FEUERLABS

# JOBS – Load Regulation

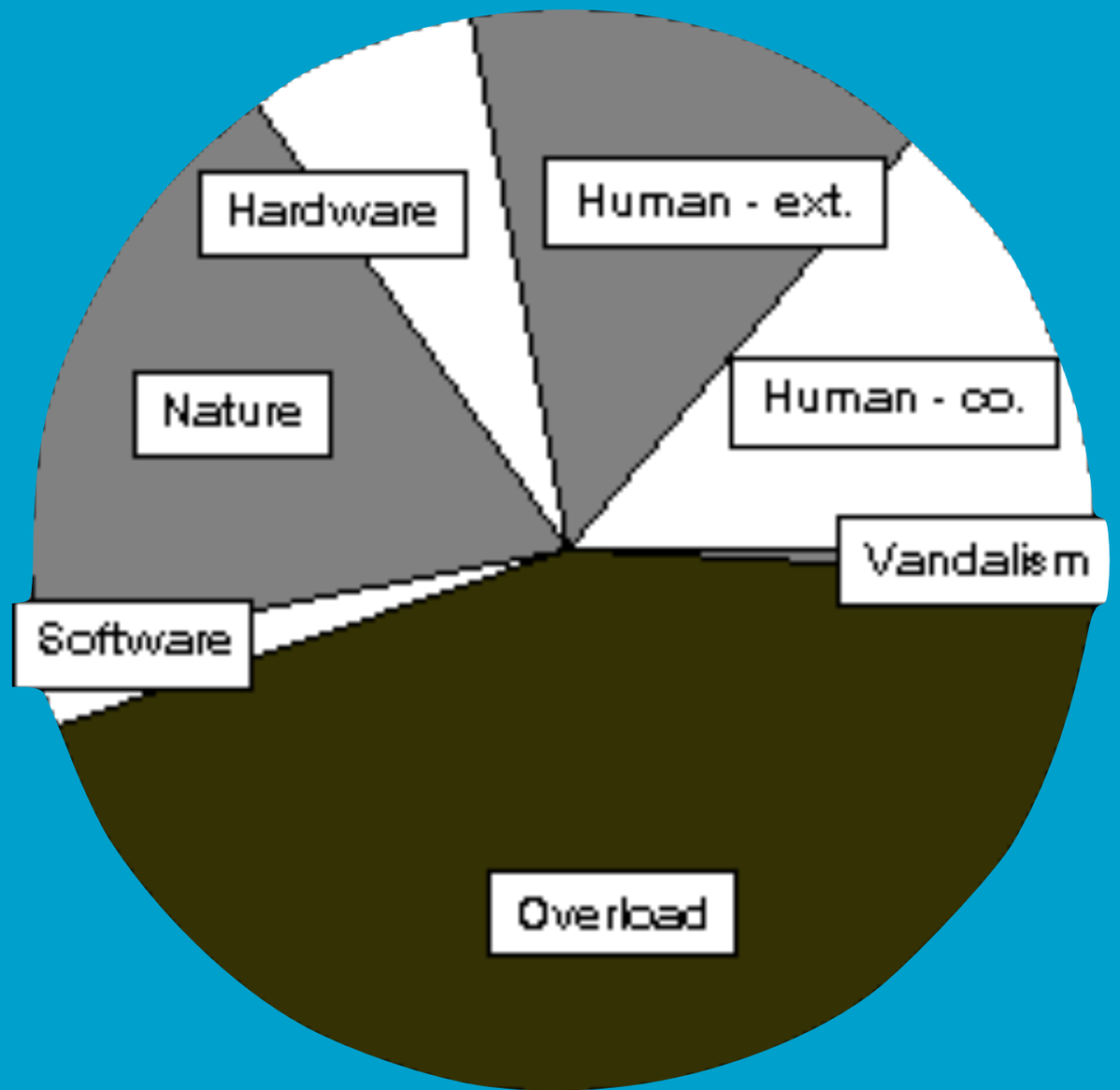
Ulf Wiger, Feuerlabs Inc

Erlang User Conference, Stockholm 14 June 2013

FEUERLABS

# Major contributor to downtime

- In Telecoms, nearly half of registered downtime is due to overload (source: FCC)



