Logging for the 21st OTP

Andrew Thompson
Helium

Logging should be boring; why talk about it?

- Logs are critical for debugging and troubleshooting
- Logs can be the only way you talk to your customers/users
- Logging is like plumbing; you ignore it until it breaks at which point it becomes a #1 priority
- Erlang has had a very mixed history on logging
- OTP finally has a new logging system in OTP 21

Why me?

- Wrote my first Erlang logger in 2008 because error_logger was terrible
- Wrote Lager in 2011 at Basho because error_logger was still terrible
- Gave a talk on Lager and Erlang logging at Erlang Factory 2013
- Have been maintaining Lager ever since (with the help of John Daily and Mark Allen)
- Participated (lightly) in the design of OTP 21’s new `logger`

What’s wrong with error_logger anyway

- Logging is completely asynchronous which leads to error_logger overload and BEAM crashing
- No oversize message protections (depth limiting was added later)
- Logs are very confusing for non-Erlang users
- Log rotation is very un-UNIX like, sysadmins hate it
- See my 2013 talk for more details

What did Lager try to change?

- Safety above everything else; never crash the node
- Try to make the logs look like more traditional application logs
- Log in English, not “giant tuples of doom”
- Sane logfile rotation
- Easy to use; simple API, try to do the right thing automatically
- Introduce metadata and structure to the logging pipeline
- Decouple formatting from storage of log messages
- Essentially, make Erlang services log like every other service does

What Lager got wrong

- I used a parse transform (because there was no ?FUNCTION macro in 2011)
- Didn’t add logging macros for a long time
- Never got buy in from the OTP team
- Didn’t follow up with Elixir integration
- The Erlang community never standardized on a reasonable logging API, mix and matching libraries using different logging libraries is/was very painful
- Flirted with structured logging but never quite got there
- Maybe a bit too opinionated

What about logger?

- New logging system in OTP 21
- Backwards compatible with error_logger
- Supports multiple formatters and handlers
- Supports log event metadata
- Supports ‘report’ messages, essentially structured log events
- Has support for lots of types of overload protection (messages too big, too many log messages, etc)
- Does log event formatting in the calling process
- Has ‘filters’ for matching messages based on metadata or message structure

This is a huge win for all BEAM languages!

Everybody wins!

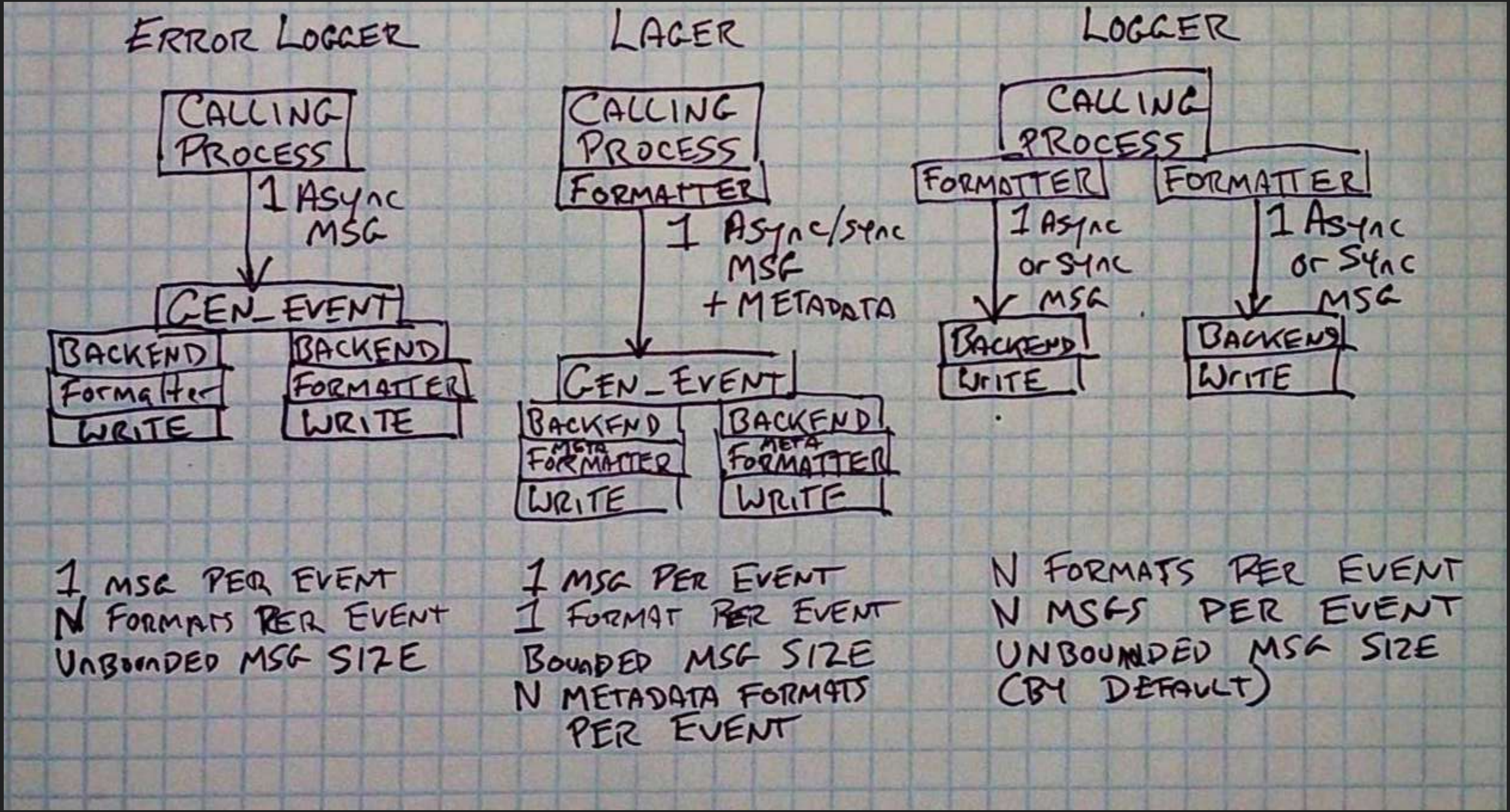
Logger is a lot like Lager

- They sound the same
- They both support the 7 syslog log levels
- Logger has all the overload protection Lager does
- Both do the event formatting in the caller
- Per process and per event metadata
- Able to direct messages based on metadata

Logger does have some differences

- The *entire* message is formatted at the caller (lager just does the format string/arguments formatting and passes the metadata & timestamp along to the logging backend)
- For each handler, the log is formatted independently (lager only does it once)
- Each handler runs as a separate process (lager uses a single gen_event for several backends)
- Filters work differently than Lager’s ‘tracing’

A tale of 3 pipelines



1 MSG PER EVENT
N FORMATS PER EVENT
UNBOUNDED MSG SIZE

1 MSG PER EVENT
1 FORMAT PER EVENT
BOUNDED MSG SIZE
N METADATA FORMATS PER EVENT

N FORMATS PER EVENT
N MSGS PER EVENT
UNBOUNDED MSG SIZE (BY DEFAULT)

Structured logging

- Instead of format string/arguments, logger allows you to log a tuple of data. This is called a ‘report’.
- Reports have a default formatting function attached
- You can, via logger configuration, supply another formatting function
- All OTP messages are specified as reports (yay!). This means that applications can control how they’re printed
- Structured logging enables a much richer set of options for working with event data

