A Beginner's Guide to Puma Internals

Learn everything you need to know to contribute to Puma

Nate Berkopec for RubyConf TH, 2022

Midjourney: "A cybernetic puma cat, growling"

speedshop

@nateberkopec/speedshop.co

nateberk.gumroad.com

I write about Rails and Ruby performance. Follow me for updates on what I am creating.

The Complete Guide to Rails Performance

★ 5.0 (23) $50

The Ruby on Rails Performance Apocrypha

★ 4.9 (19) $10

Sidekiq in Practice

★ 5.0 (32) $50

README.md

Puma: A Ruby Web Server Built For Parallelism

Puma is a **simple, fast, multi-threaded, and highly parallel HTTP 1.1 server for Ruby/Rack applications**.

Built For Speed & Parallelism

Puma processes requests using a C-optimized Ragel extension (inherited from Mongrel) that provides fast, accurate HTTP 1.1 protocol parsing in a portable way. Puma then serves the request using a thread pool. Each request is served in a separate thread, so truly parallel Ruby implementations (JRuby, Rubinius) will use all available CPU cores.

Originally designed as a server for Rubinius, Puma also works well with Ruby (MRI) and JRuby.

On MRI, there is a Global VM Lock (GVL) that ensures only one thread can run Ruby code at a time. But if you're doing a lot of blocking IO (such as HTTP calls to external APIs like Twitter), Puma still improves MRI's throughput by allowing IO waiting to be done in parallel.

puma 6.0.0

Puma is a simple, fast, threaded, and highly parallel HTTP 1.1 server for Ruby/Rack applications. Puma is intended for use in both development and production environments. It's great for highly parallel Ruby implementations such as Rubinius and JRuby as well as as providing process worker support to support CRuby well.

VERSIONS:

6.0.0 - October 14, 2022 java (237 KB)
6.0.0 - October 14, 2022 (213 KB)
5.6.5 - August 23, 2022 java (224 KB)
5.6.5 - August 23, 2022 (204 KB)
5.6.4 - March 30, 2022 java (224 KB)

Show all versions (301 total) →

RUNTIME DEPENDENCIES (1):

nio4r ~> 2.0

Show all transitive dependencies →

OWNERS:

PUSHED BY:

Star 7,262

TOTAL DOWNLOADS
248,688,773

FOR THIS VERSION
1,340,067

GEMFILE:

```
gem 'puma', '~> 6.
```

INSTALL:

```
gem install puma
```

LICENSE:
BSD-3-CLAUSE

REQUIRED RUBY VERSION:
>= 2.4

NEW VERSIONS REQUIRE MFA:
TRUE

OSS contribution should be:

- Fun
- A good way to learn

OSS contribution to Puma is:

- Fun
- A good way to learn

Midjourney: “a lazy puma cat, kawaii, cute, anime style, bright color background, simple drawing”

Midjourney: “An army of nerds with laptops”

Midjourney: “cybernetic uncle sam, pointing at camera, "i want you" poster, tron, futuristic, cyborg, android --

If you make enough contributions, we let you “name” the next minor/ major version.

Midjourney: "a clockwork puma cat, clockwork mechanism, complicated, highly detailed"

If you remember 10% of this presentation, you'll know more about Puma than when I first started maintaining it 5 years ago.

1. Design goals and purpose
2. Processes, threads
3. Code guide
4. Contributing guide

Puma's design goals:

1. More throughput for the same resources, through parallelism.
2. "Batteries included", without the need for additional tools, servers, proxies, or monitors.
3. Add less than 1ms of overhead to every request.
4. Stay simple.

Midjourney: "a studious puma cat, drafting table, architect, wearing glasses, cute, kawaii, anime style, simple cartoon, low detail, sanrio style --v 4"

A web server is an application that accepts connections on a socket, then serves HTTP apps over those connections.

Midjourney: “a puma cat chewing on a cable, cute, kawaii, anime style, simple cartoon, low detail, sanrio style”

Sockets are endpoints for streaming data to and from clients, identified by [IP_addr, port]

We create a socket for each new connection to our server. Each socket has a file descriptor.

Sockets in Puma are either TCP, TCP w/SSL, or UNIX.

Puma uses mainly the TCPSocket and UnixSocket classes from stdlib:

```
require 'socket'

s = TCPSocket.new 'localhost', 2000

while line = s.gets # Read lines from socket
  puts line         # and print them
end

s.close             # close socket when done
```

HTTP is an application protocol.