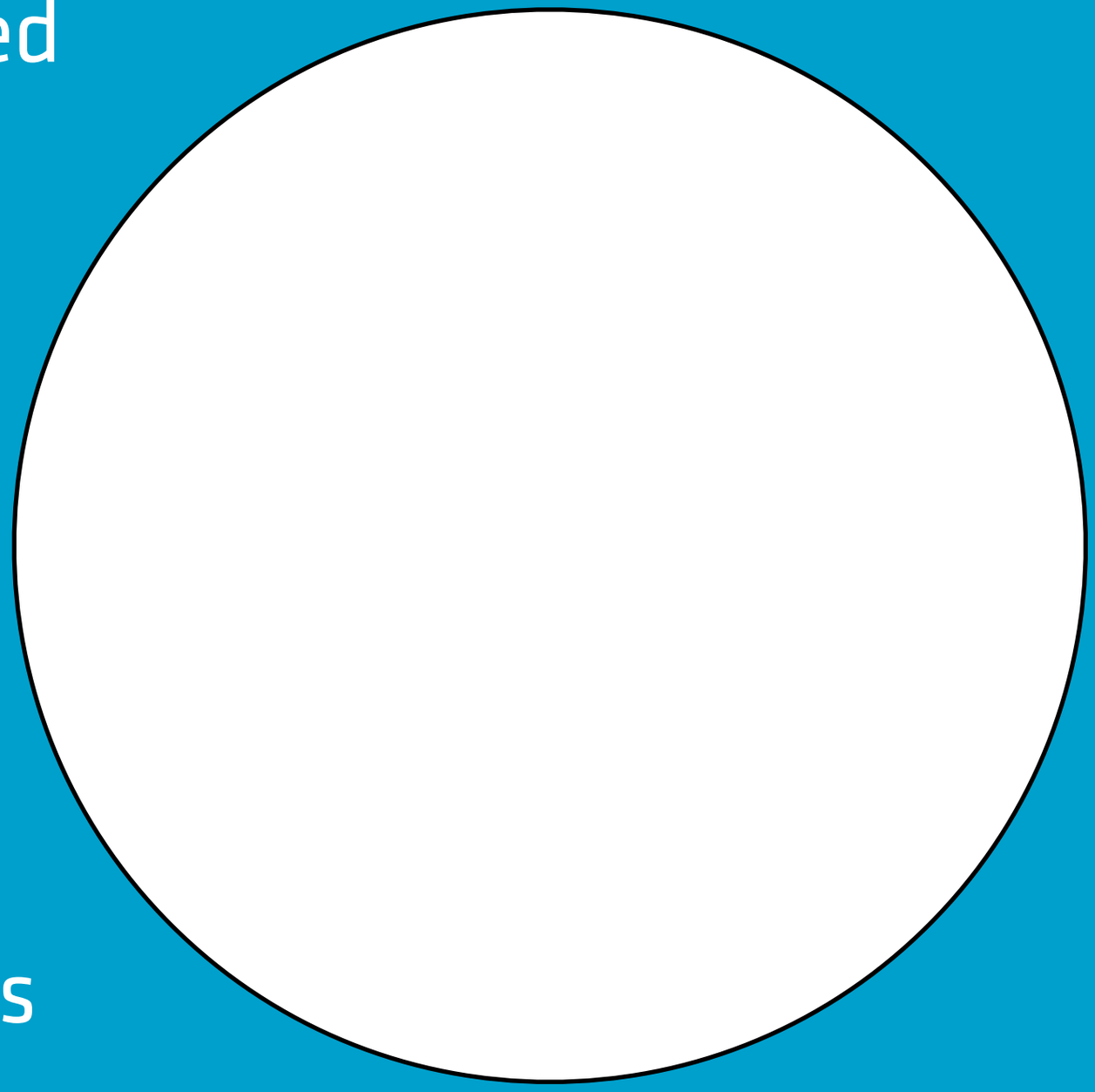
# JOBS

- First presented at SIGPLAN Erlang WS 2010
- https://github.com/uwiger/jobs.git
  https://github.com/esl/jobs.git
- Paper: jobs/doc/erlang07g-wiger.pdf
- Status: Used in anger by Feuerlabs
  (and perhaps by others)

