Process Signals in OTP-21

Lukas Larsson
lukas.larsson@erlang-solutions.com
@garazdawi

1.

Signals

“

Communication in Erlang is conceptually performed using asynchronous signaling. All different executing entities, such as processes and ports, communicate through asynchronous signals.

- ERTS User’s Guide ⇨ Communication in Erlang

ASYNCHRONOUS MESSAGES

- `Pid ! message.`
  - Send a message
- `RemotePid ! message.`
  - Send a message remotely
- `name ! message.`
  - Send a message to a locally registered process
- `{name,'n@local'} ! message.`
  - Send a message to a remotely registered process
- `Port ! {self(), {command, "hello"}}.`
  - Send a message to a port

SYNCHRONOUS MESSAGES

```
client(ServerPid) ->
  ServerPid ! {message, self()},
  receive Reply -> Reply end.

server() ->
  receive
    {message, From} ->
      From ! reply,
      server()
  end.
```

We build synchronous messages using two asynchronous messages

NON-MESSAGE SIGNALS

- `link(Pid)`, `unlink(Pid)`.
  - Create or destroy a link to Pid

NON-MESSAGE SIGNALS

- `link(Pid)`, `unlink(Pid)`.
  - Create or destroy a link to Pid
- Exit signal `{'EXIT', normal, Pid}`.
  - Sent by exit/2 or when a link breaks

NON-MESSAGE SIGNALS

- `link(Pid)`, `unlink(Pid)`.
  - Create or destroy a link to Pid
- Exit signal `{'EXIT', normal, Pid}`.
  - Sent by exit/2 or when a link breaks
- `Ref = monitor(process, Pid)`, `demonitor(Ref)`.
  - Setup or remove a monitor on Pid

NON-MESSAGE SIGNALS

- `link(Pid)`, `unlink(Pid)`.
  - Create or destroy a link to Pid
- Exit signal `{'EXIT', normal, Pid}`.
  - Sent by exit/2 or when a link breaks
- `Ref = monitor(process, Pid)`, `demonitor(Ref)`.
  - Setup or remove a monitor on Pid
- Down signal `{'DOWN', Ref, process, Pid, normal}`.
  - Send when a monitor breaks

NON-MESSAGE SIGNALS

- `link(Pid)`, `unlink(Pid)`.
  - Create or destroy a link to Pid
- Exit signal `{'EXIT', normal, Pid}`.
  - Sent by exit/2 or when a link breaks
- `Ref = monitor(process, Pid)`, `demonitor(Ref)`.
  - Setup or remove a monitor on Pid
- Down signal `{'DOWN', Ref, process, Pid, normal}`.
  - Send when a monitor breaks
- `erlang:trace(Pid, true, [call])`.
  - Change trace flags on Pid

NON-MESSAGE SIGNALS

- `link(Pid)`, `unlink(Pid)`.
  - Create or destroy a link to Pid
- Exit signal `{'EXIT', normal, Pid}`.
  - Sent by exit/2 or when a link breaks
- `Ref = monitor(process, Pid)`, `demonitor(Ref)`.
  - Setup or remove a monitor on Pid
- Down signal `{'DOWN', Ref, process, Pid, normal}`.
  - Send when a monitor breaks
- `erlang:trace(Pid, true, [call])`.
  - Change trace flags on Pid
- `erlang:process_info(Pid)`.
  - Request information about Pid

SYNCHRONOUS NON-MESSAGE SIGNALS

- `link(Pid)`, `unlink(Pid).`
- Exit signal `{'EXIT', normal, Pid}`.
- `Ref = monitor(process, Pid), demonitor(Ref).`
- Down signal `{'DOWN', Ref, process, Pid, normal}`.
- `erlang:trace(Pid, true, [call]).`
- `erlang:process_info(Pid).`

ERROR HANDLING FOR SIGNALS, link(PidOrPort)

If `PidOrPort` does not exist, the behavior of the BIF depends on if the calling process is trapping exits or not:

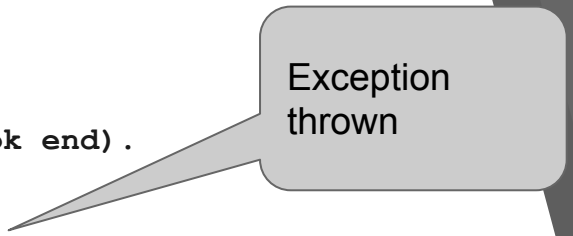
- If the calling process is not trapping exits, and checking `PidOrPort` is cheap (that is, if `PidOrPort` is local), `link/1` fails with reason `noproc`.
- Otherwise, if the calling process is trapping exits, and/or `PidOrPort` is remote, `link/1` returns `true`, but an exit signal with reason `noproc` is sent to the calling process.

ERROR HANDLING FOR SIGNALS, link(PidOrPort)

```
Eshell V10.2.3  (abort with ^G)
(test1@elxd3291v0k)1> P1 = spawn(fun() -> ok end).
<0.87.0>
(test1@elxd3291v0k)2> catch link(P1).
```

ERROR HANDLING FOR SIGNALS, link(PidOrPort)

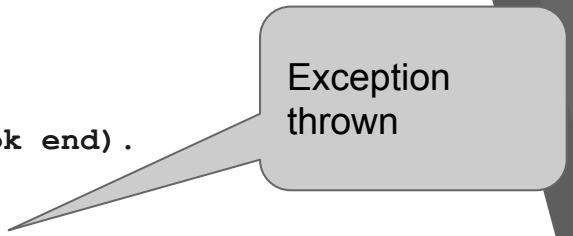
```
Eshell V10.2.3  (abort with ^G)
(test1@elxd3291v0k)1> P1 = spawn(fun() -> ok end).
<0.87.0>
(test1@elxd3291v0k)2> catch link(P1).
{'EXIT',{noproc,[{erlang,link,[<0.87.0>],[]},...}
```



ERROR HANDLING FOR SIGNALS, link(PidOrPort)

```
Eshell V10.2.3  (abort with ^G)
(test1@elxd3291v0k)1> P1 = spawn(fun() -> ok end).
<0.87.0>
(test1@elxd3291v0k)2> catch link(P1).
{'EXIT',{noproc,[{erlang,link,[<0.87.0>],[]},...}


(test1@elxd3291v0k)3> P2 = spawn('n@local',fun() -> ok end).
<8623.108.0>
(test1@elxd3291v0k)4> catch link(P2).
```



ERROR HANDLING FOR SIGNALS, link(PidOrPort)

```
Eshell V10.2.3  (abort with ^G)
(test1@elxd3291v0k)1> P1 = spawn(fun() -> ok end).
<0.87.0>
(test1@elxd3291v0k)2> catch link(P1).
{'EXIT',{noproc,[{erlang,link,[<0.87.0>],[]},...}
```

Exception thrown

```
(test1@elxd3291v0k)3> P2 = spawn('n@local',fun() -> ok end).
<8623.108.0>
(test1@elxd3291v0k)4> catch link(P2).
true
** exception error: no such process or port
```

Exit signal received

2.