Example of an OTP report

```
?LOG_ERROR(#{label=>{gen_server,terminate},
         name=>Name,
       last_message=>Msg,
       state=>format_status(terminate, Mod, get(), State),
       reason=>Reason,
       client_info=>client_stacktrace(From)},
       #{domain=>[otp],
       report_cb=>fun gen_server:format_log/1,
       error_logger=>#{tag=>error}}),
```

Filters

- Filters are a bit like firewall rules for log events
- Filters can ‘ignore’, drop or alter/forward log events
- Filters are chained together, if the filter makes it to the end of the chain it gets passed to the formatter
- If the chain ends with `ignore` as the result, the ‘filter_default’ configuration is evaluated to see whether to log or drop the event
- Filters can be global (‘primary’) or per-handler
- Primary filters run first followed by per-handler filters
- Filters have fixed state, passed in at instantiation time
- Several filters included in logger_filters.erl

The new logger, same as the old error_logger

- Logger defaults to being error_logger compatible
- No oversize message protection by default
- Legacy formatter by default
- 2 bundled handlers
    - Standard: supports console or a single logfile (that it will re-create if deleted)
    - Disk_log: Very similar to the error_logger multiple file disk log, still terrible user experience

DO NOT USE THE LOGGER DEFAULTS IN PRODUCTION, PLEASE READ THE DOCUMENTATION AND CONFIGURE IT CORRECTLY

So what’s next for Lager?

- Lager’s main reasons to exist have largely been addressed
- A lot of code out there still uses Lager, how do we get that code using logger?
- Lager still has some nice features logger lacks

Lager 4.0

- Going to start deprecating Lager’s event core in favor of logger
- Parse transform will have a `lager_use_logger` option to rewrite lager calls to logger ones (with the right metadata)
- Lager will provide a ‘report callback’ for formatting the OTP structured logging events to lager style ‘readable errors’
- Lager will provide a logger formatter that can use lager formatters
- May try to provide tools to sanity-check logger config and warn about problems

How to upgrade a lager project to logger

```
 [
 {kernel,
  [{logger,
    [{handler, default, logger_std_h,
    #{formatter => {lager_logger_formatter, #{report_cb => fun
lager_logger_formatter:report_cb/1}}}},
    ]}]},
...
{lager, [
    {lager_use_logger, true},
       …
]}
```

Upgrading a lager project to lager continued

In your rebar config (for rebar3, stop using old rebar!)

```
{overrides, [{add, [{erl_opts, [{lager_use_logger, true}]}]}]}.
```

Don’t ask me about mix or erlang.mk, I don’t know how to use those things.

Why not keep Lager alive?

- Splitting the community is bad
- Duplicating effort is bad
- Missing lager features could be added to logger, or provided by more wrappers
- Everybody on the BEAM can take advantage of logger, lager has a weaker story here
- I’m tired of maintaining a logging library :)

How you can help

- Port your lager backend to a logger handler (hard to provide a wrapper for this)
- Switch your libraries to use logger calls (if you don’t need to support pre-OTP 21)
- If your application uses lager, try lager 4.0 in logger mode and see what breaks or is missing
- File issues or PRs against logger to make it better
- Write about using the new logger (examples, tutorials, documentation PRs, etc)

What about backwards compatibility?

- Libraries used by pre OTP 21 applications are going to be tricky
- I suggest a ‘last 3 major OTP releases’ deprecation strategy; by mid 2020 & OTP 23 we should try to get everything using logger APIs
- Lager 5.0 would come out after OTP 23 and hopefully have the event core completely removed and simply be a helper library for logger
- Care is needed; lots of Erlang users seem stuck on old versions

Benchmark time!

Benchmarks

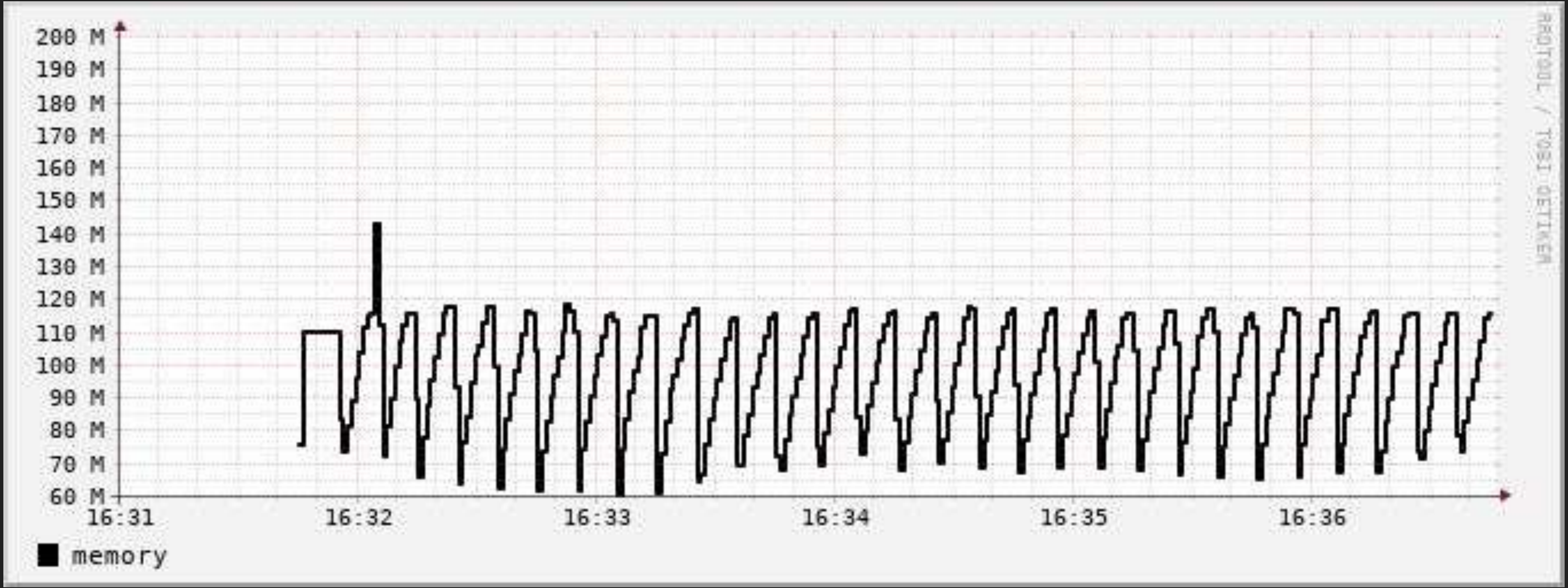
I have some old logging benchmarks lying around

- Cascading_failures - an app Fred wrote back in 2011 that uses a shared ETS table that gets deleted to cause a storm of crash messages over and over again
- Logbench - benchmarks logging performance across various workloads (differing numbers of workers and message sizes)