FEUERLABS

# Regulate at the Edges

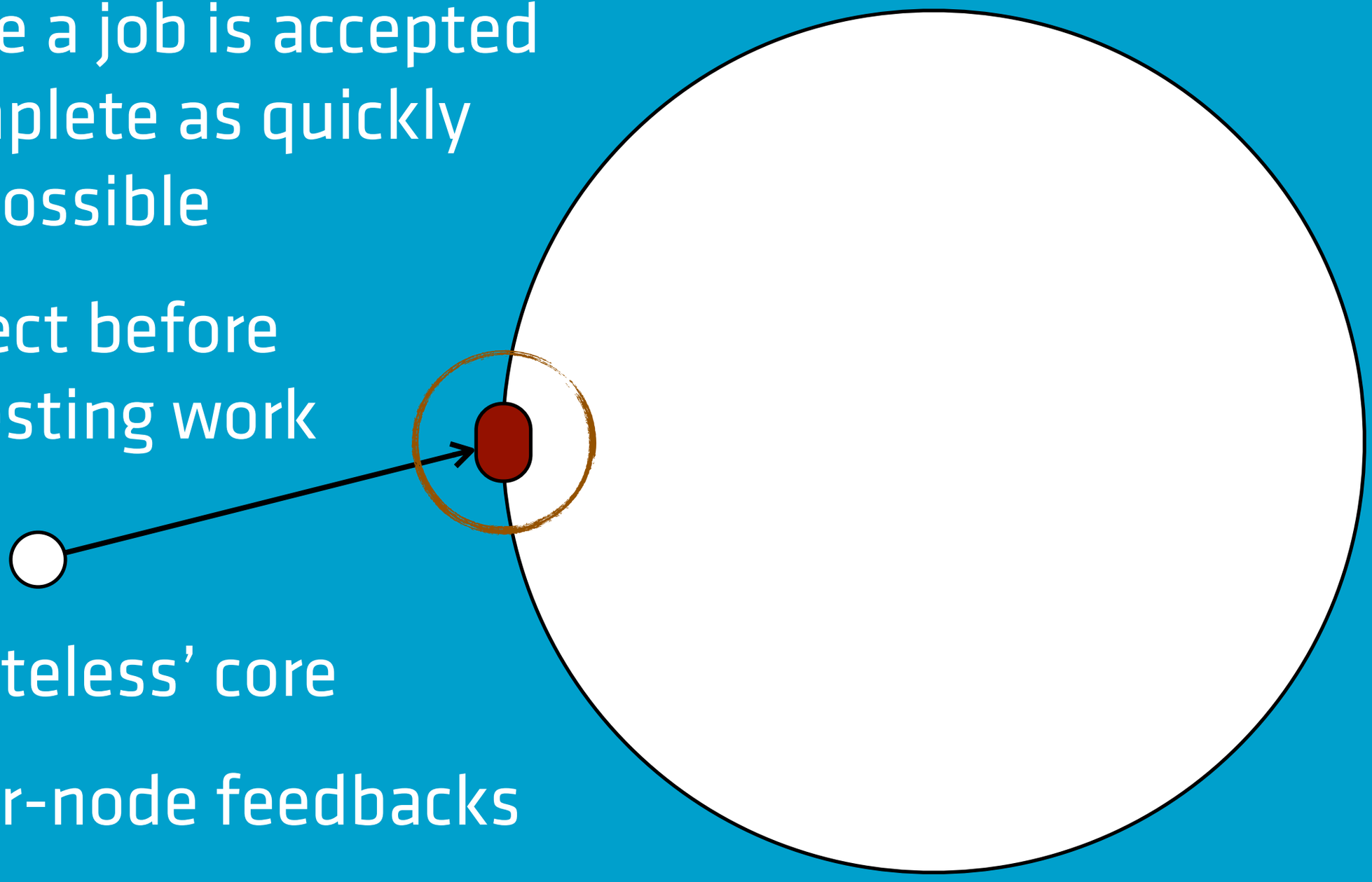
- Once a job is accepted complete as quickly as possible
- Reject before investing work



