Scaling Up from 1000 to 10 Nodes

Anton Lavrik

Alert Logic, Inc.

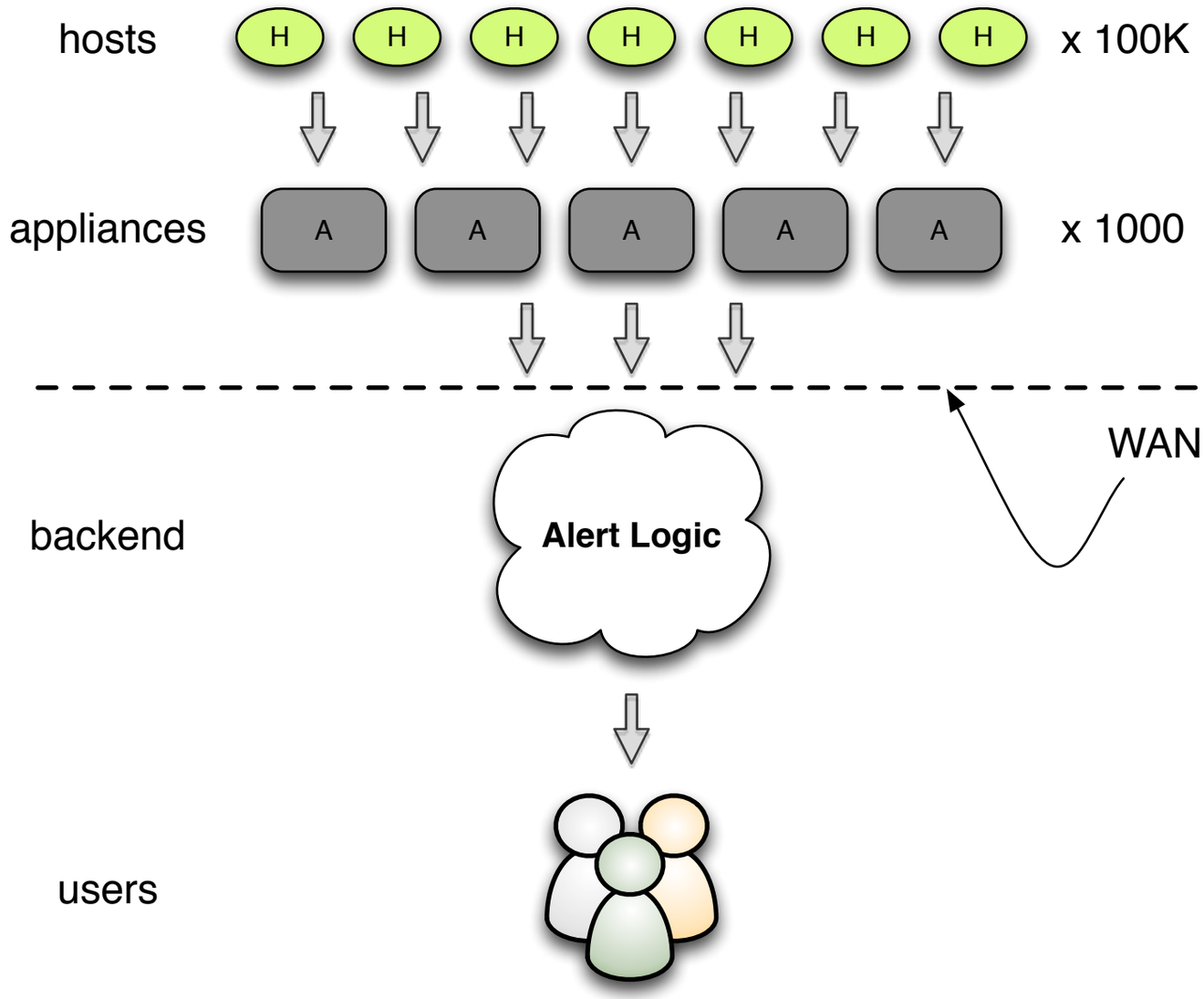
Alert Logic vs Twitter: daily numbers (2012)

	Alert Logic	Twitter
Events received	11.5B	0.5B
Volume received	5TB	70GB
Logs produced	?	100TB

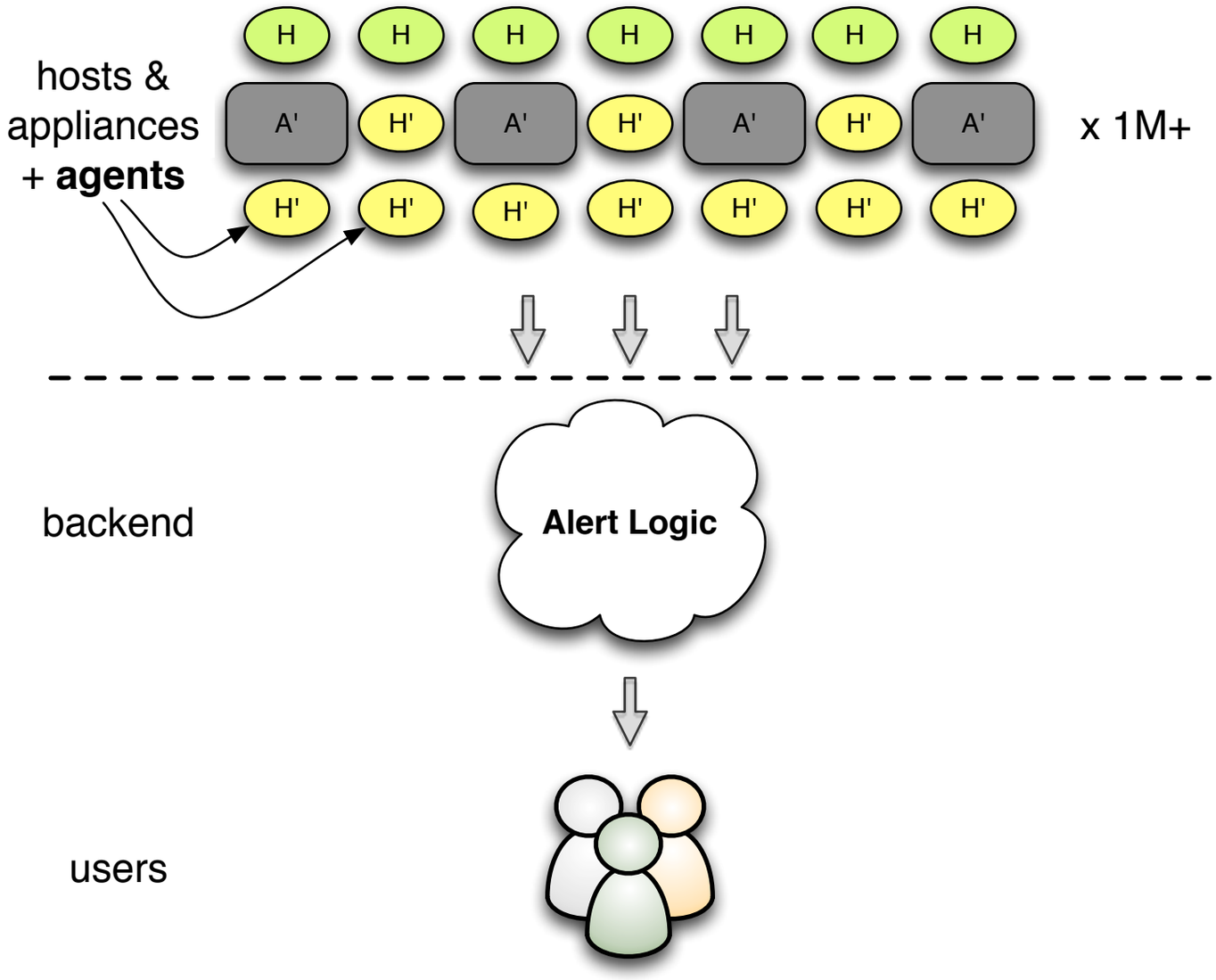
Alert Logic vs Twitter: daily numbers (2012)

	Alert Logic	Twitter
Events received	11.5B	0.5B
Volume received	5TB	70GB
Logs produced	10GB	100TB

