resty-threadpool

Re-inventing Apache httpd in nginx

Julien Desgats - Cloudflare

About me

- Working at Cloudflare London
- Edge performance team
- Worked on the nginx-based HTTP(S) proxies before that
- Code in Lua for more than 10 years

CLOUDFLARE

Cloudflare HTTP(S) infrastructure

CLOUDFLARE

Cloudflare HTTP(S) infrastructure

- Based on a customized build of nginx
- Heavily relies on lua-nginx-module
- Every requests runs Lua for:
  - Configuration loading
  - Security checks
    - IP reputation
    - Custom firewall rules
    - Web Application Firewall
  - Upstream selection
  - Response processing
  - Logging logic

CLOUDFLARE

Web Application Firewall (WAF)

- Scan requests for known attacks
  - SQL/shell/... injection attempts
  - XSS
  - Known vulnerabilities in server software
- Rule sets typically have 1,000s of rules to run
- Each has access to
  - Request path and arguments (e.g. /foo.html?arg=value%...)
  - Headers (User-Agent, Referer, ...)
  - Body
  - Metadata (client IP, geolocation, ...)
- Popular options: ModSecurity, Naxsi, lua-resty-waf

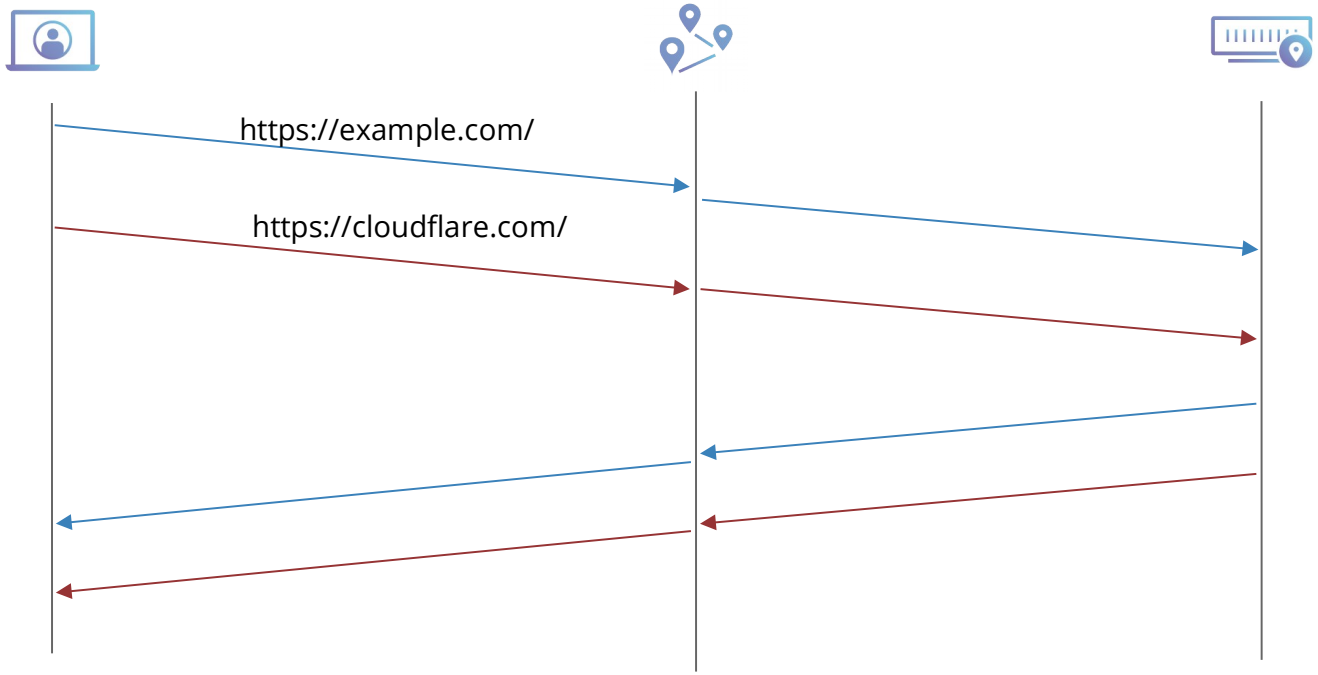
CLOUDFLARE

Web Application Firewall at Cloudflare

- Used by millions of domains
- Historically based onto ModSecurity
  - Now WAF rules are transpiled to Lua+PCRE
- Average processing time 2 to 4 ms
- Some requests are longer
- Sometimes cause slowness for other requests