- ‘Stateless’ core
- Inter-node feedbacks

# Regulate at the Edges

- Once a job is accepted complete as quickly as possible
- Reject before investing work

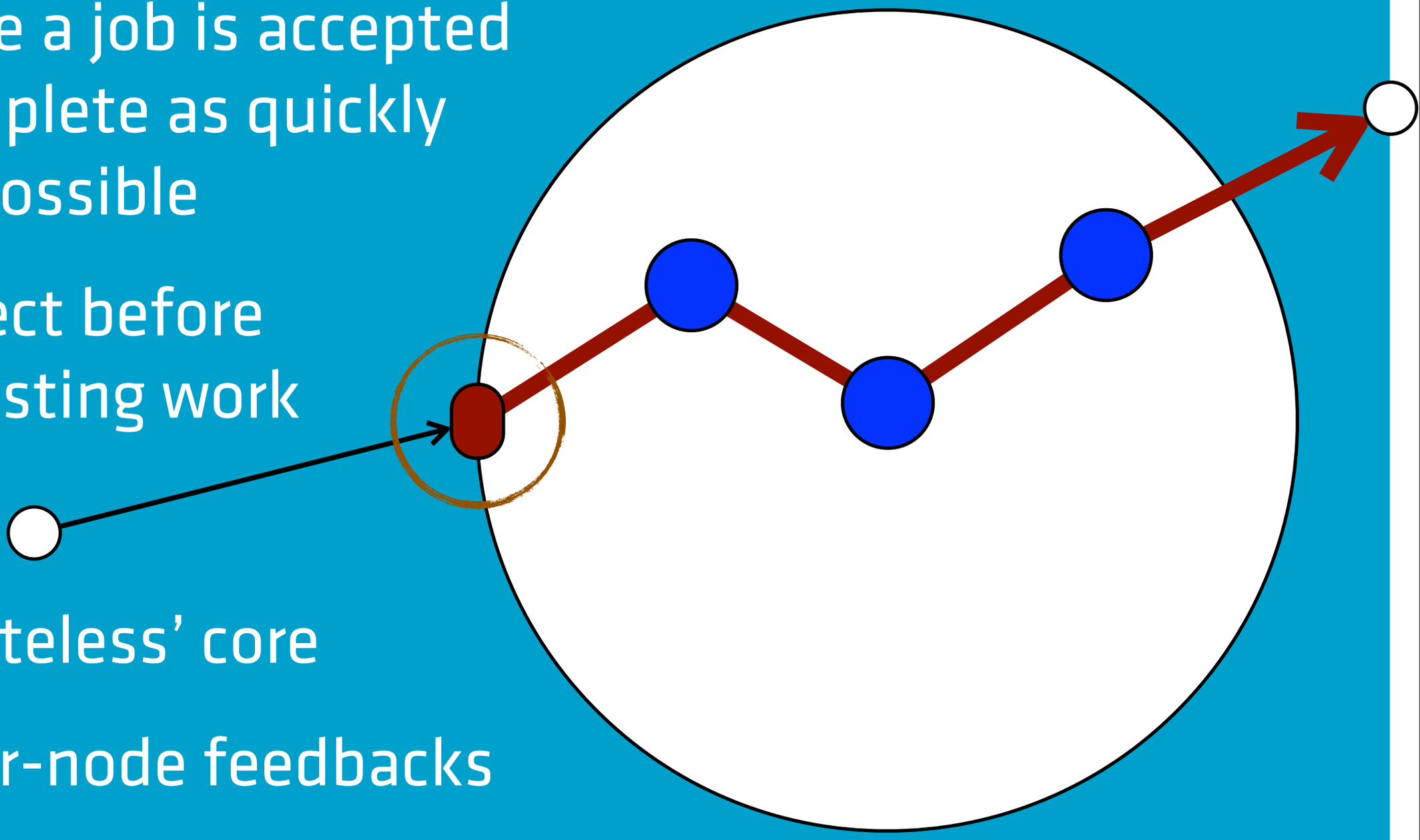


- ‘Stateless’ core
- Inter-node feedbacks

FEUERLABS

# Regulate at the Edges

- Once a job is accepted complete as quickly as possible
- Reject before investing work
- ‘Stateless’ core
- Inter-node feedbacks



