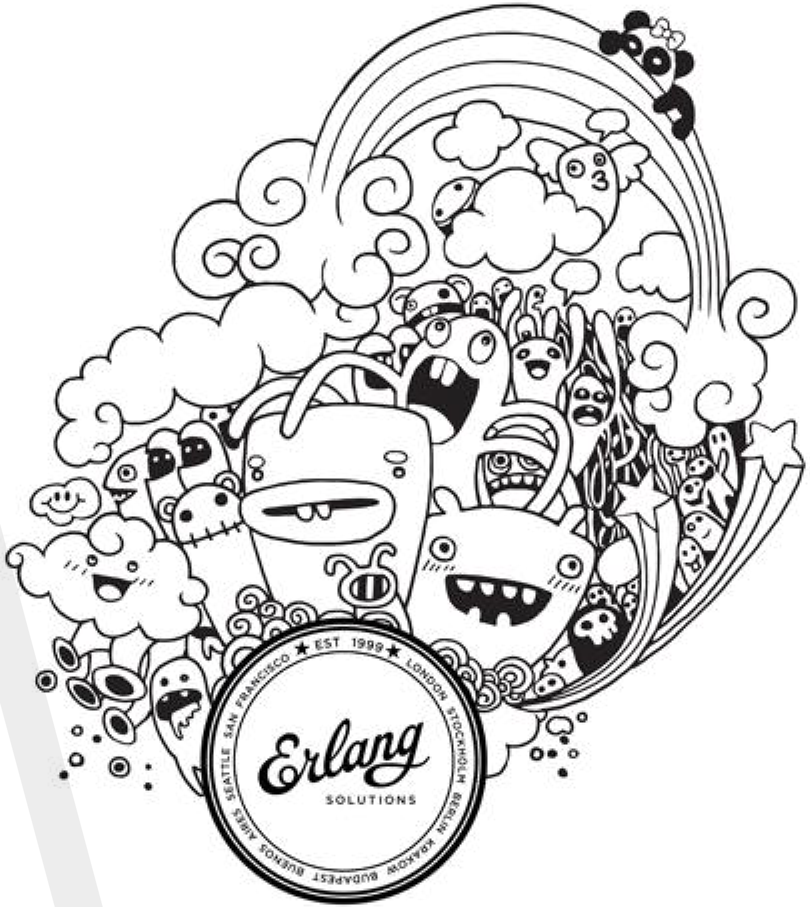

★ EST 1999 ★
LONDON STOCKHOLM BERLIN KRAKOW BUDAPEST BUENOS AIRES SEATTLE SAN FRANCISCO
Erlang
SOLUTIONS

ERLANG TRACING

HELLO!

Lukas Larsson

lukas.larsson@erlang-solutions.com
www.erlang-solutions.com
@garazdawi

Tracing today

- Overview
- Trace receivers
- Tracing events
- Meta tracing
- Match Specifications
- Sequence Tracing
- dtrace/systemtap

Tracing tomorrow

- Scalability
- Tracing nif backend
- Match Specifications everywhere
- lttng

Tracing in the future

- Vision

1.

TRACING TODAY

“

A specialized use of logging to record information about a program's execution

ERLANG TRACING

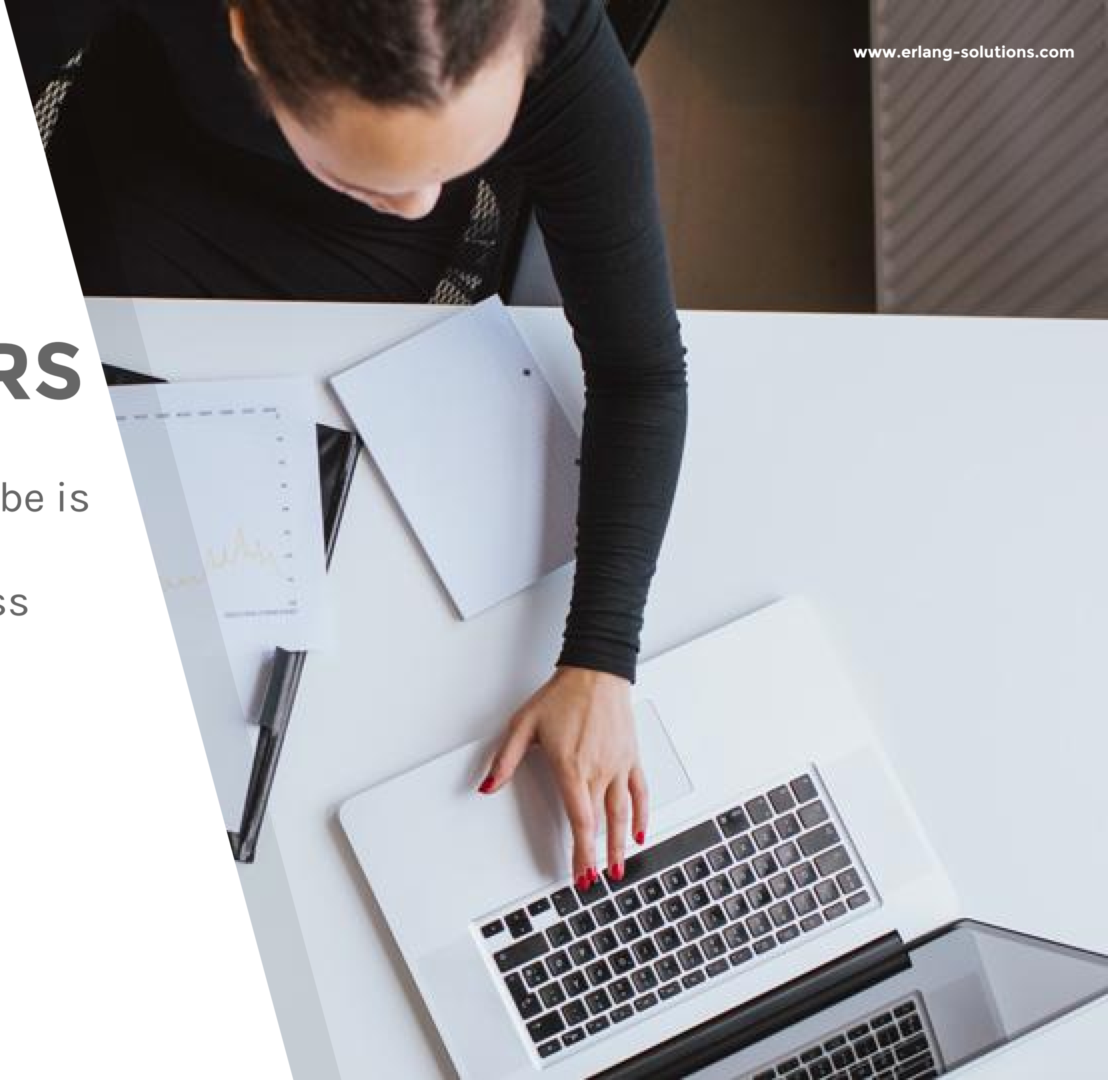
- Built into the VM
- Trace on anything
- Send to file, tcp or stdout

```
1> dbg:tracer(), dbg:p(all, c).
{ok,[{matched,nonode@nohost,26}]}
2> dbg:tp(lists, seq, x).
{ok,[{matched,nonode@nohost,2},{saved,x}]}
3> lists:seq(1,a).
(<0.32.0>) call lists:seq(1,a)
(<0.32.0>) exception_from {lists,seq,2} {error,
function_clause}
** exception error: no function clause matching lists:
seq(1,a) (lists.erl, line 228)
```

2.