CLOUDFLARE

Simple example: network requests

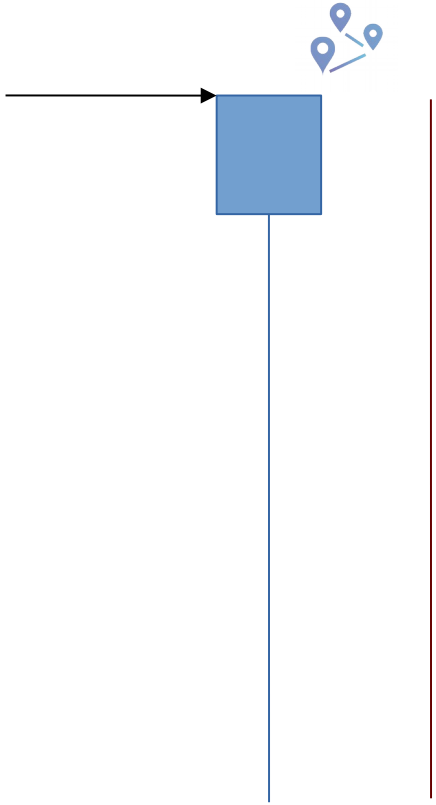


CLOUDFLARE

Thread based proxy

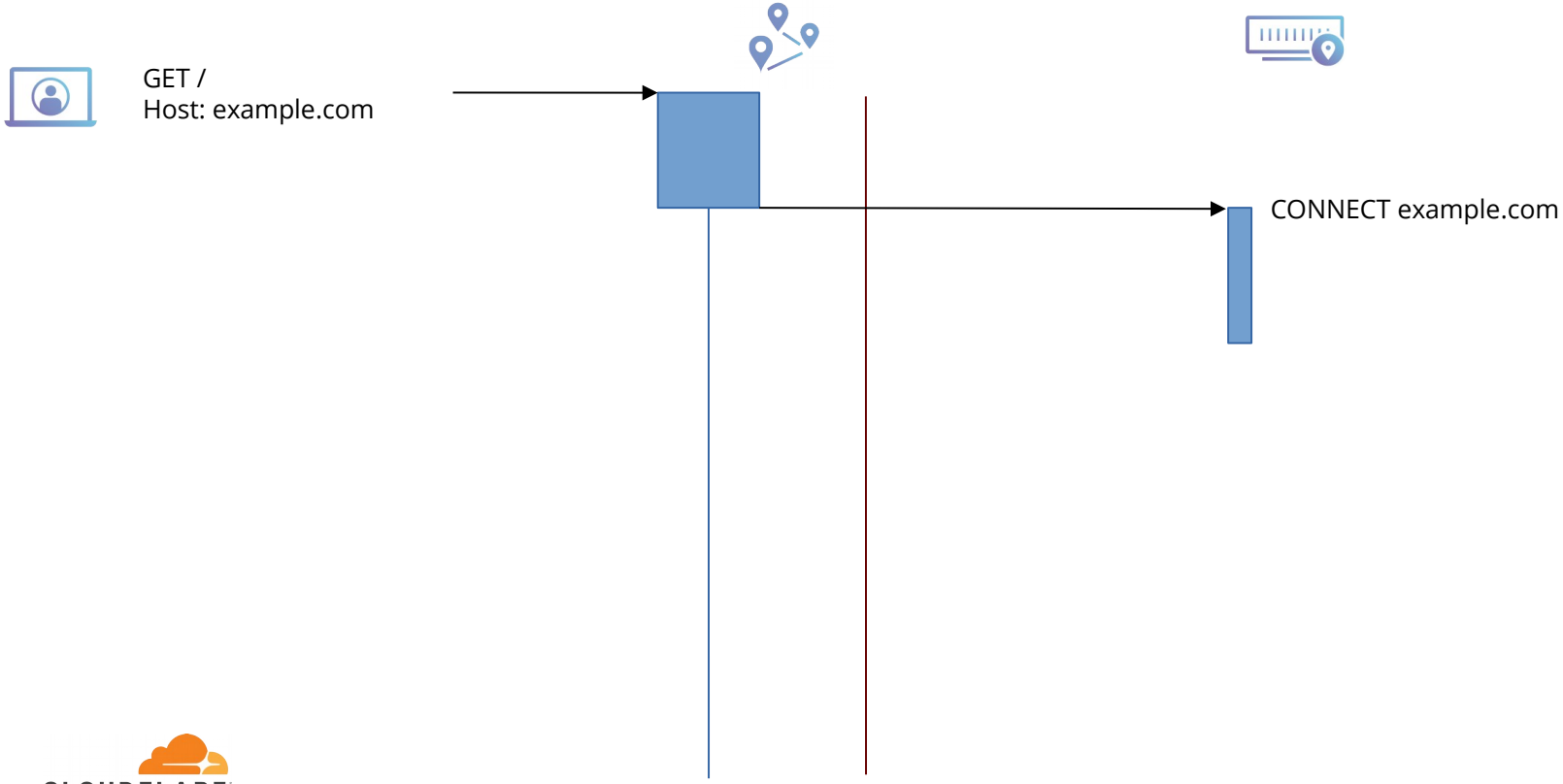
Thread based proxy

GET /
Host: example.com