FEUERLABS

# API

- `%% @spec ask(Type) -> {ok, Opaque} | {error, Reason}`
  `%% @doc Asks permission to run a job of Type.`
  `%% Returns when permission granted.`

- `%% @spec run(Type, Function::function()) -> Result`
  `%% @doc Executes Function() when permission has been granted.`

- Plus queue management functions

# Example (Feuerlabs Exosense)

```
%% @doc Handle a JSON-RPC request.
handler_session(Arg) ->
    jobs:run(
      exodm_rpc_from_web,
      fun() ->
              try
                  yaws_rpc:handler_session(
                    maybe_multipart(Arg), {?MODULE, web_rpc})
              catch
                  error:E ->
                      ...
              end
      end).
```

Queue name

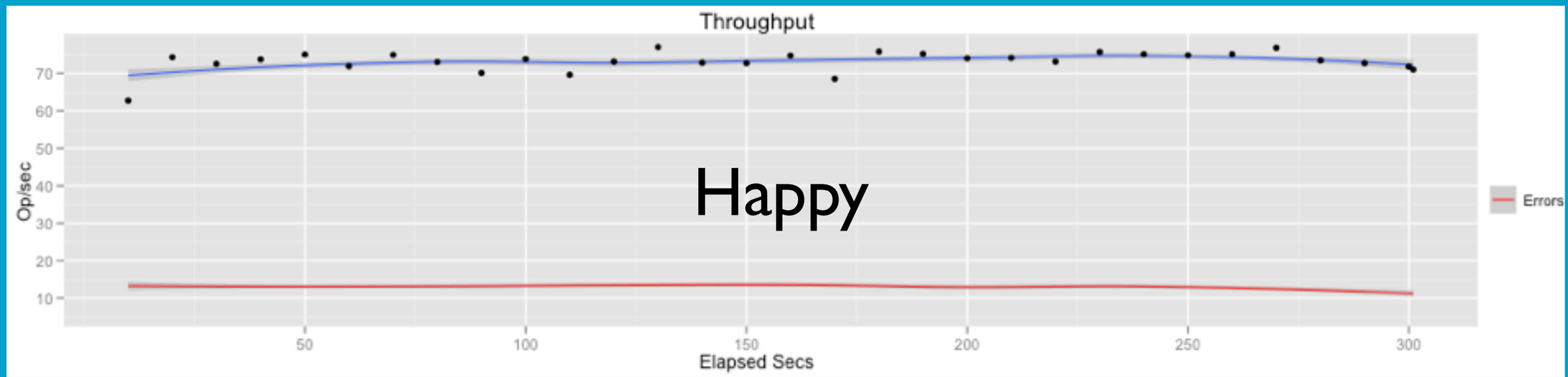
# Example (Riak prototype)

```
case jobs:ask(riak_kv_fsm) of
  {ok, JobId} ->
    try
      {ok, Pid} = riak_kv_get_fsm_sup:start_get_fsm(...),
      Timeout = recv_timeout(Options),
      wait_for_reqid(ReqId, Timeout)
    after
      jobs:done(JobId)    %% Only needed if process stays alive
    end;
  {error, rejected} ->    %% Overload!
    {error, timeout}
end
```