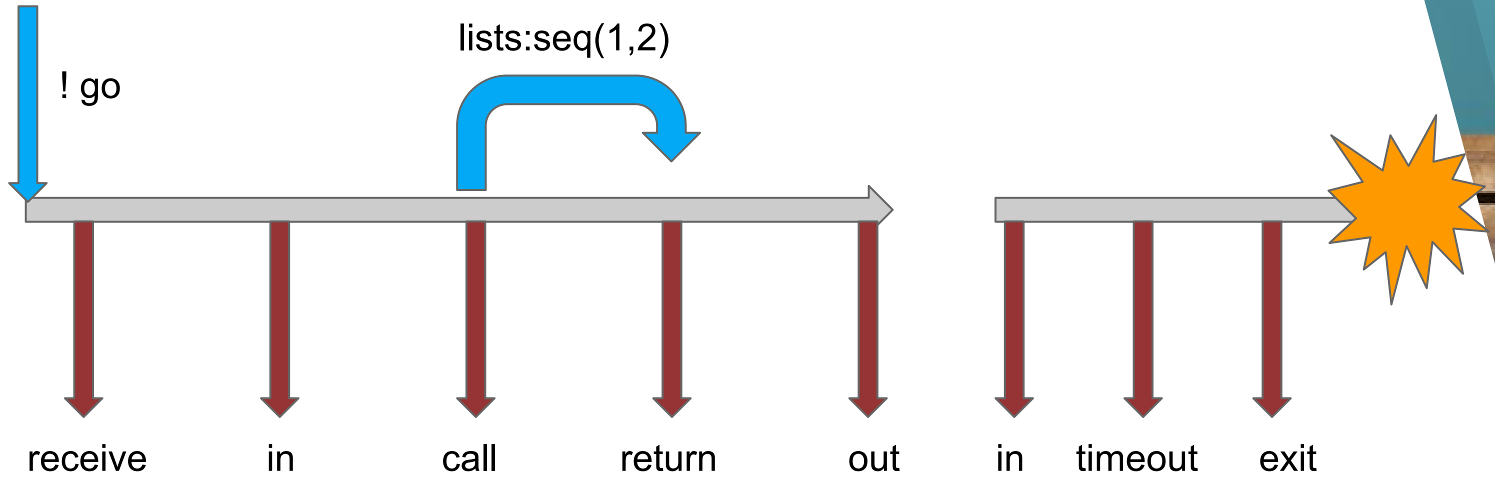
TRACE RECEIVERS

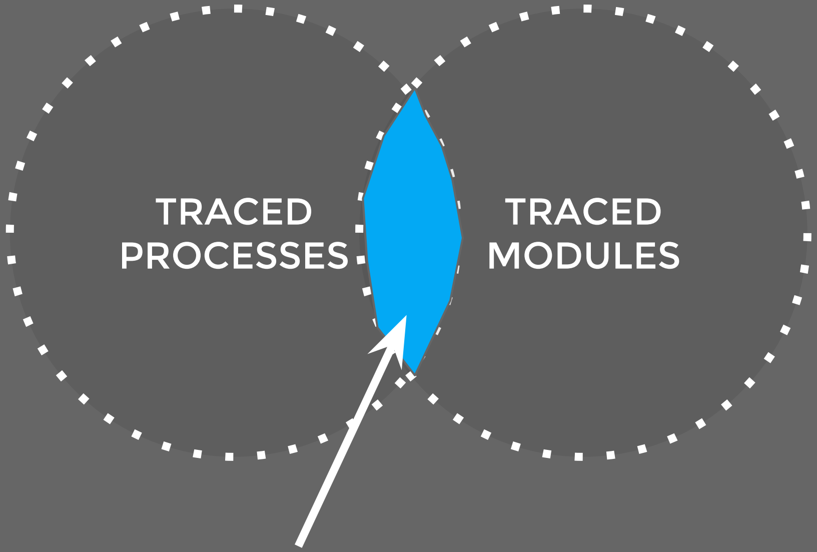
- What will happen when a probe is triggered?
  - Send message to process
    - Print to stdout
    - Analyze
  - Send message to port
    - Write to file
    - Write to tcp



3. Tracing events

```
spawn(fun() ->
                    receive go -> go end,
                    lists:seq(1,2),
                    receive after 10 -> ok end
          end).
```





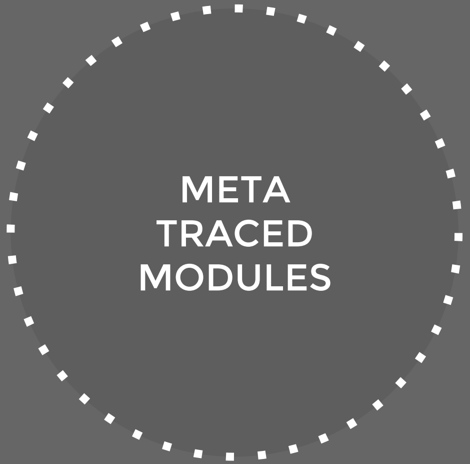
CALL TRACING

Events sent to process specific tracer

CALL TRACING GENERATES EVENTS WHEN PROCESS AND CODE ARE BEING TRACED

4.

Meta Tracing



TRACING ON CODE FOR ALL PROCESSES

Events sent to system wide meta tracer

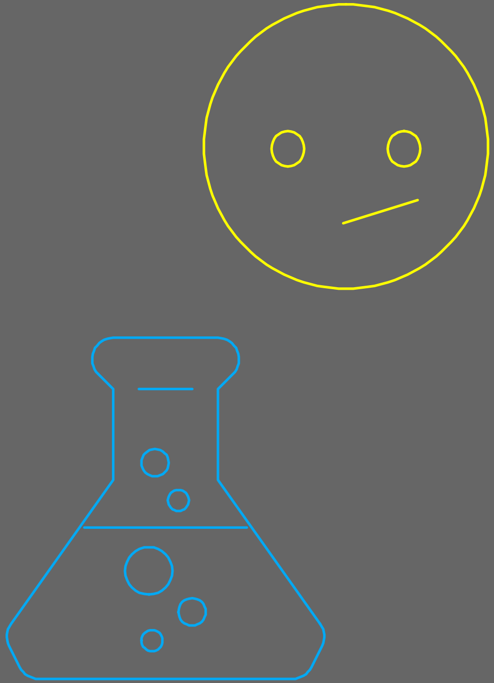
META TRACING

META
TRACED
MODULES

```
1> dbg:tracer().
{ok,<0.34.0>}
2> {ok,T} = dbg:get_tracer().
{ok,<0.35.0>}
3> erlang:trace_pattern({gen_server,cast,2},[],[{meta,T}]).
1
4> gen_server:cast(self(),hello).
(<0.32.0>) call gen_server:cast(<0.32.0>,hello)
        (Timestamp: {1450,451605,435321})
ok
```