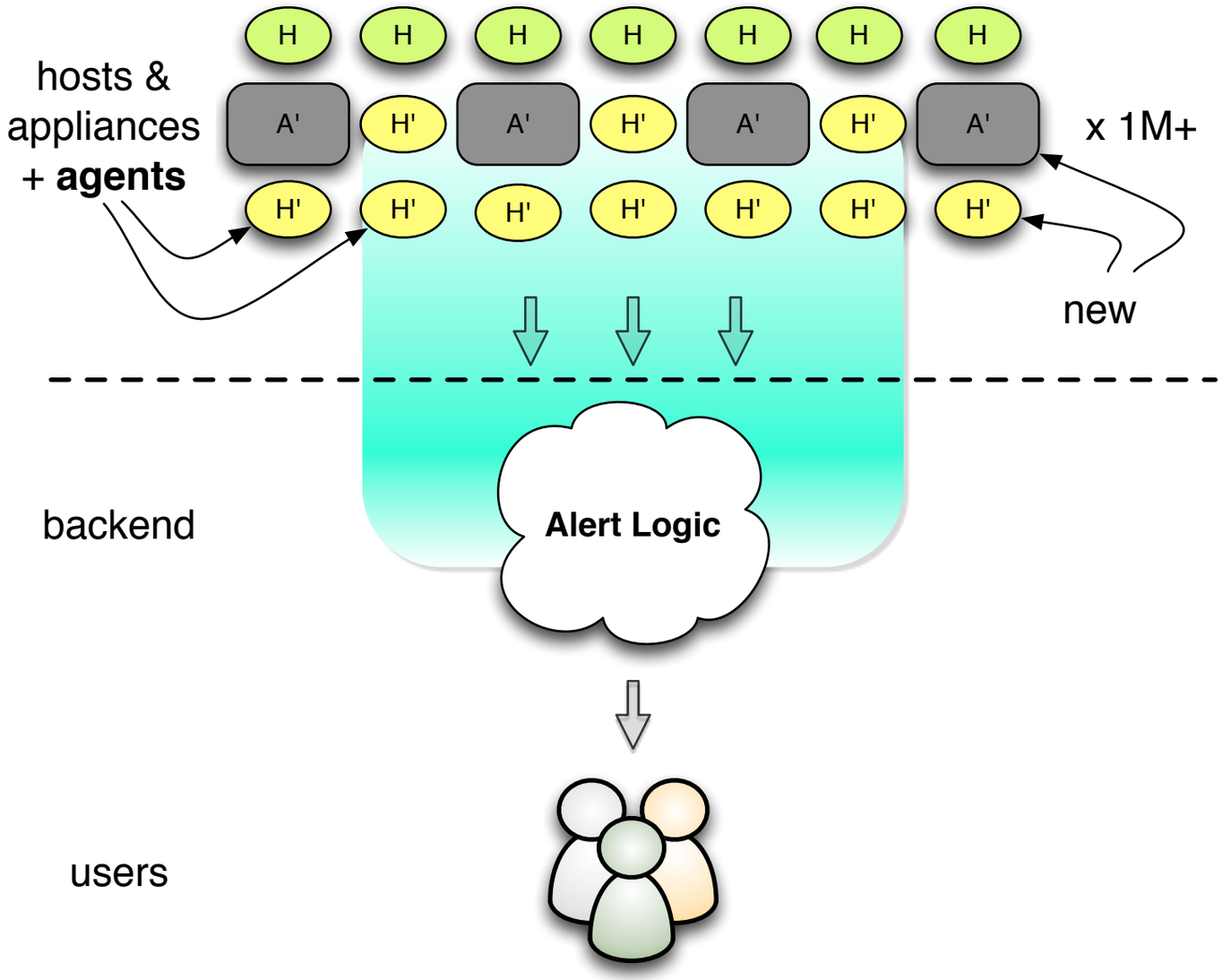
Log collection: before rewrite



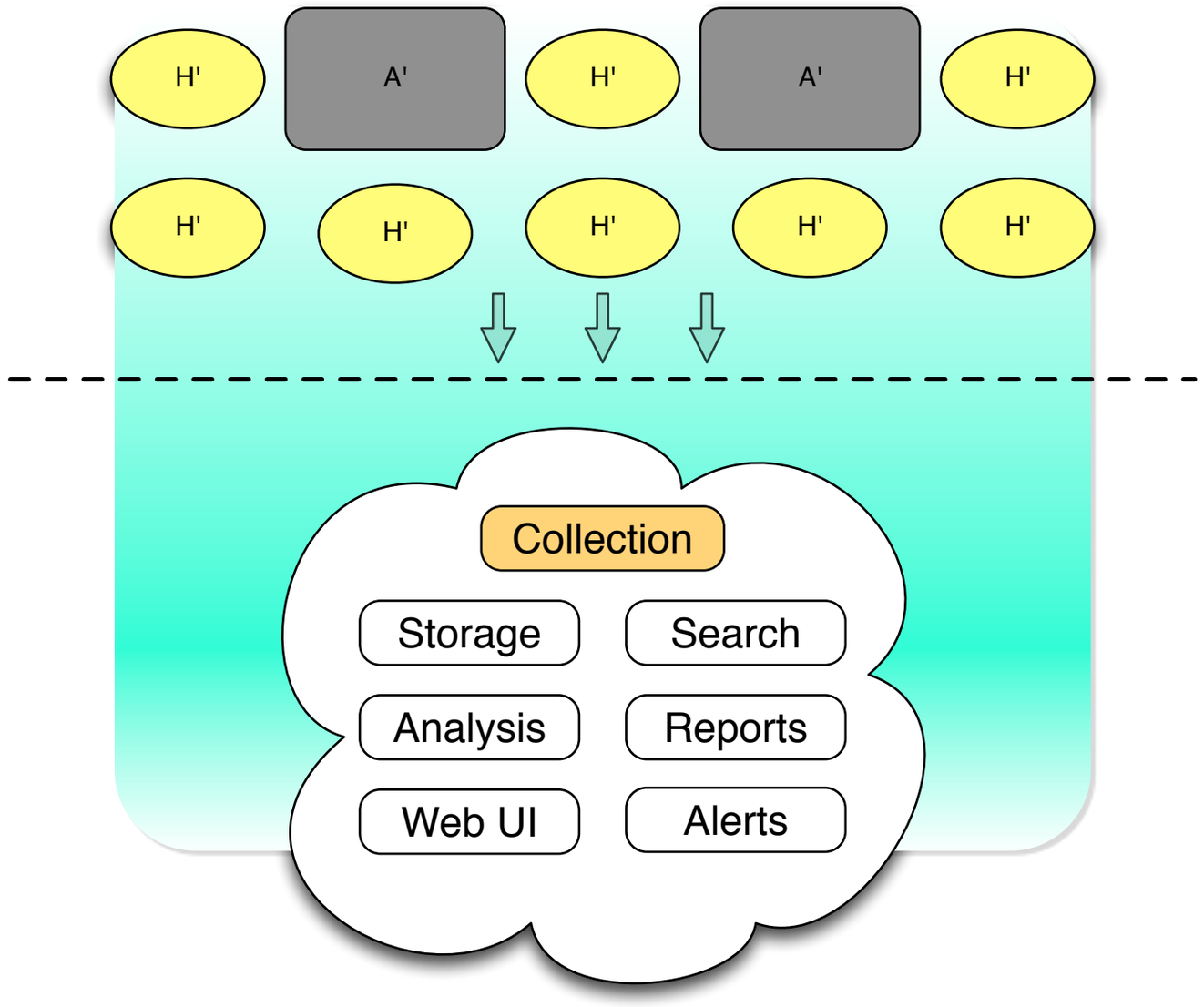
Log collection: goal



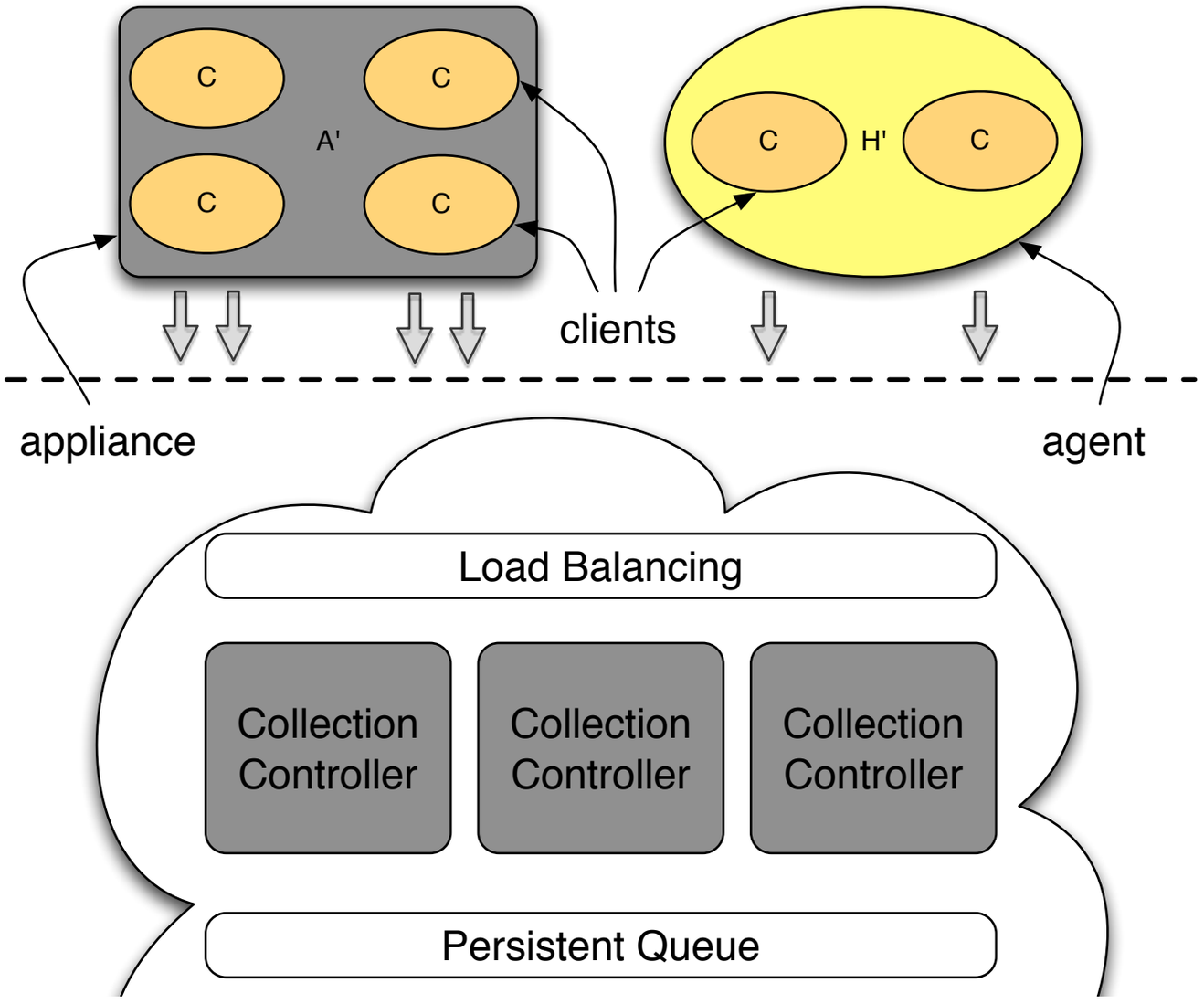
Log collection: goal



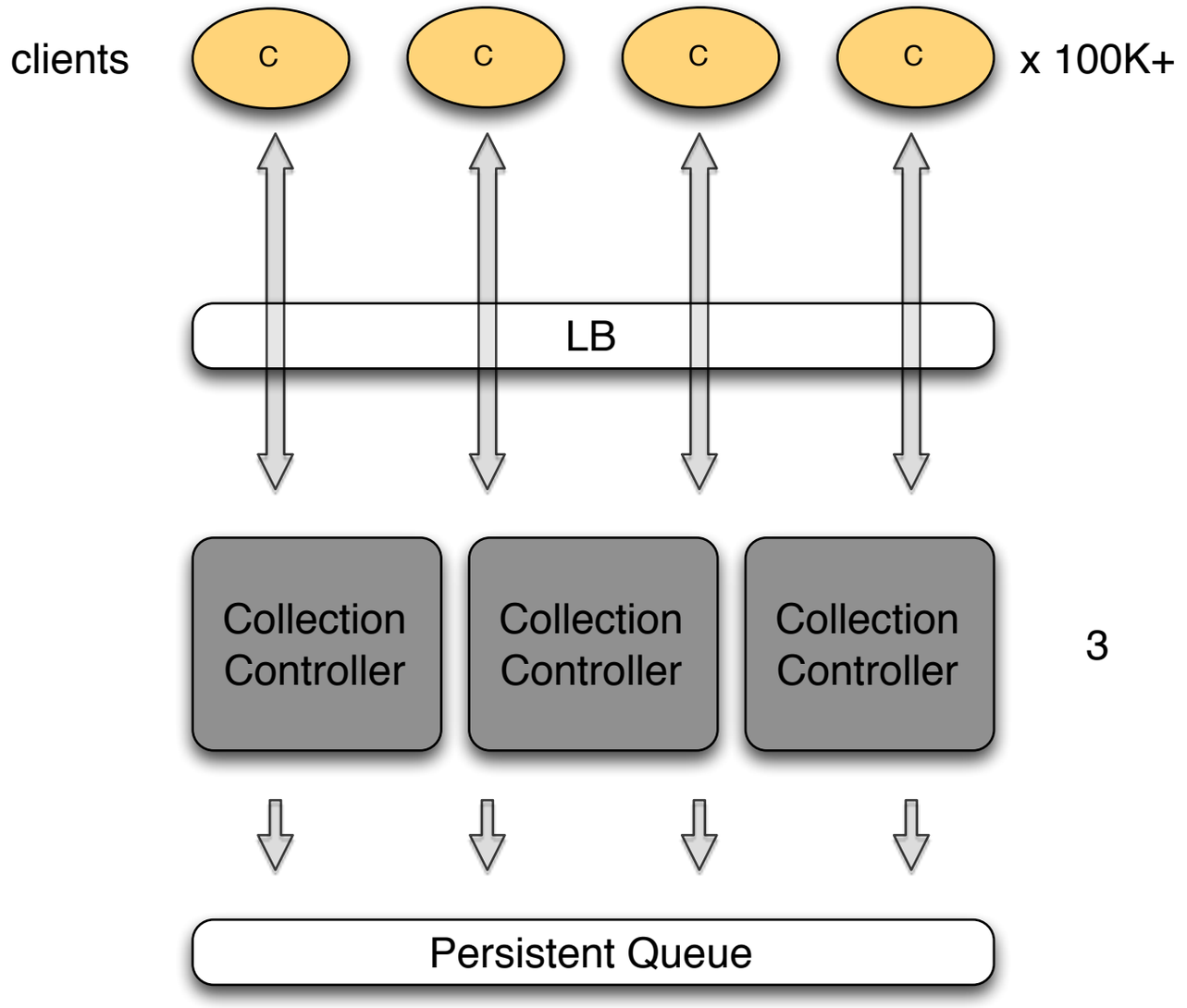
Log collection: zooming in



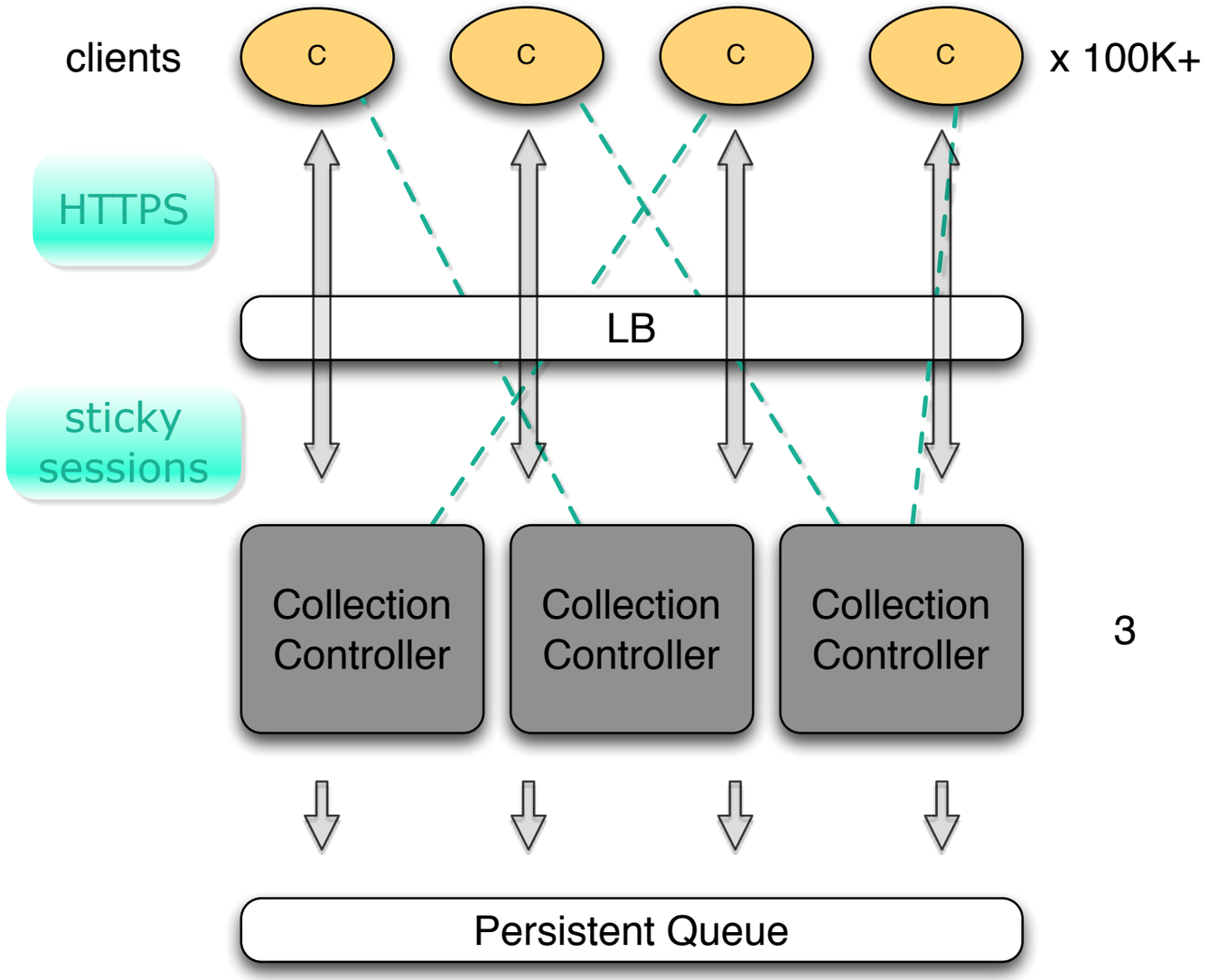
Log collection: zooming in



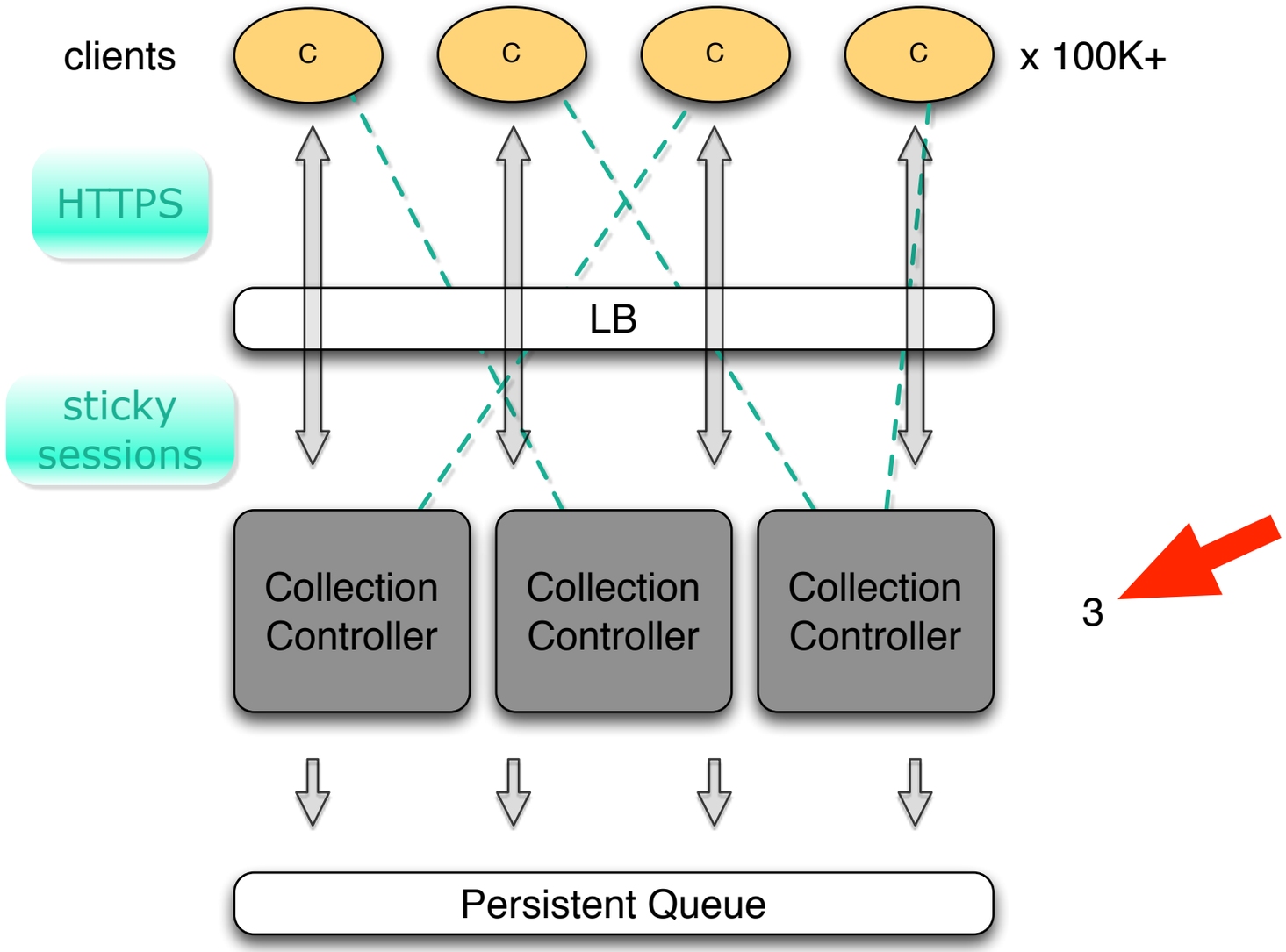
Log collection: zooming in



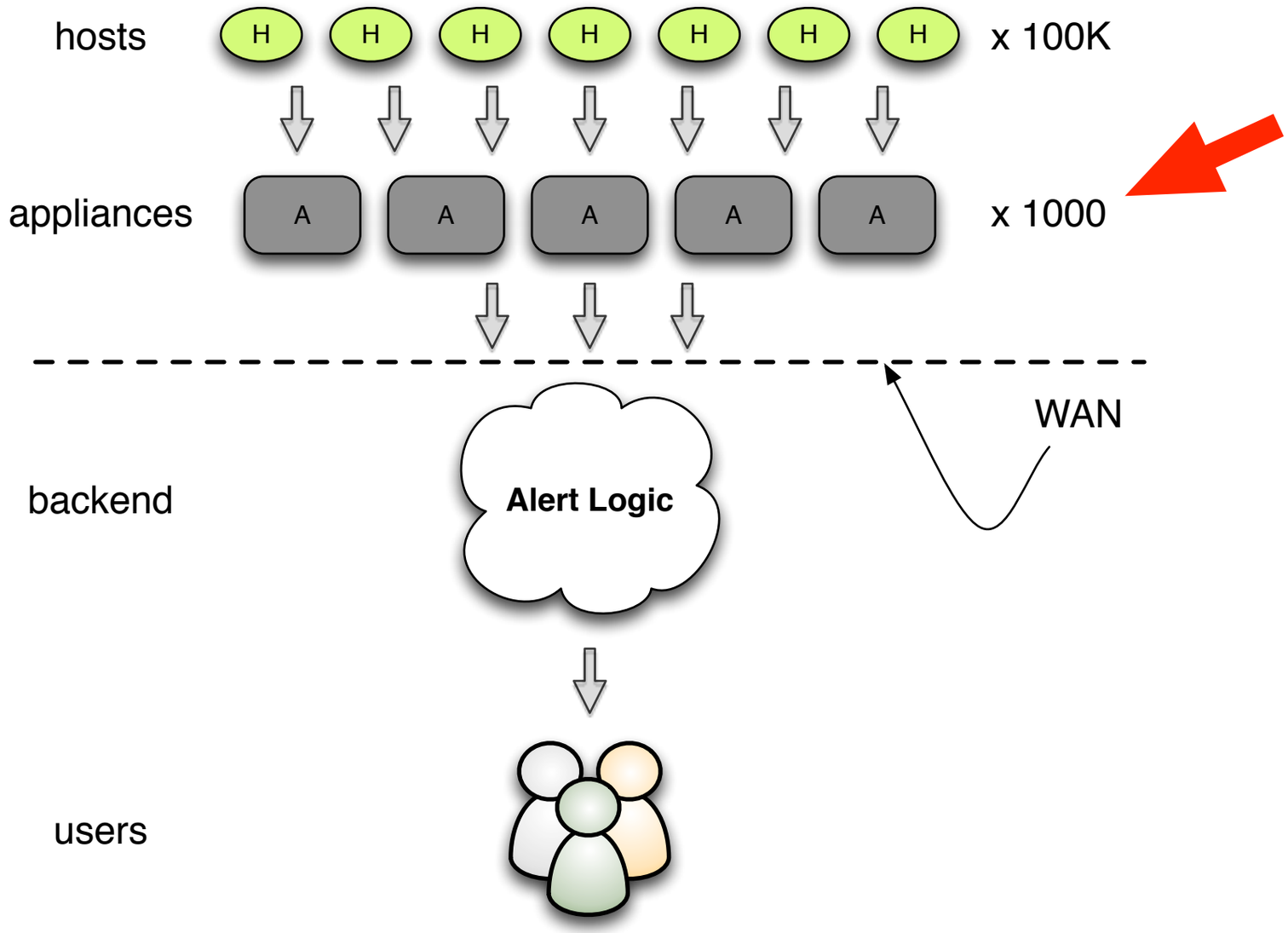
Log collection: zooming in