Puma only speaks HTTP 1.x.

```
GET /images/logo.png HTTP/1.1
Host: www.example.com

Hello!
```

A Rack server is a type of web server that serves Rack-compatible applications written in Ruby.

Midjourney: “a rack of servers, cartoon style, cute, kawaii, anime, anthropomorphic, low detail”

Rack applications are objects which respond to `call` (with an env argument) and return:

```
[
  200,
  {“Header” => “Value”},
  [“The response body”]
]
```

```
Proc.new { |env|
  [
    200,
    {“Header” => “Value”},
    [“The response body”]
  ]
}
```

You define a Rack application usually in a “Rackup” file, usually config.ru:

```
require ::File.expand_path('../config/environment', __FILE__)
run Rails.application
```

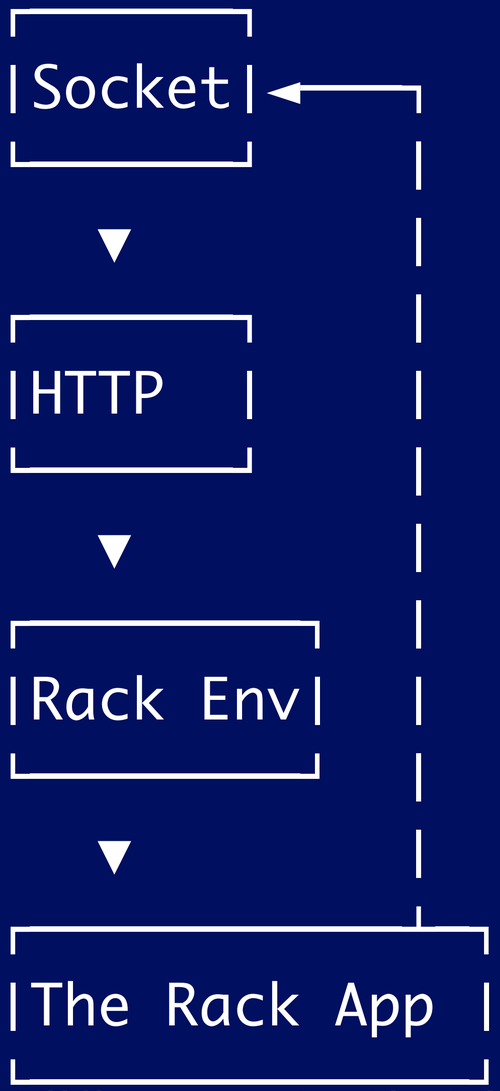
The Rack server calls the application with the environment hash.

```
{
  “REQUEST_METHOD” => “GET”,
  “PATH_INFO” => “/“,
  “HTTP_VERSION => “1.1”
  …
}
@app.call(env)
```

Rack servers are interfaces between Rack-compatible Ruby applications and a socket.

Socket -> HTTP -> The Rack Env

Our mental model:



Midjourney: “a thinking puma cat, thinking man pose, cartoon style, cute, kawaii, anime, anthropomorphic, low detail”

Puma, like almost all Ruby web servers, is a “pre-forking” web server.

A pre-forking web server boots a single “parent” process, boots the application, listens on the socket, then calls “fork(2)” to create child processes.