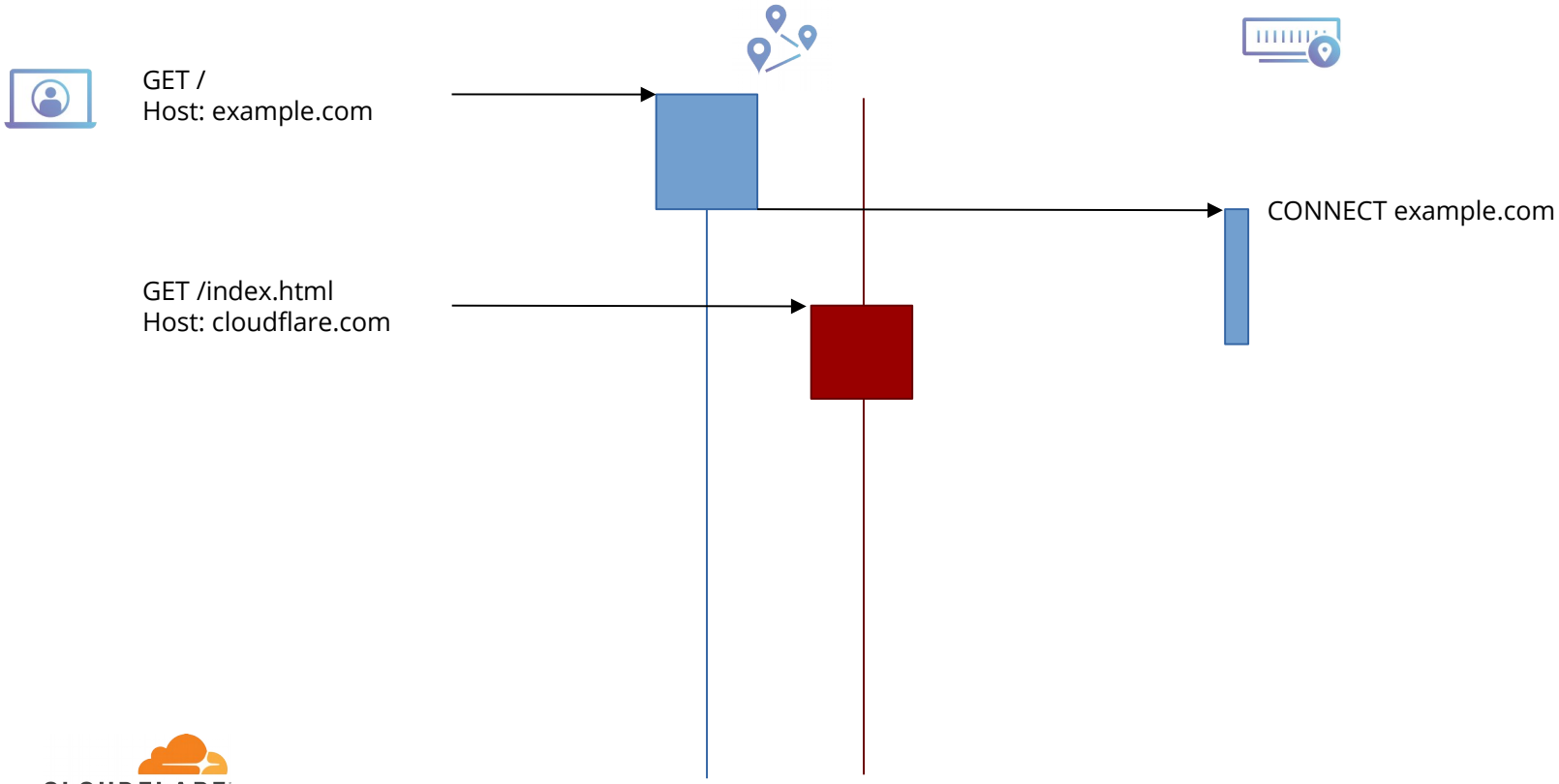
CLOUDFLARE

Thread based proxy



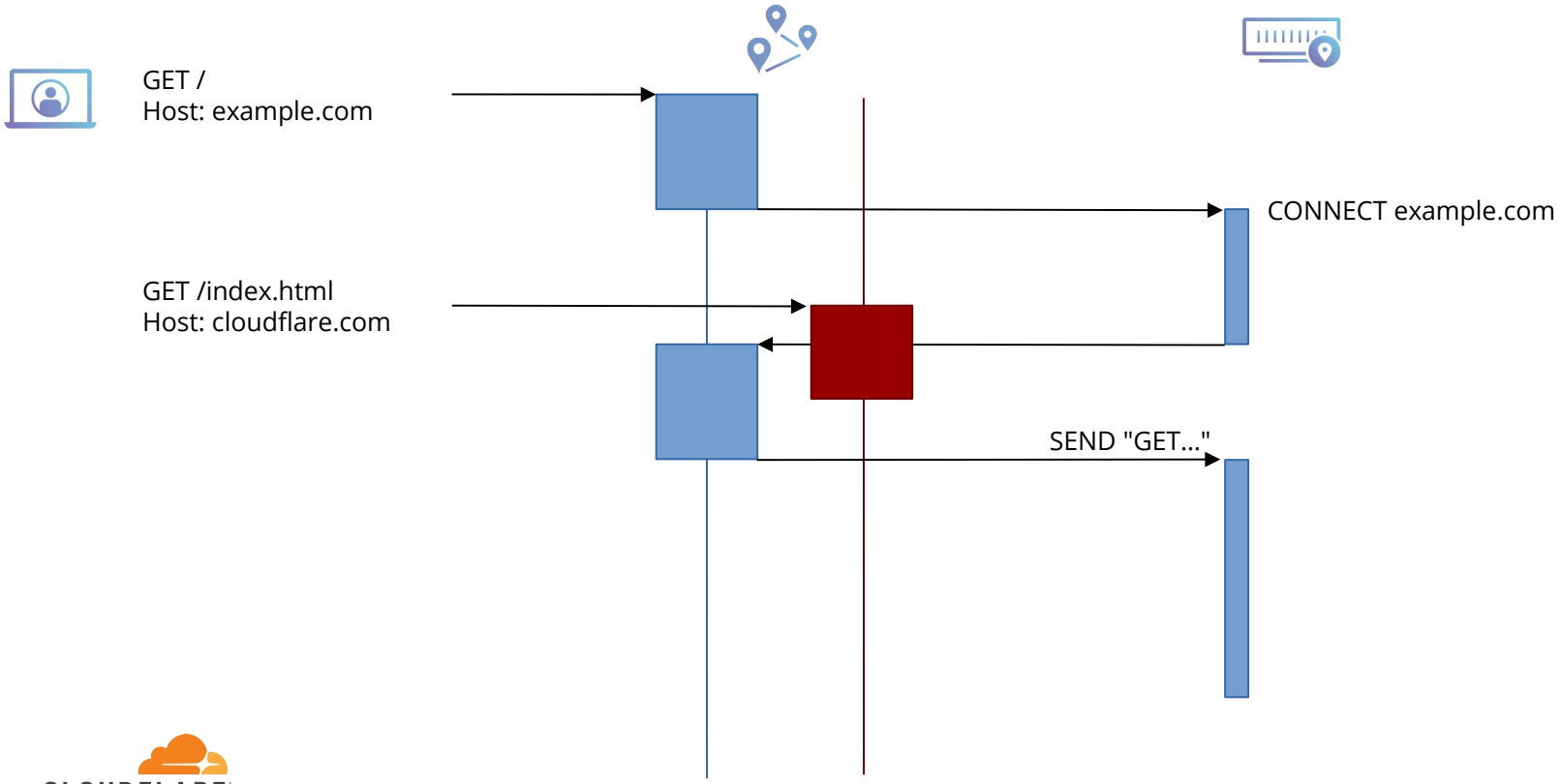
CLOUDFLARE

Thread based proxy



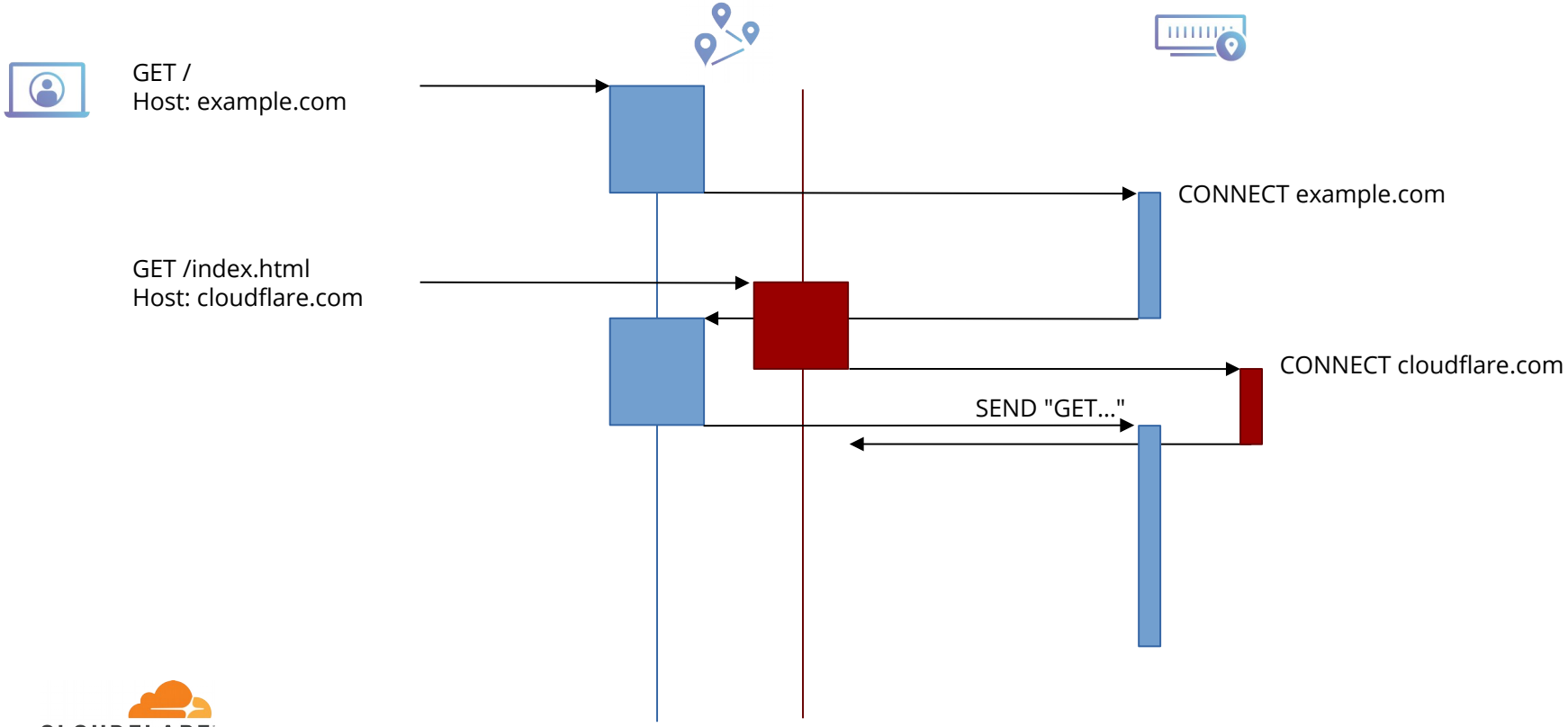
CLOUDFLARE