Signal Order

“

If an entity sends multiple signals to the same destination entity, the order is preserved; that is,

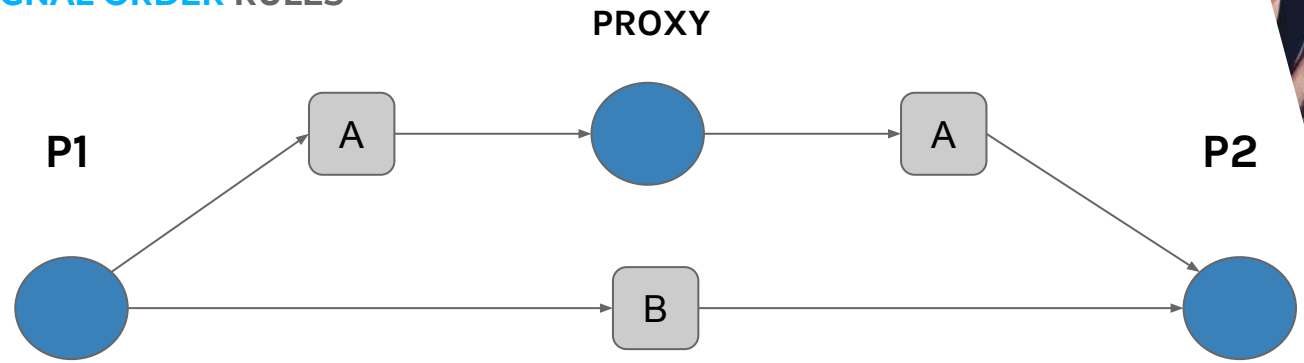
if A sends a signal S1 to B, and later sends signal S2 to B, S1 is guaranteed not to arrive after S2.

- ERTS User’s Guide ⇨ Communication in Erlang

SIGNAL ORDER RULES

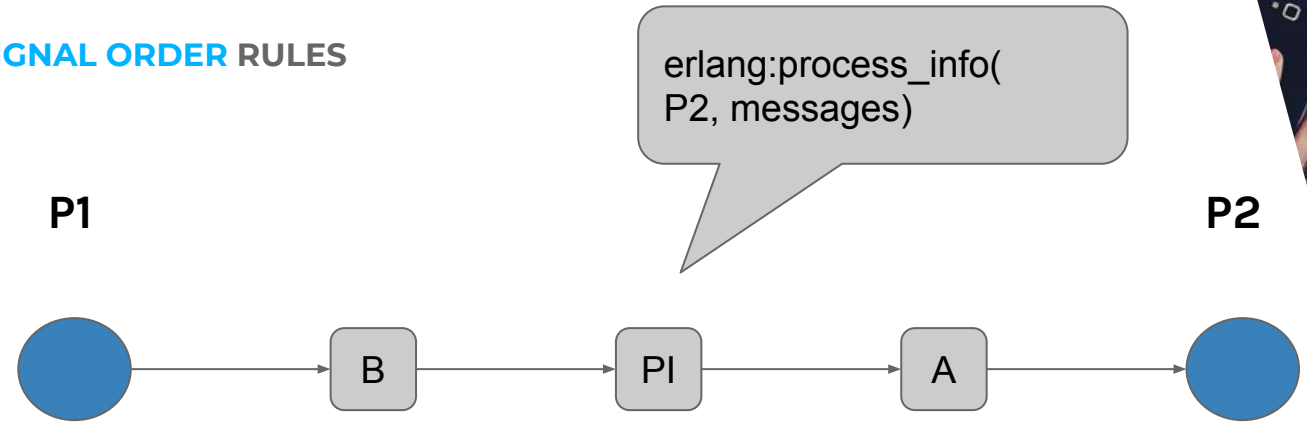
- First A then B arrives at P2

SIGNAL ORDER RULES



- First A then B arrives at P2
- First B then A arrives at P2

SIGNAL ORDER RULES



- First A then process_info then B arrives at P2

SIGNAL ORDER RULES

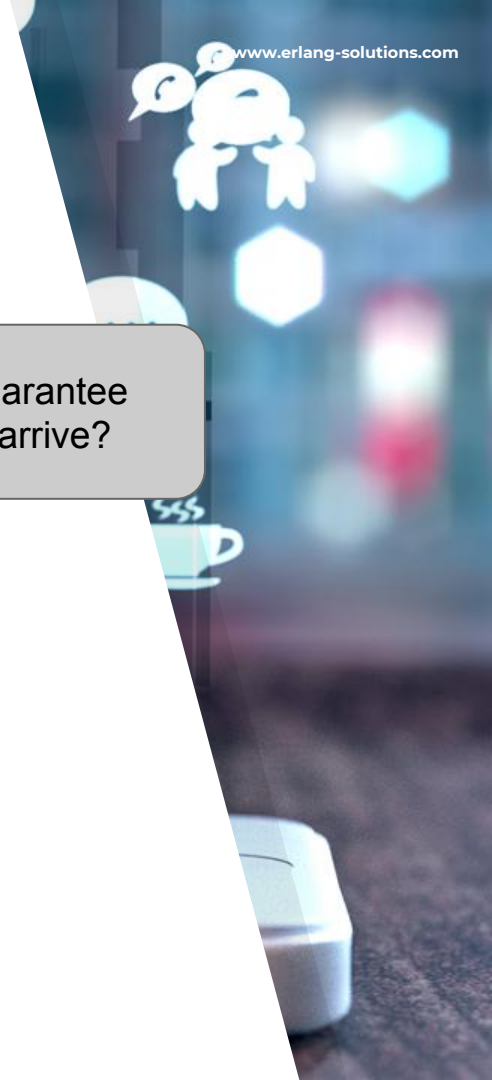
```
foo() ->
  P2 = spawn(fun() ->
                receive after infinity -> ok end
             end),
  P2 ! a,
  Msgs = process_info(P, messages),
  P2 ! b,
  Msgs.

> foo().
{messages,[a]}
```

SIGNAL ORDER RULES

```
foo() ->
  P2 = spawn(fun() ->
                receive after infinity -> ok end
             end),
  P2 ! a,
  Msgs = erlang:trace(P2, true, ['receive']),
  P2 ! b,
  Msgs.

> foo(),flush().
Shell got {trace,<0.82.0>,'receive',hello}
```



“

If an entity sends multiple signals to the same destination entity, the order is preserved; that is,

if A sends a signal S1 to B, and later sends signal S2 to B, S1 is guaranteed not to arrive after S2.

