Lux - synchronised testing of concurrent sessions

Håkan Mattsson, Cons T Åhs

Lux

- Test tool (written in Erlang - that’s why we’re here)
- System level testing
- A test script is a sequence of input and expected output (regexp)
- Allows for scenarios with multiple concurrent sessions
- Each session is a “shell” — anything taking textual commands and producing textual output

Scope

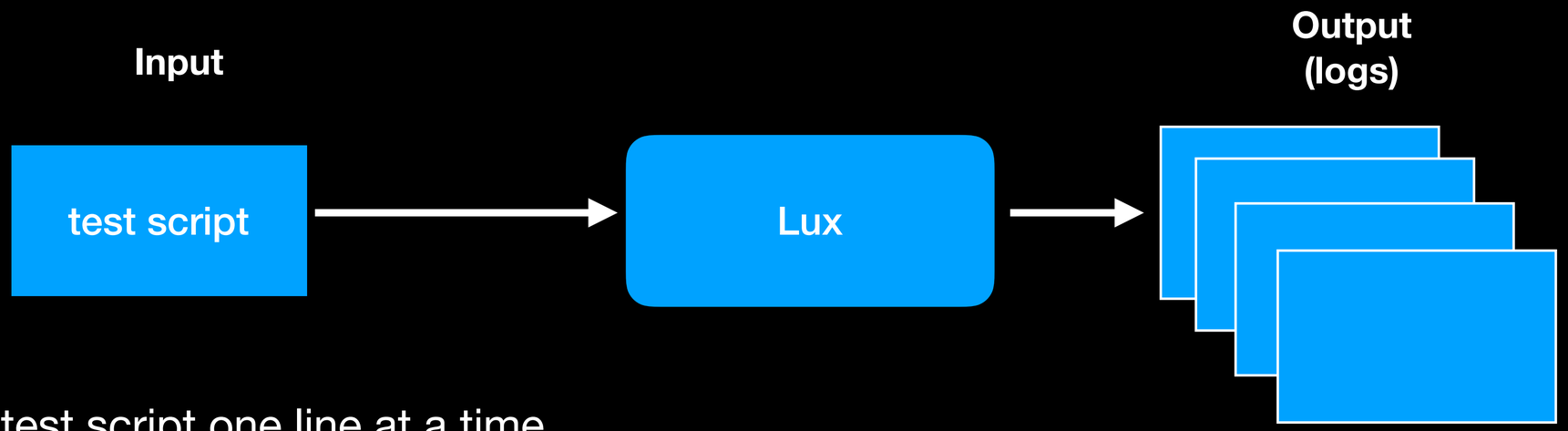
- Good fit for:
  - system level tests, text based interaction
  - simple, regular expression based output
  - orchestration of concurrent sessions
- Bad fit for:
  - GUI
  - structured output that cannot be handled by regexp

Simple example

```
[doc Demo a single shell]

# Start a shell
[shell server]
    # Send text to the active shell
    !erl -pa ../../../chatty/ebin

    # Match output from the active shell
    ?Erlang/OTP.*
    ?Eshell.*
    ?1>

    !chatty:server().
    ?Starting server
    ?2>

    !halt(3).
    ?SH-PROMPT:

    !echo "===$?==="
    ?===3===
    ?SH-PROMPT:
```