Thread based proxy



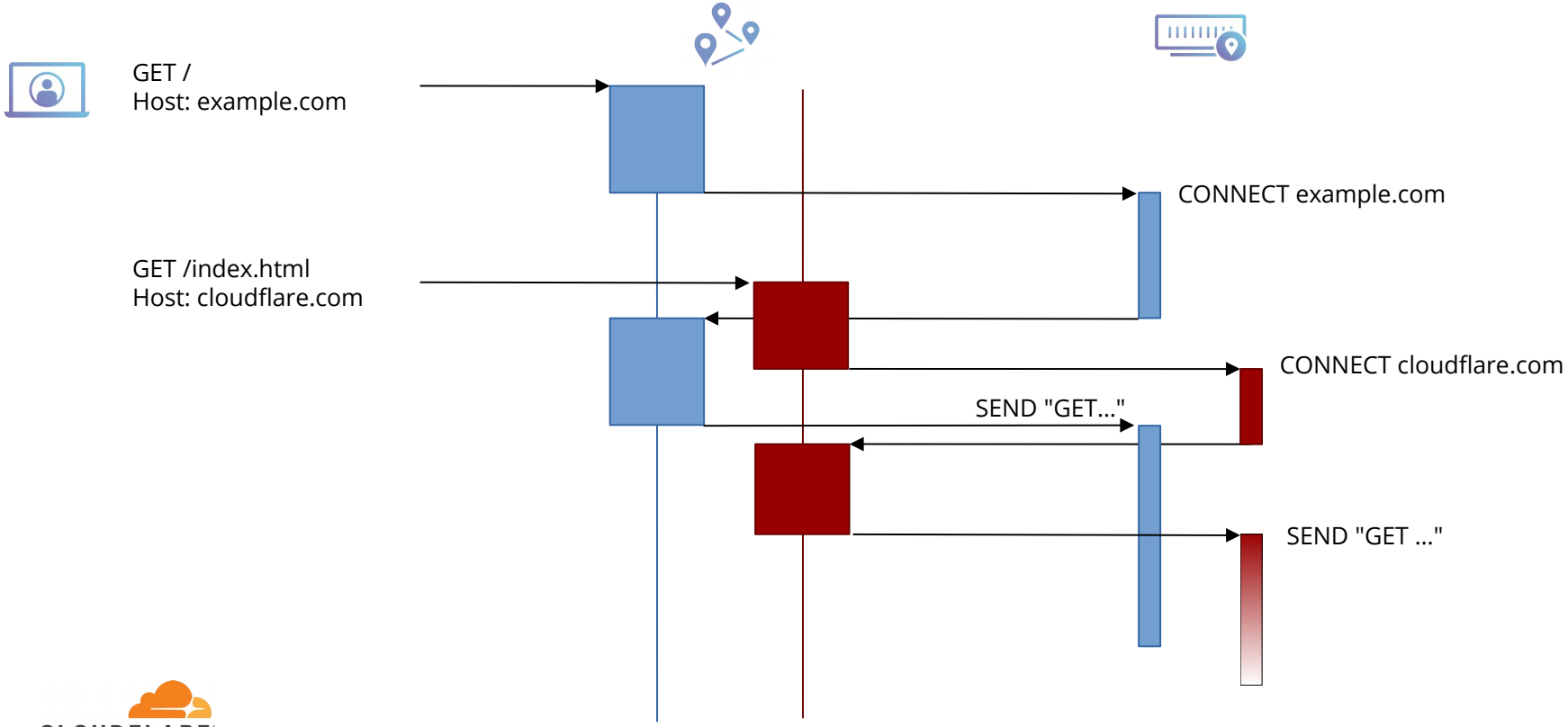
CLOUDFLARE

Thread based proxy



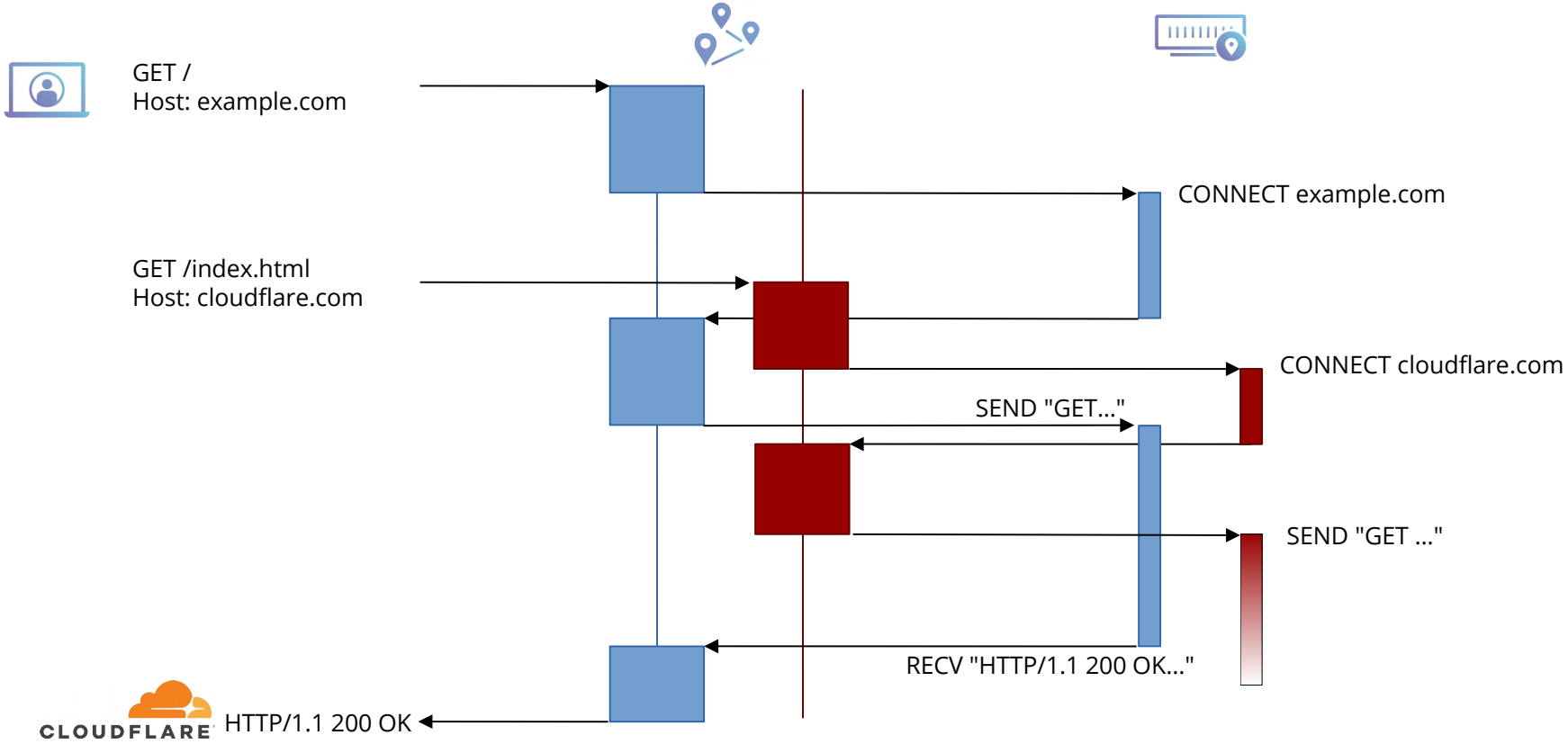
CLOUDFLARE

Thread based proxy



CLOUDFLARE

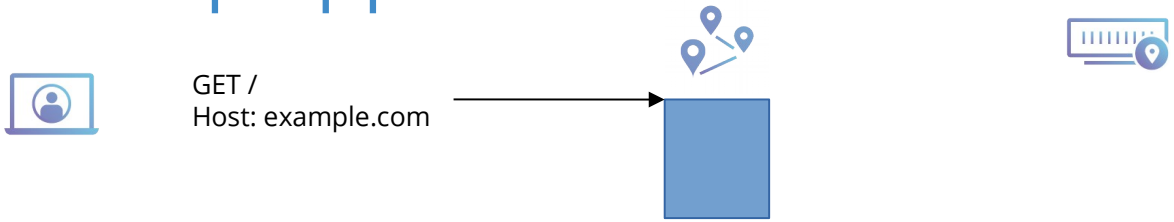