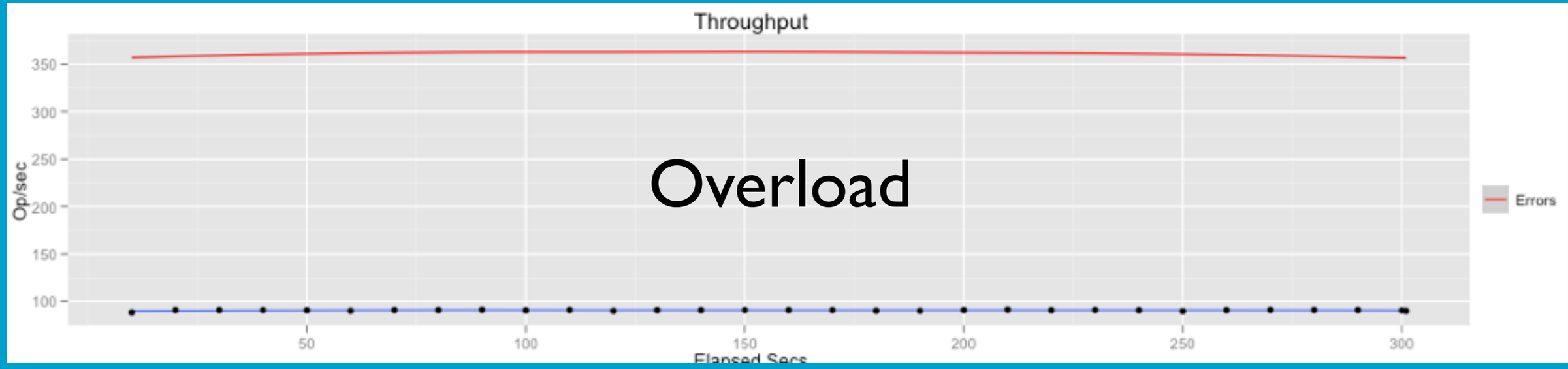
(From Dizzy Smith's EUC talk last year)

FEUERLABS

# Dizzy's Riak/Jobs experiment




Happy




Overload

FEUERLABS

# Riak/Jobs experiment, latency


FEUERLABS

# Rate-limited queues

- Fifo or lifo queue (ets-based by default)
- Incoming requests are timestamped (usec)
- For each timeslice, calculate how many jobs can be approved; calculate next timeslice
- Optional max_length and max_time limits

FEUERLABS

# Counter-limited queues

- Allow X number of concurrent jobs
- Requests are queued until a ‘slot’ available
- Counter and rate regulators can be combined
- Counter regulators can be named and referenced by other queues

# Demo – rate-limited queue

```
2> jobs:add_queue(q, [{standard_rate,1}]).
ok
3> jobs:run(q, fun() -> io:fwrite("job: ~p~n", [time()]) end).
job: {14,37,7}
ok
4> jobs:run(q, fun() -> io:fwrite("job: ~p~n", [time()]) end).
job: {14,37,8}
ok
...
5> jobs:run(q, fun() -> io:fwrite("job: ~p~n", [time()]) end).
job: {14,37,10}
ok
6> jobs:run(q, fun() -> io:fwrite("job: ~p~n", [time()]) end).
job: {14,37,11}
ok
```

# Demo – ‘Stateful’ queues

```
Eshell V5.9.2  (abort with ^G)
1> application:start(jobs).
ok
2> jobs:add_queue(q,
     [{standard_rate,1},
      {stateful,fun(init,_) -> {0,5};
                   ({call,{size,Sz},_,_},_) -> {reply, ok, {0,Sz}};
                   ({N,Sz},_) -> {N, {(N+1) rem Sz,Sz}}
                end}]).
ok
```

The fun runs in the server – must be fast!

Fun’s result is passed to the client

# Demo – ‘Stateful’ queues 2

```
3> jobs:run(q,fun(Opaque) -> jobs:job_info(Opaque) end).
0
4> jobs:run(q,fun(Opaque) -> jobs:job_info(Opaque) end).
1
5> jobs:run(q,fun(Opaque) -> jobs:job_info(Opaque) end).
2
6> jobs:run(q,fun(Opaque) -> jobs:job_info(Opaque) end).
3
7> jobs:run(q,fun(Opaque) -> jobs:job_info(Opaque) end).
4
8> jobs:run(q,fun(Opaque) -> jobs:job_info(Opaque) end).
0
9> jobs:run(q,fun(Opaque) -> jobs:job_info(Opaque) end).
1
```