5. Match Specification

EXECUTE CODE
WHEN TRACE
PROBE IS
TRIGGERED



MATCH SPECIFICATION

- Discard events early
- Change behaviour of tracing

```
[ {
    [{ ‘_’, ‘_’ }, ‘_’, ‘$1’ ],
    [{‘==’,{length,’$1’},1}],
    [{return_trace}]
  } ].
```

Function arguments match

Guard clause

Actions to take,
implicit {message, true}

META TRACING and MATCH SPECIFICATION

META TRACED MODULES

```
1> dbg:tracer().
{ok,<0.34.0>}
2> {ok,T} = dbg:get_tracer().
{ok,<0.35.0>}
3> erlang:trace_pattern({gen_server,cast,2},
    [{'_',[{'==',{self},self()}],
    [{trace,[],[send,{const,{tracer,T}}]}]}],
    [{meta,T}]).

1
4> gen_server:cast(self(),hello).
(<0.32.0>) call gen_server:cast(<0.32.0>,hello)
        (Timestamp: {1450,451605,435321})
(<0.32.0>) <0.32.0> ! {'$gen_cast',hello}
ok
```

META TRACING and MATCH SPECIFICATION

META
TRACED
MODULES

```
1> dbg:tracer().
{ok,<0.34.0>}
2> {ok,T} = dbg:get_tracer().
{ok,<0.35.0>}
3> erlang:trace_pattern({gen_server,cast,2},
    [{'_',[{'==',{self},self()}],
    [{trace,[],[send,{const,{tracer,T}}]},{message,false}]}],
    [{meta,T}]).
1
4> gen_server:cast(self(),hello).
(<0.32.0>) <0.32.0> ! {'$gen_cast',hello}
ok
```

MATCH SPEC advaced tips ‘n tricks

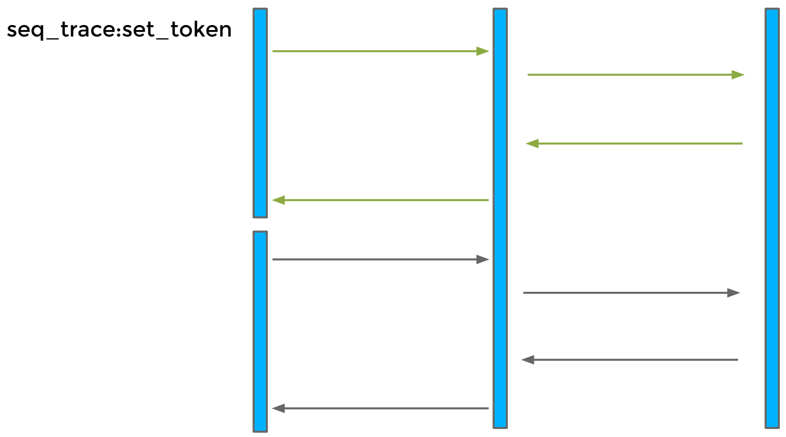
- Use **trace control word** to control when to trace
- Use **trace**, **enable_trace** and **disable_trace** to control what is traced
- Use **{message, false}** to execute match specs but not send any trace message

MATCH SPEC advanced tips 'n tricks

```
1> dbg:tracer(), {ok,T} = dbg:get_tracer().

{ok,<0.37.0>}

2> erlang:trace_pattern({lists,seq,2},
     [{'_',[{'==',{get_tcw},1}],
         [{trace, [], [{const, {tracer, T}}, send]},
          {message, false}]},
      {'_',[{'=/=',{get_tcw},1}], [{disable_trace, send},
          {message, false}]}],
     [{meta,T}]).

1

3> P = self(), spawn(fun() -> P ! lists:seq(1,2) end), flush().
Shell got [1,2]
ok
3> erlang:system_flag(trace_control_word, 1).
0
6> spawn(fun() -> P ! lists:seq(1,2) end), flush().
(<0.45.0>) <0.34.0> ! [1,2]

Shell got [1,2]
ok
```

6.