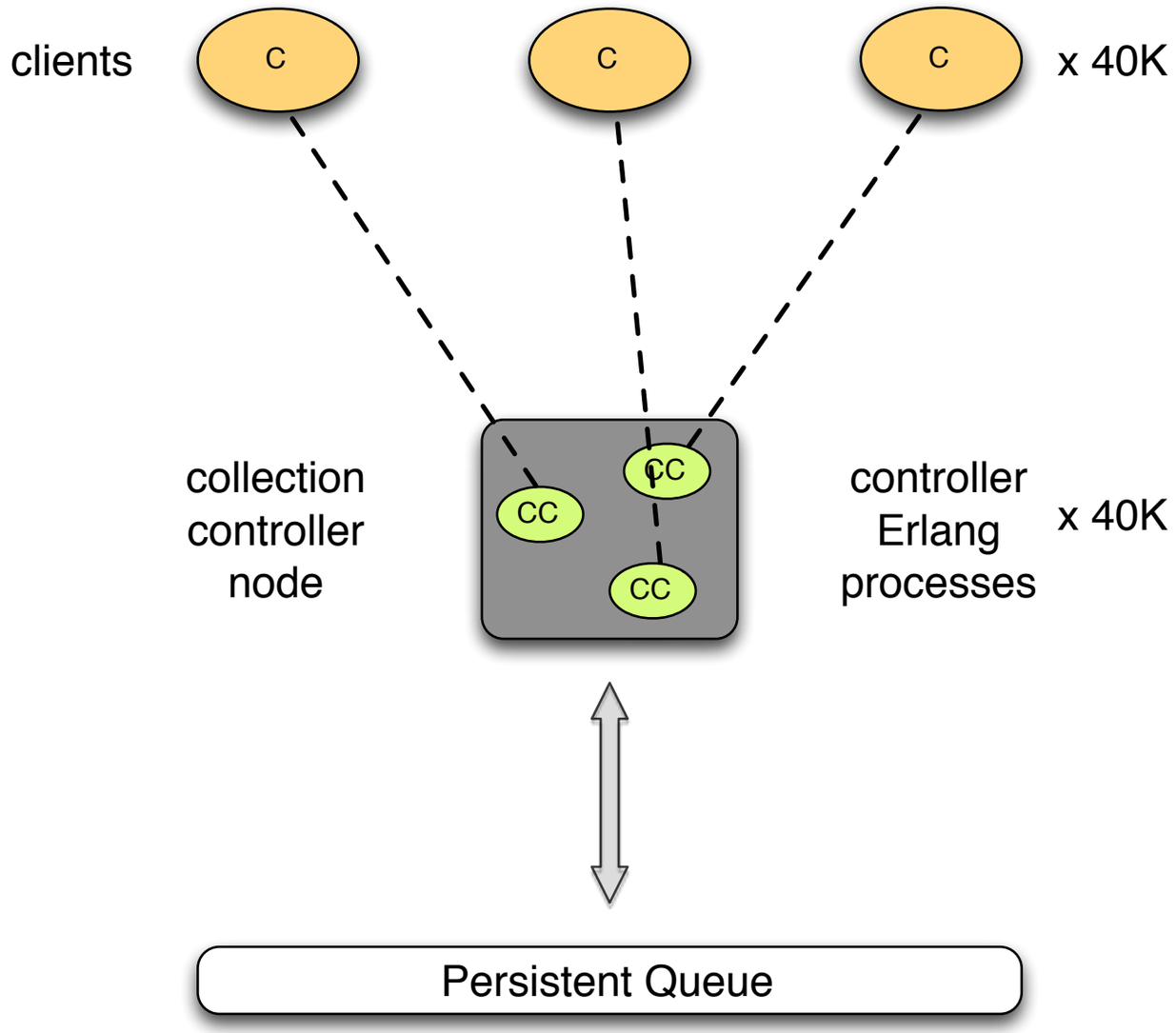
Log collection: zooming in



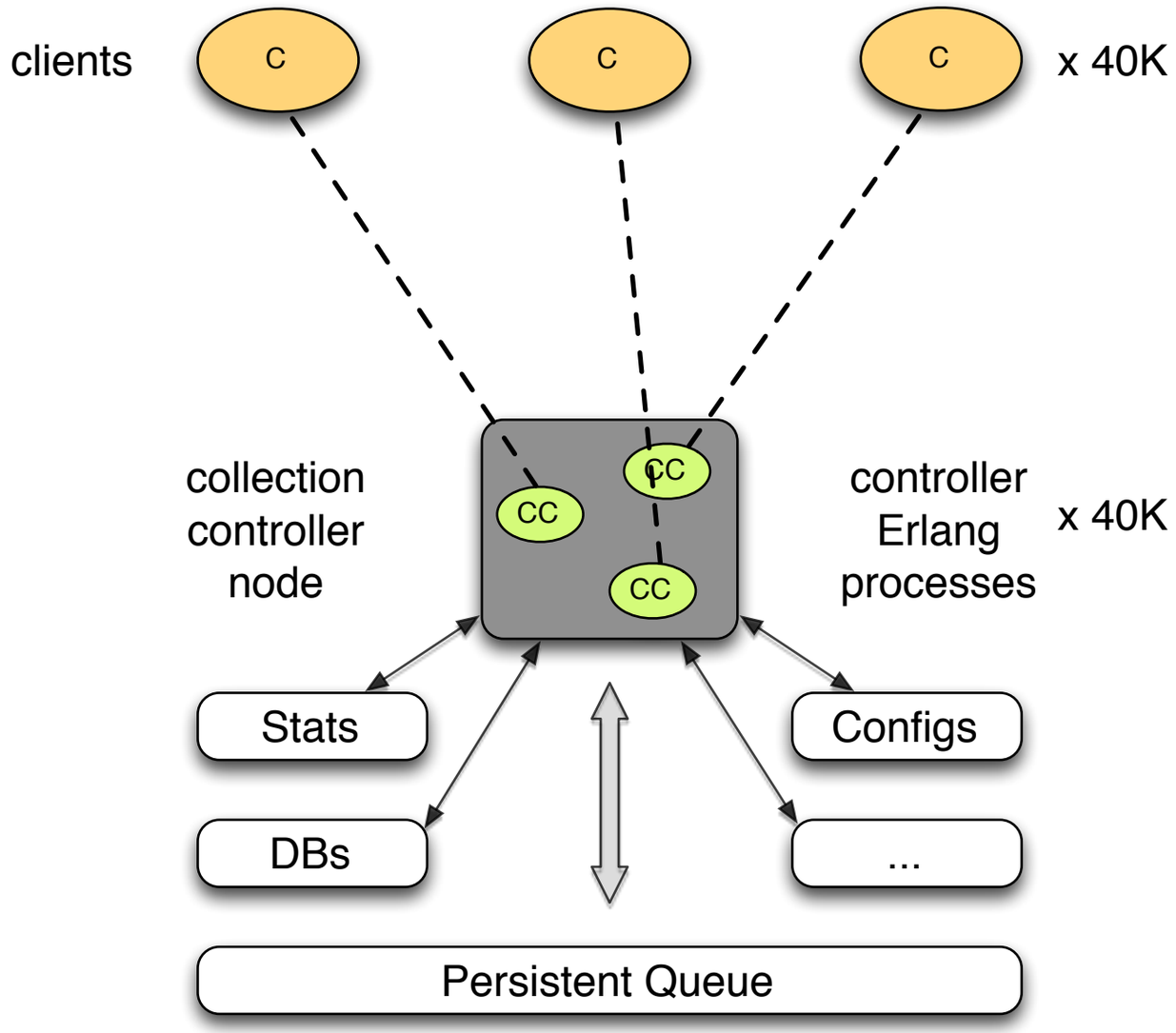
Log collection: before rewrite



Log collection: zooming in



Log collection: zooming in



What changes when you scale up from 1000 to 10 nodes?

Everything!

Increases per node/CPU core

- request rate
- data volume
- number of open connections
- number of processes per VM
- failure rate
- bugs and memory leaks get exposed

Colocated environment

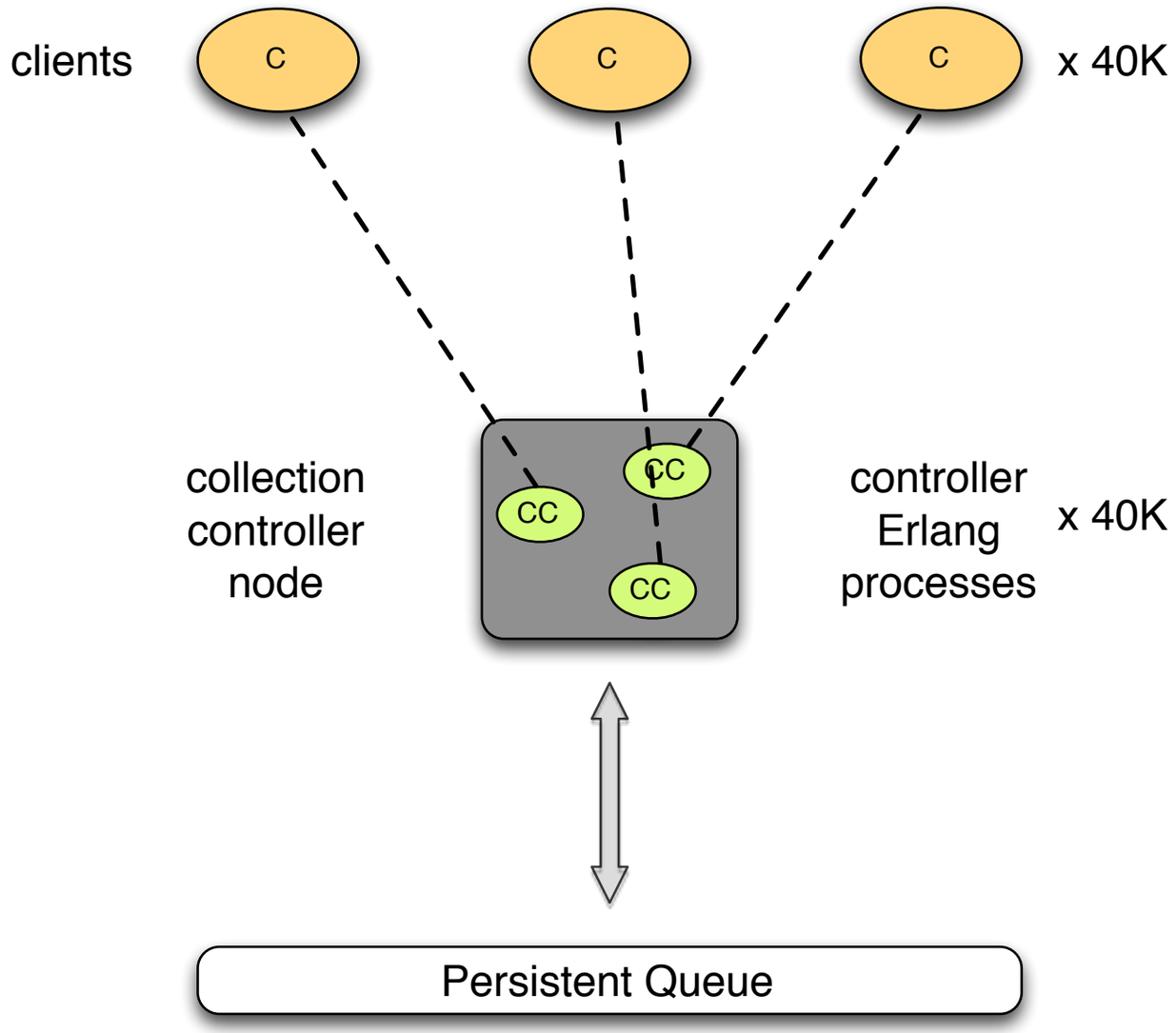
- different clients installed in different environments create different load patterns
- one set of clients can affect health of others
- can’t easily upgrade/troubleshoot individual clients
- no downtime tolerance

Application monitoring

- logging is less useful for troubleshooting
- need real-time monitoring, dashboards
- need visualization

Problem: high memory usage leading to OOM kill

Hint: many processes each dealing with lots of data



High memory usage: force garbage collection

```
garbage_collect(P)
```

Problem: memory leak

Memory leak: searching... some luck

```
[ garbage_collect(P) || P <- processes() ].
```

Memory leak: troubleshooting...

```
[{M, P, process_info(P, [registered_name, initial_call,
current_function, dictionary]), B} || {P, M, B} <-
lists:sublist(lists:reverse(lists:keysort(2, [case
process_info(P,
binary) of {_, Bins} -> SortedBins = lists:usort(Bins), {_,
Sizes, _} =
lists:unzip3(SortedBins), {P, lists:sum(Sizes), SortedBins}; _ -
> {P, 0,
[]} end || P <- processes()])), 5)].
```