- ERTS User’s Guide ⇨ Communication in Erlang

SIGNAL ORDER RULES

```
foo() ->
  P2 = spawn('n@local', fun() ->
                 receive after infinity -> ok end
               end),
  P2 ! a,
  Msgs = process_info(P, messages),
  P2 ! b,
  Msgs.

> foo().
{messages,[]}
```

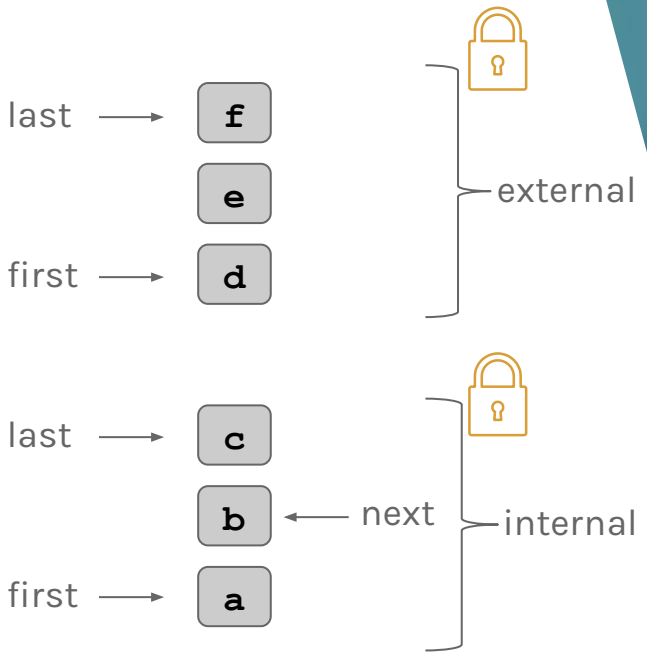
3.

MESSAGES BEFORE OTP-21

MESSAGE SIGNALS
before OTP-21

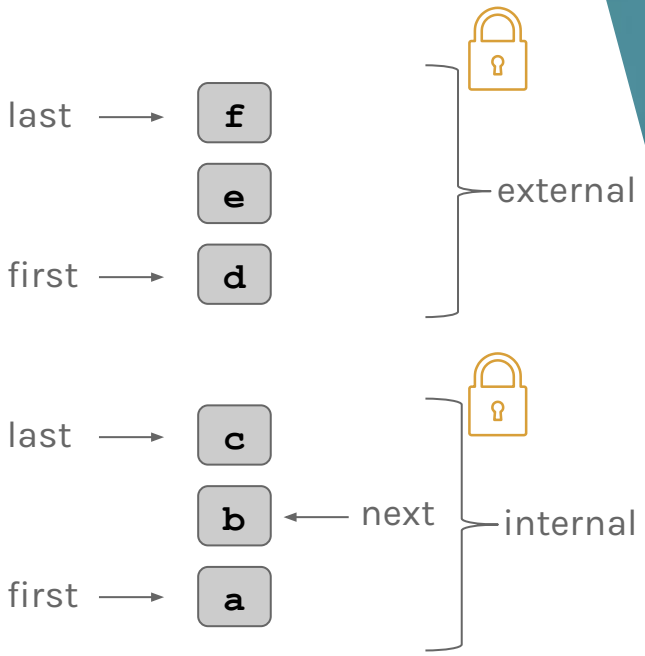
Implementation of messages:

- Two linked lists
  - External mailbox
  - Internal mailbox
- Pointer to next message to inspect
- Each mailbox protected by separate locks



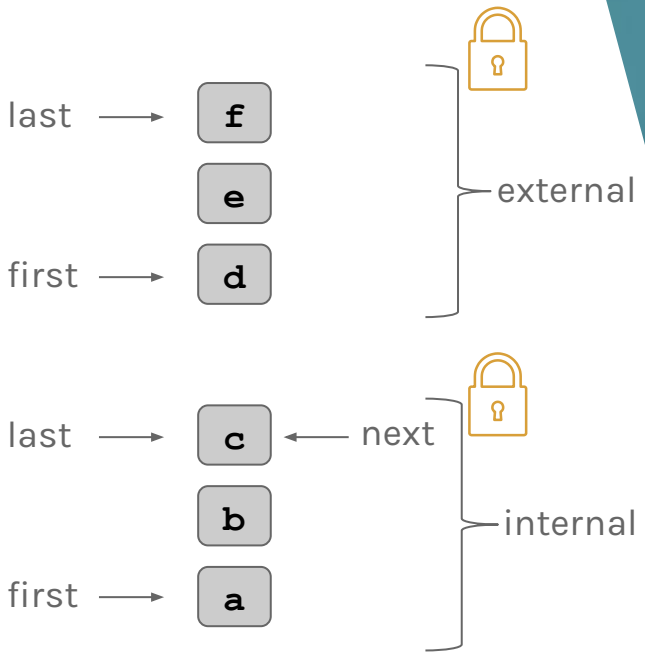
MESSAGE SIGNALS
before OTP-21

```
receive
   e -> ok
end.
```



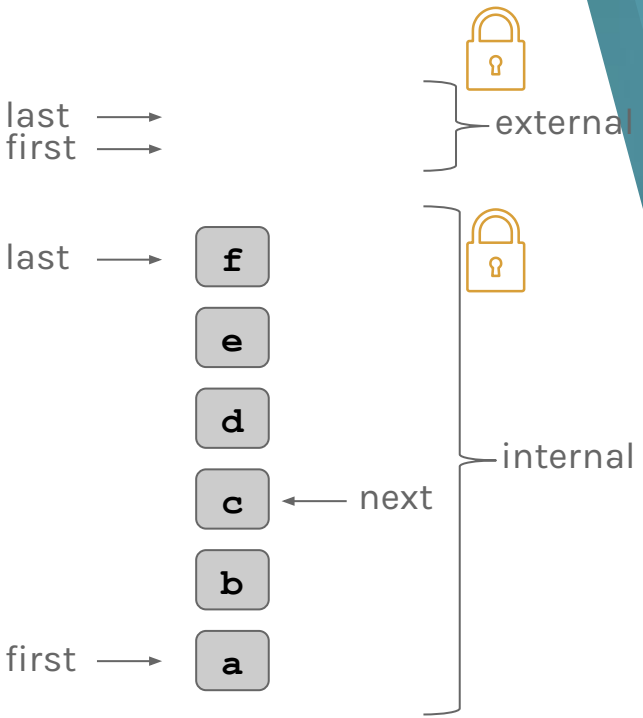
MESSAGE SIGNALS
before OTP-21

```
receive
   e -> ok
end.
```