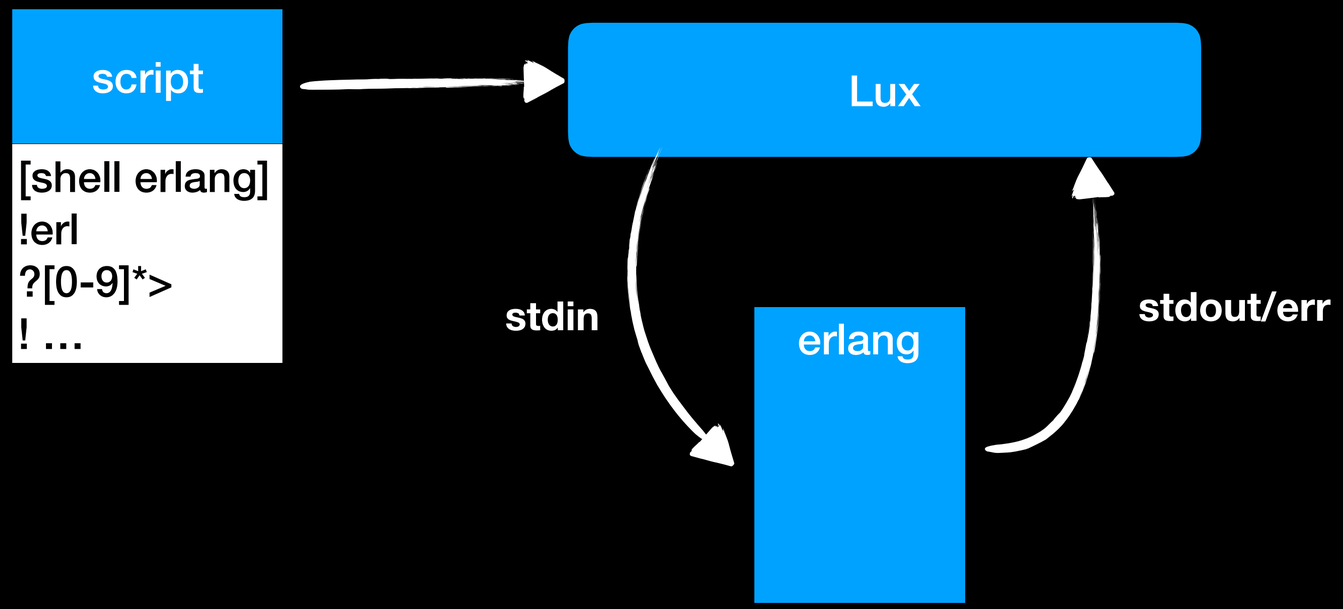
- Sequential session with a series of sending input and matching output
- Matching serves both as verifying output and synchronisation, i.e., don’t feed new input until at a known state
- Possible to have both several lines of sending or matching in sequence

The world according to Lux



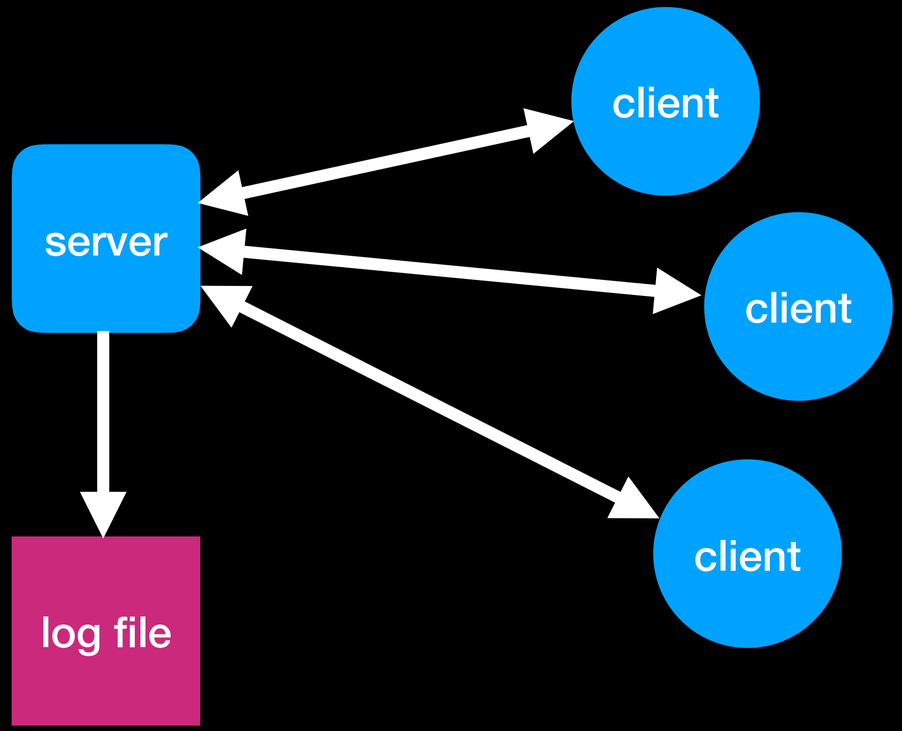
- Execute test script one line at a time
  - !.. — Feed input to shell
  - ?.. — Wait for successful match before proceeding
  - Fail if no match found
- Record what's happening in log files