Thread based proxy



Event loop approach

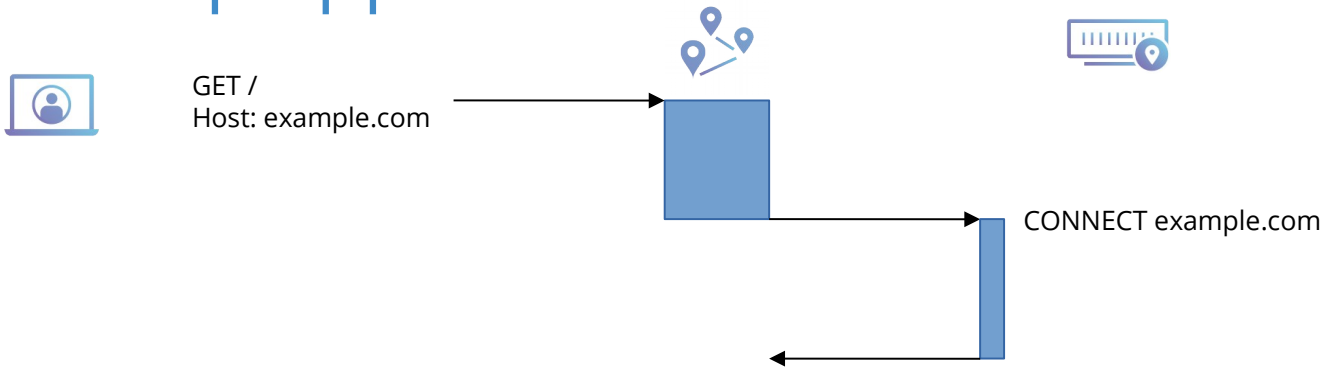
CLOUDFLARE

Event loop approach

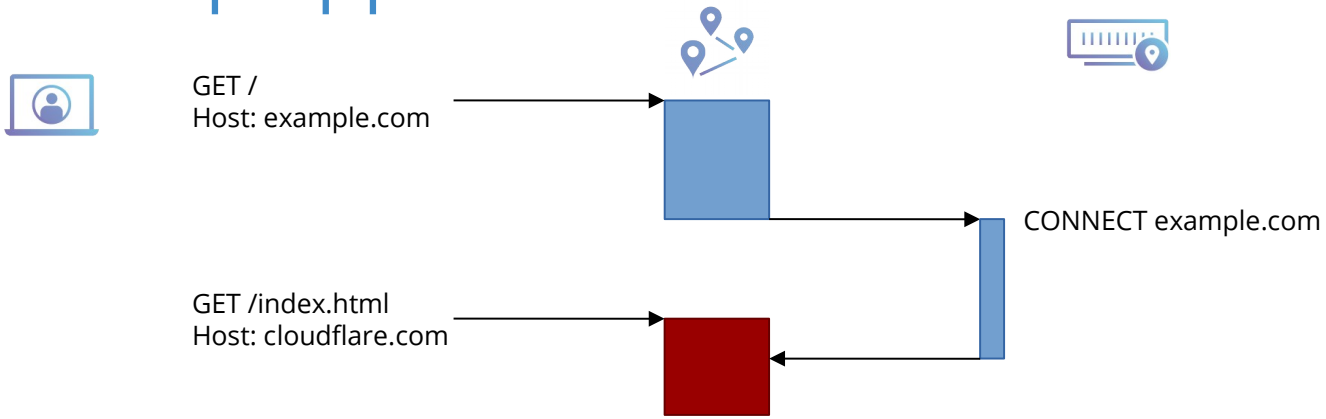


CLOUDFLARE

Event loop approach