Memory leak: troubleshooting...

```
[
   {M, P, process_info(P, [registered_name, initial_call,
current_function, dictionary]), B}
   ||
   {P, M, B} <- lists:sublist(lists:reverse(lists:keysort(2,
       [
           case process_info(P, binary) of
               {_, Bins} ->
                   SortedBins = lists:usort(Bins),
                   {_, Sizes, _} = lists:unzip3(SortedBins),
                   {P, lists:sum(Sizes), []};
               _ ->
                   {P, 0, []}
           end
           ||
           P <- processes()
       ]
   )), 5)
].
```

Memory leak: searching...

```
[{561889757,<0.1087.0>,
 [{registered_name,alcollect_client_sup},
  {initial_call,{proc_lib,init_p,5}},
  {current_function,{gen_server,loop,6}},
  {dictionary,[{'$ancestors',[alcollect_sup,lmcollect_sup,
                              <0.1057.0>]},
              {'$initial_call',
{supervisor,alcollect_client_sup,1}}]}],
 []},

{6550836,<0.1039.0>,
 [{registered_name,gproc},
  {initial_call,{proc_lib,init_p,5}},
  {current_function,{gen_server,loop,6}},
  {dictionary,[{'$ancestors',[gproc_sup,<0.1037.0>]},
              {'$initial_call',{gproc,init,1}}]}],
 []},
```

Memory leak: searching... victory!

```
supervisor:terminate_child(lmcollect_sup, lmcollect_ls_config).

application:stop(lmcollect).

...
```

Memory leak: sub-binary references

- process heaps
- ets tables
- message queues
- driver queues

Memory leak: preventing binary references

1. `binary:copy(X)`

2. `-define(COPY_BINARIES(X),`
   `binary_to_term(term_to_binary(X))).`

Problem: supervisors shutdown after children exceed max restart rate

Preventing supervisor shutdowns

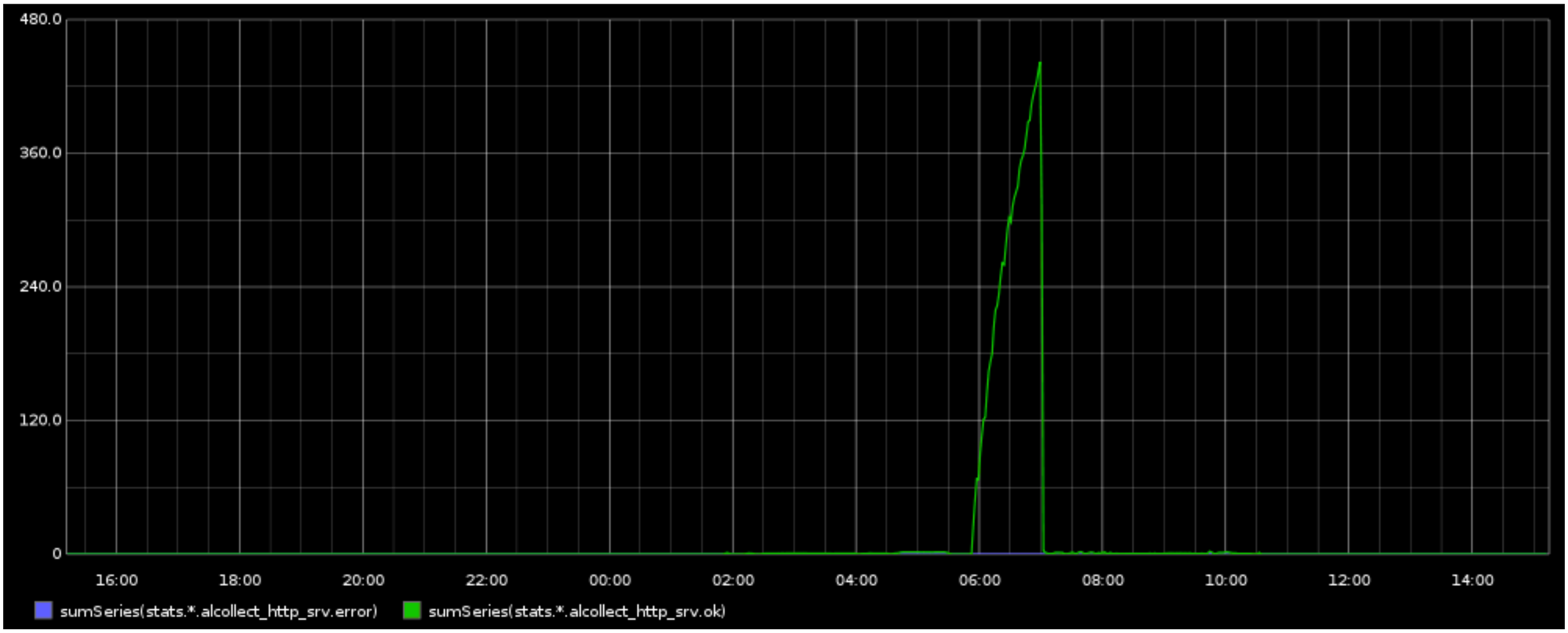
Delay child crashes: 2 options

1. `supervisor2`

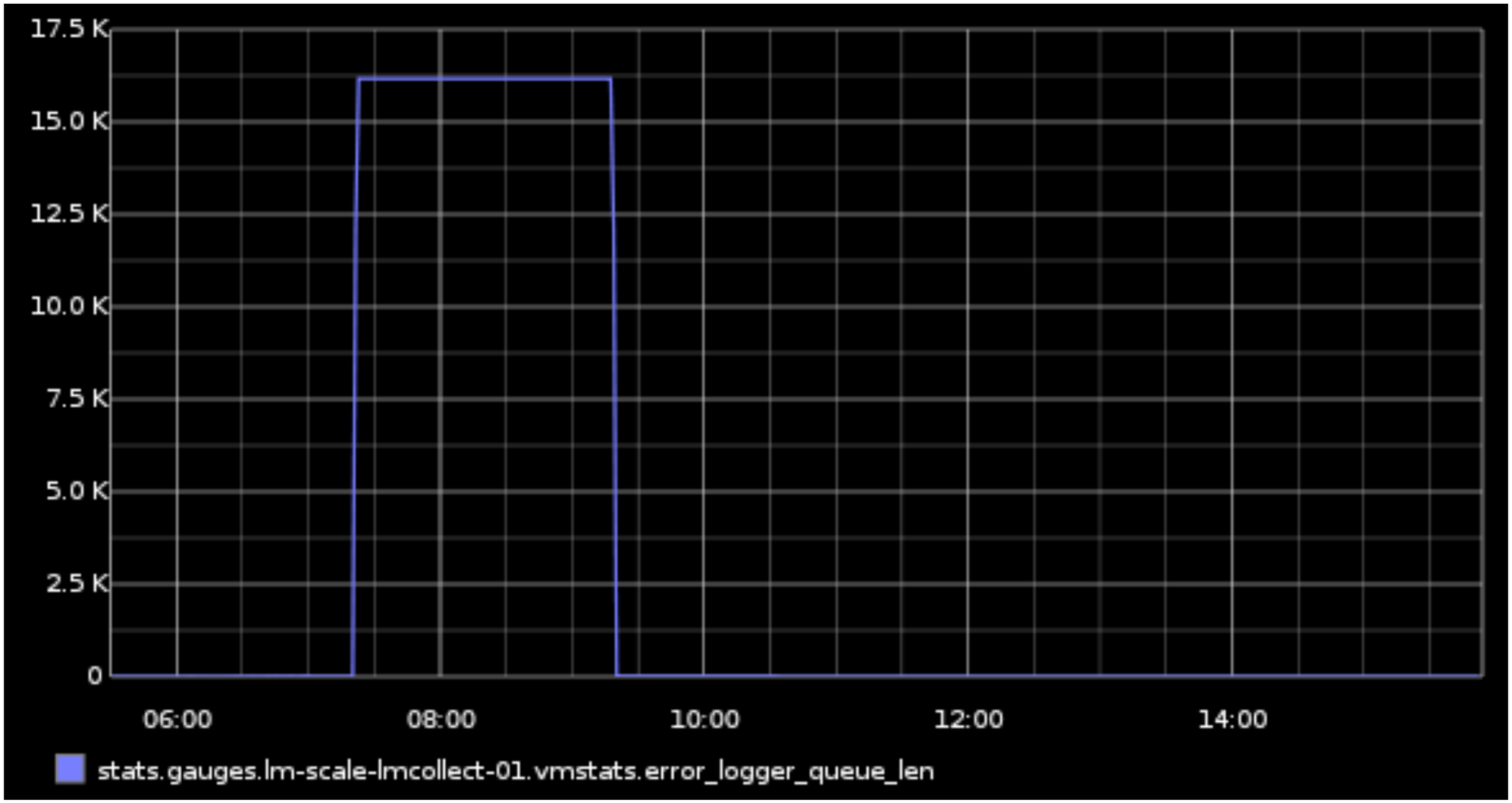
2. manually insert delays for each gen_server's Mod:handle_X

```
handle_call() ->
    try
        do_handle_call()
    catch Class:Reason ->
       timer:sleep(1000),
       exit({crash, Class, Reason, get_stracktrace()})
end
```

Problem: stall while using gen_udp



Problem: stall while using gen_udp



usr/erlang/erts-5.8.5/src/prim_inet.erl

```
send(S, Data, OptList) when is_port(S), is_list(OptList) ->
    ?DBG_FORMAT("prim_inet:send(~p, ~p)~n", [S,Data]),
    try erlang:port_command(S, Data, OptList) of
        false -> % Port busy and nosuspend option passed
            ?DBG_FORMAT("prim_inet:send() -> {error,busy}~n", []),
            {error,busy};
        true ->
            receive
                {inet_reply,S,Status} ->
                    ?DBG_FORMAT("prim_inet:send() -> ~p~n",
[Status]),
                    Status
            end
    catch
        error:_Error ->
            ?DBG_FORMAT("prim_inet:send() -> {error,einval}~n", []),
             {error,einval}
    end.
```