Sequence Tracing

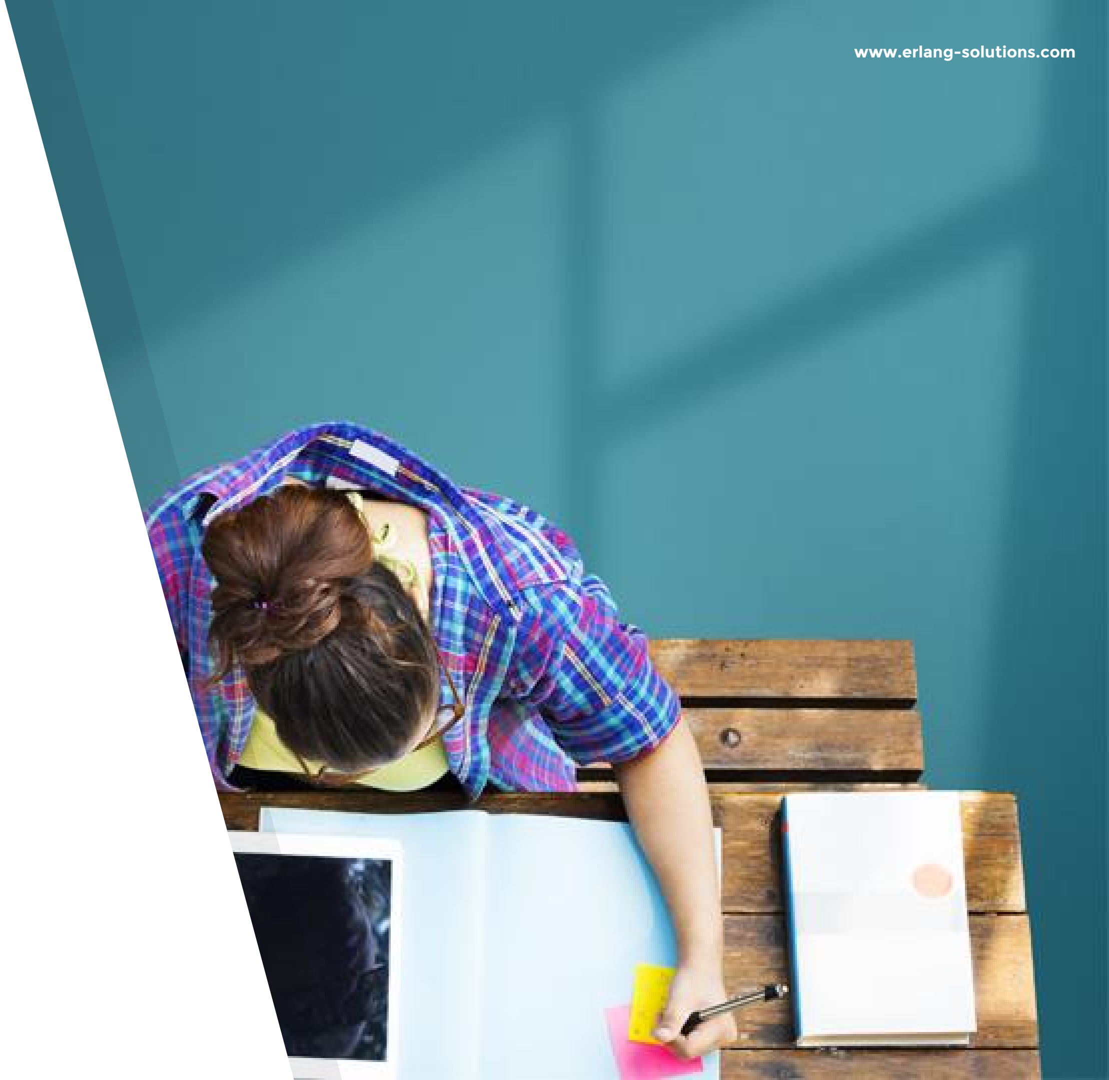


- Follow events from one source
- Spreads via message passing
- Read/write in match specs
  - i.e. we can filter on it
- Works over distribution

7.

dtrace/systemtap

- Trace on ERTS and Erlang events
  - almost all normal trace events + many ERTS internal probes
- Filter using D/stap scripts
- Correlate Erlang events with external events
  - kernel, filesystem, tcp stack etc

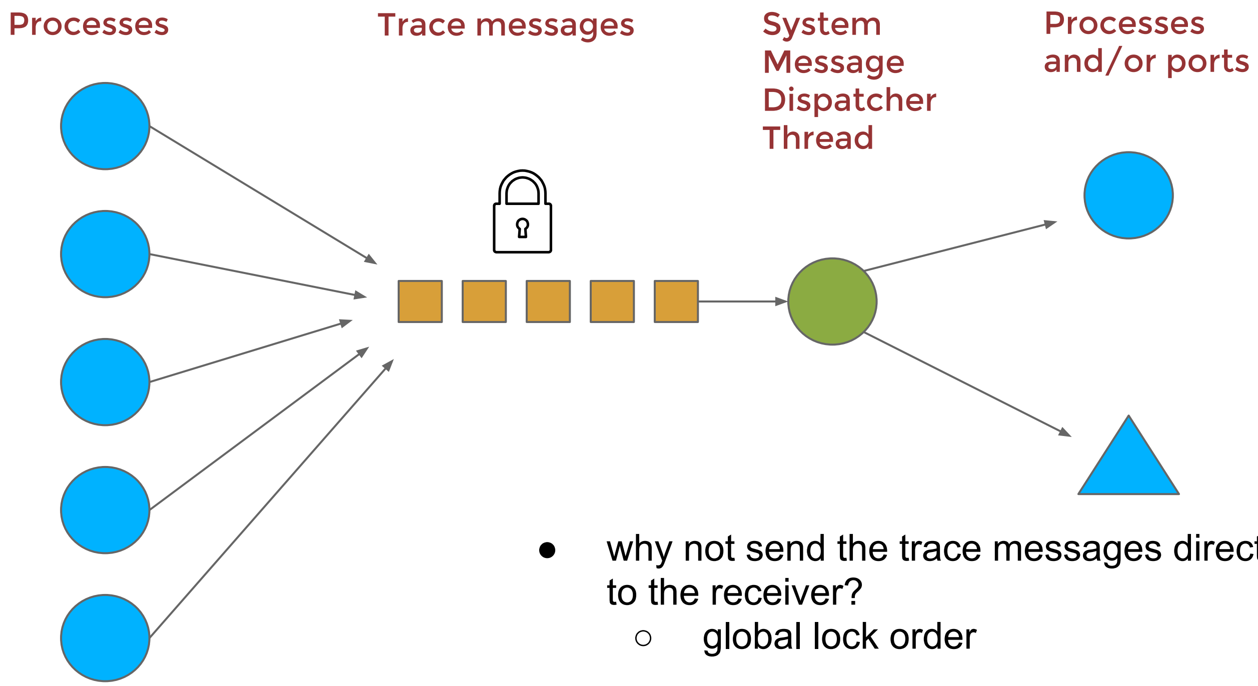


TRACING TOMORROW

8.

Scalability

Tracing in Erlang/OTP before 19.0



- why not send the trace messages directly to the receiver?
    - global lock order