One shell view



- A match is successful when the regexp is found in the output
- Future matching is done from the point after the last match
- Not all output needs to be matched
- A match fails when matching output has not been found within some specified time

Example: chat server and clients



Example: chat server and clients

```
(9) chatty > erl -pa ebi
Starting server mytopic
Trying to open log file
```

```
(5) chatty > erl -pa ebin -sname cons -noshell -s chatty client mytopic
Trying to join the mytopic chat room...
Welcome to the chat room mytopic!!!
Enter text and press enter. Exit chat with ^d.
cons>
hawk: Client joined.
cons> ping
cons>
hawk: pong
cons>
```

```
(3) chatty > erl -pa ebin -sname hawk -noshell -s chatty client mytopic
Trying to join the mytopic chat room...
Welcome to the chat room mytopic!!!
Enter text and press enter. Exit chat with ^d.
hawk>
cons: ping
hawk> pong
hawk>
```

When in doubt: automate!

```
[shell server]
    !erl -pa $ebin -sname $topic -s chatty server
    ?Starting server

[shell cons]
    !erl -pa $ebin -sname cons -noshell -s chatty client $topic
    ?cons>

[shell hawk]
    !erl -pa $ebin -sname hawk -noshell -s chatty client $topic
    ?hawk>

[shell cons]
    ?hawk: Client joined
    !ping

[shell hawk]
    ?cons: ping
    !pong

[shell cons]
    ?hawk: pong
```

- One Lux shell for each terminal
- Mimic the sequence of interactions from the manual session

Lux is faster than you

```
test case          : naive.lux
progress           : ..:..:.:..:..:.:..:.:.:....:.:..:.:..:..:.:..:.:.:.:.:.:.:.:.:.
13????13..
result             : FAIL at 13 in shell cons
expected*
        cons>
actual match_timeout
        erl -pa ../../../chatty/ebin -sname cons -noshell -s chatty client mytopic
        Trying to join the mytopic chat room...
        <ERROR> Failed to join 'mytopic@CAHS-M-K1Y7'. Is the server started?
        {"init terminating in do_boot",shutdown}
        init terminating in do_boot (shutdown)
```

expected prompt

Multi shell execution

```
[shell server]
    !erl -pa $ebin -sname $topic -s chatty server
    ?Starting server

[shell cons]
    !erl -pa $ebin -sname cons -noshell -s chatty c
    ?cons>

[shell hawk]
    !erl -pa $ebin -sname hawk -noshell -s chatty c
    ?hawk>

[shell cons]
    ?hawk: Client joined
    !ping

[shell hawk]
    ?cons: ping
    !pong

[shell cons]
    ?hawk: pong
```

- The shell command switches the shell being used
- A shell not ending with a match will just proceed to the next shell (and the program kept running)
- We need the possibility to know when the server has been started before proceeding
- Matching serves as synchronisation

