$500 for your favorite OSS project

herokulove.com

# THANK YOU!

TO _____

LET SOMEONE KNOW THEY'RE SPECIAL - SEND A HUG WITH THIS CARD.

# Rails Core + Contributors Office Hours

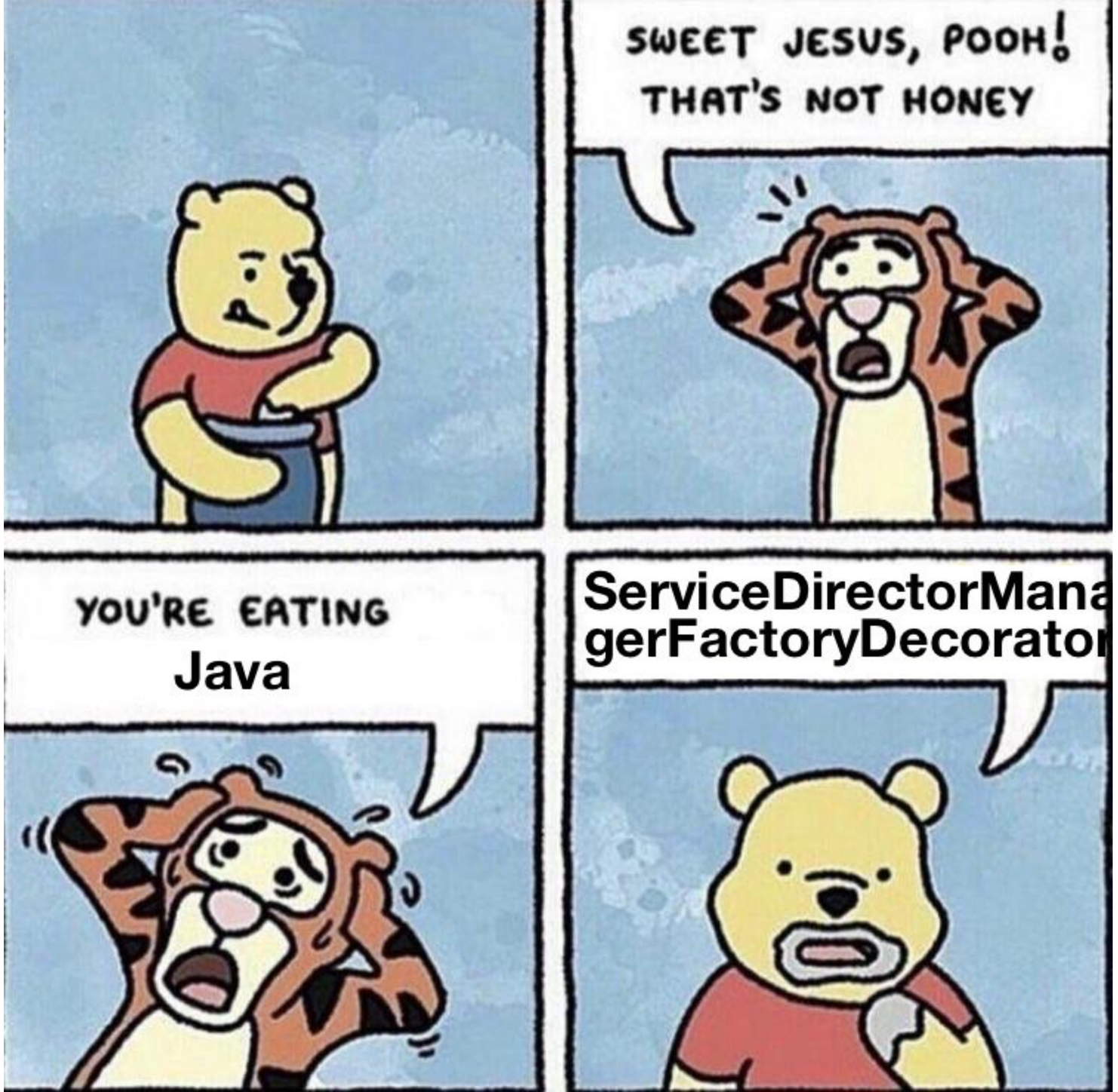
Right after this talk @ Heroku Booth
4:10-5:30 PM

# YOUR APP SERVER CONFIG IS WRONG

# WHO THE HECK AM I?

MEMELORD


DOES HE BITE?
NO, BUT HE CAN HURT YOU IN OTHER WAYS
Would you like to see some code you wrote six months ago?