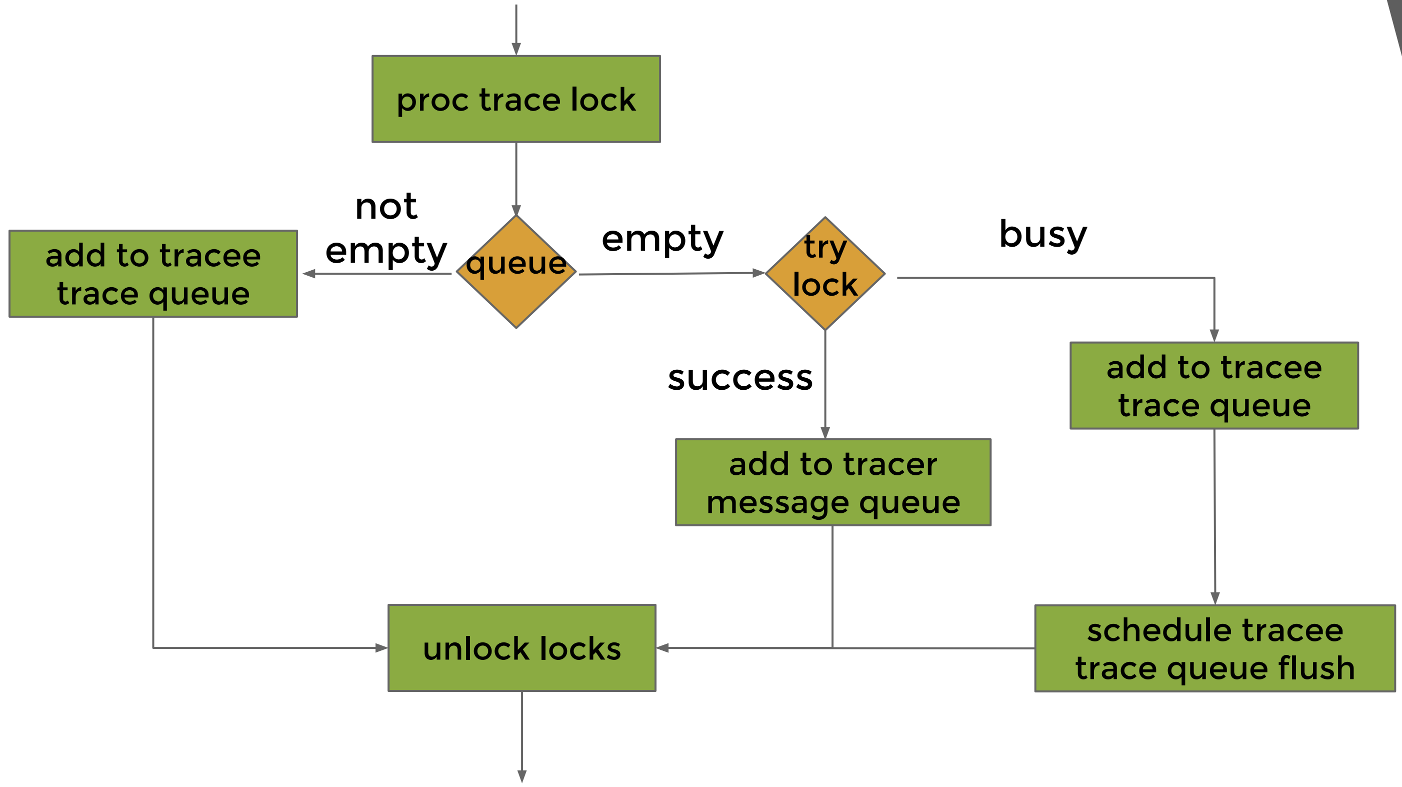
Global Lock Order

- All locks have an order in which they have to be taken
  - port > proc main > proc link > proc msg > proc status > erts internal
  - #Port<0.25> > #Port<0.26.0>
  - About 90 locks in total
- Message sending needs proc msg lock
  - therefore we cannot have anything under proc link locked when tracing
  - some traces needs to have proc msg and/or proc status locked to order trace messages
- Problem!

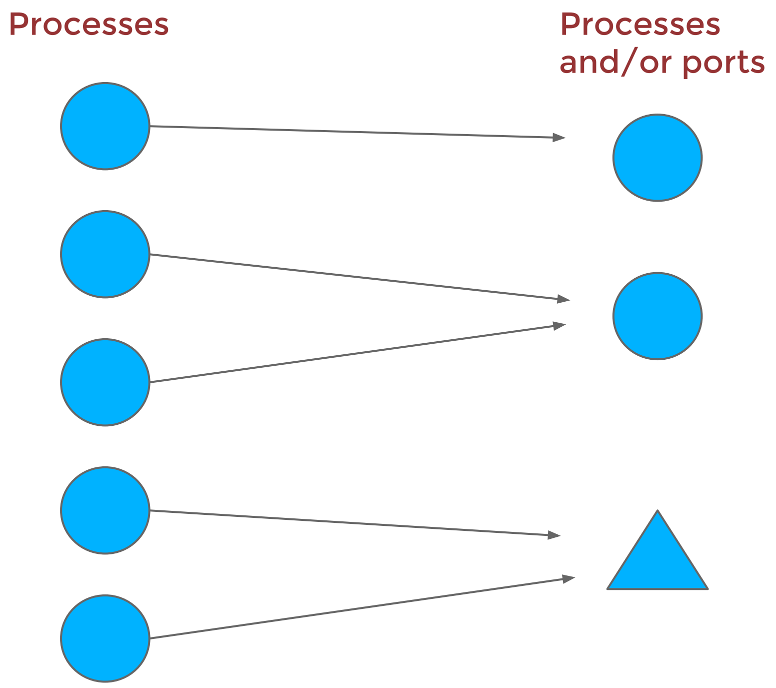
Tracing in Erlang/OTP 19.0

- Removed need for proc status to be taken
  - Only have to deal with proc msg
- Always try lock proc msg lock
  - Allows lock order to be circumvented
  - Have to deal with what happens if try lock fail
    - spinning is not an option…
- Introduce per process trace message queue
  - protected by a low order per process lock
  - has to be taken when generating a trace message

Tracing in Erlang/OTP 19.0


proc trace lock
queue
not empty
add to tracee trace queue
empty
try lock
busy
success
add to tracer message queue
add to tracee trace queue
schedule tracee trace queue flush
unlock locks

Tracing in Erlang/OTP 19.0



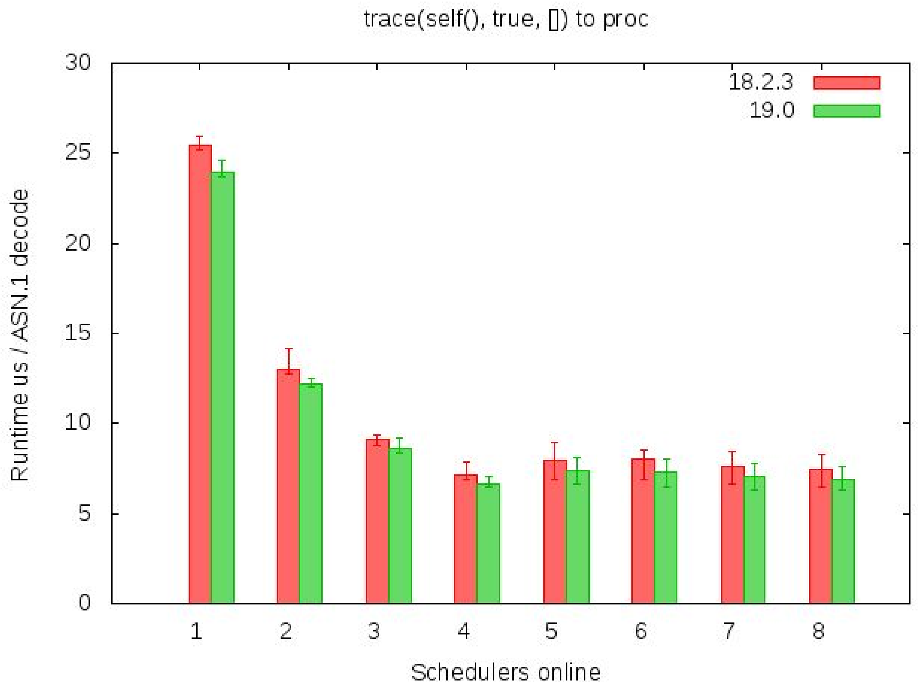
Functional Implications of new trace algorithm

- Trace messages will more likely be out of order with normal messages
  - Was possible before, but unlikely
- Trace message timestamps may arrive out of order to tracer
  - not possible before as timestamp was taken when holding system message lock
- Tracing to port does not need this as the port msg lock is very low
  - still some small reordering may happen