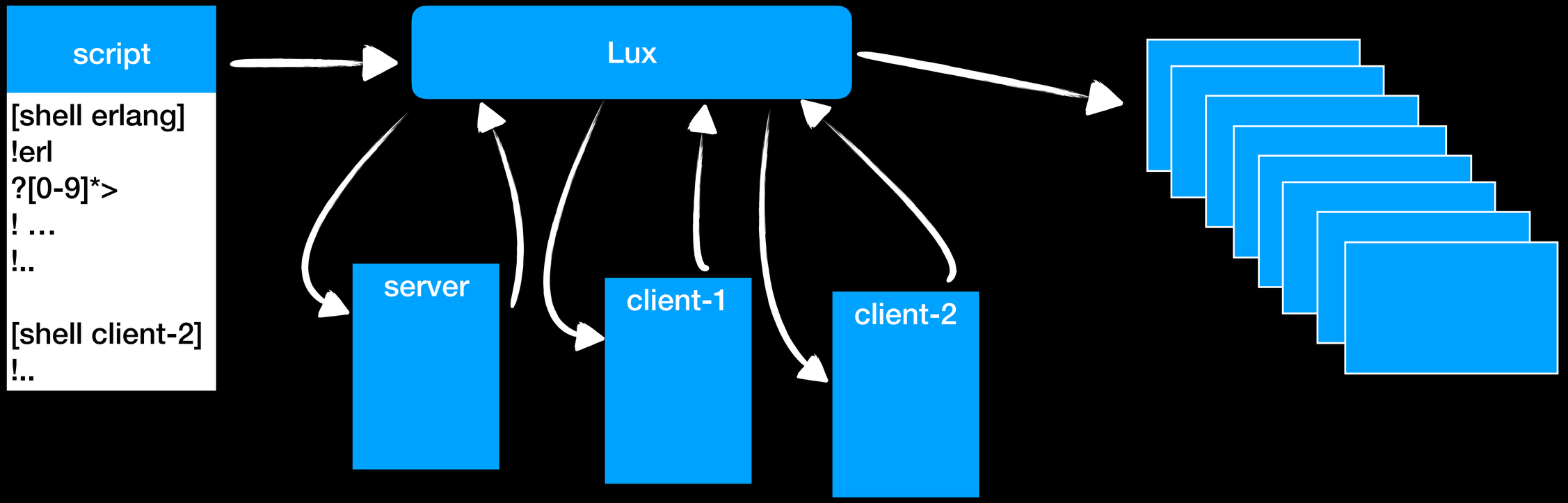
Proper synchronisation

bookkeeping

```
[shell server]
    !rm -f chatty_mytopic.log
    ?SH-PROMPT
    !erl -pa $ebin -sname $topic -s chatty server
    ?Starting server

[shell logger]
    !tail -F chatty_mytopic.log
    ?Server started

[shell cons]
    !erl -pa $ebin -sname cons -noshell -s chatty client $topic
    ?cons>
```

wait for server to start

Multiple shells view



Test cases lead to..

- Automating manual tests, running them and see them fail will give us a number of test cases after a while. Put each in its own file.
  - start client with no server — client-start.lux
  - start server and clients, verify connection — server-start.lux
  - test interaction — interact.lux
  - ..

..Test Suites

- Lux can run a number of test cases in a test suite
  - results are collected
  - normal test case hygiene needed to not have interacting test cases
    - don’t use data from other test cases
    - don’t overwrite results from previous test (save them before it happens)

When tests fail

- A test case will fail when a required match is not seen
- Reported as a match timeout as the timer expired
- Lux will show the line of failure (with a “call stack” if needed)
- Use the logs!

The truth is in the logs..

- Running a Lux script `simple.lux` with a single shell called `server` will by default produce 12 log files
  - simple.lux.server.stdout.log — bytes received from stdout (and stderr) of the shell named server
  - simple.lux.server.stdin.log — bytes sent to stdin of the shell named server
  - simple.lux.event.log — trace of internal Lux events
    - simple.lux.event.log.html — pretty printed event log with links to other logs
    - simple.lux.event.log.csv — low level info about actual duration of timers

The truth is in the logs..

- simple.lux.config.log - test case specific configuration
- simple.lux.orig — the test script itself
- lux_config.log —general configuration for the entire test suite
- lux_summary.log — summary of the outcome of the test (suite)
  - lux_summary.log.html — same in html
- lux.tap — summary log on TAP format

Matching in different flavours

- use regular expressions for matching
- ? - with regexps and variable expansion
- ?? - without regexps, with variable expansion
- ??? - verbatim, no regexps, no variable expansion
- “““ — start multiline match

Extract sub patterns

```
(9) chatty > erl -pa ebin -noshell -sname mytopic -s chatty server
Starting server mytopic …
Trying to open log file chatty_mytopic.log...ok.
```

```
[shell server]
    !erl -pa $ebin -sname $topic -s chatty server
    ?Starting server

    # Match sub-expressions
    ?Trying to open log file (.*)\.\.\.ok.
    [global logfile=$1]

# Start another shell
[shell server-log]
    # Match in log file
    !tail -F $logfile
    ?Server started
```

Fail patterns

- Ordinary matching is positive, i.e., we want to verify that we see certain patterns
- There might also be strings that we don’t want to see and fail directly when we do
- A fail pattern can be defined for each shell
- If the fail pattern is seen, the test case fails

```
[global fail_pattern=[Ee][Rr][Rr][Oo][Rr]]

[shell eshell]
    -$fail_pattern|SH-PROMPT:
    !erl -s badmod
    ?Erlang/OTP
    ?1>
    !goodmod:start().
    ?2>
```