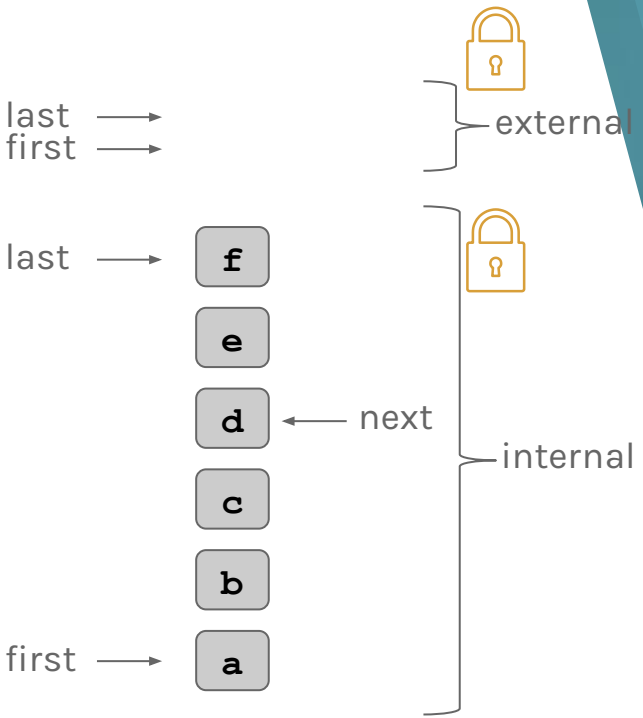
MESSAGE SIGNALS
before OTP-21

```
receive
   e -> ok
end.
```



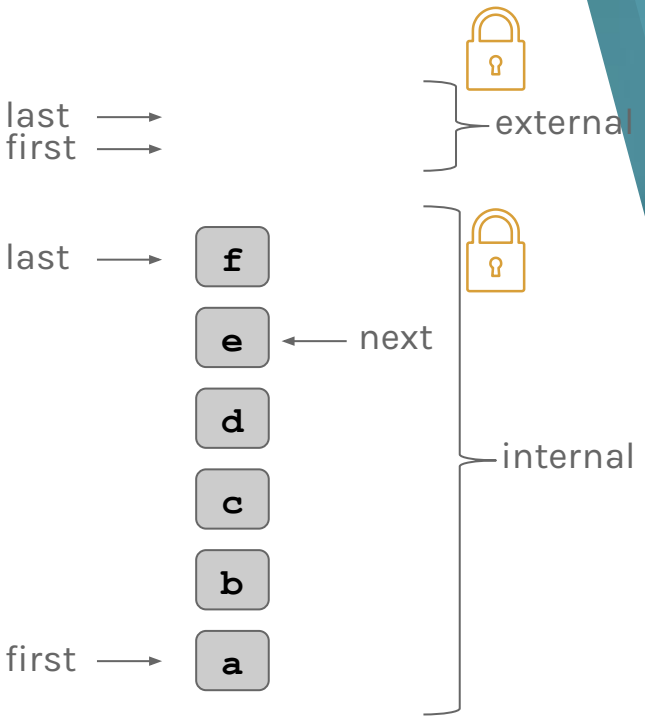
MESSAGE SIGNALS
before OTP-21

```
receive
   e -> ok
end.
```



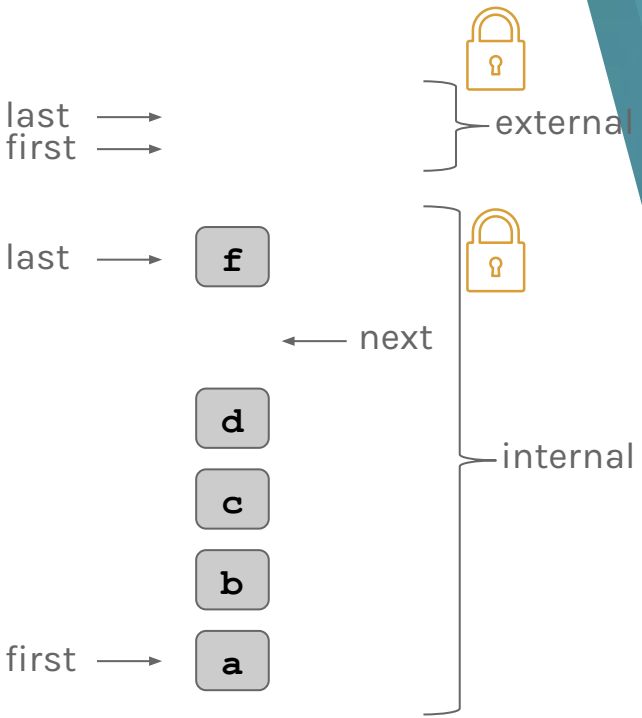
MESSAGE SIGNALS
before OTP-21

```
receive
   e -> ok
end.
```



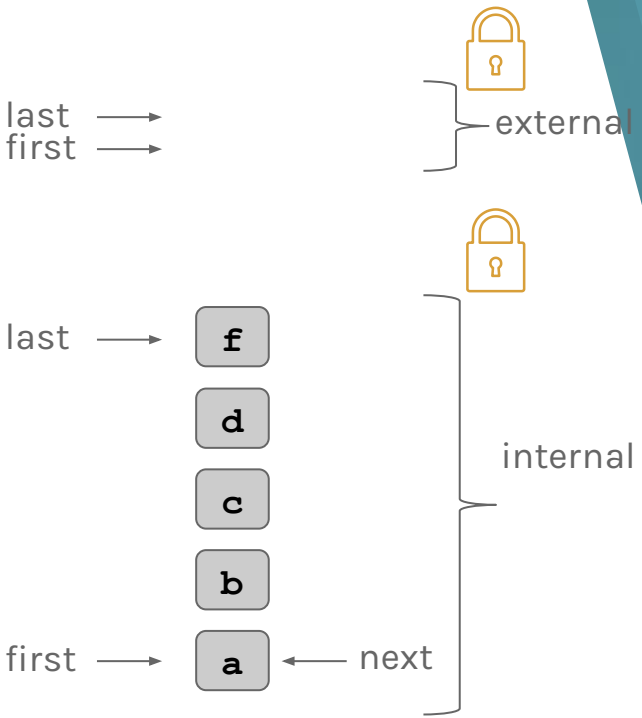
MESSAGE SIGNALS
before OTP-21

```
receive
   e -> ok
end.
```



MESSAGE SIGNALS
before OTP-21

```
receive
   e -> ok
end.
```



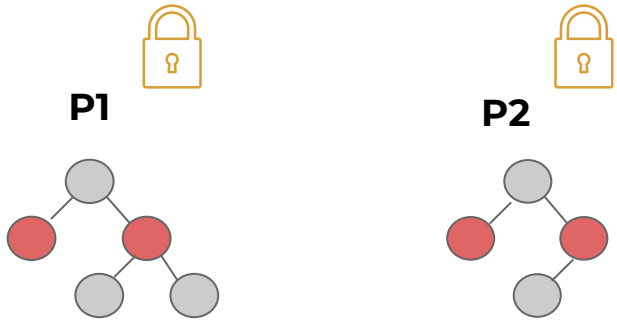
4.