SWEET JESUS, POOH!
THAT'S NOT HONEY
YOU'RE EATING
Java
ServiceDirectorMana
gerFactoryDecorator

# Who would win

popular web framework


512MB of RAM

# Understanding Ruby GC through GC.stat

by **Nate Berkopec** (@nateberkopec) of ***speedshop*** (who?), a Rails performance consultancy.

**Summary:** Have you ever wondered how the heck Ruby's GC works? Let's see what we can learn by reading some of the statistics it provides us in the GC.stat hash. *(1560 words/8 minutes)*



Most Ruby programmers don't have any idea how garbage collection works in their runtime - what triggers it, how often it runs, and what is garbage collected and what isn't. That's not entirely a bad thing - garbage collection in dynamic languages like Ruby is usually pretty complex, and Ruby programmers are better off just focusing on writing code that matters for their users.

I call that an object leak.

But, occasionally, you get bitten by GC - either it's running too often or not enough, or your process is using tons of memory but you don't know why. Or maybe you're just curious about how GC works!

RAILSSPEED.COM

INCORRECT APP SERVER CONFIGURATION IS THE MOST COMMON ISSUE I SEE ON CLIENT APPLICATIONS.

# OVERPROVISIONING

# GENERAL RULE:

## IF YOU SPEND MORE ON HEROKU PER MONTH THAN YOU HAVE RPM, YOU'RE OVERPROVISIONED

# UNDERSIZING

# DEFINITIONS

# CONTAINER (DYNO)

# WORKER (PROCESS)

# THREAD

# PROCESS:

1. Determine theoretical capacity
2. Determine memory usage per worker
3. Choose container size + worker counts
4. Check connection limits
5. Deploy and monitor queue depths, CPU, memory, restarts, timeouts.