Variables

- Different scope
  - local (for shell), global (for all shells), macro local
  - matching against contents of variable makes it convenient for matching changing parts, such as prompts
  - numeric variable names are parts of last match

Macros

- Abstraction possible by use of macros with arguments
- Define a sequence of Lux commands executed in the context of the invoking shell
- Makes scripts compact, abstract and introduces reusability

```
[macro ok]
  !echo =$?=
  ?=0=
[endmacro]

[macro prompt user]
  ?^$user>
[endmacro]

[shell cons]
  [local me=cons]
  [invoke start-client $me]
  [invoke prompt $me]
    !$_CTRL_D_
    [invoke ok]
```

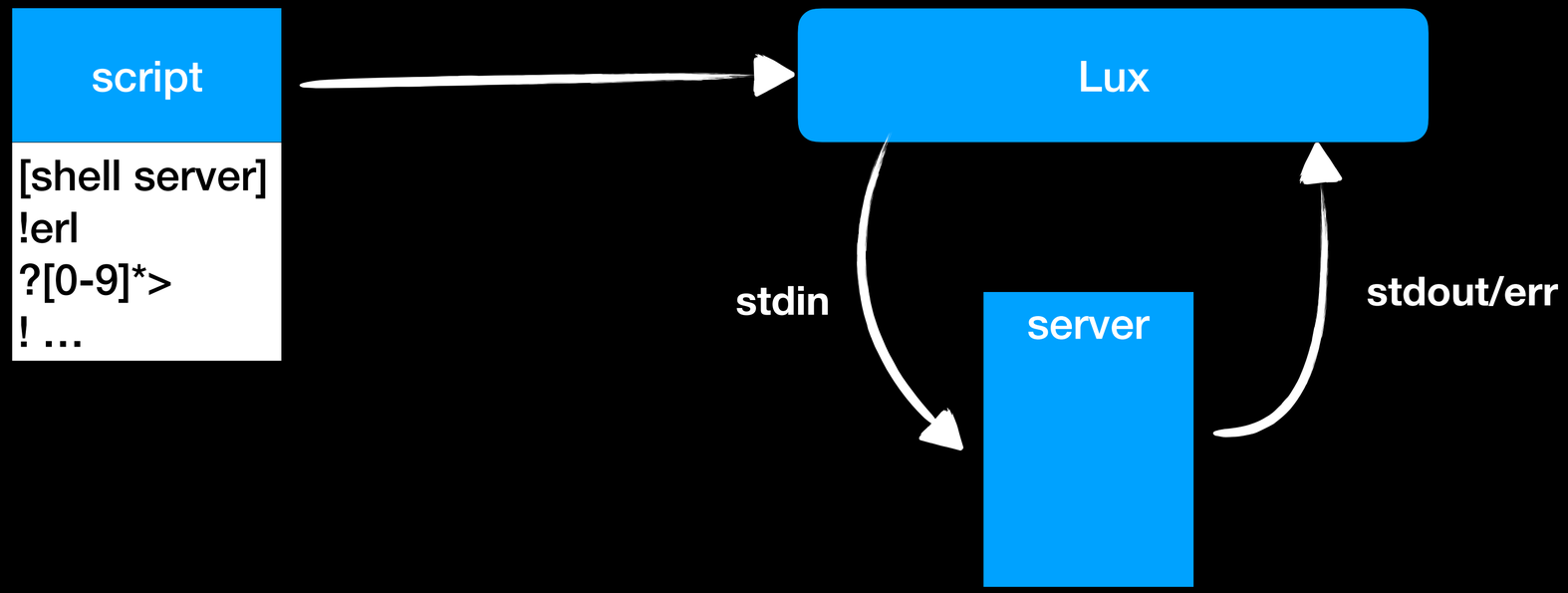
Additional Convenience

- Loops
  - allows for iterating over items, with the possibility to end as soon as a certain pattern has been seen
- Configurable timeout
- Special cleanup shell
  - always executed, regardless of whether the tests succeeds or fails
  - used to terminate running programs, saving logs, checking general conditions on exit etc