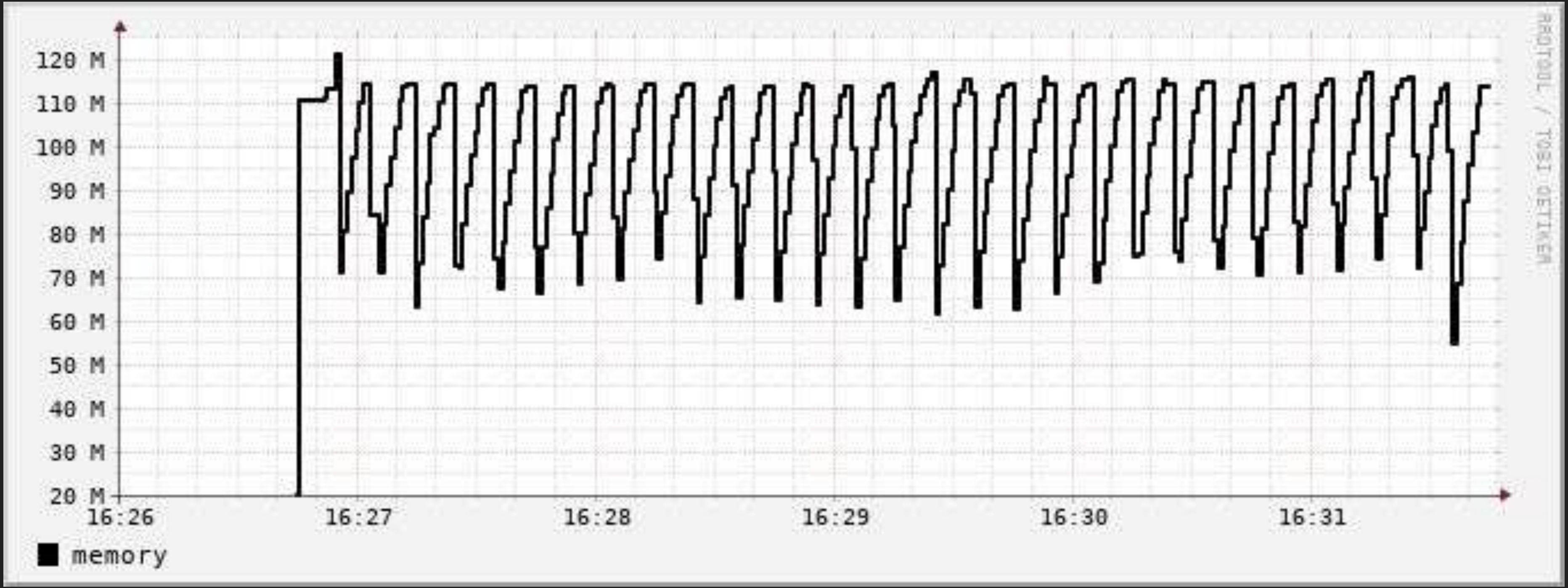
Both are on github, all the following measurements are relative to the machine they're run on.

Cascading Failures

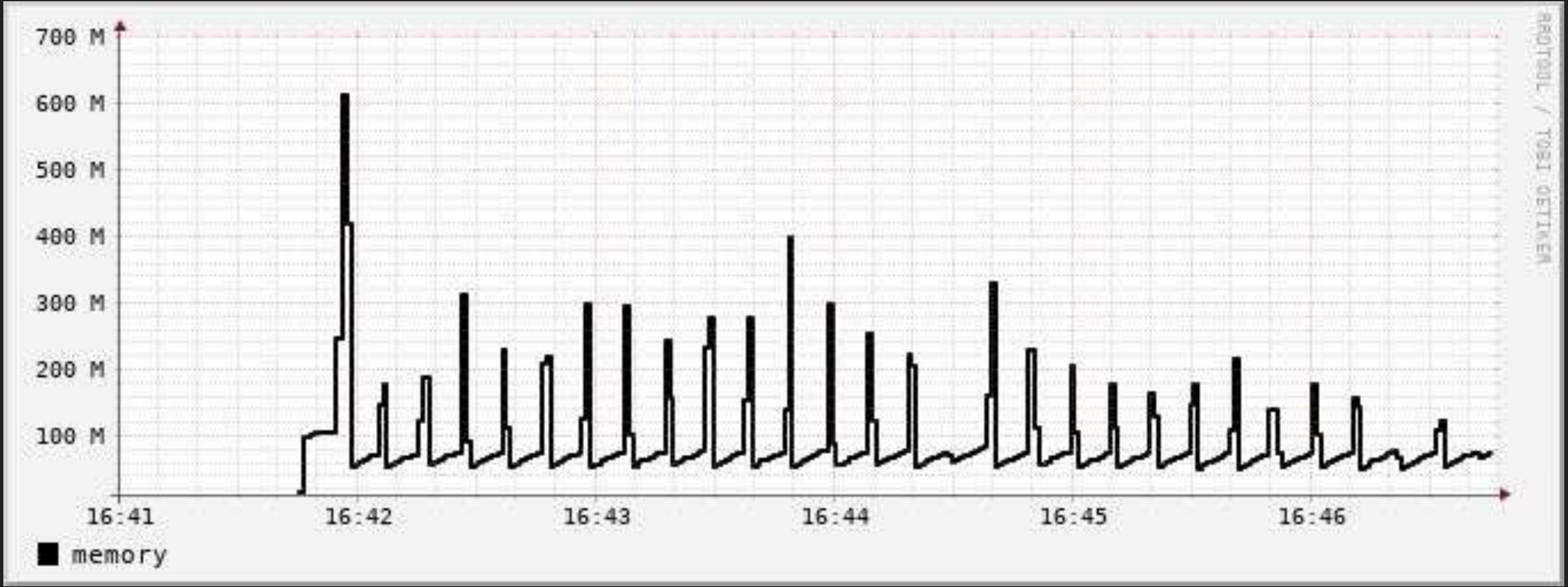
Benchmark memory usage over 5 minutes of endless crashing



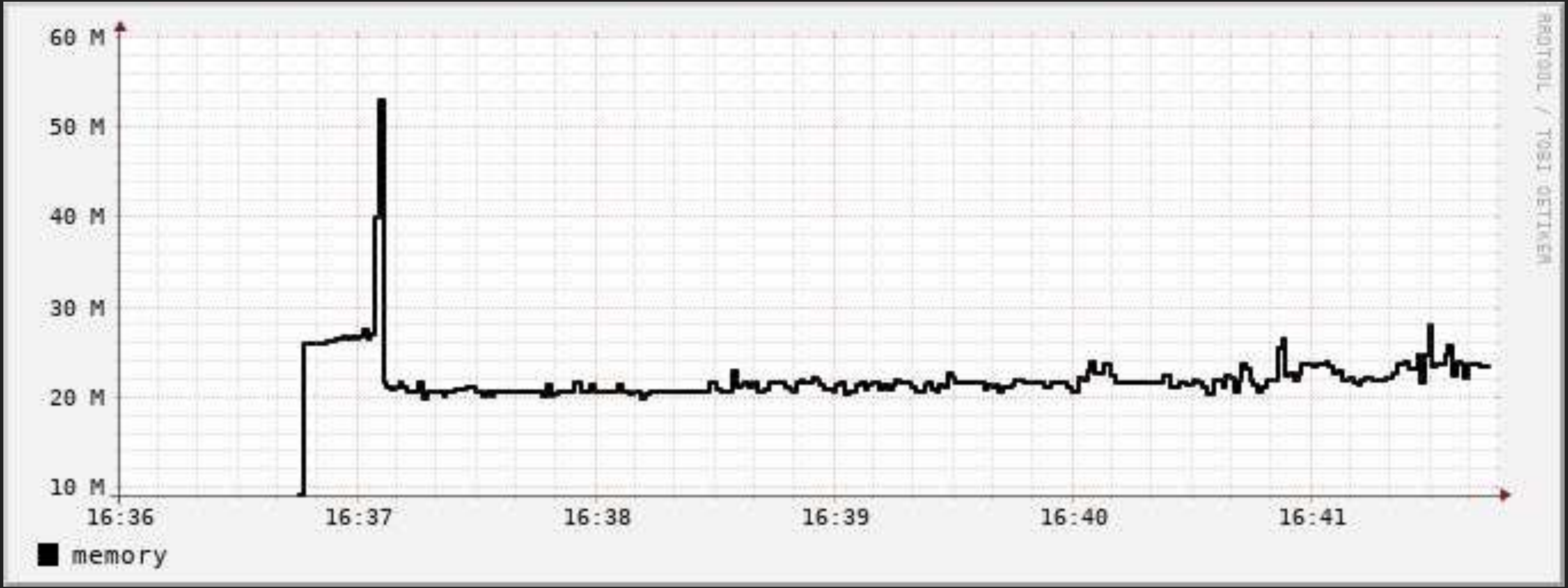
Lager 4.0 alpha (default config; 3 log files & console)