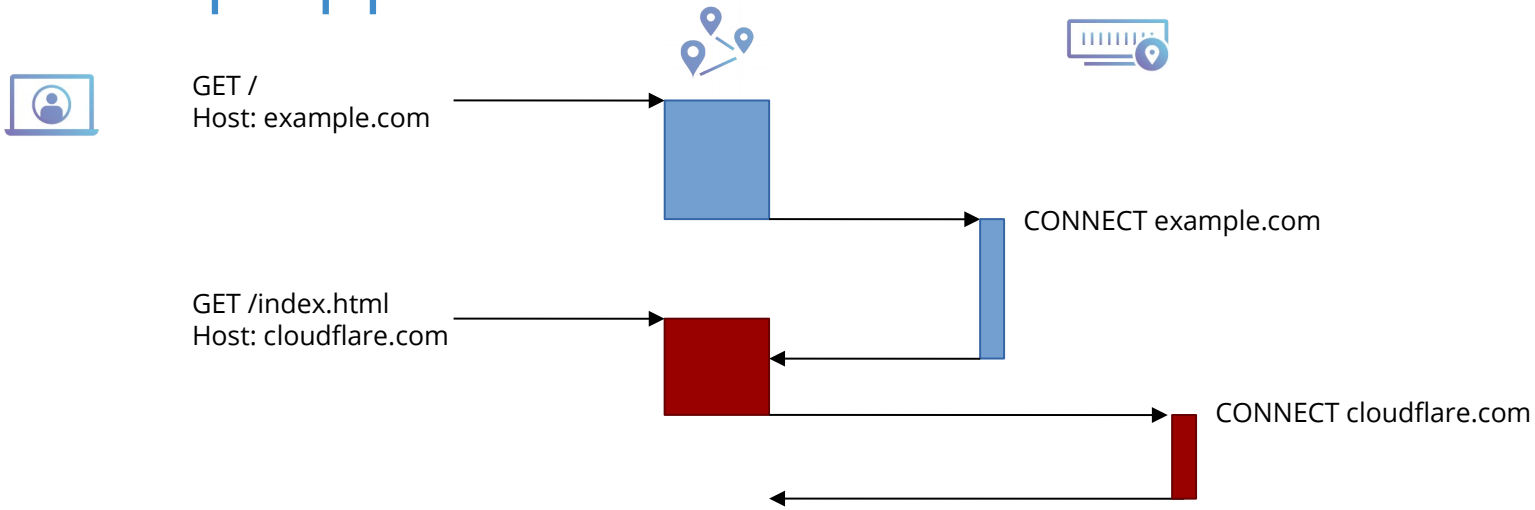


Event loop approach



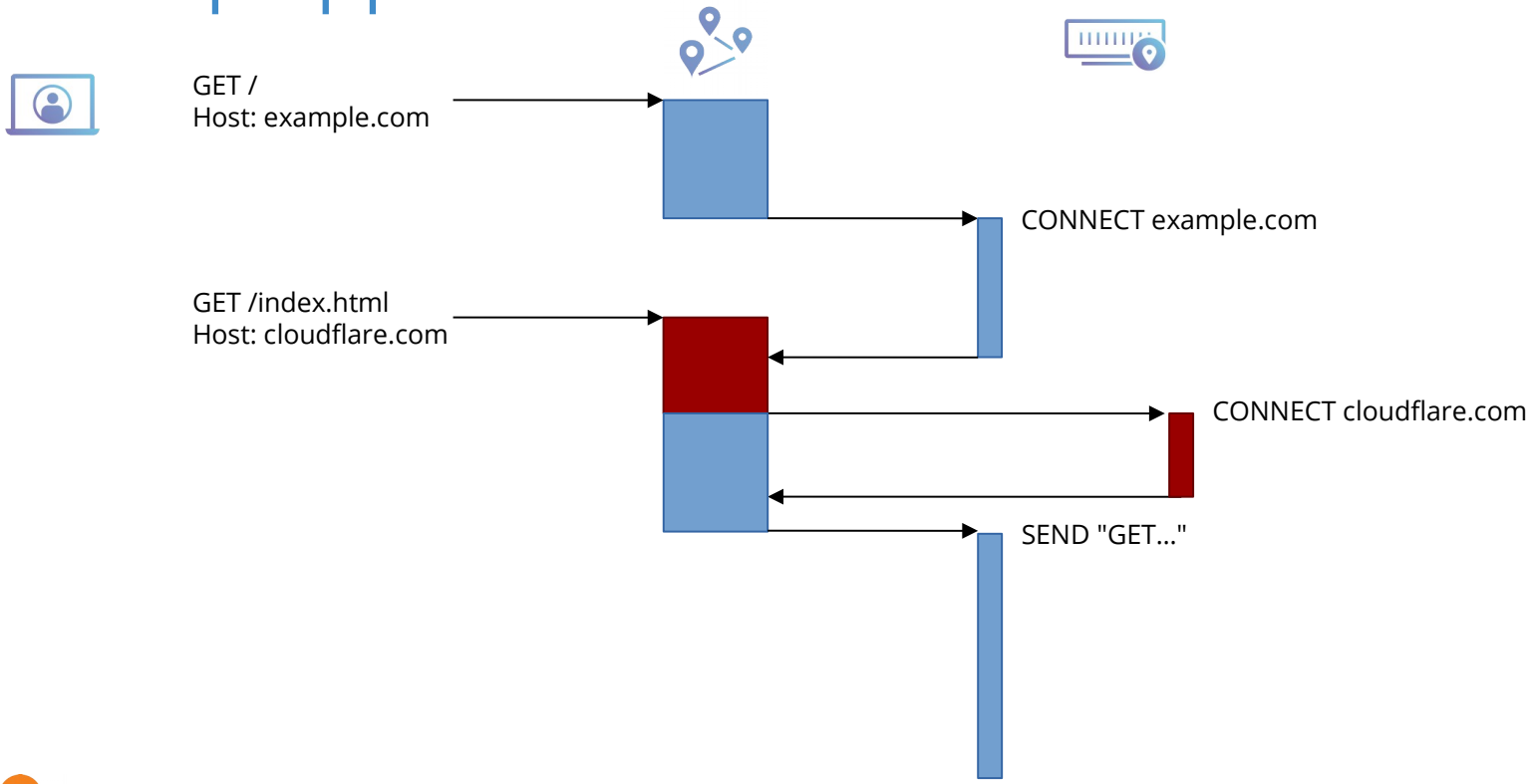
CLOUDFLARE

Event loop approach



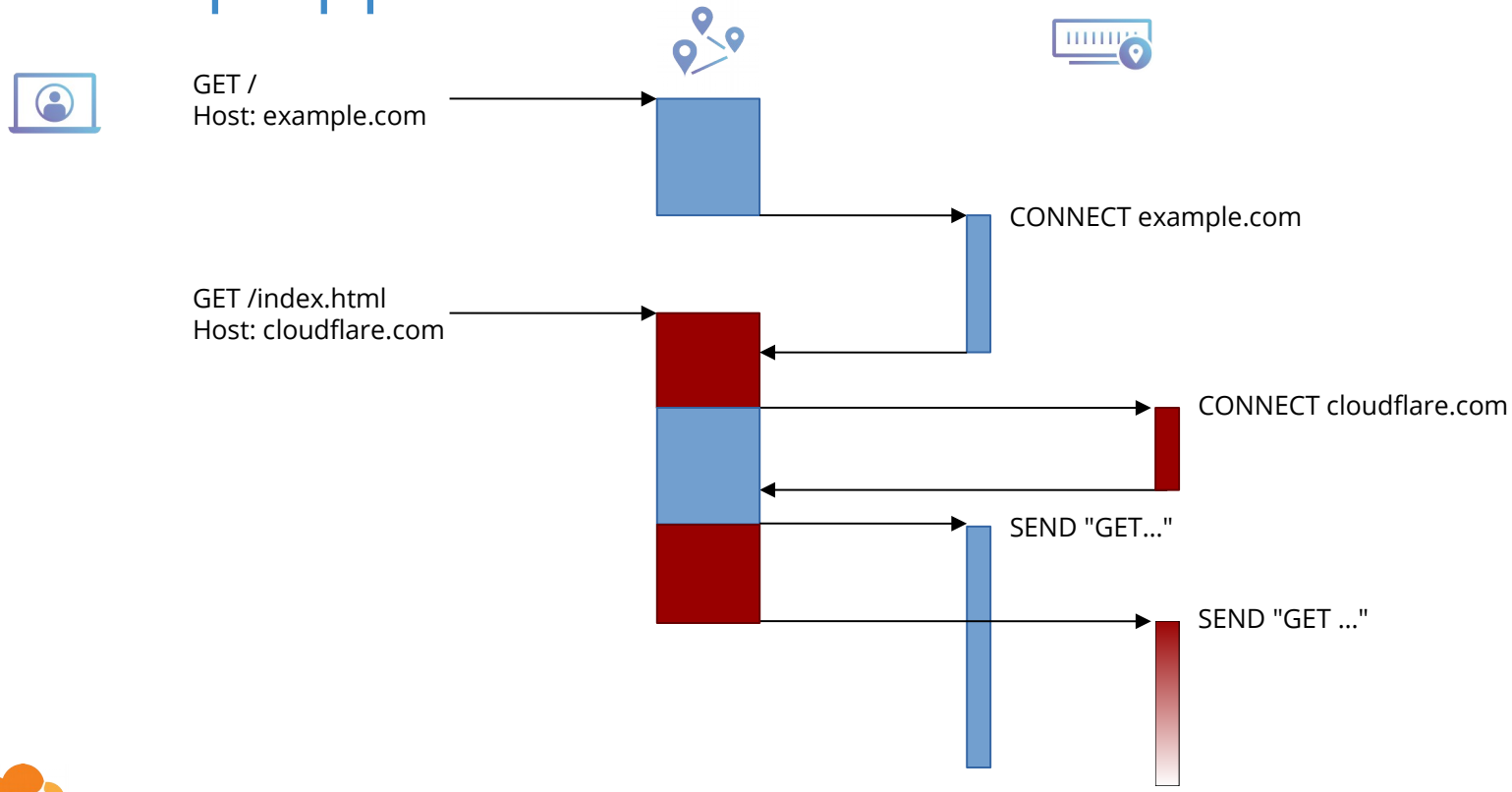
CLOUDFLARE

Event loop approach