# LITTLE'S LAW

$$l = \lambda * W$$

# THINGS IN SYSTEM = ARRIVAL RATE X TIME SPENT IN SYSTEM

# REQUESTS IN SYSTEM = REQUESTS/SECOND X AVERAGE RESPONSE TIME

DIVIDING AVERAGE REQUESTS IN SYSTEM BY HOW MANY WORKERS WE HAVE GIVES US A UTILIZATION PERCENTAGE

# AVERAGES

# ENVATO, 2013

» Envato receives 115 requests per second

» They run an average of 147ms response time

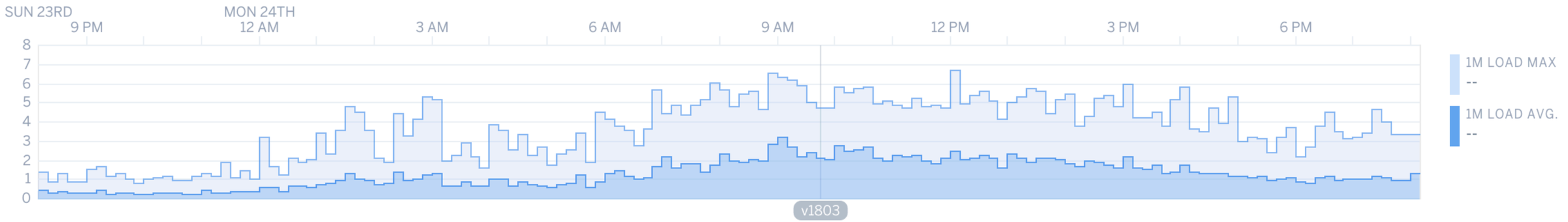
» They run 45 workers

$$16.9 = 115 * 0.147$$

# 16.9 / 45 = 37% UTILIZATION

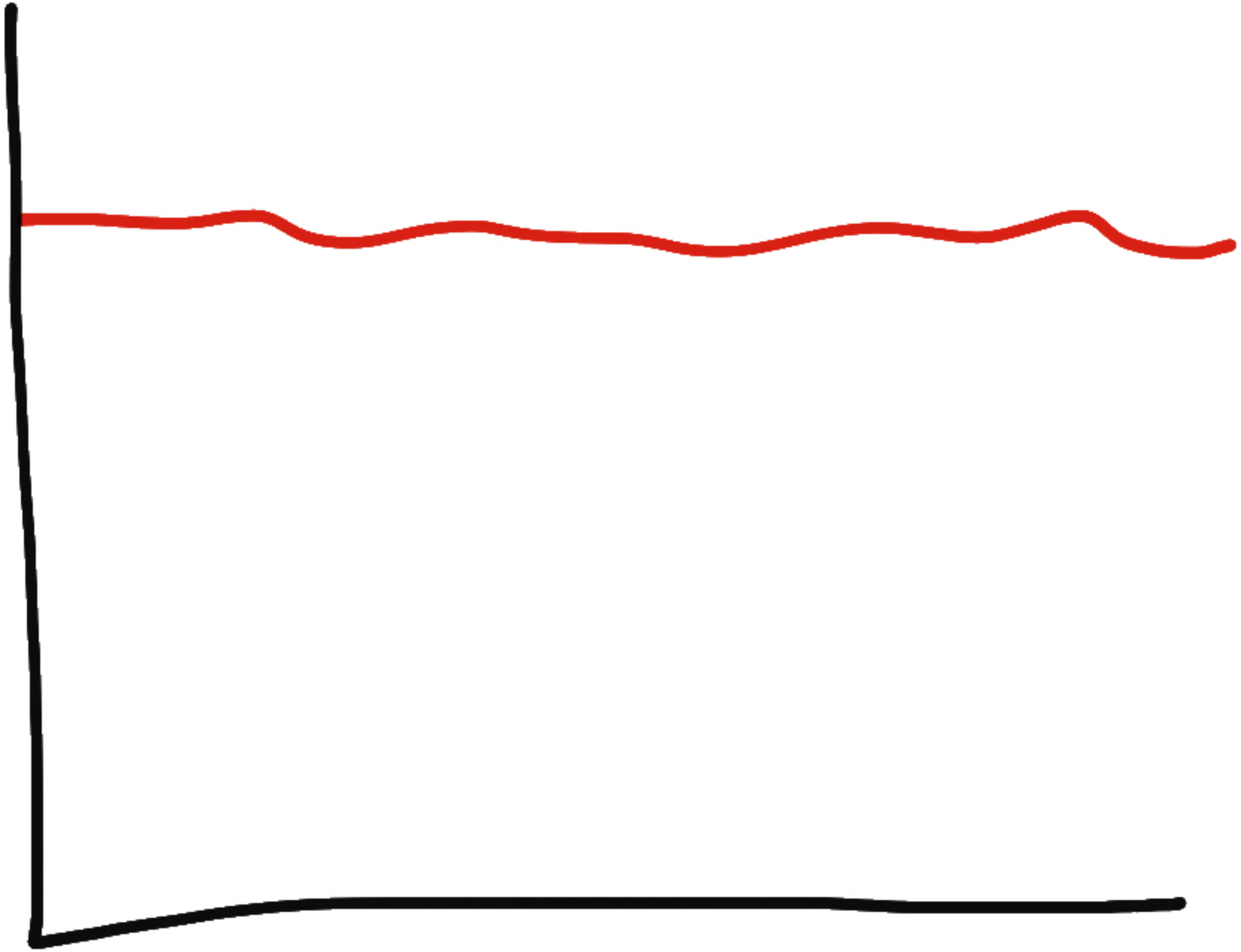
# TAKE LITTLE'S LAW, MULTIPLY BY A FACTOR OF 5 TO GET YOUR INITIAL ESTIMATE

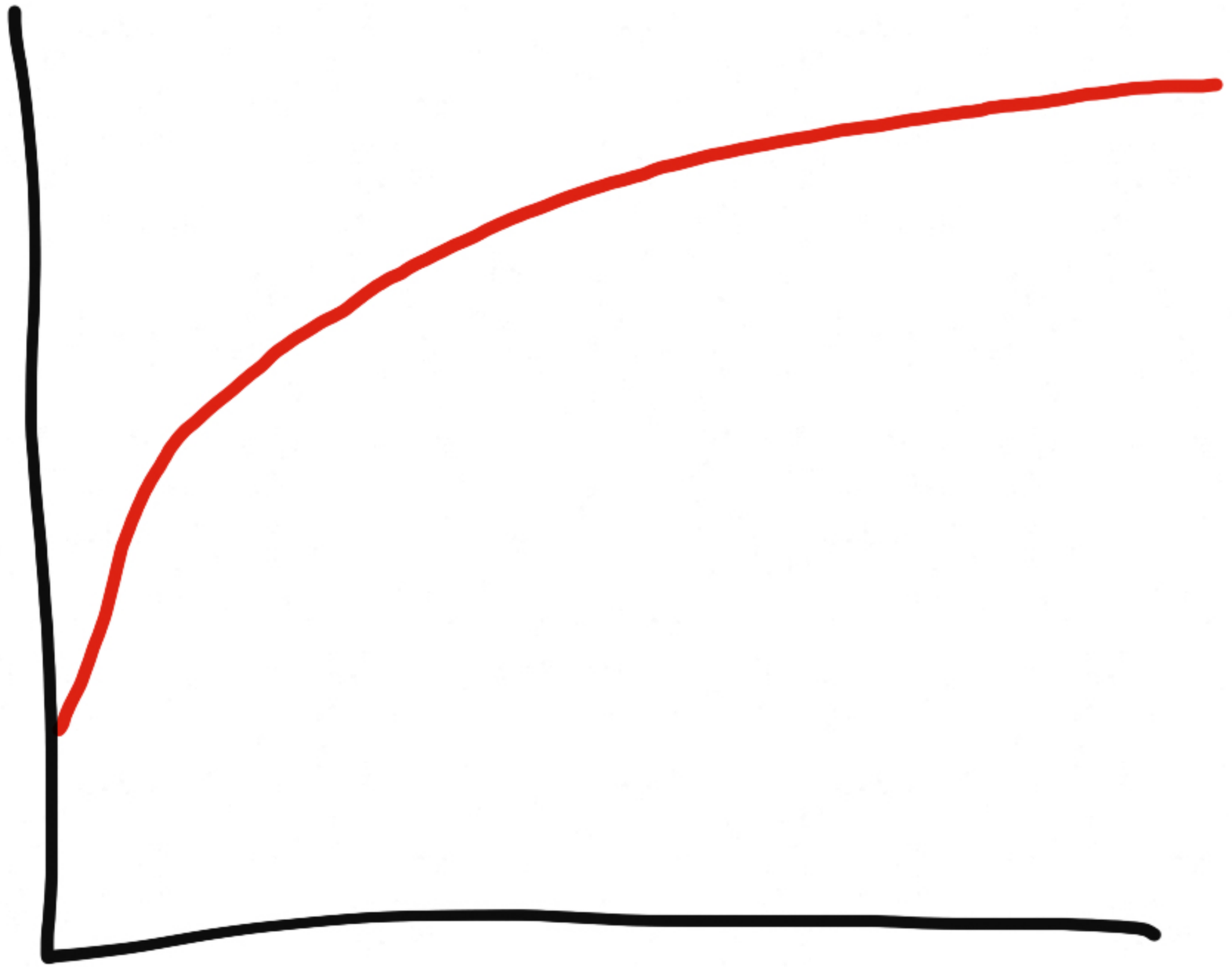
## Dyno Load

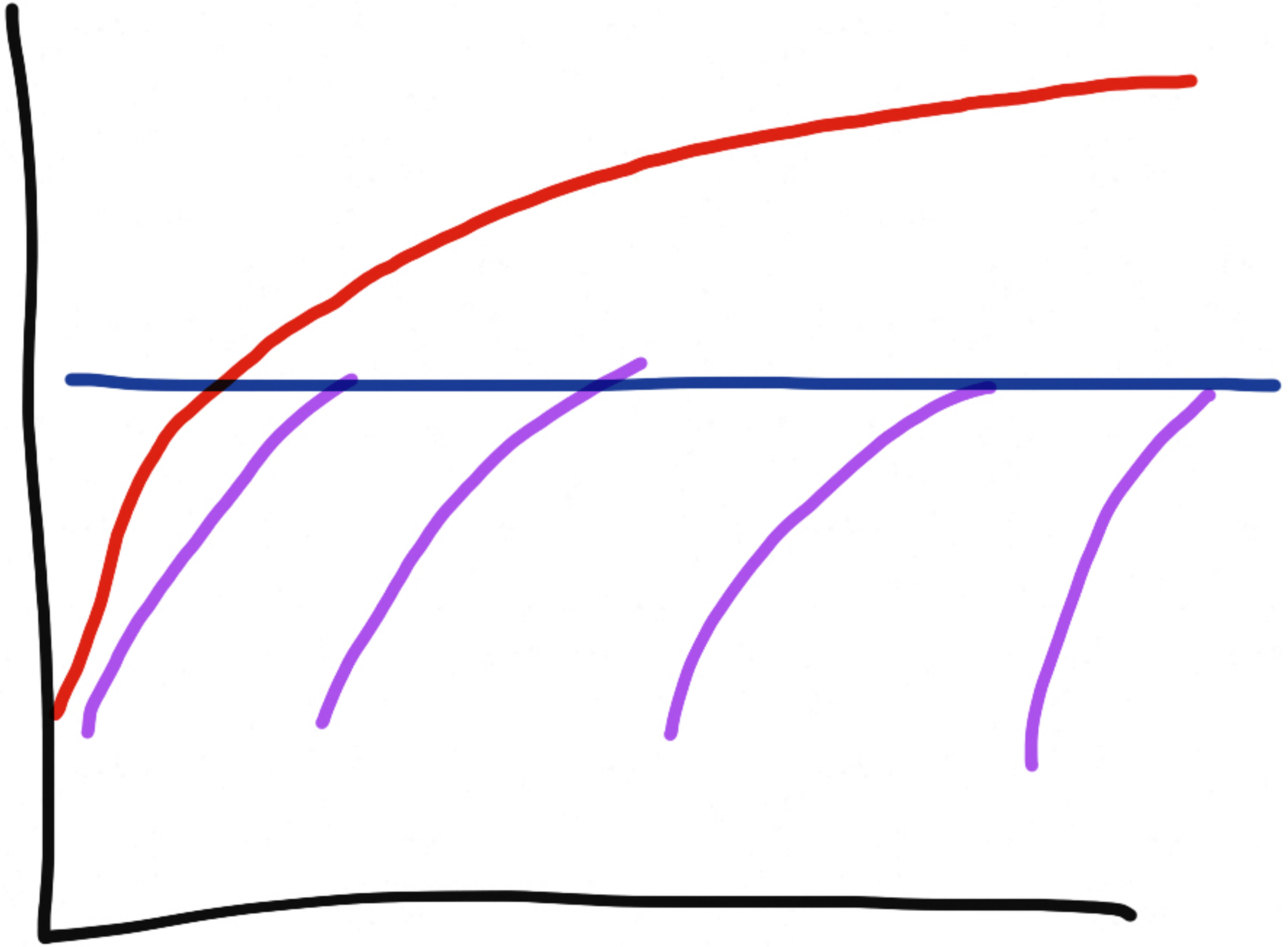