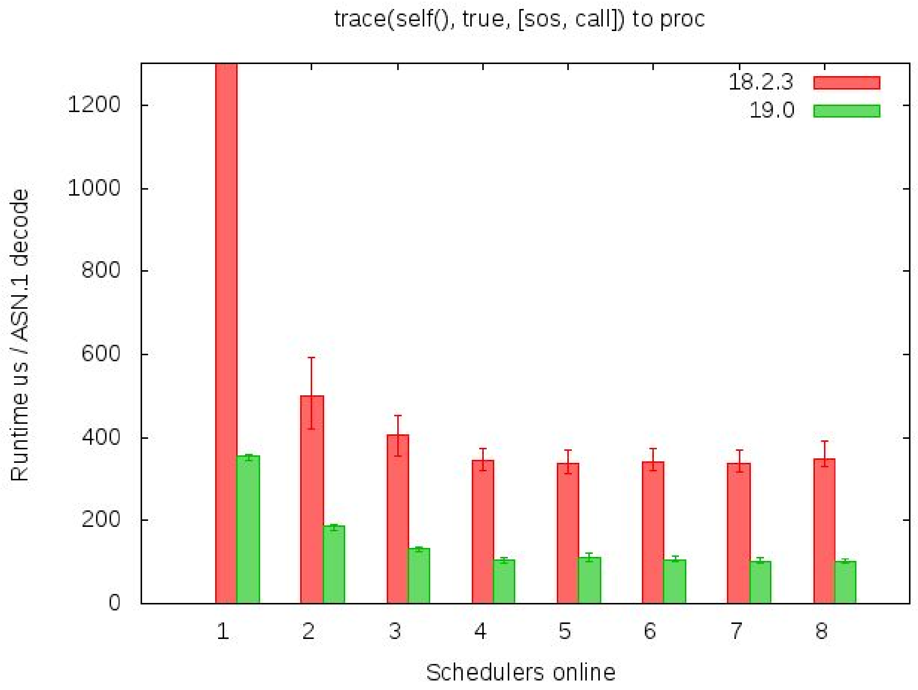
Benchmark

- 10 processes
- 1250 ASN.1 encode per process
- 7,062,500 trace messages in total
- Measure time until completion
- Measure time until all trace messages delivered
- Measure erlang:memory(total) at 10 Hz