# Demo – ‘Stateful’ queues 3

Resize the ‘pool’

```
10> jobs:ask_queue(q, {size,3}).
ok
11> jobs:run(q,fun(Opaque) -> jobs:job_info(Opaque) end).
0
12> jobs:run(q,fun(Opaque) -> jobs:job_info(Opaque) end).
1
13> jobs:run(q,fun(Opaque) -> jobs:job_info(Opaque) end).
2
14> jobs:run(q,fun(Opaque) -> jobs:job_info(Opaque) end).
0
...
```

# Producers (load *generation*)

- Rate- or counter limited dispatch of a predefined function
- A new process spawned for each job

# Demo – Producers

```
Eshell V5.9.2  (abort with ^G)

1> application:start(jobs).
ok
2> jobs:add_queue(p,
  [{producer, fun() -> io:fwrite("job: ~p~n",[time()]) end},
   {standard_rate,1}]).
job: {14,33,51}
ok
3> job: {14,33,52}
job: {14,33,53}
job: {14,33,54}
job: {14,33,55}
...
```

# Demo – Passive queues

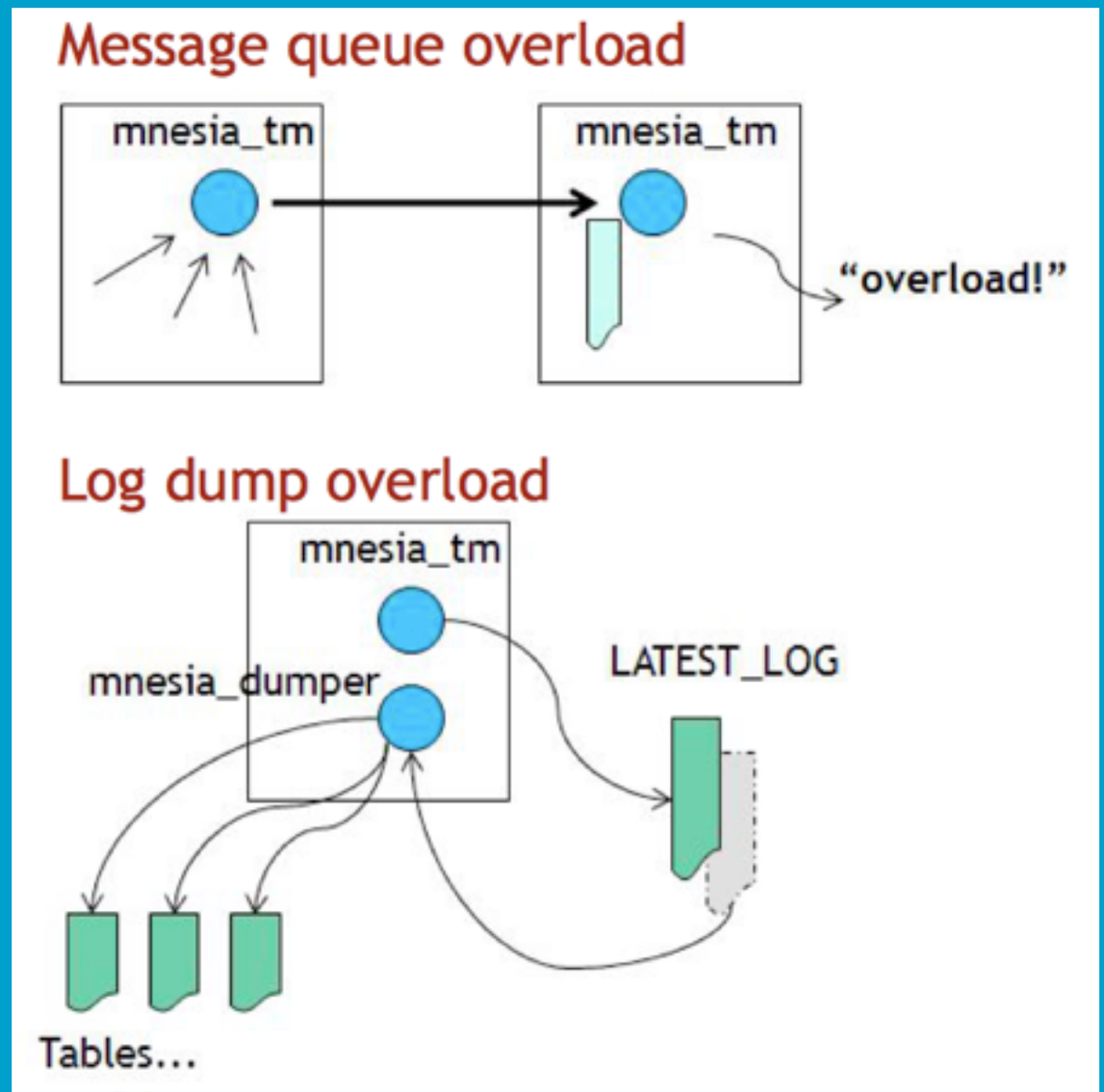
```
2> jobs:add_queue(q,[passive]).
ok
3> Fun = fun() -> io:fwrite("~p starting...~n",[self()]),
3>                Res = jobs:dequeue(q, 3),
3>                io:fwrite("Res = ~p~n", [Res])
3>        end.
#Fun<erl_eval.20.82930912>
4> jobs:add_queue(p, [{standard_counter,3},{producer,Fun}]).
<0.47.0> starting...
<0.48.0> starting...
<0.49.0> starting...
ok
5> jobs:enqueue(q, job1).
Res = [{113214444910647,job1}]
ok
<0.54.0> starting...
```