LINKS/MONITORS BEFORE OTP-21

LINK/MONITOR SIGNALS
before OTP-21

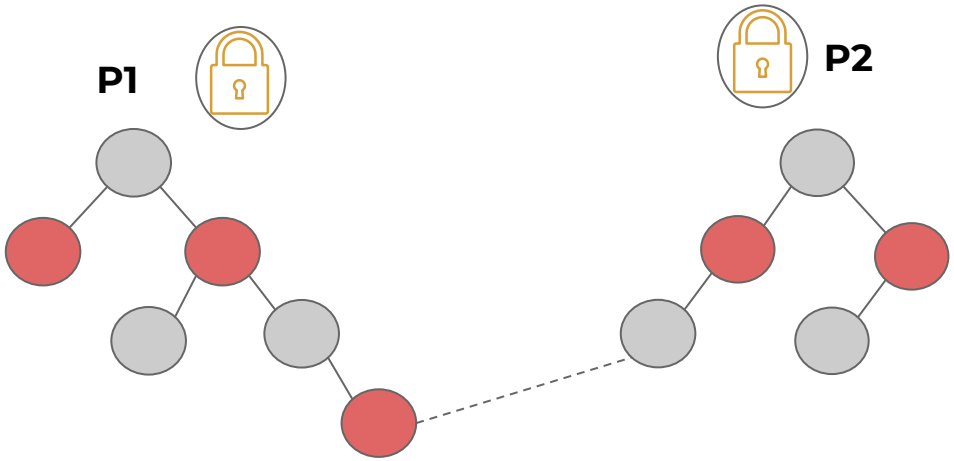
Implementation of links/monitors:

- One R/B-tree each
  - Sorted on Pid/Port/Ref
  - Contains both origins and targets
- One lock protecting links and monitors



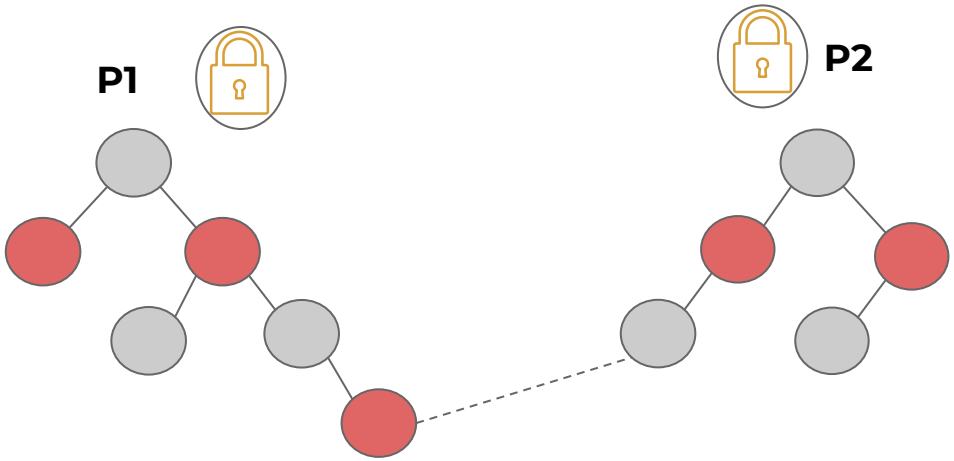
LINK/MONITOR SIGNALS
before OTP-21

```
> link(P2).
```



LINK/MONITOR SIGNALS
before OTP-21

```
> link(P2).
```



SUMMARY SIGNALS before OTP-21

- Messages
  - Two linked lists
    - External mailbox
    - Internal mailbox
- Non-message signals
  - Protected by locks
  - R/B tree for link/monitor
  - Lots of ad-hoc implementations

5.

A gen_server call

gen_server:call/2

```
gen:do_call(Process, Request, Timeout) ->
    Mref = erlang:monitor(process, Process),
    Process ! {'GEN_CALL', {self(), Mref}, Request),
    receive
        {Mref, Reply} ->
            erlang:demonitor(Mref, [flush]),
            {ok, Reply};
        {'DOWN', Mref, _, _, Reason} ->
            exit(Reason)
    after Timeout ->
            erlang:demonitor(Mref, [flush]),
            exit(timeout)
    end.
```

gen_server:call/2

```
%% Take 2 locks + 2 r/b tree inserts
Mref = erlang:monitor(process, Process),

%% Take 1 lock + linked list insert
Process ! {'GEN_CALL', {self(), Mref}, Request),

%% Take 2 locks + 2 r/b tree deletions
erlang:demonitor(Mref, [flush])
```

5 locks + 4 R/B Tree ops + 1 linked list op

R/B Tree ops become more expensive as tree grows

gen_server:call benchmark

Processes	Total Calls	Total Time (s)
1	3 000 000	6.4
10	3 000 000	6.3
20	3 000 000	7.1
50	3 000 000	13.4
1000	3 000 000	23.6
10000	3 000 000	19.2

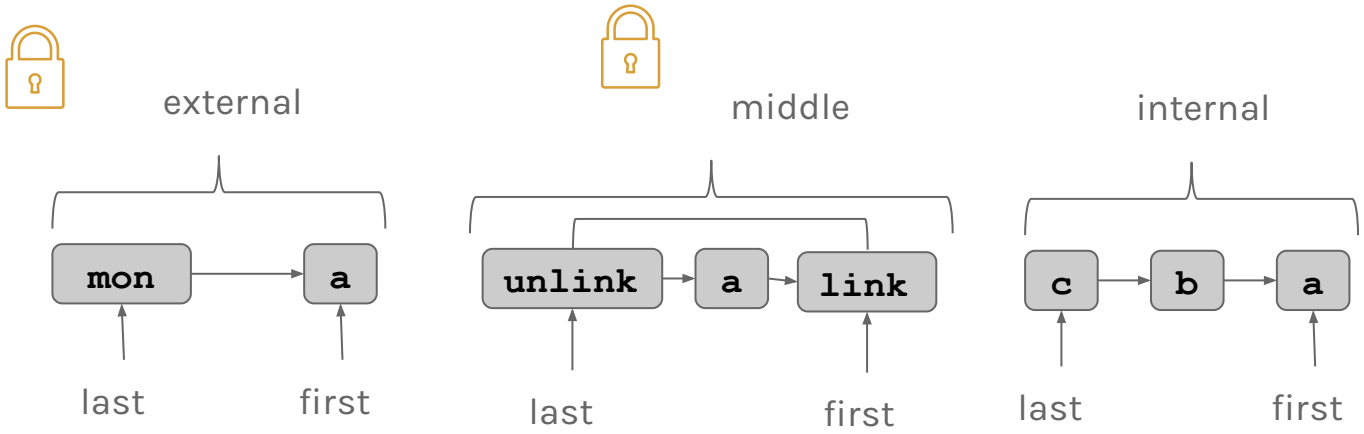
6.