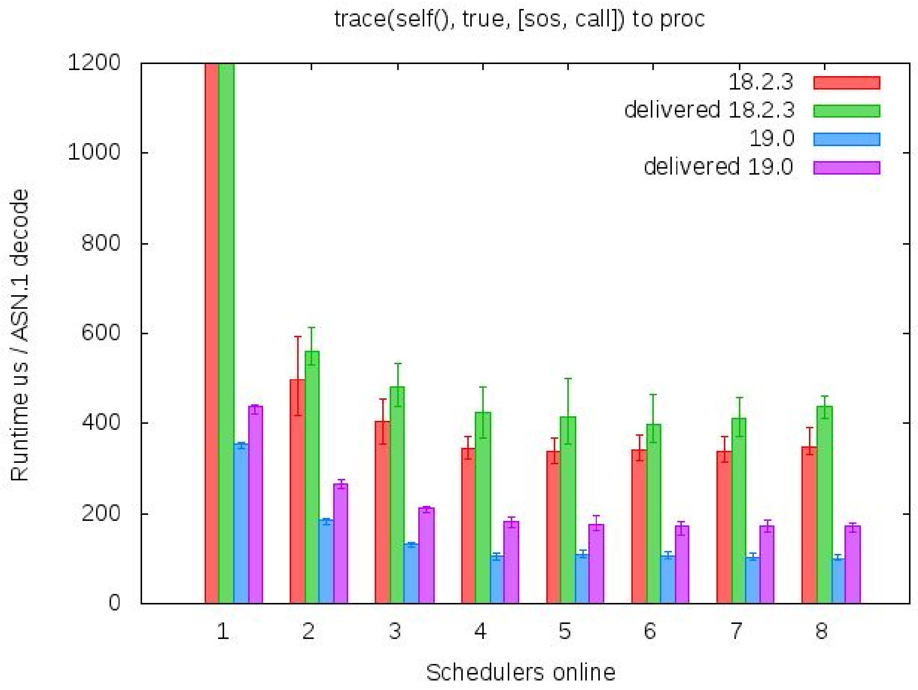
Baseline – No tracing



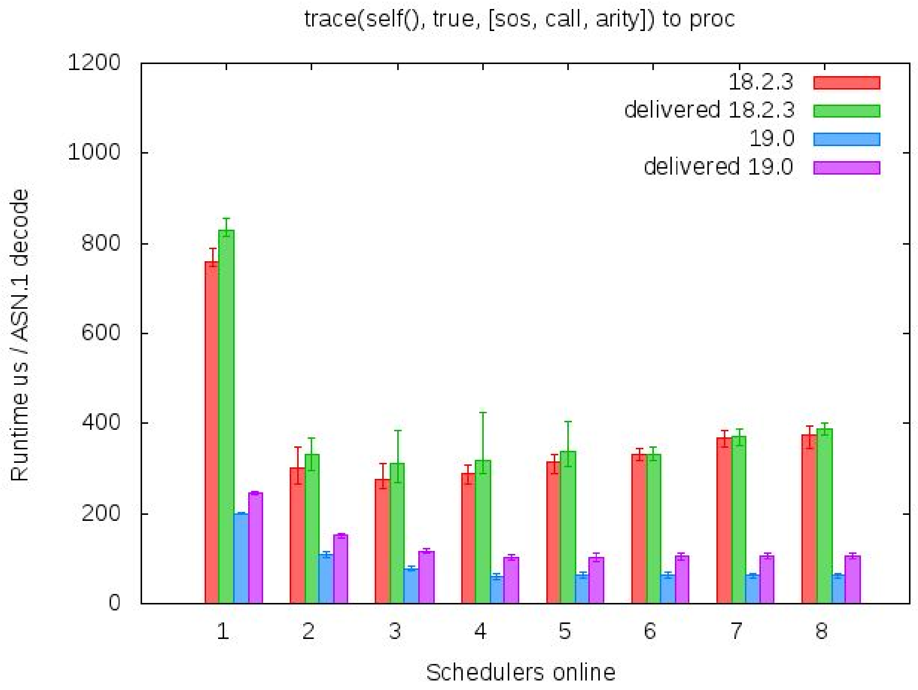
Call trace to process



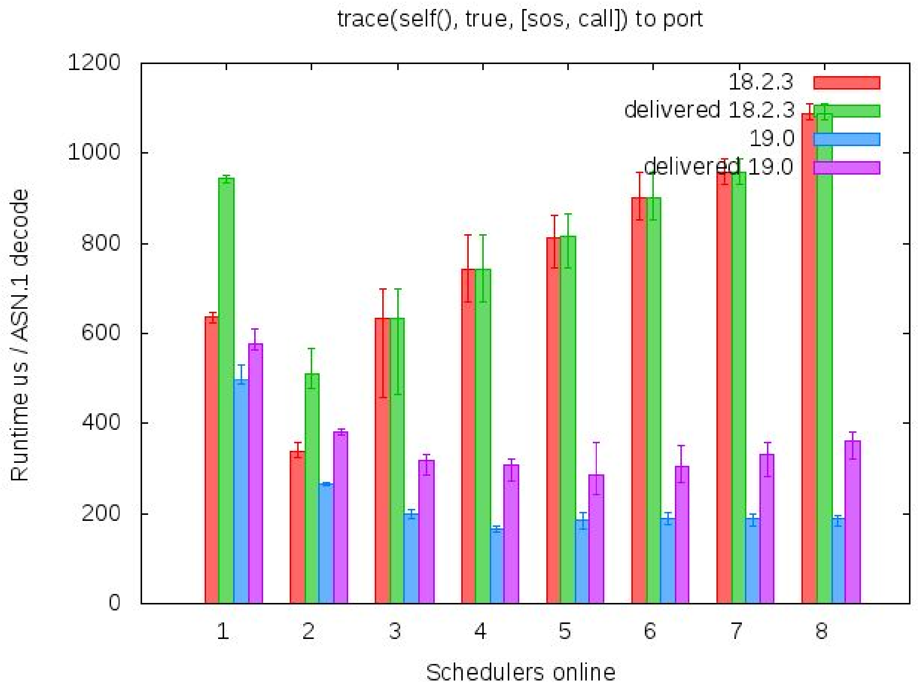
Call trace to process



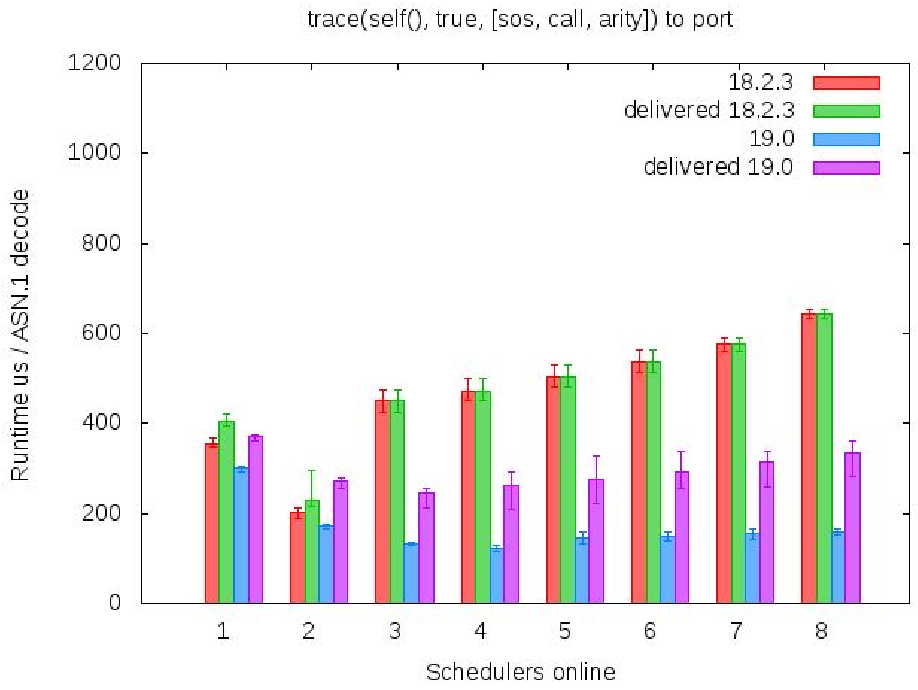
Call arity trace to process



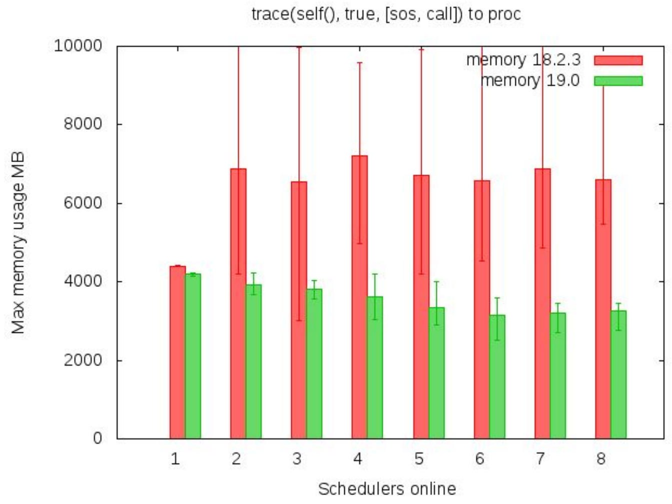
Call trace to port

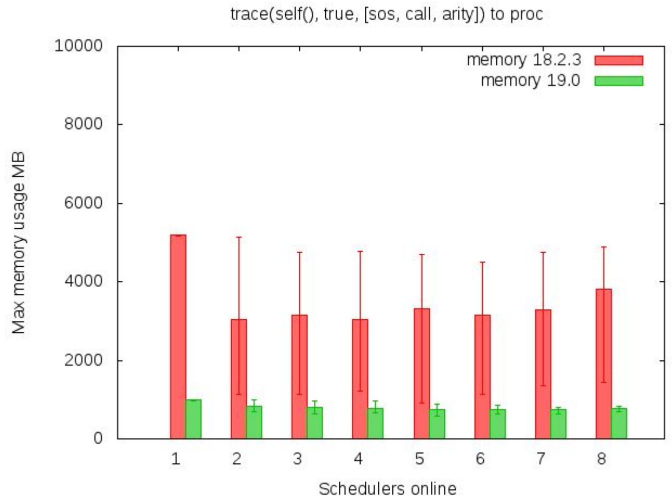


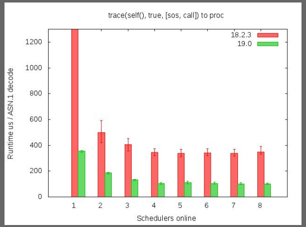
Call arity trace to port

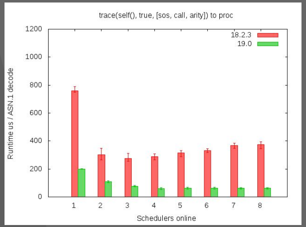


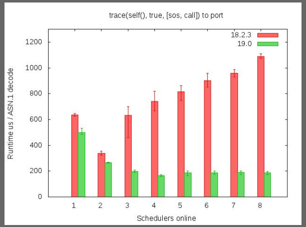
Max mem usage during trace to process

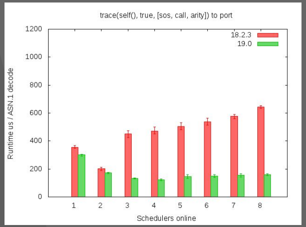




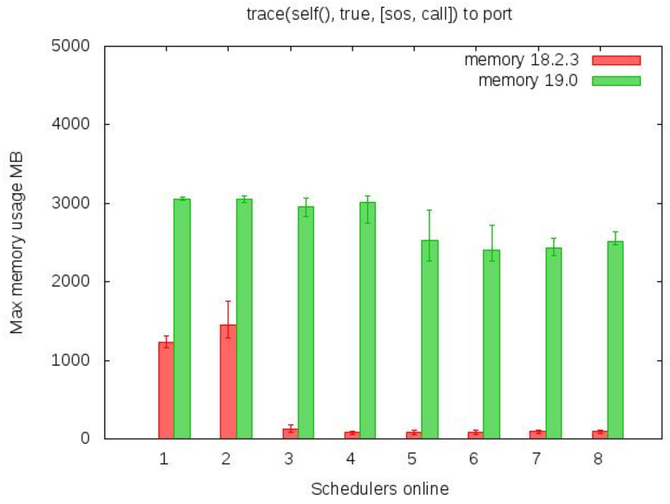


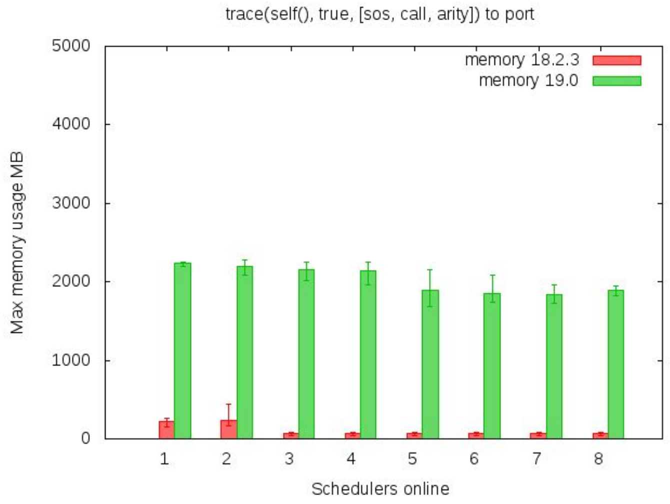






Max mem usage during trace to port





9.