SUN 23RD
MON 24TH
9 PM
12 AM
3 AM
6 AM
9 AM
12 PM
3 PM
6 PM
8
7
6
5
4
3
2
1
0
v1803
1M LOAD MAX
--
1M LOAD AVG.
--

# STEP 1: ESTIMATE WORKER COUNT

# CONTAINER SIZE?







# DEPLOY WITH 1X/2X DYNOS, 1 WORKER PER DYNO, 5 THREADS PER WORKER, AND MEASURE AVERAGE RSS AFTER 24 HOURS.

# AVERAGE APP IS 256-512MB PER WORKER.

# you have been visited by pixel DHH

he is very rare and appears only once in every 5000 davids

## like in 5 seconds or less to receive good memory usage for life

# STEP 2: DETERMINE MEMORY PER WORKER

# CONTAINER CHOICE ON HEROKU

| NAME | RAM | VCPU COUNT | "COMPUTE" | $/MONTH |
|---|---|---|---|---|
| 1x | 512MB | 8 | 1-4x | 25 |
| 2x | 1GB | 8 (2x share) | 4-8x | 50 |
| Perf-M | 2.5GB | 2 | 12x | 250 |
| Perf-L | 14GB | 8 | 50x | 500 |

| NAME | RAM/$ | $/COMPUTE |
| --- | --- | --- |
| 1x | 21 | 10 |
| 2x | 21 | 8.3 |
| Perf-M | 10 | 20 |
| Perf-L | 28 | 10 |

# NOTE THAT 2X DYNOS ACCESS 8 CPUS

## HIGHER THREAD COUNTS REQUIRED?

# IF > ~25 APP INSTANCES, USE PERF-L. OTHERWISE, USE 2X.

# AIM FOR AT LEAST 3 WORKERS PER DYNO (MAY HAVE TO PERF-M OTHERWISE)

# HIGHER WORKERS PER DYNO EQUALS BETTER ROUTING

YOU CAN REDUCE THREAD COUNT TO 3 OR USE JEMALLOC TO TRY TO SQUEEZE ANOTHER WORKER INTO A DYNO

# WORKERS CAN BE 3-4X CORE COUNTS

# KEEP THREAD COUNTS TO 3-5.

# HOW DO I KNOW IF MY APP IS THREADSAFE?

» Start slow, with 2 threads.

» Try minitest/hell.