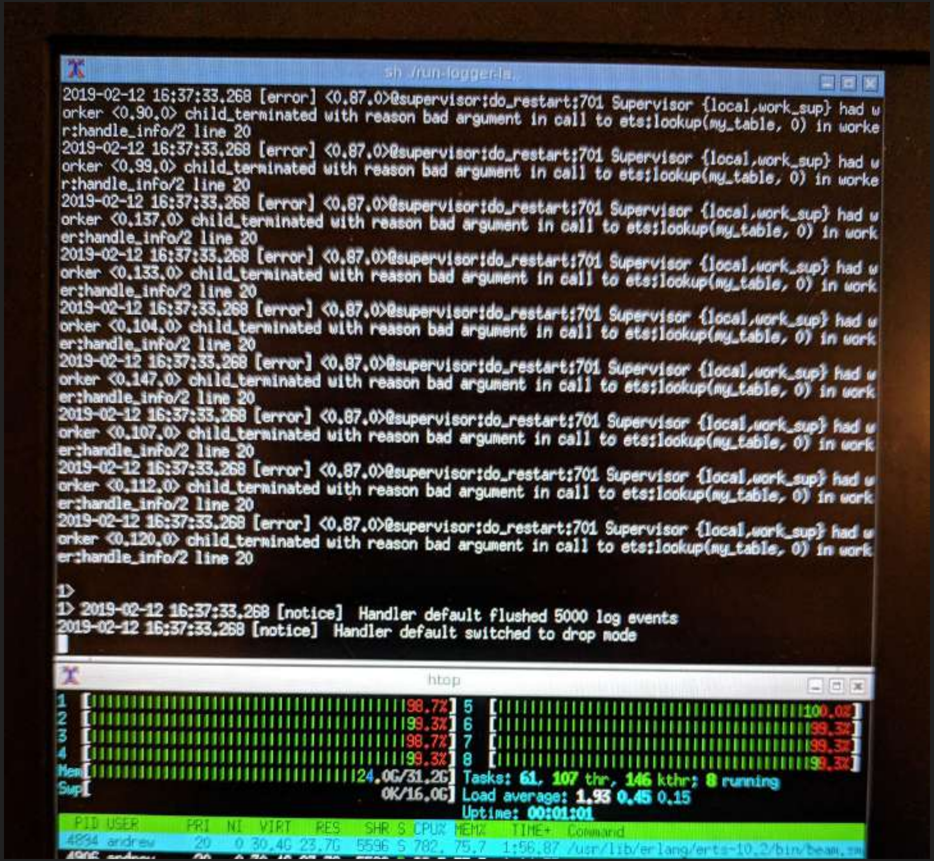
Lager 4.0 alpha (console only)



Logger OTP 21.2 (default config; console only)



Logger OTP 21.2 (max_size & chars_limit set to 1024)



```
2019-02-12 16:37:33.268 [error] <0.87.0>@supervisor:do_restart:701 Supervisor {local,work_sup} had w
orker <0.90.0> child_terminated with reason bad argument in call to ets:lookup(my_table, 0) in worke
r:handle_info/2 line 20
2019-02-12 16:37:33.268 [error] <0.87.0>@supervisor:do_restart:701 Supervisor {local,work_sup} had w
orker <0.99.0> child_terminated with reason bad argument in call to ets:lookup(my_table, 0) in worke
r:handle_info/2 line 20
2019-02-12 16:37:33.268 [error] <0.87.0>@supervisor:do_restart:701 Supervisor {local,work_sup} had w
orker <0.137.0> child_terminated with reason bad argument in call to ets:lookup(my_table, 0) in work
er:handle_info/2 line 20
2019-02-12 16:37:33.268 [error] <0.87.0>@supervisor:do_restart:701 Supervisor {local,work_sup} had w
orker <0.133.0> child_terminated with reason bad argument in call to ets:lookup(my_table, 0) in work
er:handle_info/2 line 20
2019-02-12 16:37:33.268 [error] <0.87.0>@supervisor:do_restart:701 Supervisor {local,work_sup} had w
orker <0.104.0> child_terminated with reason bad argument in call to ets:lookup(my_table, 0) in work
er:handle_info/2 line 20
2019-02-12 16:37:33.268 [error] <0.87.0>@supervisor:do_restart:701 Supervisor {local,work_sup} had w
orker <0.147.0> child_terminated with reason bad argument in call to ets:lookup(my_table, 0) in work
er:handle_info/2 line 20
2019-02-12 16:37:33.268 [error] <0.87.0>@supervisor:do_restart:701 Supervisor {local,work_sup} had w
orker <0.107.0> child_terminated with reason bad argument in call to ets:lookup(my_table, 0) in work
er:handle_info/2 line 20
2019-02-12 16:37:33.268 [error] <0.87.0>@supervisor:do_restart:701 Supervisor {local,work_sup} had w
orker <0.112.0> child_terminated with reason bad argument in call to ets:lookup(my_table, 0) in work
er:handle_info/2 line 20
2019-02-12 16:37:33.268 [error] <0.87.0>@supervisor:do_restart:701 Supervisor {local,work_sup} had w
orker <0.120.0> child_terminated with reason bad argument in call to ets:lookup(my_table, 0) in work
er:handle_info/2 line 20

1>
1> 2019-02-12 16:37:33.268 [notice]  Handler default flushed 5000 log events
2019-02-12 16:37:33.268 [notice]  Handler default switched to drop mode
```

```
1  [|||||||||||||||||||||||||||||||||||||||98.7%]  5  [||||||||||||||||||||||||||||||||||||||||100.0%]
2  [|||||||||||||||||||||||||||||||||||||||99.3%]  6  [|||||||||||||||||||||||||||||||||||||||||99.3%]
3  [|||||||||||||||||||||||||||||||||||||||98.7%]  7  [|||||||||||||||||||||||||||||||||||||||||99.3%]
4  [|||||||||||||||||||||||||||||||||||||||99.3%]  8  [|||||||||||||||||||||||||||||||||||||||||99.3%]
Mem[|||||||||||||||||||||||||||||||||||24.0G/31.2G]  Tasks: 61, 107 thr, 146 kthr; 8 running
Swp[                                       0K/16.0G]  Load average: 1.93 0.45 0.15
                                                    Uptime: 00:01:01
  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 4894 andrew     20   0 30.4G 23.7G  5596 S 782. 75.7  1:56.87 /usr/lib/erlang/erts-10.2/bin/beam.sm
```