Tracer Modules

When and what to trace

GOALS erl_tracer

- Earlier programmable filters for all trace events
  - complement/replace match specifications
- Pluggable backends types
  - syslog, lttng, sampling profiler
- Performant
  - zero-copying of trace terms
- Erlang code

erl_tracer

API:s may change before release

```
1> erlang:trace(new, true, [send,{tracer, erl_msg_tracer, Tracer}]).
0
```

API:s may
change
before
release

erl_tracer

```
-module(erl_msg_tracer).
-export([enabled/3, trace/6]).

enabled(Event, Tracer, Tracee) when Event =:= send;
                                    Event =:= trace_status ->
   trace;
enabled(Event, Tracer, Tracee) ->
   disable.

trace(send, Tracer, Tracee, Msg, To, _Opts) when node(To) =:= node() ->
   Tracer ! {my_trace, Tracee, Msg, To};
trace(send, _Tracer, _Tracee, _Msg, _To, _Opts) -> ok.
```

erl_tracer in Erlang/OTP 19.0

- Callbacks have to be NIFs
- Handle all trace and seq trace events
- process + port + dtrace/systemtap/lttng backends

erl_tracer after Erlang/OTP 19.0

- Some trace events handled in Erlang code
  - especially call, hopefully more
- Handle all system profile, system monitor error logger events
- Specialized backends
  - sampling profiler (e.g. github. com/slfritchie/eflame)
  - load shedding
  - udp
- Lots of backends implemented by you!

10. More match specs

I heard you liked match specs, so I put match specs in your match specs

More match specs

API:s may change before release

- `erlang:trace_pattern(send, [{['$1','_'],[{'=/=',{node},{node,'$1'}}],[]}]).`
  - only trace on message to other nodes
- `erlang:trace_pattern('receive', [{['_', '$2'],[{'==',{const,'$gen_cast'},{element,1,'$2'}}],[]}]).`
  - only trace on received {'$gen_cast, …} messages

- In future add to more/all trace events
  - Maybe can be superseded by more flexible tracer modules

11.

Port tracing

```
1> dbg:tracer(), {ok, T} = dbg:get_tracer().
{ok,<0.37.0>}
2> erlang:trace(new_ports, true, [ports, send, 'receive', {tracer, erl_tracer, T}]).
0
```

11.

Port tracing

```
1> dbg:tracer(), {ok, T} = dbg:get_tracer().
{ok,<0.37.0>}
2> erlang:trace(new_ports, true, [ports, send, 'receive', {tracer, erl_tracer, T}]).
0
3> {ok, D} = file:open("/tmp/tmp.data",[write]).
(#Port<0.495>) open <0.40.0> efile
(#Port<0.495>) getting_linked <0.40.0>
(#Port<0.495>) << {<0.40.0>,{command,[<<1,0,0,0,2,47,116,109,112,47,116,109,112,46,100,97,116,97,0>>]}}
(#Port<0.495>) <0.40.0> ! {#Port<0.495>,{data,[3,0,0,0,0,0,0,0,12]}}
{ok,<0.40.0>}
4> erlang:trace(new_ports, false, [send, 'receive']).
0
```

11.

Port tracing

```
1> dbg:tracer(), {ok, T} = dbg:get_tracer().
{ok,<0.37.0>}
2> erlang:trace(new_ports, true, [ports, send, 'receive', {tracer, erl_tracer, T}]).
0
3> {ok, D} = file:open("/tmp/tmp.data",[write]).
(#Port<0.495>) open <0.40.0> efile
(#Port<0.495>) getting_linked <0.40.0>
(#Port<0.495>) << {<0.40.0>,{command,[<<1,0,0,0,2,47,116,109,112,47,116,109,112,46,100,97,116,97,0>>]}}
(#Port<0.495>) <0.40.0> ! {#Port<0.495>,{data,[3,0,0,0,0,0,0,0,12]}}
{ok,<0.40.0>}
4> erlang:trace(new_ports, false, [send, 'receive']).
0
5> io:format(D, "Hello world\n", []).
(#Port<0.495>) << {<0.40.0>,{command,[<<4>>,<<"Hello world\n">>]}}
(#Port<0.495>) <0.40.0> ! {#Port<0.495>,{data,[3,0,0,0,0,0,0,0,12]}}
ok
```

11.

Port tracing

```
1> dbg:tracer(), {ok, T} = dbg:get_tracer().
{ok,<0.37.0>}
2> erlang:trace(new_ports, true, [ports, send, 'receive', {tracer, erl_tracer, T}]).
0
3> {ok, D} = file:open("/tmp/tmp.data",[write]).
(#Port<0.495>) open <0.40.0> efile
(#Port<0.495>) getting_linked <0.40.0>
(#Port<0.495>) << {<0.40.0>,{command,[<<1,0,0,0,2,47,116,109,112,47,116,109,112,46,100,97,116,97,0>>]}}
(#Port<0.495>) <0.40.0> ! {#Port<0.495>,{data,[3,0,0,0,0,0,0,0,12]}}
{ok,<0.40.0>}
4> erlang:trace(new_ports, false, [send, 'receive']).
0
5> io:format(D, "Hello world\n", []).
(#Port<0.495>) << {<0.40.0>,{command,[<<4>>,<<"Hello world\n">>]}}
(#Port<0.495>) <0.40.0> ! {#Port<0.495>,{data,[3,0,0,0,0,0,0,0,12]}}
ok
6> file:close(D).
(#Port<0.495>) << {<0.40.0>,{command,[<<23>>]}}
(#Port<0.495>) <0.40.0> ! {#Port<0.495>,{data,[0]}}
(#Port<0.495>) << {<0.40.0>,close}
(#Port<0.495>) closed normal
(#Port<0.495>) unlink <0.40.0>
ok
```

12.

Vision

Implemented
Framework
To be done

Flexibility

- Many tracers
- Early filtering
- Overload prot.

Documentation

- Basic dbg
- dbg + ttb interaction
- Match Specs

Tools

- lttng/dtrace/ systemtap
- Common Trace Format

Performance

- Overhead
- Scalability
- Throughput

THANK YOU!

Any questions?

lukas.larsson@erlang-solutions.com
www.erlang-solutions.com
@garazdawi