Preventing gen_server stall when using gen_udp

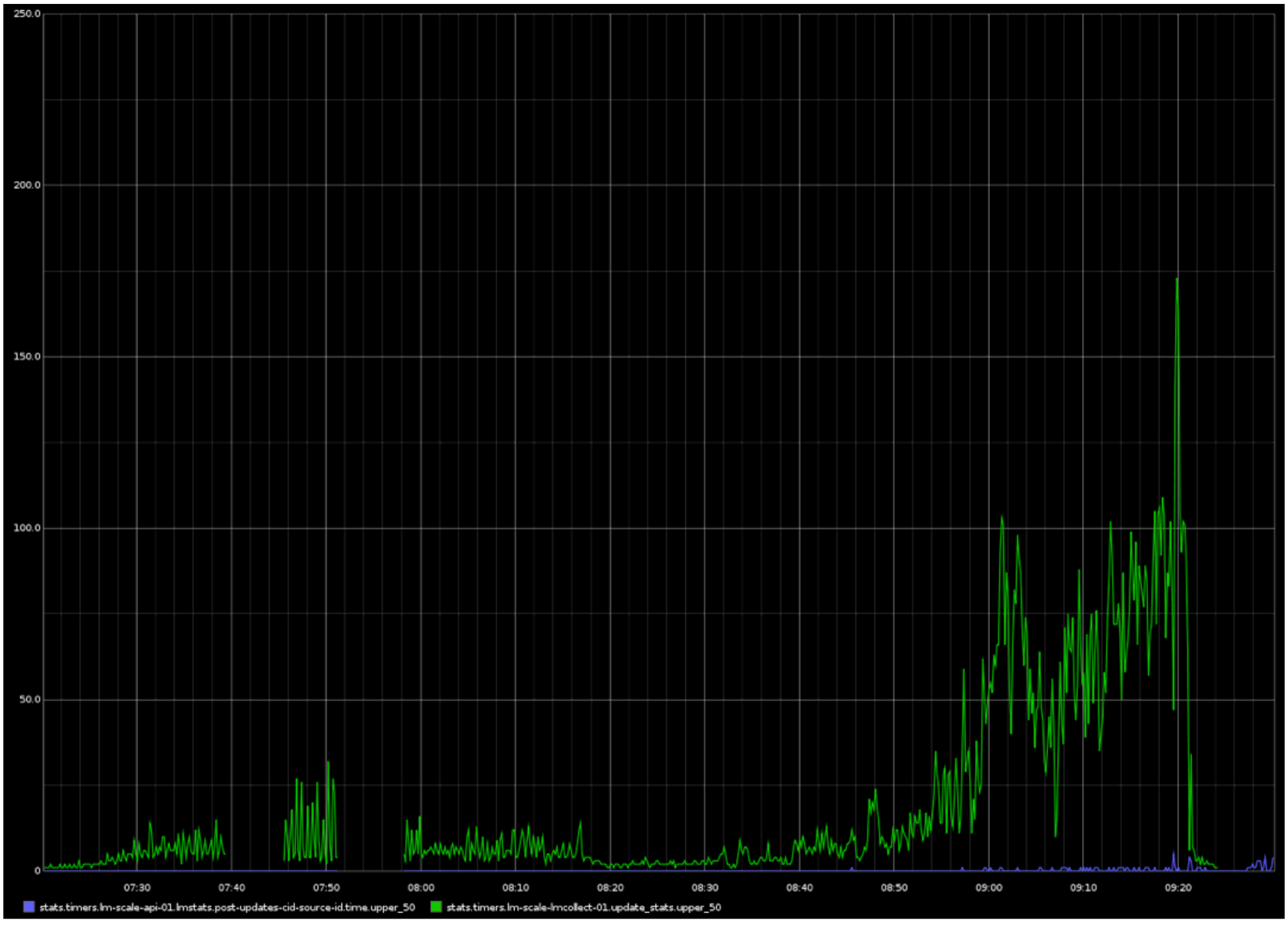
1. gen_server2

2. asynchronous gen_udp:send

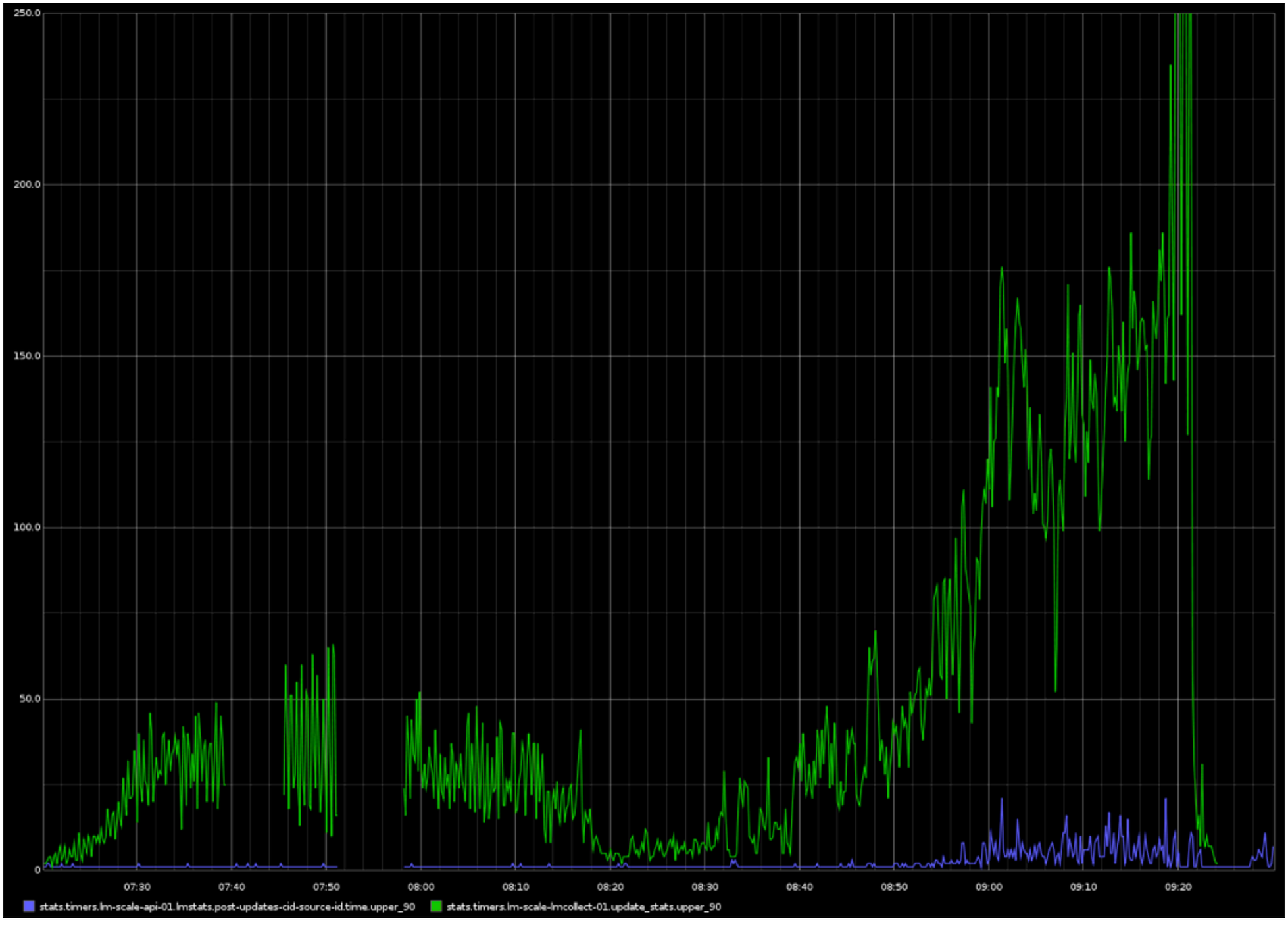
```
-      gen_udp:send(Socket, Address, Port, SyslogMessage);
+      try erlang:port_command(Socket,
+                              [[((Port) bsr 8) band 16#ff,
+                                (Port) band 16#ff],
+                               [A band 16#ff, B band 16#ff,
+                                C band 16#ff, D band 16#ff],
+                               SyslogMessage])of
+          true -> ok
+      catch ...

+handle_info({inet_reply, _, ok}, State) ->
+    {noreply, State};
```

3. fix gen_udp!