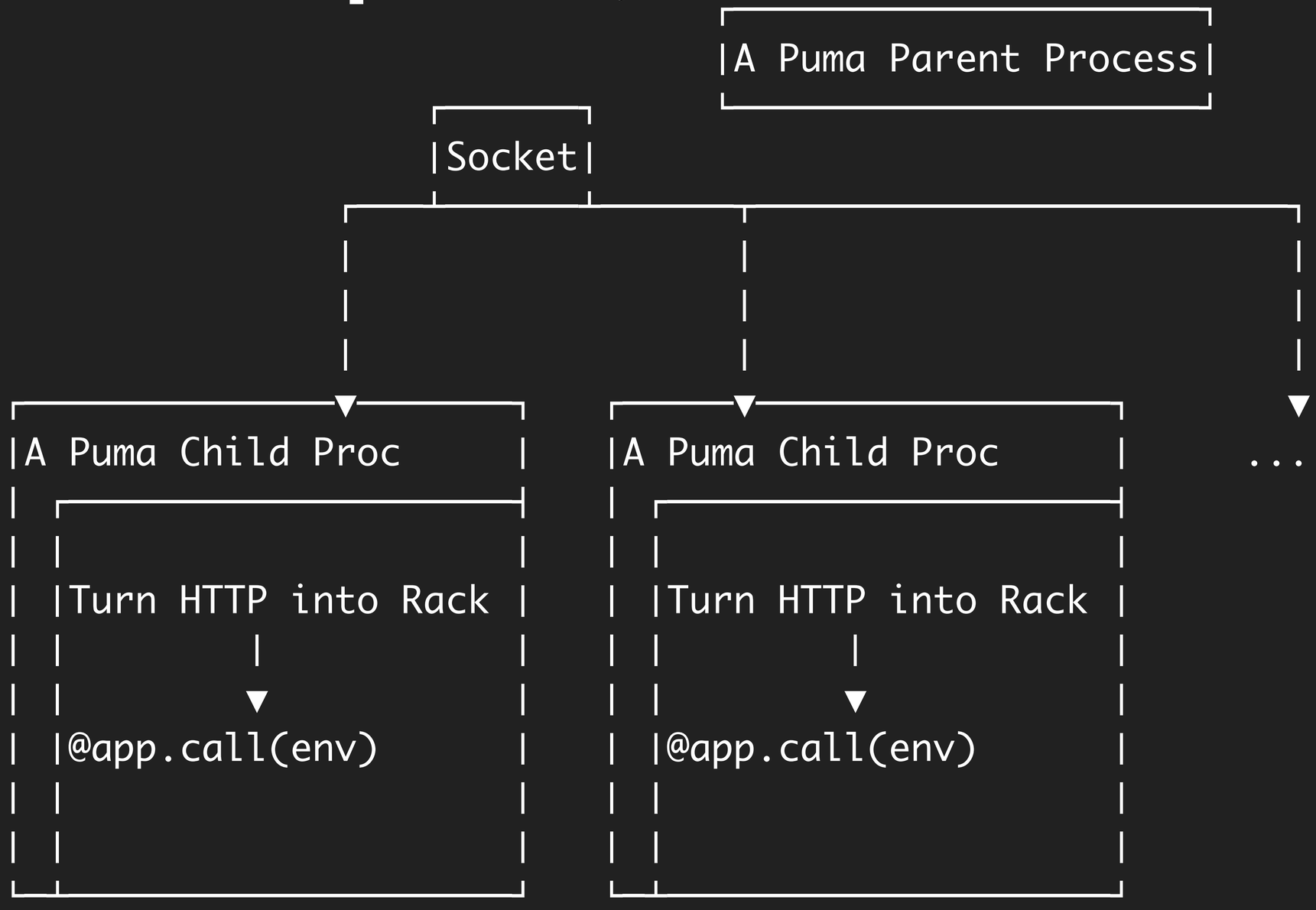
After the app boots, child processes serve requests. The parent does not.

The parent processes's job is to receive signals, and to boot new child processes.

puma -w 4

Midjourney: “group of 5 puma cats, cartoon style, cute, kawaii, anime, anthropomorphic, low detail”

At this point, our mental model is:



The Threadpool

Midjourney: “a pool of threads”

```
work_to_do = []
thread_pool = Array.new(5) { Thread.new {
block_that_consumes(work_to_do) } }
# …
work_to_do << Request.new(this_request)
```

The Global VM Lock (GVL) allows only one thread to run Ruby code at any point in time.

The GVL is like a special machine being passed around between customs agents at a border checkpoint.

The main thing Ruby processes do other than run Ruby is to wait on I/O.

Midjourney: “a sleepy puma, lazy, cartoon style, cute, kawaii, anime, anthropomorphic, low detail”

The Ractor VM Lock (RVL???) allows only one thread in a Ractor to run Ruby code at any point in time.

This is Ruby-implementation specific.