SIGNALS
AFTER OTP-21

SIGNALS
after OTP-21

Implementation of signals:

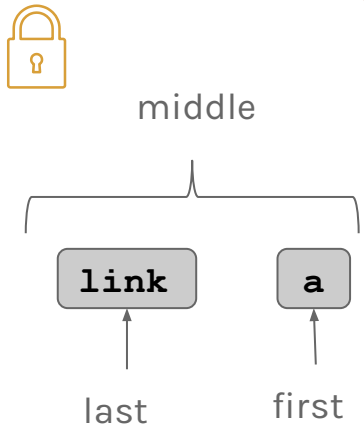
- One external queue for all signals
- One inner queue of only message signals
- One middle queue for all signals
- Skip list for non-message signals



SIGNALS
after OTP-21

Implementation of signals:

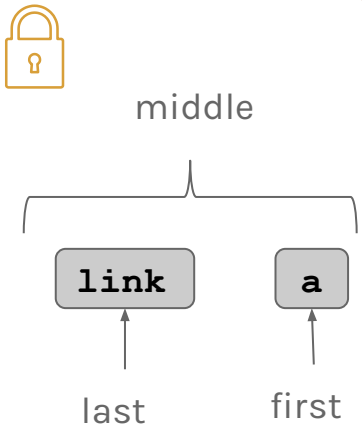
- Messages and non-message signals in outer + middle queue
- All non-message signals handled when transferred from middle to inner queue
  - link/unlink/exit
  - monitor/demonitor/down
  - group_leader
  - is_process_alive
  - process_info
  - suspend/resume
  - Trace change



SIGNALS
after OTP-21

Implementation of signals:

- Messages and non-message signals in outer + middle queue
- All non-message signals handled when transferred from middle to inner queue
- All inspection BIFs now send internal messages to the receiving process
  - Great for scalability and performance
  - Not always great for latency of those operations



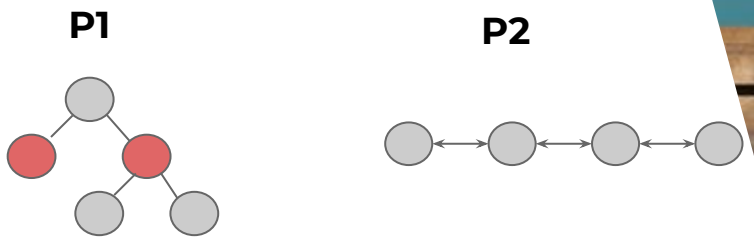
7.

MONITORS AFTER OTP-21

MONITOR SIGNALS
after OTP-21

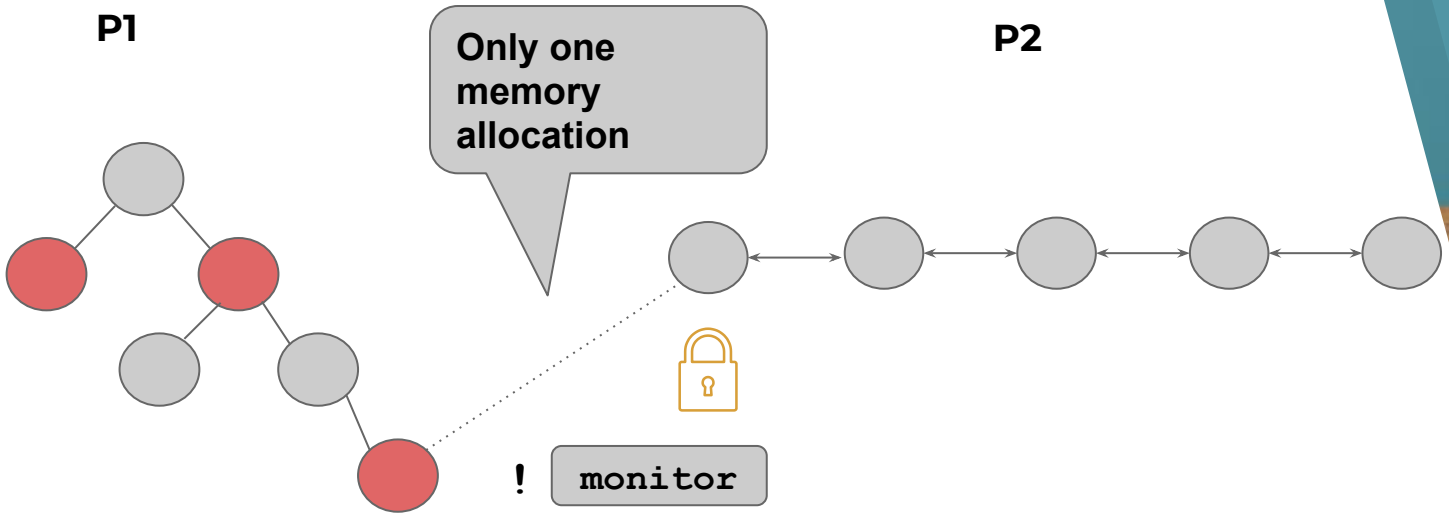
Implementation of monitors:

- One R/B-tree each
  - Sorted on Pid/Port/Ref
  - Contains only origin
- One double linked list with target monitors
- No locks!
  - Or rather only the message queue lock



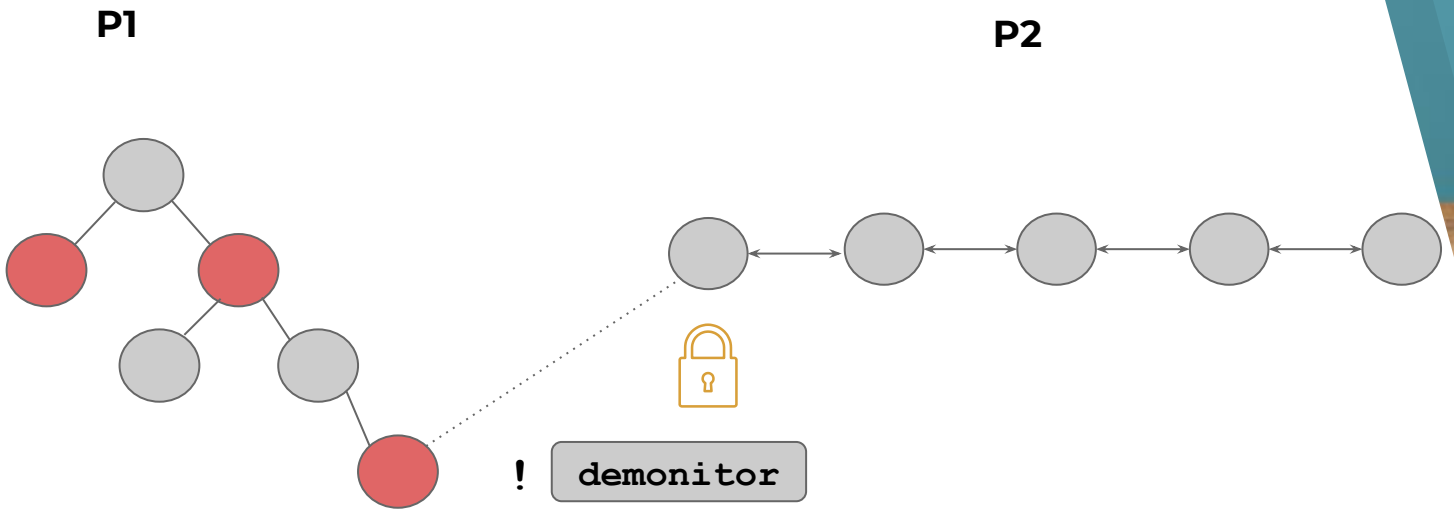
MONITOR SIGNALS
after OTP-21

```
> Ref = monitor(process, P2).
```