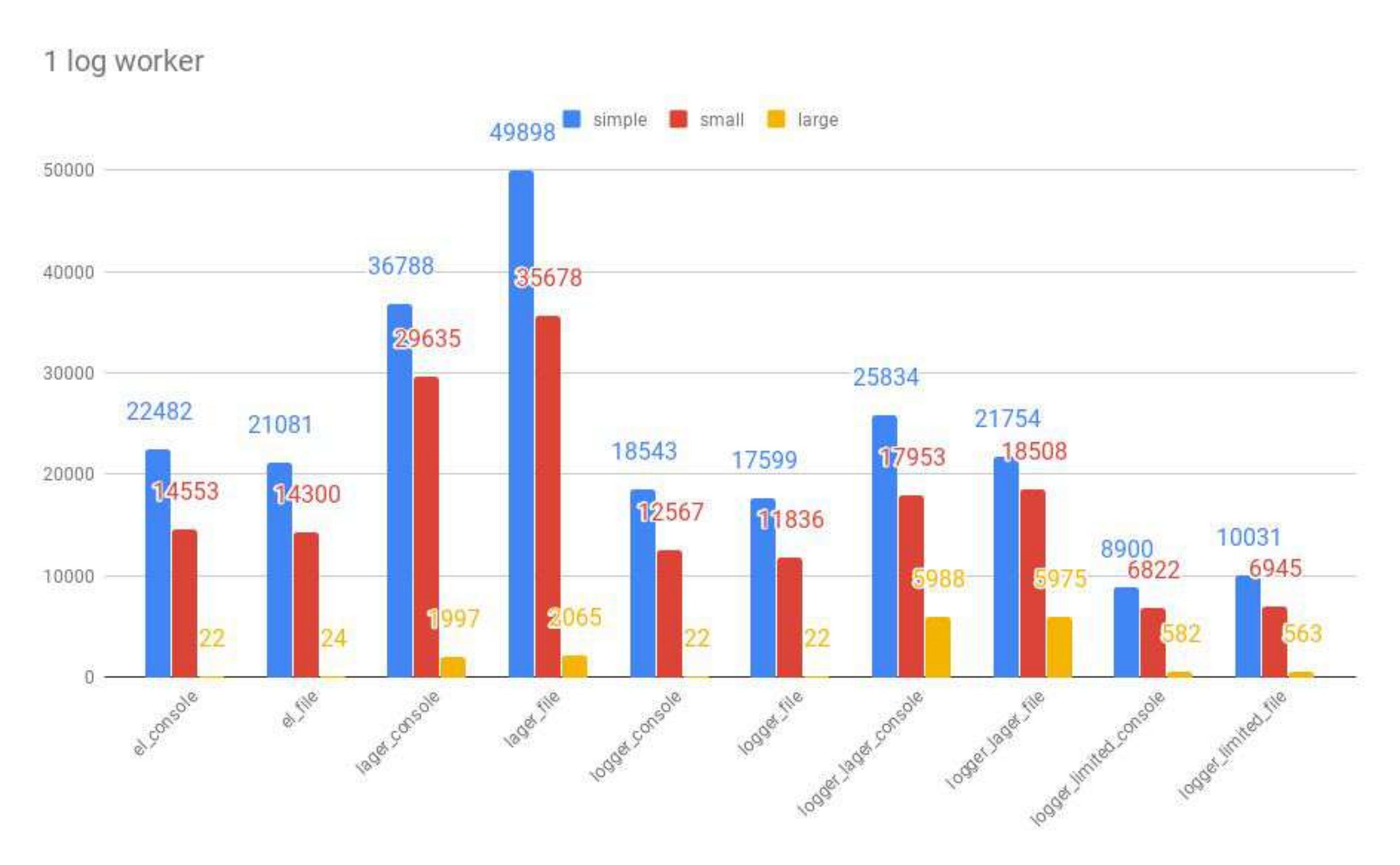
I tried a custom formatter with logger, but it went into drop mode, used all my RAM and froze my computer...

Notes on the benchmark

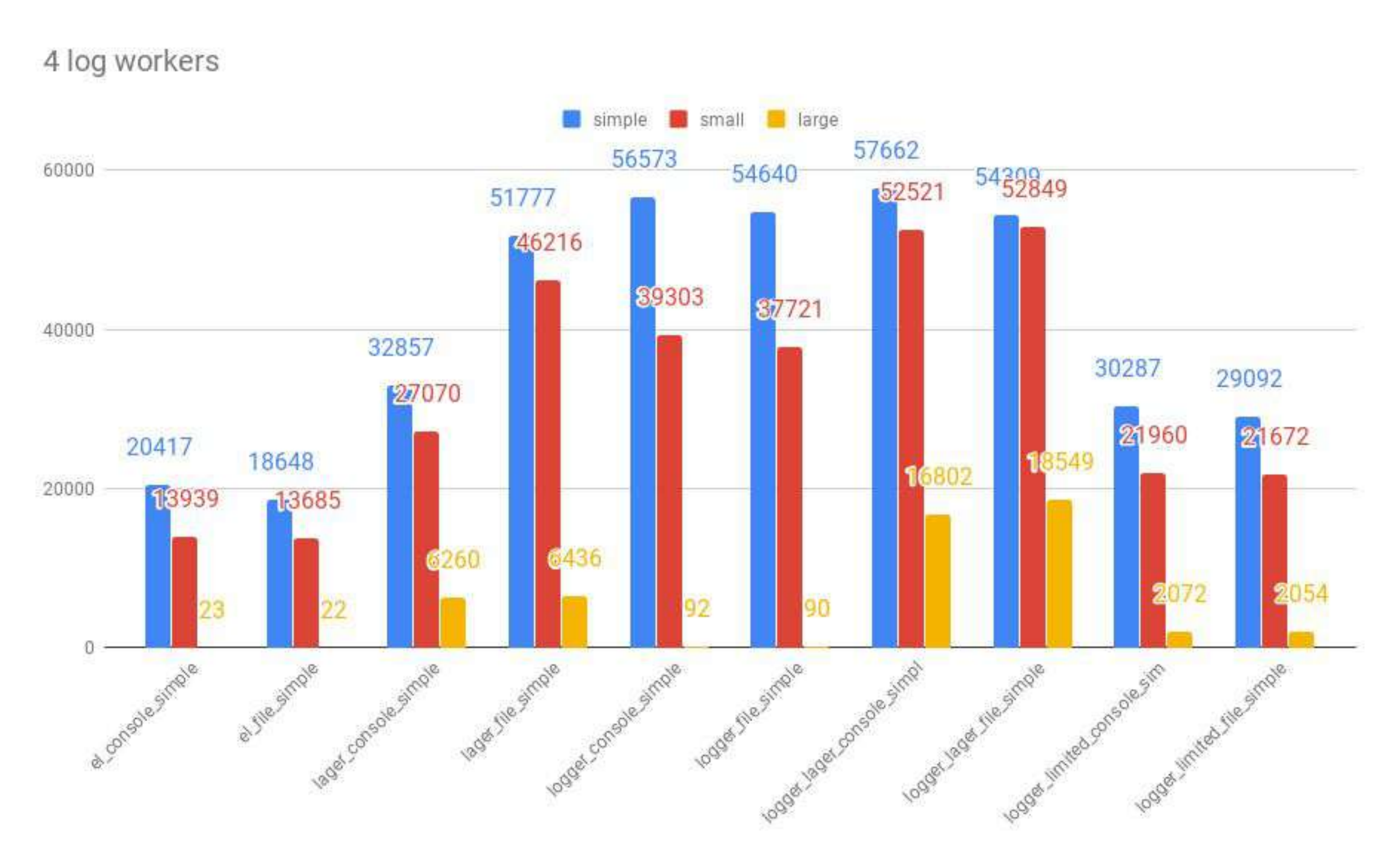
- Graphs have different scales
- Logger by default was only logging to console
- Lager by default was logging to console, error.log, console.log and crash.log
- Logger RAM spiked over 2x as high as Lager
- Lager seems consistent even with more backends
- max_size/chars_limit drops a LOT of log messages, probably 90%+
- I have no idea why my custom formatter blows up so dramatically...