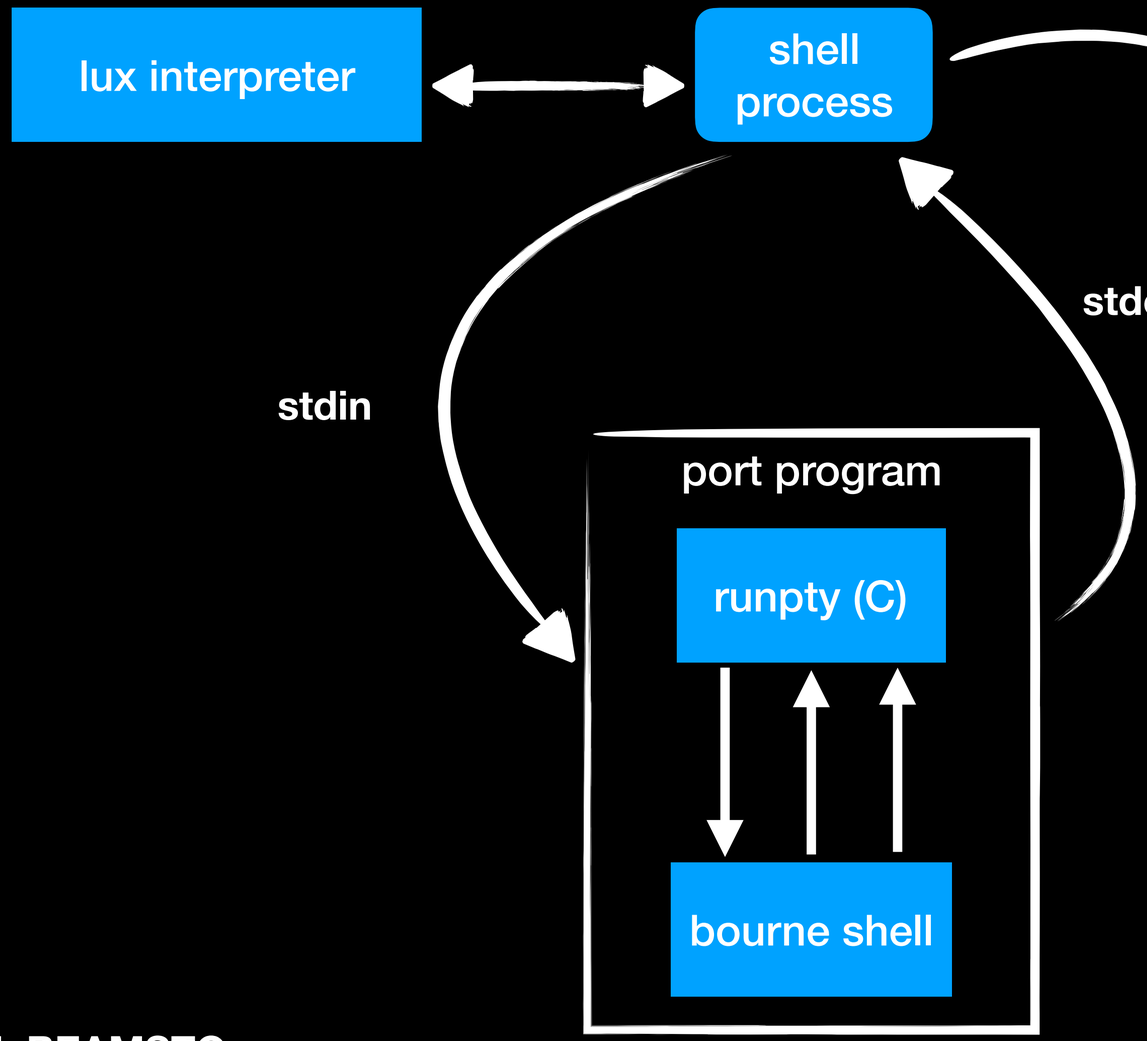
Implementation

- Lux is implemented (almost) entirely in Erlang
  - Concurrency - simple controlling of several shells
  - Port programs - simple running of external programs
  - Built in regular expression - Lux does a lot of matching..
  - Timers are simple