MONITOR SIGNALS
after OTP-21

```
> demonitor(Ref).
```

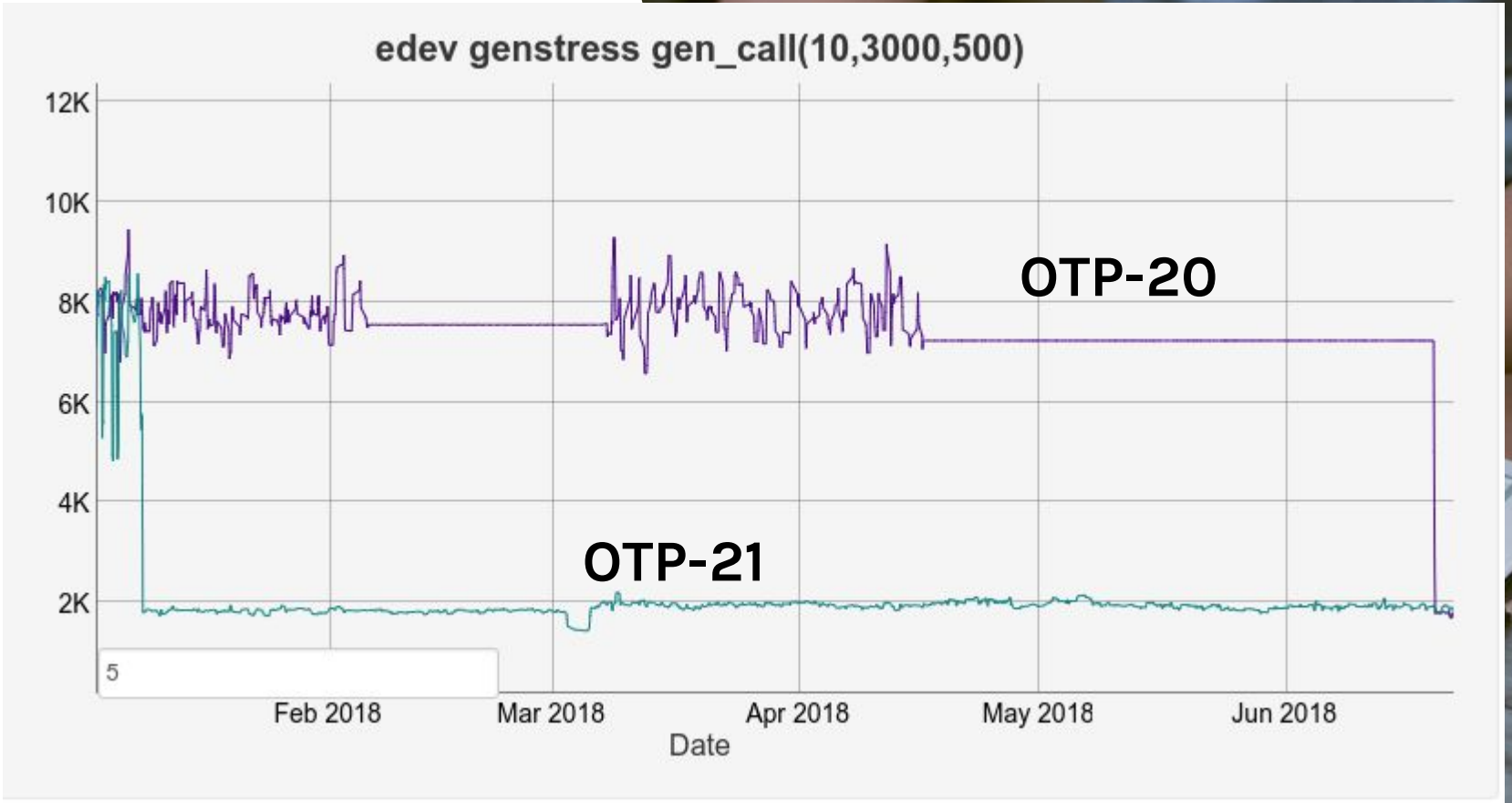


SUMMARY SIGNALS after OTP-21

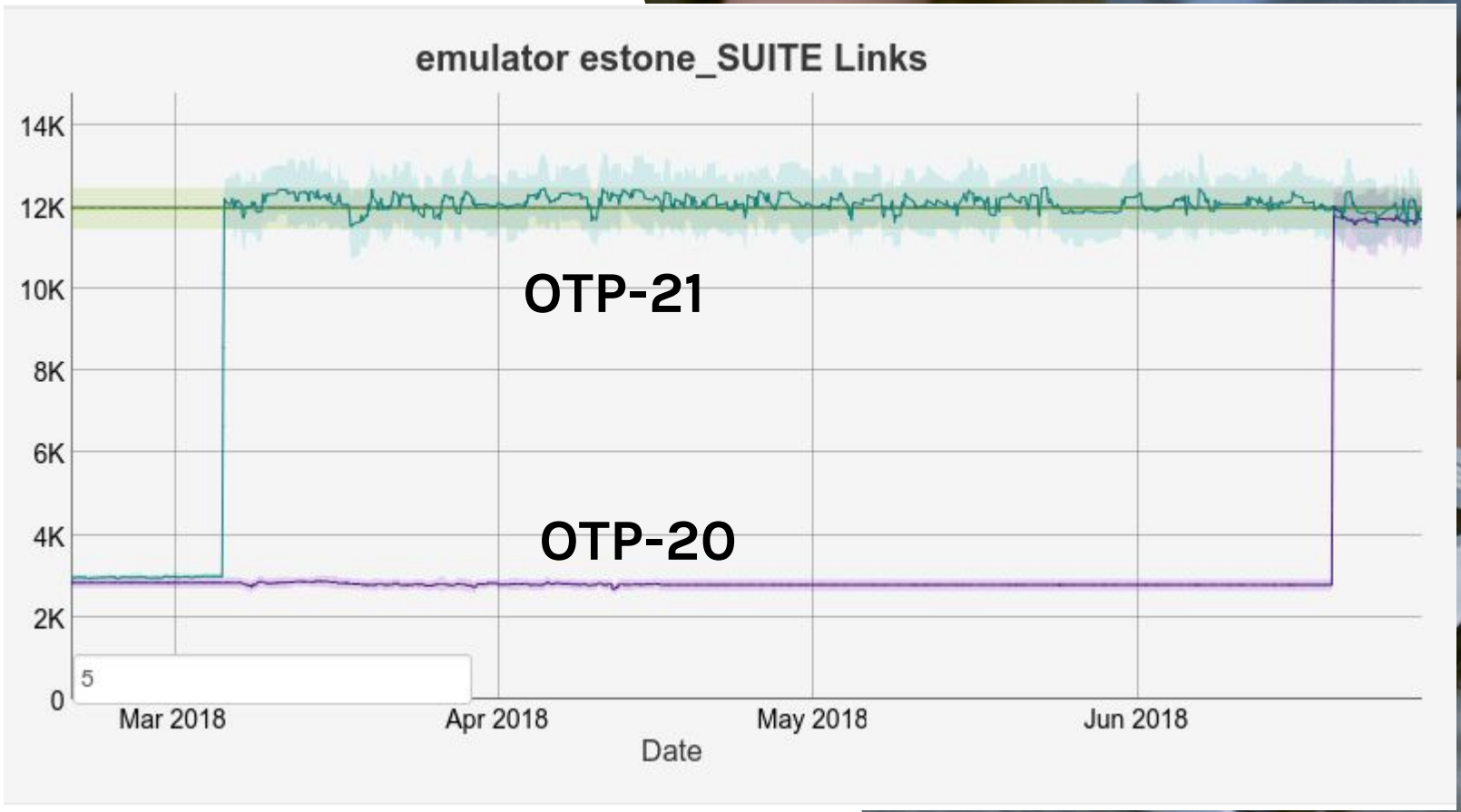
- Signals
  - Three linked lists
    - External mailbox
    - Middle mailbox
    - Internal mailbox
  - Skip list for non-message signals
- Non-message signals
  - No locks!
  - R/B tree for link/monitor, double linked list for monitor target