# Queue status

```
(a@uwair)1> jobs:queue_info(q).
{queue,[{name,q},
        {mod,jobs_queue},
        {type,fifo},
        {group,undefined},
        {regulators,[{rr,[{name,{rate,q,1}},
                          {rate,{rate,[{limit,1},
                                       {preset_limit,1},
                                       {interval,1.0e3},
                                       {modifiers,
                                        [{cpu,10},{memory,10}]},
                                       {active_modifiers,[]}
                                      ]}}]}]},
        {max_time,undefined},
        {max_size,undefined},
        {latest_dispatch,113216378663298},
        {approved,4},
        {queued,0},
        ...,
        {stateful,undefined},
        {st,{st,45079}}]}
```

# Distributed overload

- Mnesia sends 'overload' events only on the node where overload was detected
- The cause of the overload may well be on other nodes
- Feedback modifiers in JOBS lower the rate of relevant job queues.



FEUERLABS

# Mnesia overload sampler

```
init(Opts) ->
    mnesia:subscribe(system),
    Levels = proplists:get_value(levels, Opts, default_levels()
    {ok, #st{levels = Levels}}.

default_levels() ->
    {seconds, [{0,1}, {30,2}, {45,3}, {60,4}]}.

handle_msg({mnesia_system_event, {mnesia,{dump_log,_}}}, _T, S) ->
    {log, true, S};

handle_msg({mnesia_system_event,{mnesia_tm, message_queue_len, _}}, _T, S) ->
    {log, true, S};

handle_msg(_, _T, S) ->
    {ignore, S}.

sample(_T, S) ->
    {is_overload(), S}.

calc(History, #st{levels = Levels} = S) ->
    {jobs_sampler:calc(time, Levels, History), S}.
```

# CPU load sampler

```
init(Opts) ->
    cpu_sup:util([per_cpu]),  % first return value is rubbish
    Levels = proplists:get_value(levels, Opts, default_levels()),
    {ok, #st{levels = Levels}}.

default_levels() -> [{80,1},{90,2},{100,3}].

sample(_Timestamp, #st{} = S) ->
    Result = case cpu_sup:util([per_cpu]) of
                 ...
             end,
    {Result, S}.

calc(History, #st{levels = Levels} = St) ->
    L = jobs_sampler:calc(value, Levels, History),
    {L, St}.
```

FEUERLABS

# Future work

- Code cleanup, Q&A, optimizations
- Add more interesting queueing algorithms?
- Sliding window for rate estimation (experimental)
- Multiple Jobs servers (scalability)