» It's probably fine (on MRI), watch for globals (Redis) or class-level state (User.current)

# STEP 3: CHOOSE CONTAINER SIZE AND WORKER COUNT

# THINGS THAT USE CONNECTIONS: ACTIVERECORD, REDIS, MEMCACHED, ETC

# CONNECTION POOLS AND LIMITS TO WATCH

» ActiveRecord (database.yml). 1 connection per thread.

» Postgres/database.

» Redis (Sidekiq, etc).

» Memcache (usually unlimited).

# YOU MAY NEED MORE THAN 1 DB CONNECTION PER THREAD

| NAME | CONNECTIONS |
| --- | --- |
| Hobby | 20 |
| Standard-0 | 120 |
| Standard-2 | 400 |
| Standard-4 | 500 |

# NEED MORE THAN 500 CONNECTIONS? USE PGBOUNCER.

# DO THE MATH TO FIGURE OUT HOW MANY DYNOS OUTSCALE YOUR LIMITS.

# EXAMPLE

» I have a Perf-L dyno with 20 app workers, each with thread count of 5.

» That's 100 threads (and 100 DB connections) per dyno.

» I can scale to 5 of these dynos before outscaling my connection limits.

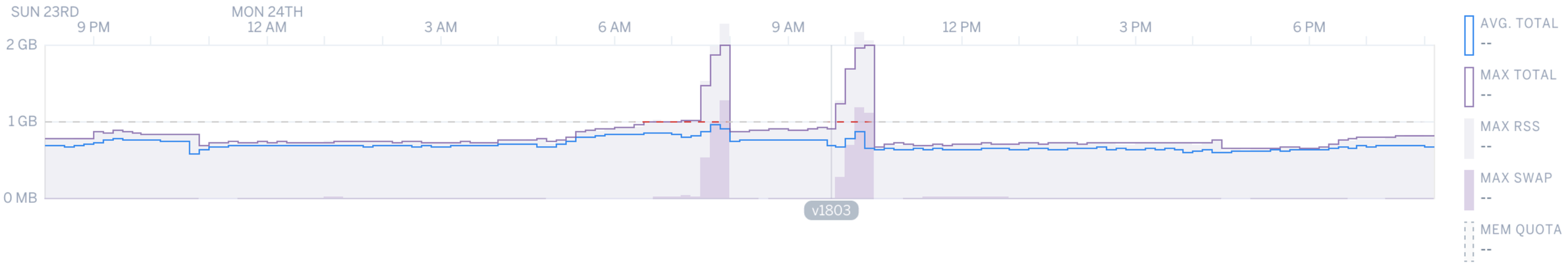
# STEP 4: CHECK CONNECTION LIMITS

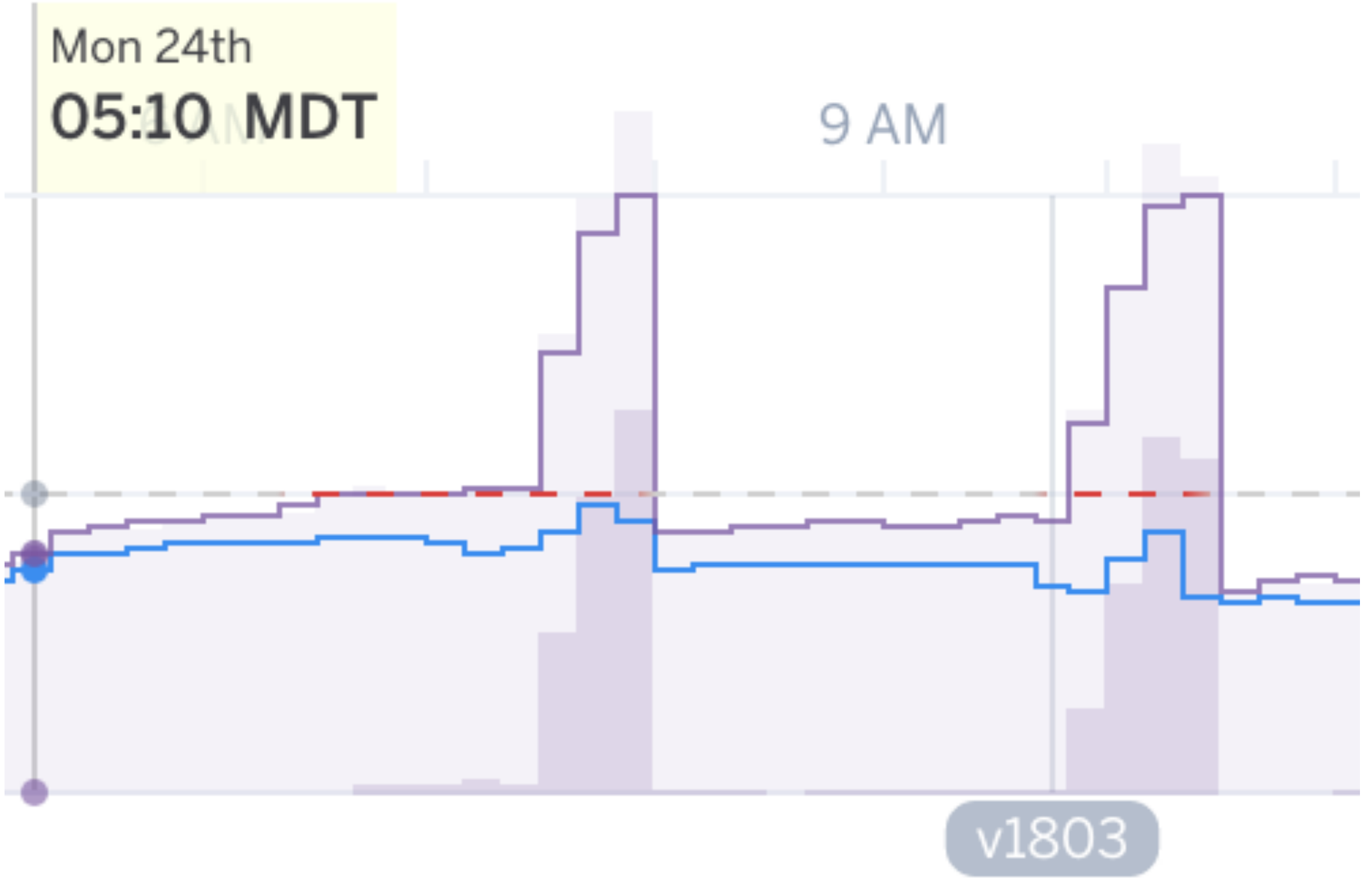
# THINGS TO WATCH AFTER DEPLOYMENT

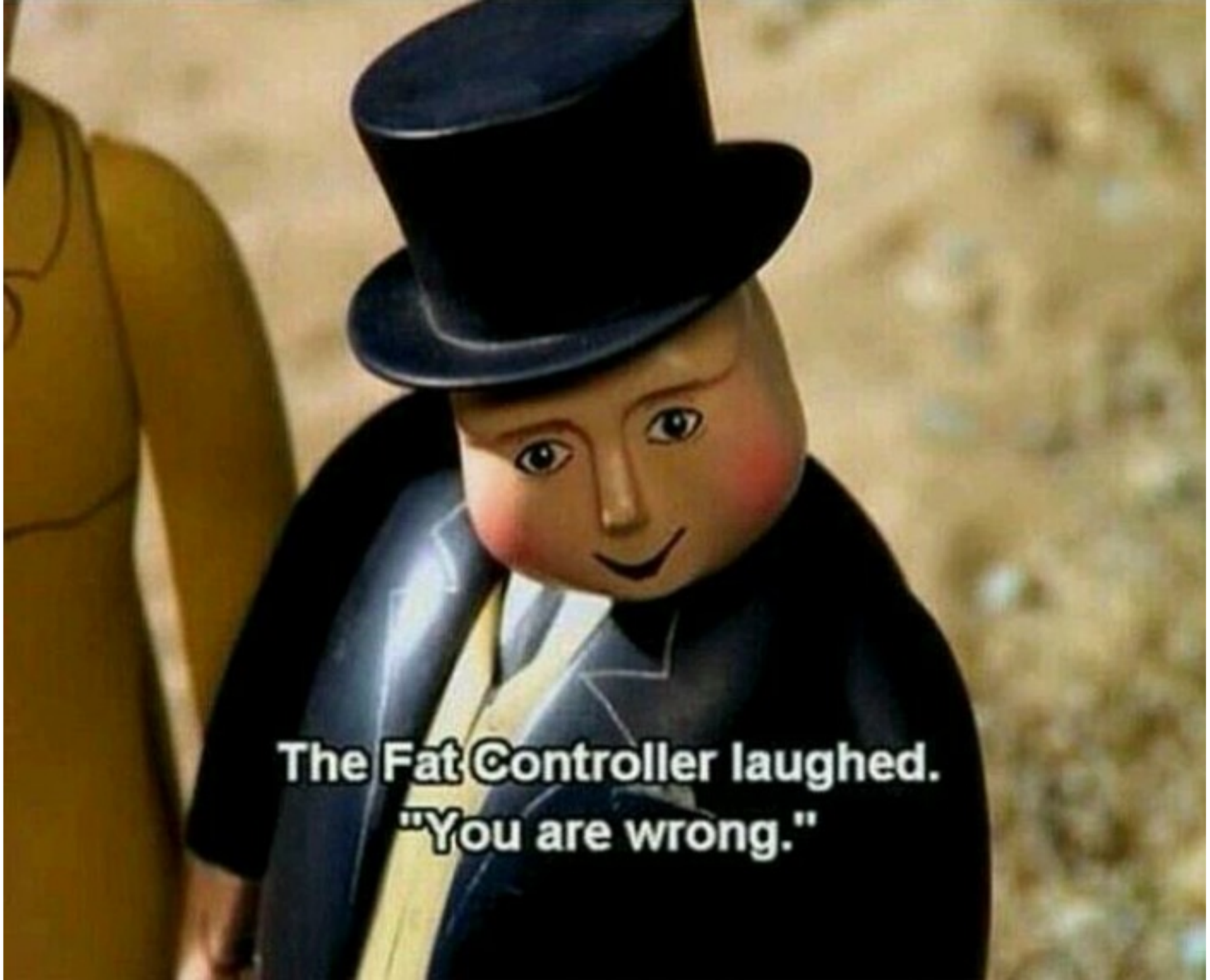
# MEMORY

## Memory Usage

| Latest | Max | Average |
| --- | --- | --- |
| 82.4 % 833.0 MB | ⚠ 229.7 % 2,352.6 MB | 70.2 % 718.7 MB |




Mon 24th
05:10 MDT
9 AM
v1803


The Fat Controller laughed.
"You are wrong."

SKYLIGHT
SCOUT
oink

**IF TOO MUCH, SCALE DOWN**

WEB_CONCURRENCY

**IF TOO LITTLE, SCALE UP**

WEB_CONCURRENCY

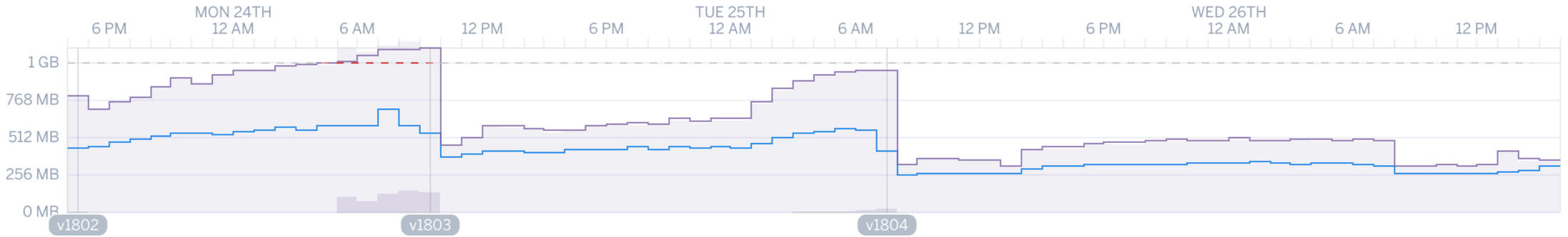
# CAN ALSO TWEAK THREAD COUNTS

jemalloc **OR**

MALLOC_ARENA_MAX

## Memory Usage

| Latest | Max | Average |
| --- | --- | --- |
| 35.1 % 363.4 MB | ⚠ 115.2 % 1,179.8 MB | 40.2 % 411.6 MB |



# QUEUE TIMES

# HIGH QUEUE TIMES? MORE DYNOS.

# CPU USAGE

# IF LOW, MAY BENEFIT FROM MORE THREADS.

# RESTARTS

nurse: sir... you've been in a dyno since 31 seconds ago

me: wow, i can't wait to get back to my user and give them this response I worked so hard on

# APPS WHICH HAVE HIGH 95TH PERCENTILE TIMES MAY BENEFIT FROM HIGH WORKERS-PER-DYNO

# passenger_max_request_time and worker_timeout

# PROCESS:

1. Determine theoretical capacity
2. Determine memory usage per worker
3. Choose container size + worker counts
4. Check connection limits
5. Deploy and monitor queue depths, CPU, memory, restarts, timeouts.

preload_app!

# lowlevel_error_handler

queue_requests