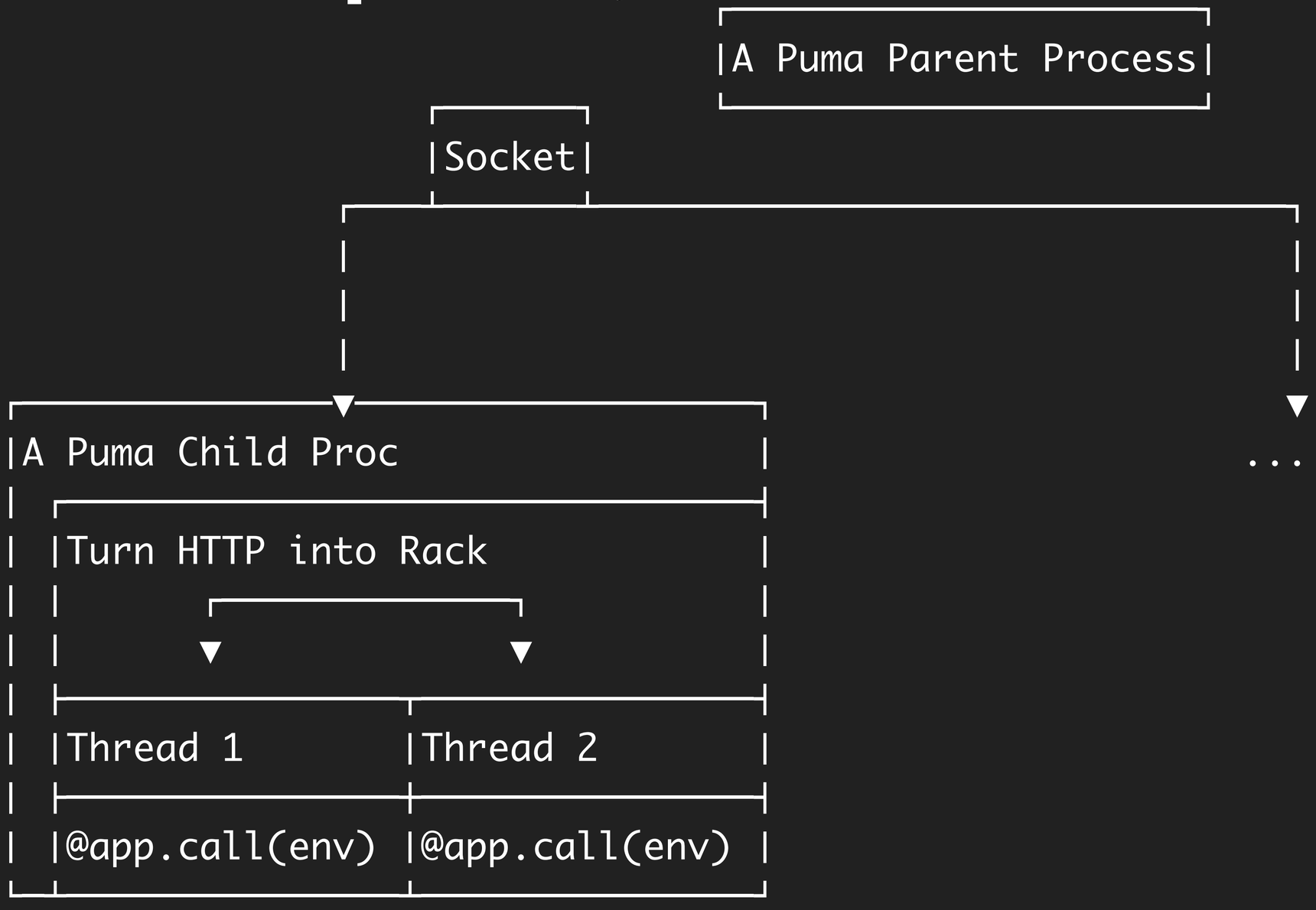
Midjourney: “a gemstone shaped cartoon character, ruby, cute, kawaii, anime, sanrio style, thumbs up pose, low detail”

A Puma process running a 50%-of-time-in I/O wait app, in MRI, with about 4 threads can process 2x as many requests.

This is Puma’s primary benefit: more throughput for a similar amount of memory resources.

Midjourney: “a puma cat in a chinese propaganda poster”

At this point, our mental model is:



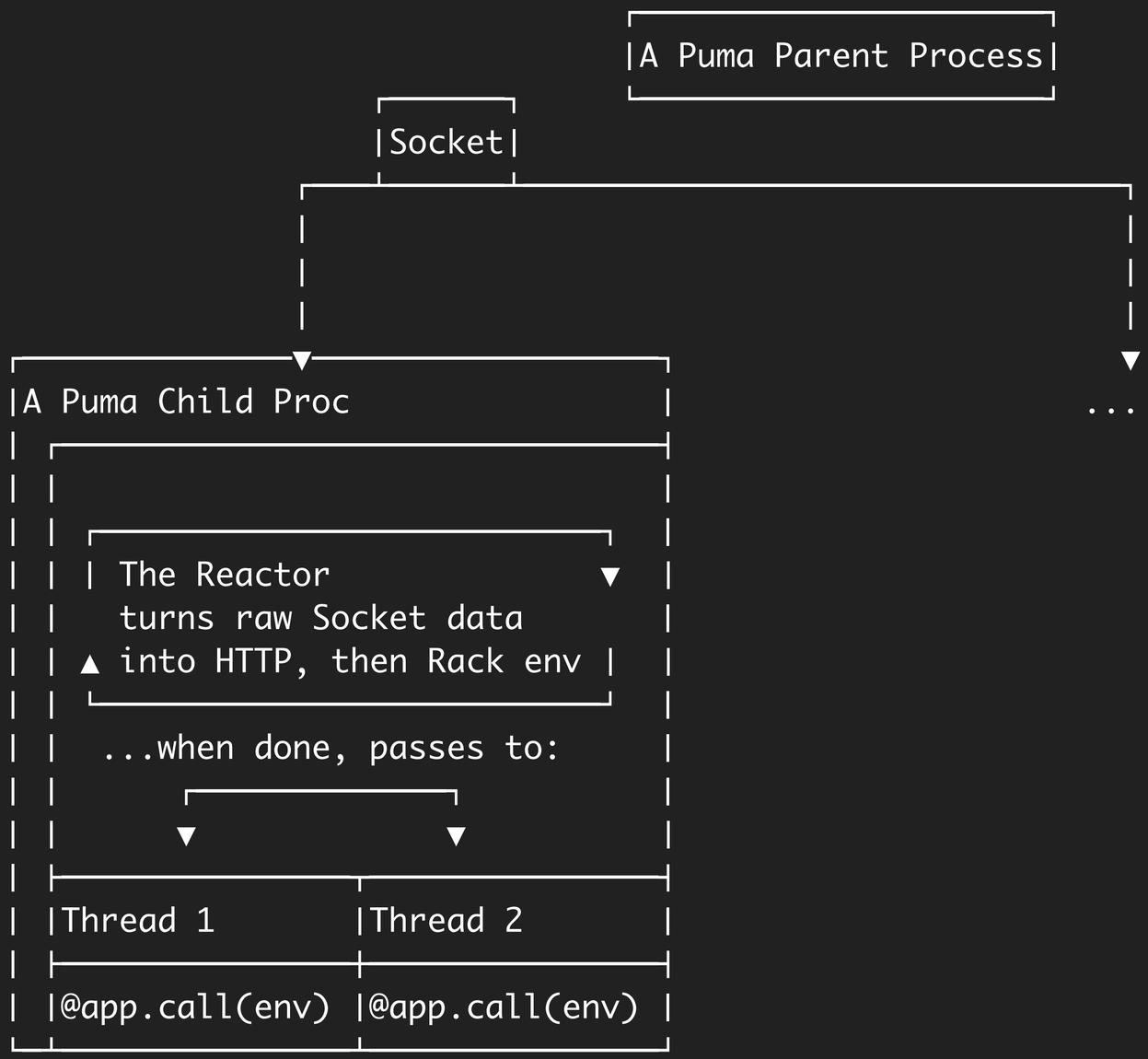
The Reactor’s job is to buffer requests, so that only complete requests are sent to the thread pool.

This prevents slow clients (uploads, attackers) from consuming our ThreadPool.

Unicorn lacks a Reactor, which is why you should always use a proxy in front of it, like nginx or apache.

Midjourney: “a unicorn cartoon character, ruby, cute, kawaii, anime, sanrio style, thumbs up pose, very low detail”

At this point, our final model is:



cloc --by-file --exclude_dir=test,tmp,pkg,benchmarks,examples --include_ext=rb,c,java .

```
     109 text files.
      98 unique files.
      66 files ignored.

github.com/AlDanial/cloc v 1.94  T=0.57 s (87.1 files/s, 22109.7 lines/s)
```