logbench

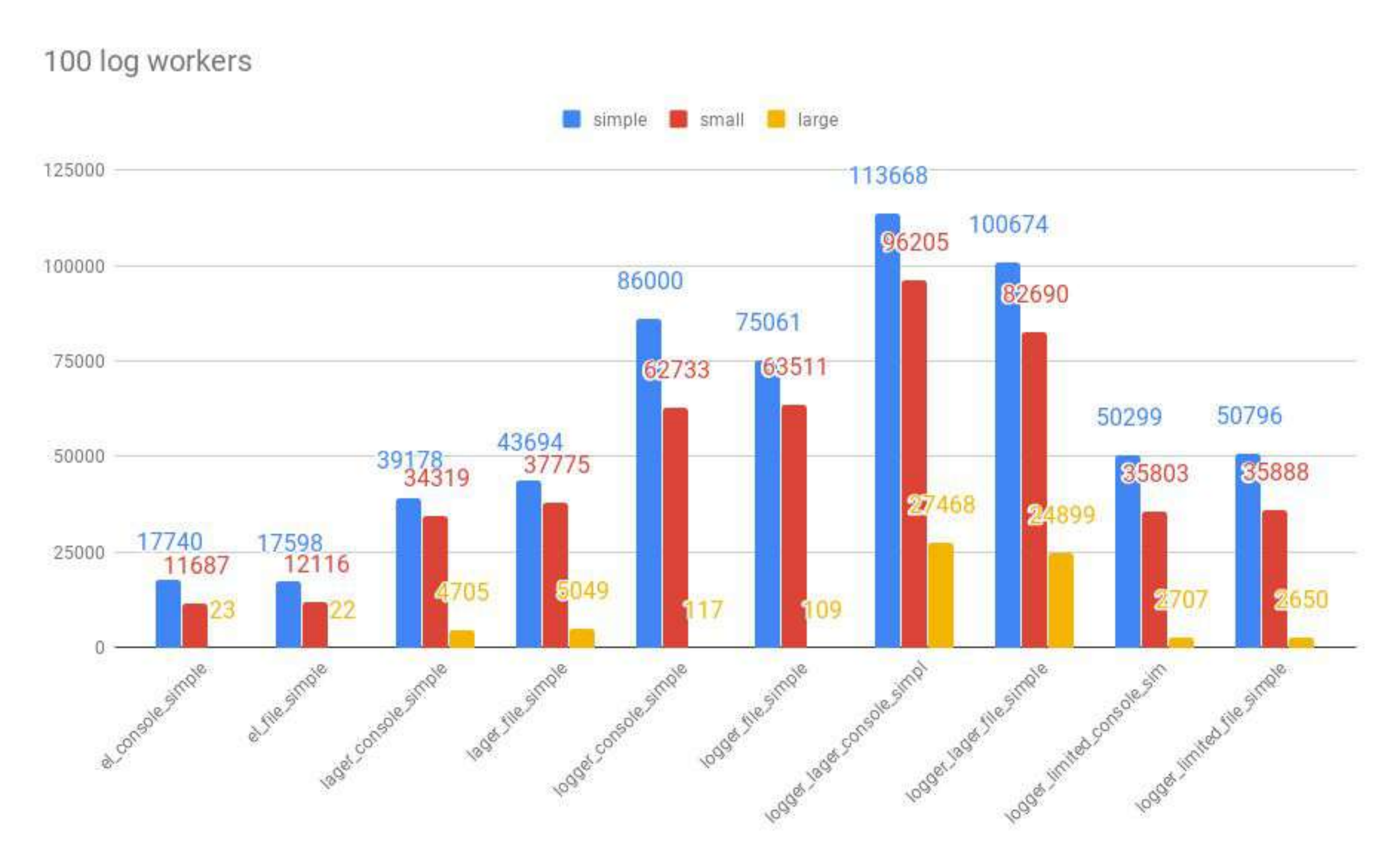
Logs different size messages to different logging libraries from a variable number of processes


1 log worker
simple
small
large
50000
40000
30000
20000
10000
0
22482
14553
22
21081
14300
24
36788
29635
1997
49898
35678
2065
18543
12567
22
17599
11836
22
25834
17953
5988
21754
18508
5975
8900
6822
582
10031
6945
563
el_console
el_file
lager_console
lager_file
logger_console
logger_file
logger_lager_console
logger_lager_file
logger_limited_console
logger_limited_file

