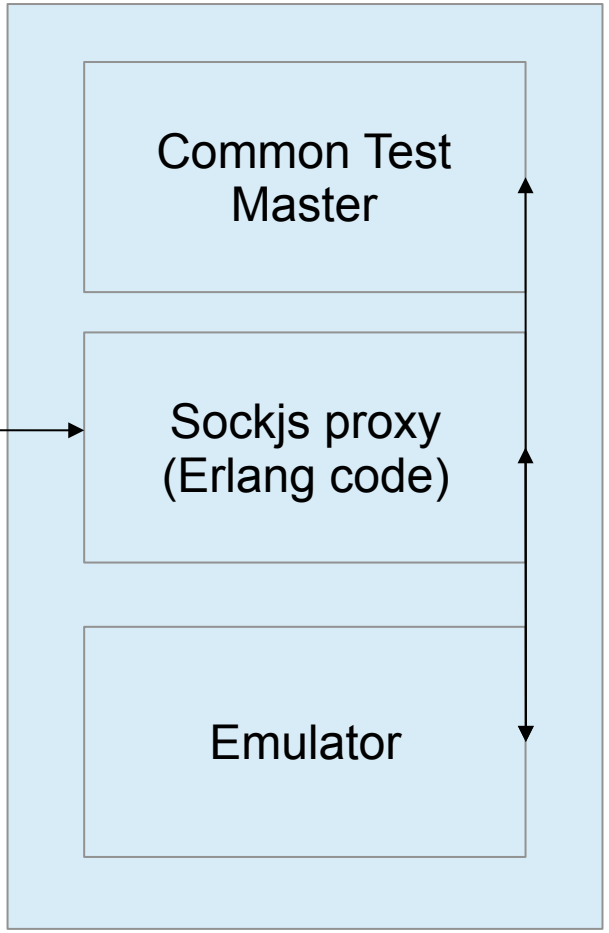
Distribution mechanism

- Using sockjs to connect to server
- Messages encoded in the external term format
- Server uses proxy (similar to the ssl_dist proxy)
- Makes it possible to run common test master node logging and collecting statistics on server
- Example code available in tester.erl
- Client browser cannot communicate directly with other browsers, only server (same origin policy)

Test Setup using Distribution



Common Test Client



Experiences

- BEAM opcodes are beautiful in their simplicity
- Implementing BIF:s is a lot of work
- Error handling is hard
- Bitstrings ops are tricky
- The OTP test server is great!
- Javascript engines differ – a lot

Characteristics

- Factor 10-20 slower than BEAM on same HW
- BIF dispatch significantly slower
- Lists use more memory
- Garbage collection on system level
- Timers likely to be less exact
- erlang:now()

“Near” future plans

- Bitstrings
- ETS – a huge undertaking
- Fully compliant to Erlang External Term format
- Improving the Pass Rate for OTP test suites
- Modularize
  - Make it possible to plug in a more efficient loader
  - Split out ETS and “file driver”
- Make support and using some minifier/js code optimizer
- More efficient BIF calls
- Make it more object oriented

Future plans – AKA wild ideas

- Compiling Erlang to directly to javascript for hot functions
- Local persistent storage using HTML5 localStorage()
- Multithreading using HTML5 web workers
- Distribution proxy
- Quickcheck style testing

Show me the code...

https://github.com/svahne/browserl

Demo:

http://svahne.github.com/browserl/

Contributing?

- Please contribute (or fork or write your own)!

- Contributions must be made available under MIT and GPL licenses
- Contributions will generally be rejected if they
  - Make the emu slower
  - Decrease the pass rate

Thanks for your time!

Questions?