File	blank	comment	code
./ext/puma_http11/http11_parser.c	30	13	1014
./ext/puma_http11/mini_ssl.c	156	35	554
./lib/puma/client.rb	101	60	434
./ext/puma_http11/org/jruby/puma/MiniSSL.java	72	19	417
./lib/puma/server.rb	90	121	405
./lib/puma/cluster.rb	101	49	399
./lib/puma/request.rb	77	147	385
./lib/puma/dsl.rb	97	589	370
./ext/puma_http11/org/jruby/puma/Http11Parser.java	66	27	362
./lib/puma/binder.rb	85	53	360
./lib/puma/launcher.rb	88	68	341
./ext/puma_http11/puma_http11.c	91	84	317
./lib/puma/minissl.rb	68	69	299
./lib/puma/control_cli.rb	53	11	245
./lib/puma/thread_pool.rb	66	89	244
./lib/puma/configuration.rb	58	89	240
./ext/puma_http11/org/jruby/puma/Http11.java	39	5	182
./lib/puma/cli.rb	49	16	174
./lib/puma/const.rb	40	40	168
./lib/puma/rack/builder.rb	39	92	166
./lib/puma/runner.rb	38	12	148
./lib/puma/cluster/worker.rb	25	14	136
./lib/puma/util.rb	25	15	101
./lib/puma/log_writer.rb	25	25	87
./lib/rack/handler/puma.rb	21	6	87
./lib/puma/plugin.rb	23	5	83
./lib/puma/launcher/bundle_pruner.rb	21	4	79
./lib/puma/reactor.rb	12	28	76
./lib/puma/json_serialization.rb	9	14	73
./lib/puma/minissl/context_builder.rb	19	0	73
./lib/puma/cluster/worker_handle.rb	15	10	72
./lib/puma/error_logger.rb	21	19	72
./lib/puma/commonlogger.rb	17	23	68
./lib/puma/app/status.rb	19	10	67
./lib/puma/rack/urlmap.rb	19	15	59
./lib/puma.rb	14	13	53
./ext/puma_http11/extconf.rb	13	10	50
./lib/puma/state_file.rb	8	10	50
./lib/puma/single.rb	15	10	42
./lib/puma/null_io.rb	14	7	40
./tools/trickletest.rb	7	0	37
./lib/puma/events.rb	14	7	36
./lib/puma/systemd.rb	10	3	34
./lib/puma/io_buffer.rb	7	2	27
./lib/puma/plugin/tmp_restart.rb	6	3	27
./lib/puma/detect.rb	13	7	24
./lib/puma/jruby_restart.rb	4	1	22
./ext/puma_http11/PumaHttp11Service.java	4	0	13
./ext/puma_http11/no_ssl/PumaHttp11Service.java	4	0	11
./lib/puma/rack_default.rb	2	1	6
SUM:	1910	1950	8829

98 unique files.

File	blank	comment	code
SUM:	910	1950	8829

File	blank	comment	code
./ext/puma_http11/http11_parser.c	30	13	1014
./ext/puma_http11/mini_ssl.c	156	35	554
./lib/puma/client.rb	101	60	434
./ext/puma_http11/org/jruby/puma/MiniSSL.java	72	19	417
./lib/puma/server.rb	90	121	405
./lib/puma/cluster.rb	101	49	399
./lib/puma/request.rb	77	147	385
./lib/puma/dsl.rb	97	589	370
./ext/puma_http11/org/jruby/puma/Http11Parser.java	66	27	362
./lib/puma/binder.rb	85	53	360
./lib/puma/launcher.rb	88	68	341
./ext/puma_http11/puma_http11.c	91	84	317
./lib/puma/minissl.rb	68	69	299

The EXT directory contains native extensions written in C and Java

```
extconf.rb
ext_help.h
http11_parser.c
http11_parser_common.rl
http11_parser.h
http11_parser.java.rl
http11_parser.rl
mini_ssl.c
no_ssl
org
puma_http11.c
PumaHttp11Service.java
```

We roll our own SSL implementation in mini_ssl

Midjourney: “a puma in the style of Kandinsky”

We have an HTTP parser based on Zed Shaw's work in Mongrel

Midjourney: "a puma cat in the style of caravaggio"

The parser uses Ragel, a library that helps us to create the parser:

```
%%{

  machine puma_parser_common;

#### HTTP PROTOCOL GRAMMAR
# line endings
  CRLF = "\r\n";

# character types
  CTL = (cntrl | 127);
  safe = ("$" | "-" | "_" | ".");
  extra = ("!" | "*" | "'" | "(" | ")" | ",");
  reserved = (";" | "/" | "?" | ":" | "@" | "&" | "=" | "+");
```

We are in dire need of contribution to /ext, particularly regarding:

Java
OpenSSL
C

Midjourney: “a cartoon character that looks like a cup of coffee, anthromorphic, kawaii, cute, sanrio style, low detail”

The Server Classes:

- Server
- Cluster
- Cluster::Worker
- Runner
- Threadpool

Midjourney: “a puma drawn in the style of Yoji Shinkawa”

The Runner is created, either the Cluster or Single class. Runners create 1 Server, which has a Threadpool. Cluster Runners have many Cluster::Workers.