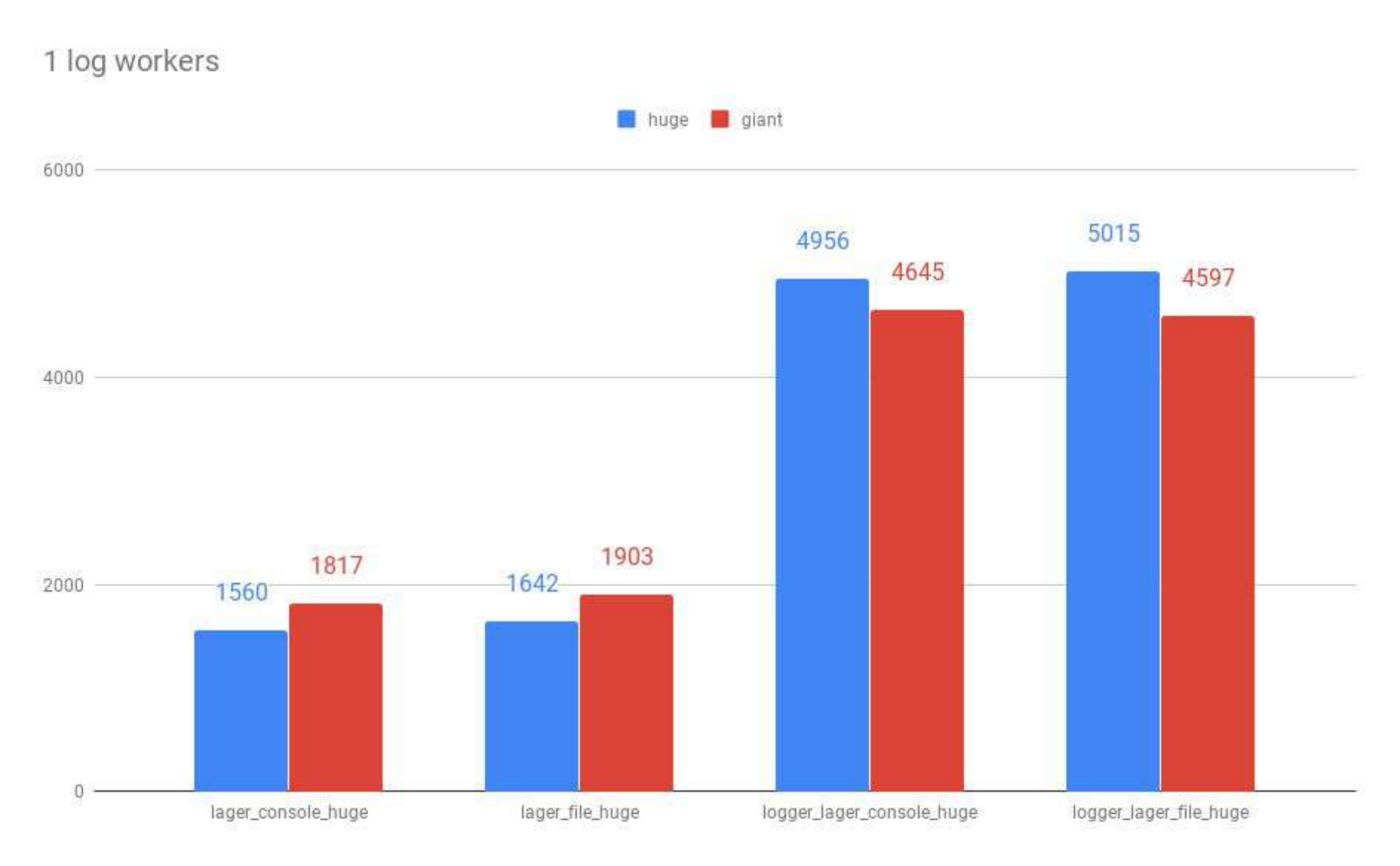
1 log worker



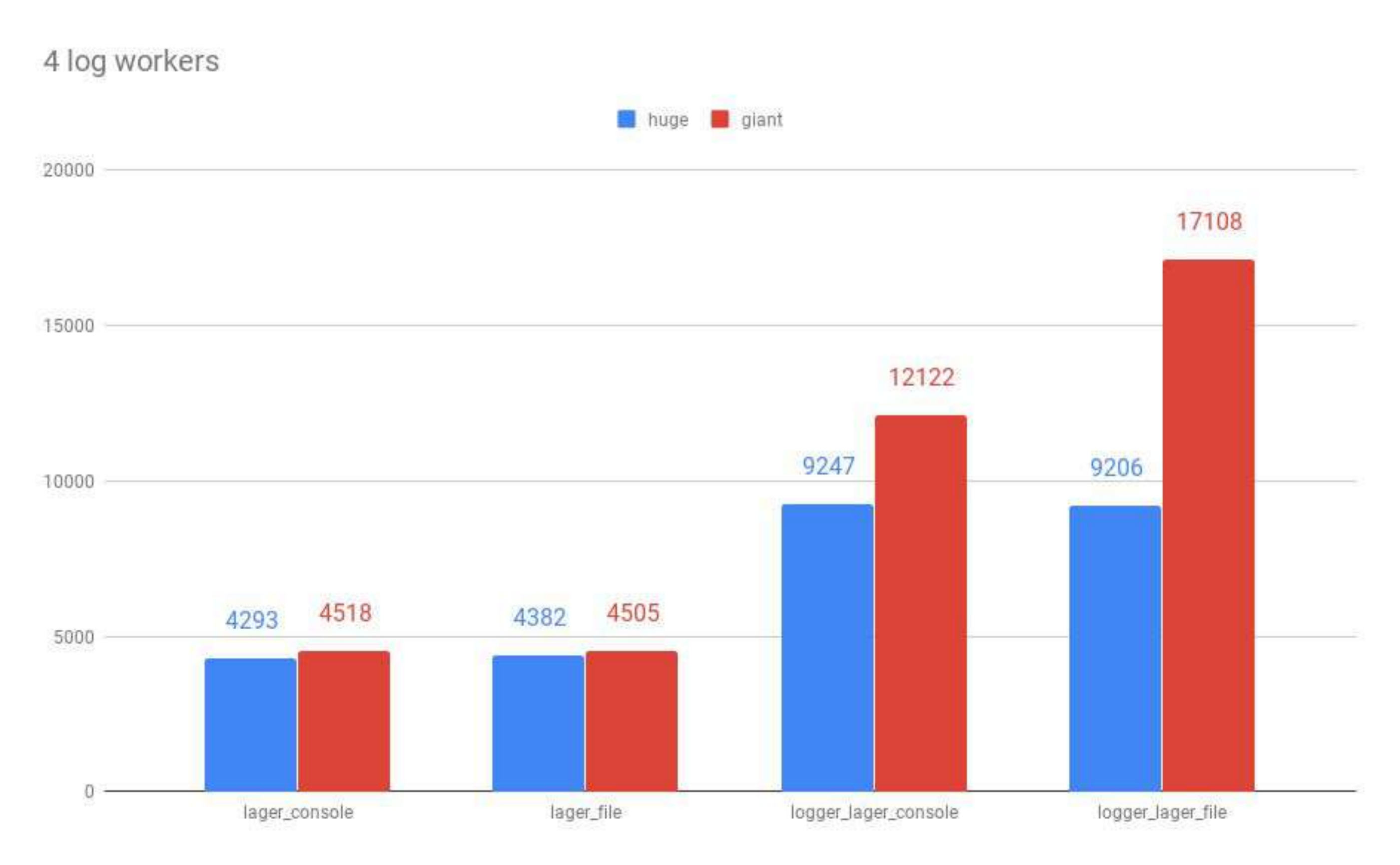
4 log workers



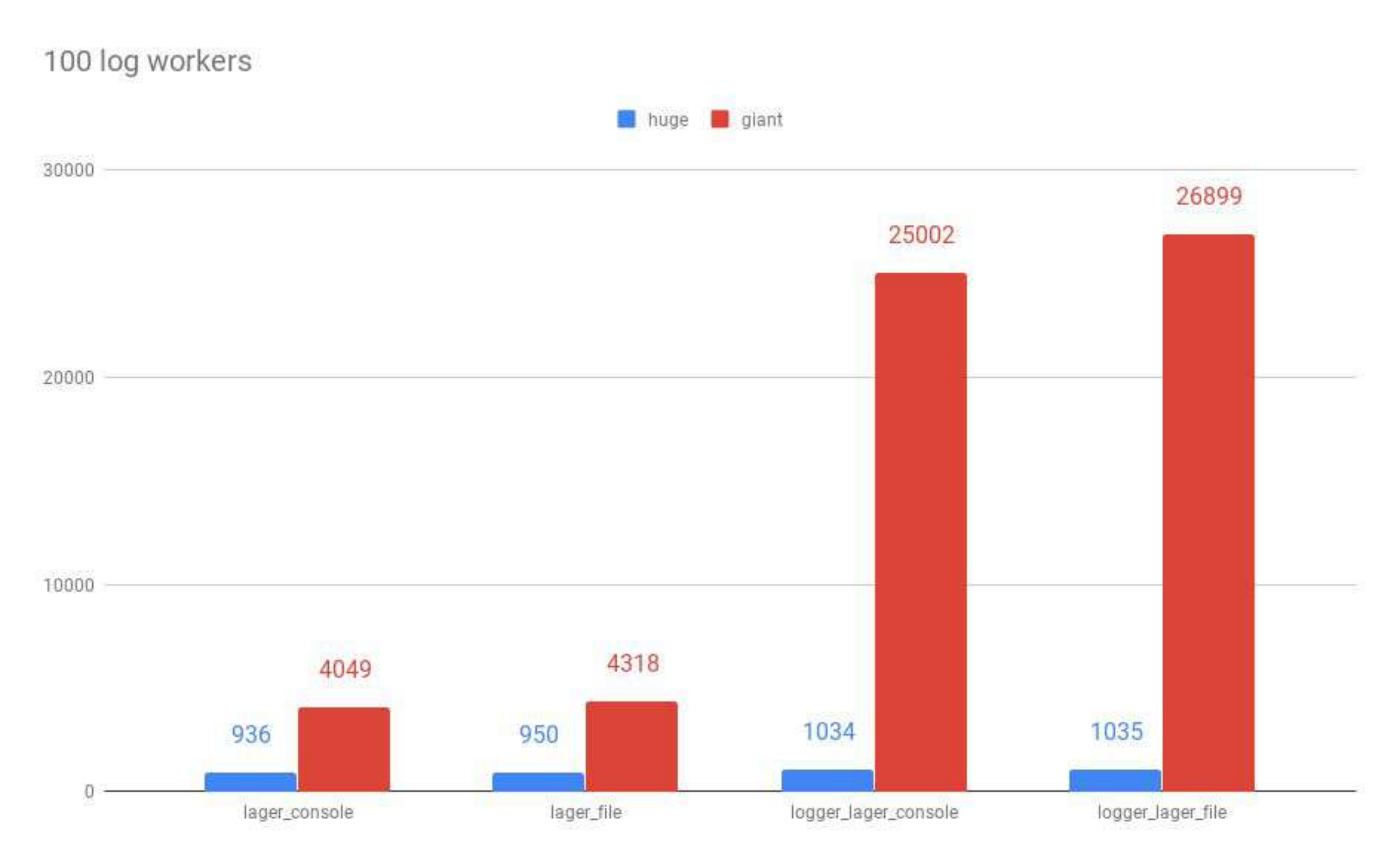
100 log workers



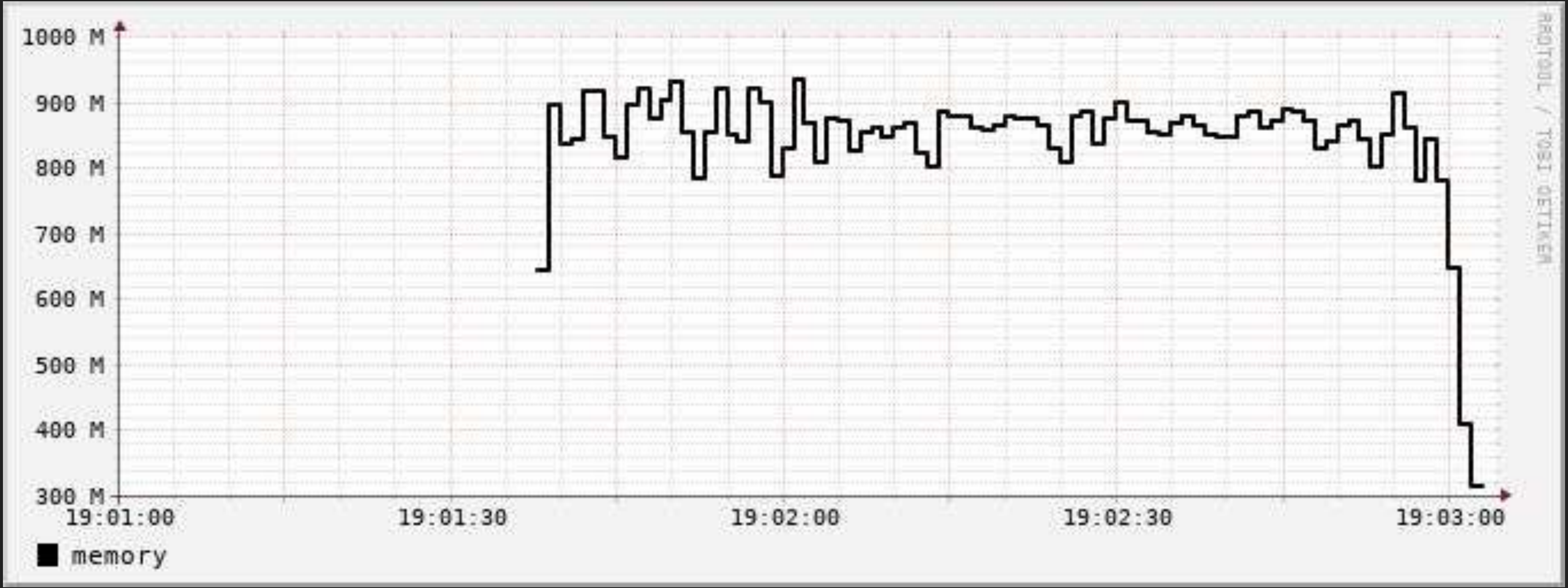
1 log workers



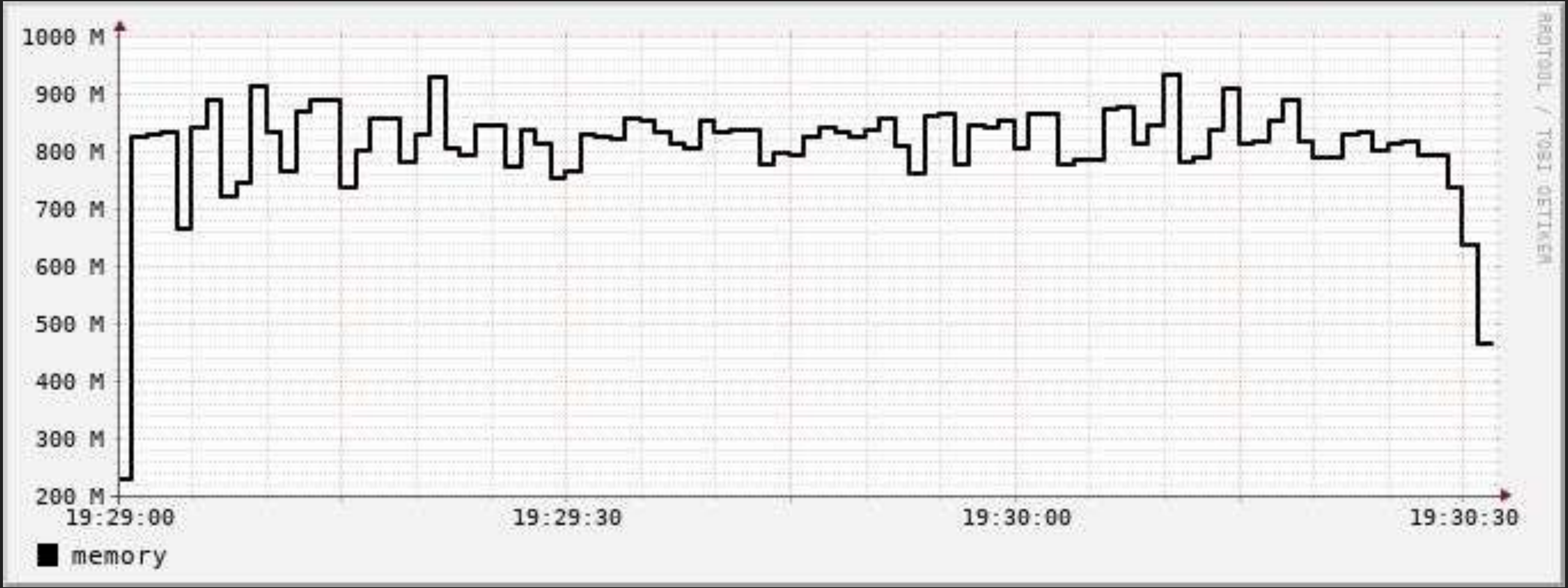
4 log workers



100 log workers



Logger console, 100 workers, large



Logger file, 100 workers, large

Logbench notes & conclusions

- ‘Simple’ messages are 5 bytes, ‘small’ is 56 bytes, large are ~96kb, huge are 4mb, giant are 16mb
- Logbench measures message flow, not log messages written
- Logger’s message size limiters are slow (OTP team working on it)
- Lager is quite a lot faster than logger when a single process is generating all the log events
- Without message size limiting, logger gets overwhelmed
- Logger aggressively drops ‘giant’ messages
- Using lager’s message format limiting with logger’s event pipeline is the fastest combination

Wrap up

- Logger is still maturing, it still needs work and user feedback
- Logger has less features than lager
- Logger does not ship with a safe default configuration
- Extra logger handlers incur a higher cost
- If you’re using lager today, be cautious about switching just yet
- If you’re using logger today, make sure you check your configuration under load
- If you’re using error_logger pre OTP 21, please upgrade or use literally anything else
- Going forward, let’s try to standardize on logger (or at least its API) and stop reinventing the logging wheel

Questions?