Implementation



Implementation



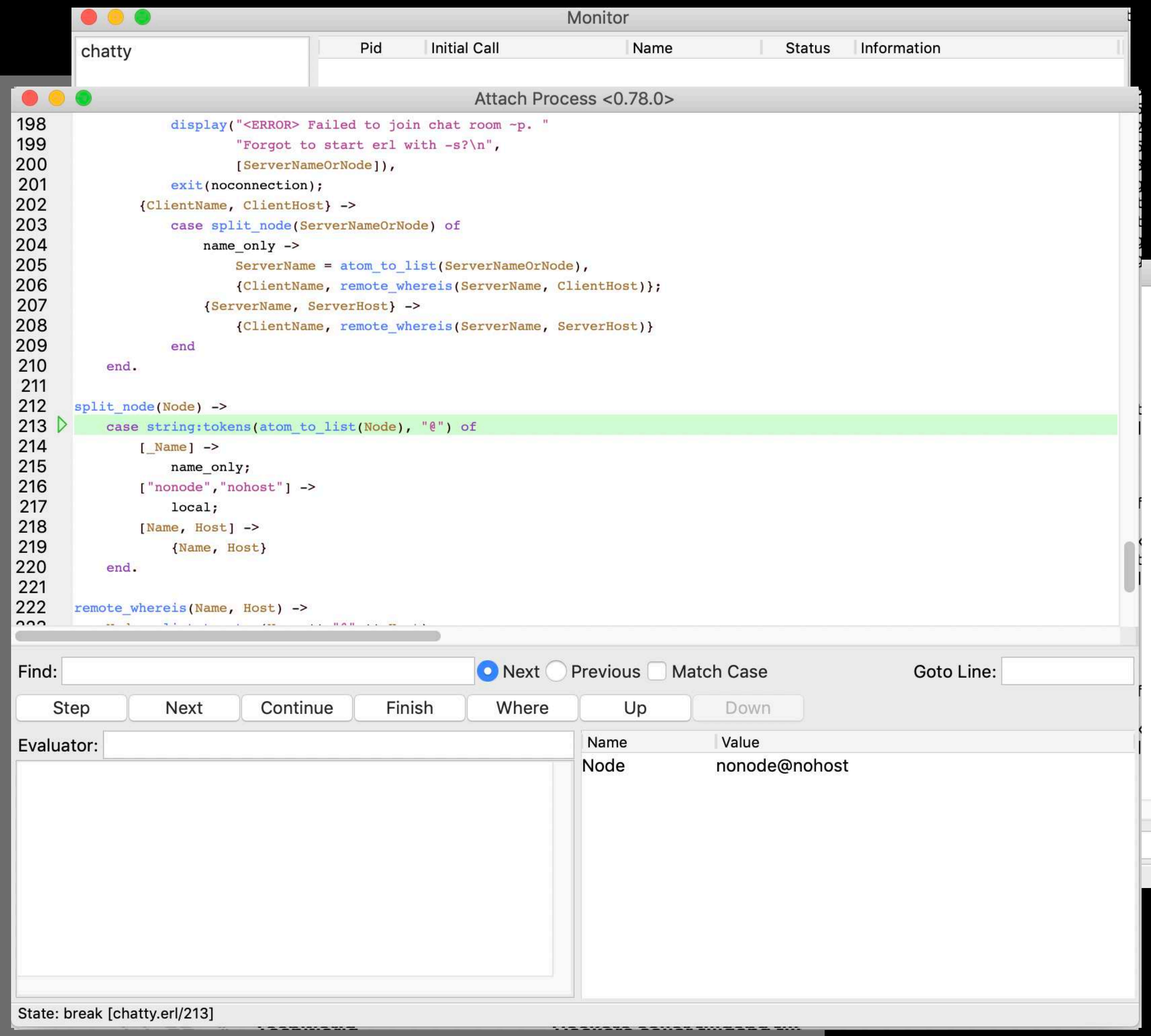
manipulate terminal settings
make bourne shell believe it is interactive
normalise output
echo input
redirect stderr to stdout

where the actual “shell” is run

Debugging test cases

- Interactive debugger with the possibility to
  - Run in verbose mode
  - Break at specified line numbers
  - Single step
  - Connect to and interact with a shell
  - View logs