gen_server:call benchmark

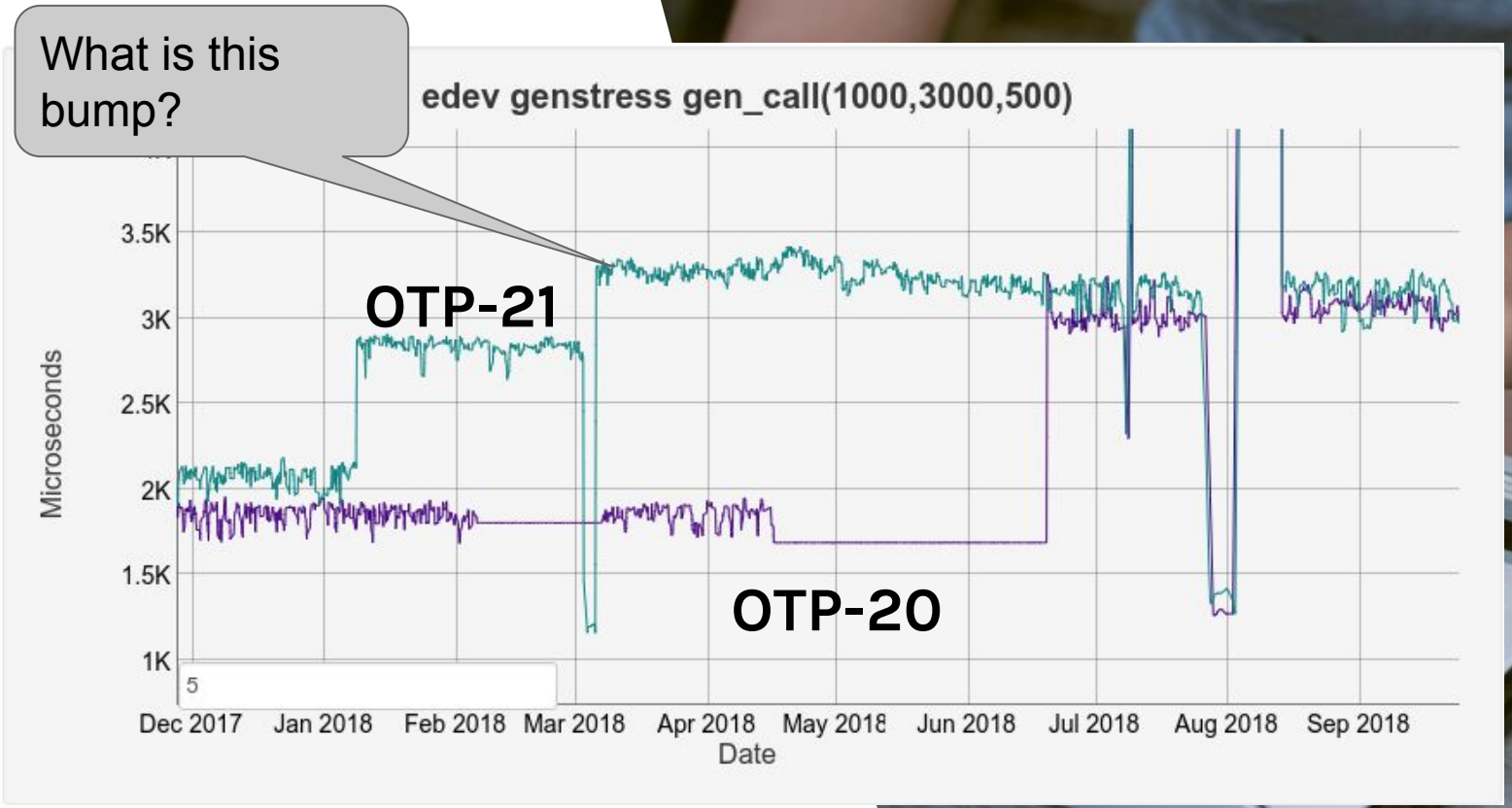
Processes	Total Calls	Total Time OTP-20 (s)	Total Time OTP-21 (s)
1	3 000 000	6.4	6.8
10	3 000 000	6.3	7.1
20	3 000 000	7.1	6.3
50	3 000 000	13.4	6.7
1000	3 000 000	23.6	8.3
10000	3 000 000	19.2	10.5



LOWER IS BETTER



HIGHER IS BETTER



HIGHER IS BETTER

8.

Combining signals

COMBINING SIGNALS

```
gen:do_call(Process, Request, Timeout) ->
    Mref = erlang:monitor(process, Process),
    Process ! {'GEN_CALL', {self(), Mref}, Request},
    receive
        {Mref, Reply} ->
            erlang:demonitor(Mref, [flush]),
            {ok, Reply};
        {'DOWN', Mref, _, _, Reason} ->
            exit(Reason)
    after Timeout ->
            erlang:demonitor(Mref, [flush]),
            exit(timeout)
    end.
```

COMBINING SIGNALS

```
Mref = erlang:monitor(process, Process),
Process ! {'GEN_CALL', {self(), Mref}, Request},
```

- Delay monitor signal until next message
  - If target process is same, combine into one signal
  - Else send signals as normal
- If process is scheduled out without sending any message, send monitor signal anyway.

“

Questions?

https://github.com/erlang/otp/blob/master/erts/emulator/beam/erl_proc_sig_queue.h