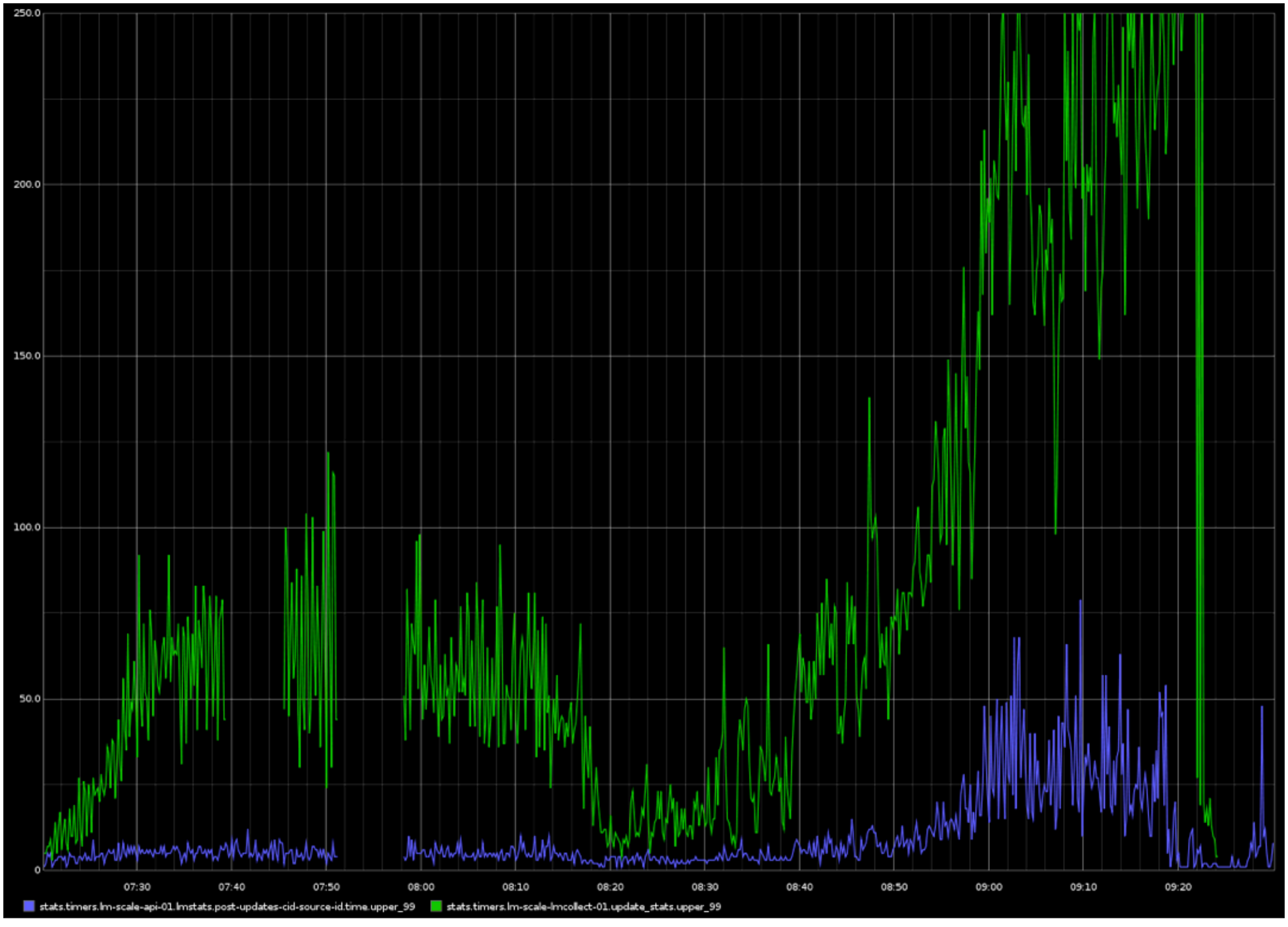
Problem: client-side latencies: 50%-ile



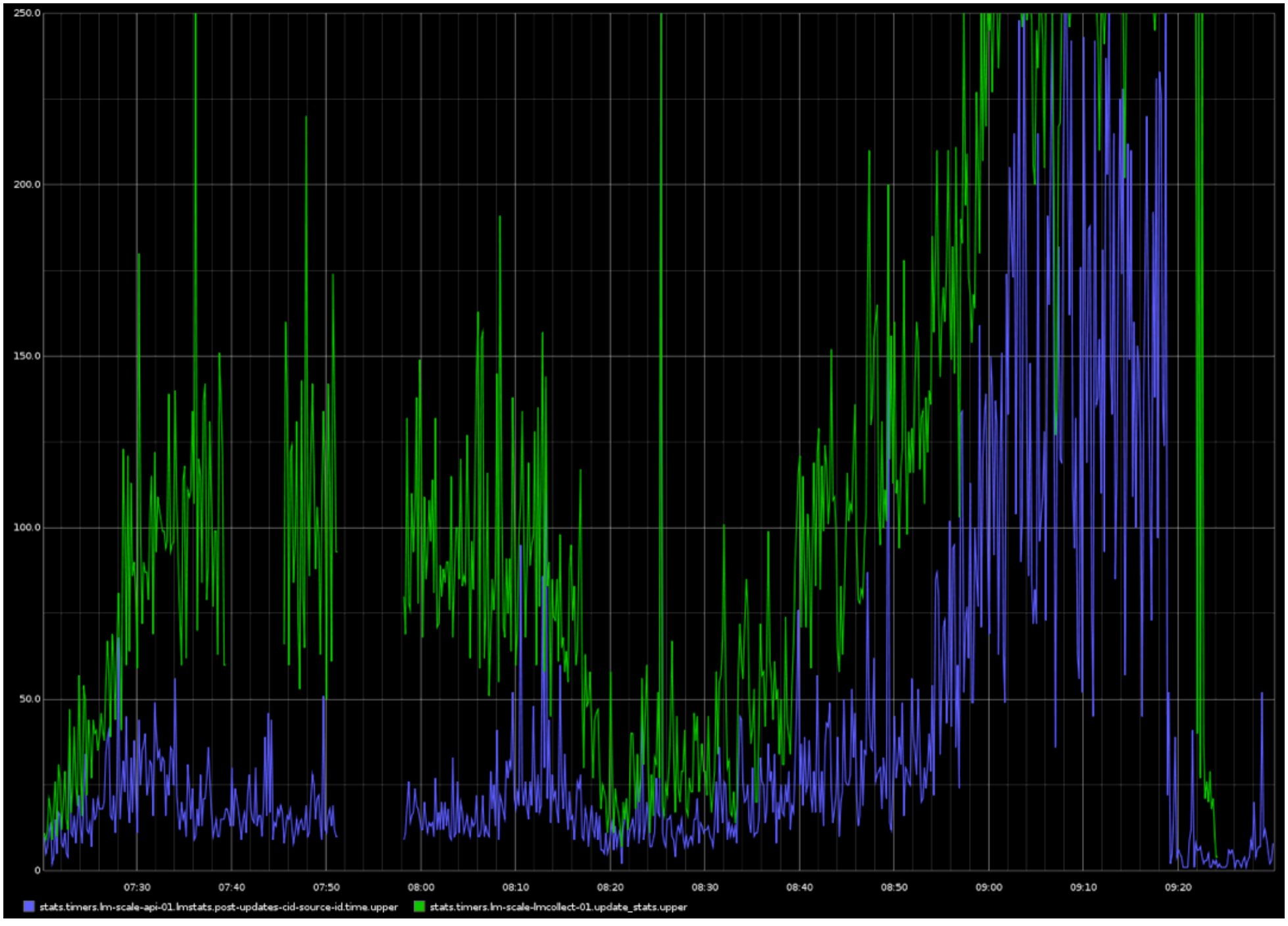
Client-side latencies: 90%-ile



Client-side latencies: 99%-ile



Client-side latencies: max



Summary

Erlang is a very efficient tool*

Erlang is a very efficient tool*

* you need to know your tools

- Excellent runtime for running highly concurrent applications + soft real-time
- Simple and powerful language
- Fault-tolerance
- Stable
- Rich troubleshooting capabilities
- Growing community

A dream: Erlang/OSP (**O**pen **S**erver **P**latform)

- ~~revise OTP~~ establish OSP principles and building blocks
- packaging and deployment: e.g. get rid of reltool.config
- logging out of the box: standard error, syslog (including sasl)
- http client that works well
- clients for popular databases: MySQL, Postgres
- alternative distributed Erlang: cluster membership, leader election, distributed locking

Questions?

Thank you

Optional: if there’s enough time and interest

Problem: httpc - Erlang/OTP HTTP client

httpc bug: mixes up responses

Server responds with 200:

```
Oct 23 02:20:36 lm-scale-api-01 al-hostmeta[17814]: ALH00002I
10.3.12.102 - - "GET /host_metadata/1710/
e1924d5bfc58ac9576d6b31f261e0d137d56e142 HTTP/1.1" 200 547 --
performed in 4 ms
```

Client magically receives 204:

```
Oct 23 02:20:36 lm-scale-lmcollect-02 al-lmcollect[9436]:
ALL00000T al_httpc:61: http get http://lm-scale-
balance-01.pd.alertlogic.net:8090/host_metadata/1710/
e1924d5bfc58ac9576d6b31f261e0d137d56e142 performed in 41.124000 ms

Oct 23 02:20:36 lm-scale-lmcollect-02 al-lmcollect[9436]:
AHM00001E Error querying host metadata for key 1710,
e1924d5bfc58ac9576d6b31f261e0d137d56e142: {unknown_http_response,
{{"HTTP/1.1",204,"No Content"},[{"date","Tue, 23 Oct 2012 07:20:36
GMT"},{"server","MochiWeb/1.0 (Any of you quaids got a smint?)"},
{"content-length","0"}],<<>>}}
```

httpc bug: mixes up responses

Server responds with 204:

Oct 23 02:20:38 lm-scale-api-01 al-hostmeta[17814]: ALH00002I 10.3.12.102 - - **"PUT** /host_metadata/1712/fd09bc458d5ee0a250668cf9bf41b0bea4886571 HTTP/1.1" **204** 0 -- performed in 4 ms

Client receives 404:

Oct 23 02:20:38 lm-scale-lmcollect-02 al-lmcollect[9436]: ALL00000T al_httpc:86: http put http://lm-scale-balance-01.pd.alertlogic.net:8090/host_metadata/1712/fd09bc458d5ee0a250668cf9bf41b0bea4886571 performed in 7.443000 ms

Oct 23 02:20:38 lm-scale-lmcollect-02 al-lmcollect[9436]: AHM00000E Error updating host metadata for key 1712, fd09bc458d5ee0a250668cf9bf41b0bea4886571: **{unknown_http_response, {404,<<"No metadata for key">>}}**