The Single runner

Output the “booting” message

Bind to sockets

Create bg threads for plugins

Inform the Launcher object

Start control server, if desired

Create Server object

Fire events

Let the server thread take over

```
def run
  output_header "single"

  load_and_bind

  Plugins.fire_background

  @launcher.write_state

  start_control

  @server = server = start_server
  server_thread = server.run

  log "Use Ctrl-C to stop"
  redirect_io

  @events.fire_on_booted!

  begin
    server_thread.join
  rescue Interrupt
    # Swallow it
  end
end
```

Cluster.rb’s run method is more complicated, because it has to:

1. Setup signal handling
2. Preload the application
3. Spawn child workers

The Server

Fire events

Create thread pool

Create Reactor

Fire more events

Create a new thread to listen

```
def run(background=true, thread_name: 'srv')
  BasicSocket.do_not_reverse_lookup = true

  @events.fire :state, :booting

  @status = :run

  @thread_pool = ThreadPool.new(thread_name, @options) {

  if @queue_requests
    @reactor = Reactor.new(@io_selector_backend) { |c| re
    @reactor.run
  end

  @thread_pool.auto_reap! if @options[:reaping_time]
  @thread_pool.auto_trim! if @options[:auto_trim_time]

  @check, @notify = Puma::Util.pipe unless @notify

  @events.fire :state, :running

  if background
    @thread = Thread.new do
      Puma.set_thread_name thread_name
      handle_servers
    end
    return @thread
  else
    handle_servers
  end
end
```

The Server

If not shutting down…
IO.select()

Wait if full or busy

Accept

Add Client to pool

```
def handle_servers
  begin
    #...
    while @status == :run || (drain && shutting_down?)
      begin
        ios = IO.select sockets, nil, nil, (shutting_down? ? 0 :
        break unless ios
        ios.first.each do |sock|
          if sock == check
            break if handle_check
          else
            pool.wait_until_not_full
            pool.wait_for_less_busy_worker(@options[:wait_for_les

            io = begin
              sock.accept_nonblock
            rescue IO::WaitReadable
              next
            end
            drain += 1 if shutting_down?
            pool << Client.new(io, @binder.env(sock)).tap { |c|
              c.listener = sock
              c.send(addr_send_name, addr_value) if addr_value
            }
          end
        end
```

IO.select takes an array of IO objects, calls the select(2) sys call, and waits until one of those IO objects is available for reading, then returns that IO object.

IO#accept_nonblock take the first connection request from the queue, creates a new socket, and returns that Socket object, but doesn’t block (duh!)

Threadpool

#new

```
@mutex.synchronize do
  @min.times do
    spawn_thread
    @not_full.wait(@mutex)
  end
end
```

Shift work

```
  work = todo.shift
end

if @clean_thread_locals
  ThreadPool.clean_thread_locals
end

begin
  @out_of_band_pending = true if block.call(work, *extra)
```

Call block

The Config/Startup Classes

- CLI
- Launcher
- DSL
- Binder
- Configuration

Midjourney: “a puma cat in a bear suit, cute, kawaii, low detail cartoon, in the style of sanrio, anime”

The CLI is created by bin/puma. It creates a Configuration object based on user input. A Launcher object is created with that Configuration. Finally, the Launcher creates a Binder and a Runner.

CLI and Launcher

bin/puma

```
require 'puma/cli'

cli = Puma::CLI.new ARGV

cli.run
```

Create Launcher

```
  @launcher = Puma::Launcher.new(@conf,
  :log_writer => @log_writer, :events =>
  @events, :argv => argv)
end

attr_reader :launcher

# Parse the options, load the rackup, start the
server and wait
# for it to finish.
#
def run
  @launcher.run
end
```

Call #run

Launcher#new

Create Binder

```
@binder         = Binder.new(@log_writer, conf)
@binder.create_inherited_fds(ENV).each { |k| ENV.delete k }
@binder.create_activated_fds(ENV).each { |k| ENV.delete k }
```

Create Runner

```
if clustered?
  @options[:logger] = @log_writer

  @runner = Cluster.new(self)
else
  @runner = Single.new(self)
end
Puma.stats_object = @runner
```

Set @status

```
@status = :run
```

Launcher#run

Stop or restart,
Close listeners

```ruby
# Run the server. This blocks until the server is stopped
def run
  previous_env = get_env

  @config.clamp

  @config.plugins.fire_starts self

  setup_signals
  set_process_title
  integrate_with_systemd

  # This blocks until the server is stopped
  @runner.run

  do_run_finished(previous_env)
end
```

The Request/ Response Classes

- Reactor
- Client
- Request

Midjourney: “puma cat on the moon as an astronaut, epic, cinematic, dramatic lighting, in the style of Hieronymus Bosch”

ThreadPool worker threads call Server#process_client with a Client object. This adds the Client to the Reactor. When the Client is fully buffered, Request creates the Rack env, calls the app, and writes the response.