Debugging test cases



 # Run the test case and stop before line 1:
> lux -d a_simple_server.lux

cisco

Infrastructure support

- Skip test cases on some or all architectures
- Mark test cases as unstable on some or all architectures
- Logs on TAP format can be generated
- Logs on JUnit format can be generated
- History of multiple test runs can be assembled and visualized

Visualisation

Lux history overview (jenkins_hist) generated at 2019-05-15 07:12:03.157097

1569 runs (0 errors) within this range of repository revisions

Latest: **190515.053503.6e587ffa4853** at 2019-05-15 2005-35-03
First: **190501.053500.e104fd9e0f5f** at 2019-05-01 2005-35-00

Legend

First fail	Secondary fails on same host	Warning	Skipped	Success	No data

Overview

Still failing test suites

Still failing test cases

Only one config. No config page generated.

Only one host. No host page generated.

Only one config. No latest run generated.

All runs

All test suites	190515.053503.6e587ffa4853 **190515.053503.6e587ffa4853** 2019-05-15 2005-35-03 origin/confd-5.4	190514.053500.be3eb3b4d107 **190514.053500.be3eb3b4d107** 2019-05-14 2005-35-00 origin/confd-5.4	190513.053500.be3eb3b4d107 **190513.053500.be3eb3b4d107** 2019-05-13 2005-35-00 origin/confd-5.4	190512.053503.be3eb3b4d107 **190512.053503.be3eb3b4d107** 2019-05-12 2005-35-03 origin/confd-5.4	190511.053503.be3eb3b4d107 **190511.053503.be3eb3b4d107** 2019-05-11 2005-35-03 origin/confd-5.4	190510.053503.be3eb3b4d107 **190510.053503.be3eb3b4d107** 2019-05-10 2005-35-03 origin/confd-5.4
	jslaveX x86_64_linux	**jslaveX** x86_64_linux	**jslaveX** x86_64_linux	**jslaveX** x86_64_linux	**jslaveX** x86_64_linux	**jslaveX** x86_64_linux
	11 (5401)	**16 (5401)**	**16 (5401)**	**15 (5155)**	**19 (5401)**	**18 (5401)**
test_core-cli-new-tr	0 (247)	1 (247)	1 (247)	1 (247)	1 (247)	1 (247)
test_core-cli-old-tr	0 (247)	0 (247)	0 (247)	0 (247)	0 (247)	0 (247)
test_core-cli-rt	0 (201)	0 (201)	0 (201)	0 (201)	0 (201)	0 (201)
test_core-confd	0 (270)	0 (270)	0 (270)	0 (270)	0 (270)	0 (270)
test_core-examples	0 (128)	0 (128)	0 (128)	0 (128)	0 (128)	0 (128)
test_core-java-api	0 (7)	0 (7)	0 (7)	0 (7)	0 (7)	0 (7)
test_core-jsonrpc	0 (77)	0 (77)	0 (77)	0 (77)	1 (77)	1 (77)
test_core-junit	0 (1)	0 (1)	0 (1)	0 (1)	0 (1)	0 (1)
test_core-ncm	0 (1)	0 (1)	0 (1)	0 (1)	0 (1)	0 (1)
test_core-netconf	0 (13)	0 (13)	0 (13)	0 (13)	0 (13)	0 (13)

Use at Cisco, Stockholm office

- ConfD - device configuration
  - Model driven configuration management framework for a network element
  - Render northbound interfaces such as CLI, NETCONF, SNMP, Rest, RestConf
  - Tracable internal interfaces
- NSO - Network Service Orchestrator
  - massive number of heterogeneous network elements
  - same northbound interfaces as ConfD, standard southbound interfaces
  - adapters (NEDs) for network elements lacking standard interfaces

Use at Cisco, Stockholm office

- Lux widely used for testing ConfD and NSO
- Automated test environment using Jenkins
  - ConfD needs to be tested on different architectures and OSes
- More than 4000 test cases written in Lux
- We also use eunit, common test, JUnit, hand written tests..
- No scheduled manual testing before release

Download Lux

- github.com/hawk/lux (Apache license)
- Tutorial in directory tutorial
- Source code from this presentation in directory tutorial/chatty

Questions?


Thank you

Extras