Server#process_client

Add to reactor, exit

Wait for client ready

Handle request

```ruby
# Return true if one or more requests were processed.
def process_client(client, buffer)
  # ...
  begin
    if @queue_requests &&
      !client.eagerly_finish

      client.set_timeout(@first_data_timeout)
      if @reactor.add client
        close_socket = false
        return false
      end
    end

    with_force_shutdown(client) do
      client.finish(@first_data_timeout)
    end

    while true
      @requests_count += 1
      case handle_request(client, buffer, requests + 1)
```

Client#try_to_finish

Read from socket
Manage errors

Add to our buffer string

Send to the parser
If parser is done, we're done

```ruby
def try_to_finish
  return read_body if in_data_phase

  begin
    data = @io.read_nonblock(CHUNK_SIZE)
  rescue IO::WaitReadable
    return false
  rescue EOFError
    # Swallow error, don't log
  rescue SystemCallError, IOError
    raise ConnectionError, "Connection error detected during read"
  end

  # No data means a closed socket
  unless data
    @buffer = nil
    set_ready
    raise EOFError
  end

  if @buffer
    @buffer << data
  else
    @buffer = data
  end

  return false unless try_to_parse_proxy_protocol

  @parsed_bytes = @parser.execute(@env, @buffer, @parsed_bytes)

  if @parser.finished?
    return setup_body
  elsif @parsed_bytes >= MAX_HEADER
    raise HttpParserError,
      "HEADER is longer than allowed, aborting client early."
  end

  false
end
```

Request#handle_request

Call app!

```
status, headers, app_body = @thread_pool.with_force_shutdown do
  @app.call(env)
end
```

Cleanup

```
  prepare_response(status, headers, res_body, io_buffer, requests, client)
ensure
  io_buffer.reset
  uncork_socket client.io
  app_body.close if app_body.respond_to? :close
  client.tempfile&.unlink
  after_reply = env[RACK_AFTER_REPLY] || []
  begin
    after_reply.each { |o| o.call }
  rescue StandardError => e
    @log_writer.debug_error e
  end unless after_reply.empty?
end
```

Midjourney: “an infinite repeating pattern of puma cats, cartoon, low detail, kawaii, cute, anime, sanrio style”

CONTRIBUTING.md

Midjourney: “an anthropomorphic puma cat, "we can do it!" poster”

Book time with me, open Github Discussions or Issues freely.

Midjourney: “16-bit puma cat video game character sprite”

1. git clone
2. brew install ragel
3. bundle exec rake compile
4. bundle exec rake

```
bundle exec bin/puma test/rackup/hello.ru

gem "puma", path: "/path/to/local/puma"
```

Don’t “claim” issues.

Post draft PRs instead.

It’s not just about pushing new code.

Midjourney: “a puma cat cartoon character, chibi, kawaii, cute, sanrio style, low detail, anime, hacker, computers, programmer”

contrib-wanted label

Midjourney: “a puma cat cartoon character coding at a laptop, kawaii, cute, sanrio style, low detail, hacker, anime --v 4”

needs-repro label

docs label

Midjourney: “a puma cartoon character, dressed as a librarian, cute, kawaii, low detail, in the style of sanrio, books

Code review isn’t just for “maintainers”

You can fix bugs or backport bug fixes and features.

Midjourney: “a puma cat cartoon character, chibi, kawaii, cute, sanrio style, low detail, anime”

Finally: features.

1. Contrib-wanted
2. Needs-repro
3. Docs
4. Code review
5. Bugs
6. Features

Midjourney: “jumping puma cat, cute, kawaii, anime style, simple cartoon, anthropomorphic, low detail, happy, mouth open, sanrio style

- workingwithruby.com
- The Rack SPEC
- Ruby docs for Socket, IO.pipe

Midjourney: “a studious puma cat, cute, kawaii, anime style, simple cartoon, low detail, wearing glasses, sanrio style”

Thank you for your time. See you on Github!

Midjourney: “anime puma cat, waving goodbye, cartoon style, cute, kawaii, bright solid color background”

Chris Seaton