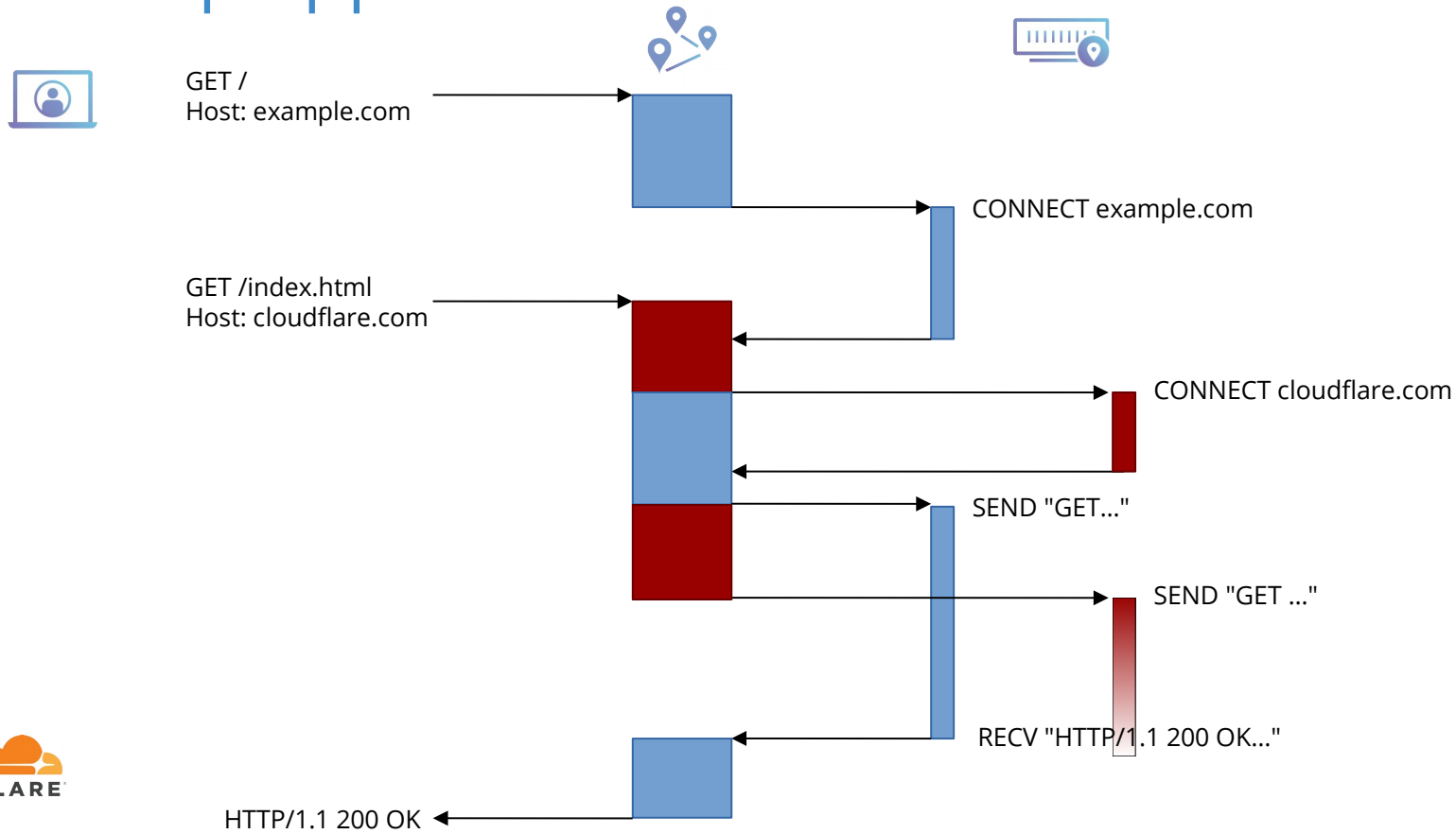
CLOUDFLARE

Event loop approach



CLOUDFLARE

Event loop approach



CLOUDFLARE

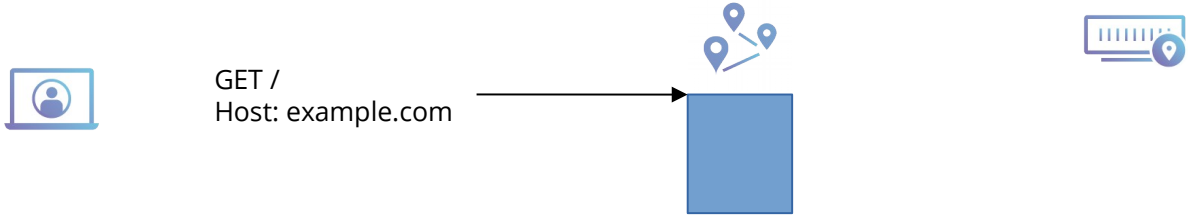
nginx architecture

- Event loop approach
- Multiple worker processes
  - Each has its own event loop
  - Each accept and process connections
  - Limited shared state
- Designed primarily to be used as a (caching) proxy or a static file server

CLOUDFLARE

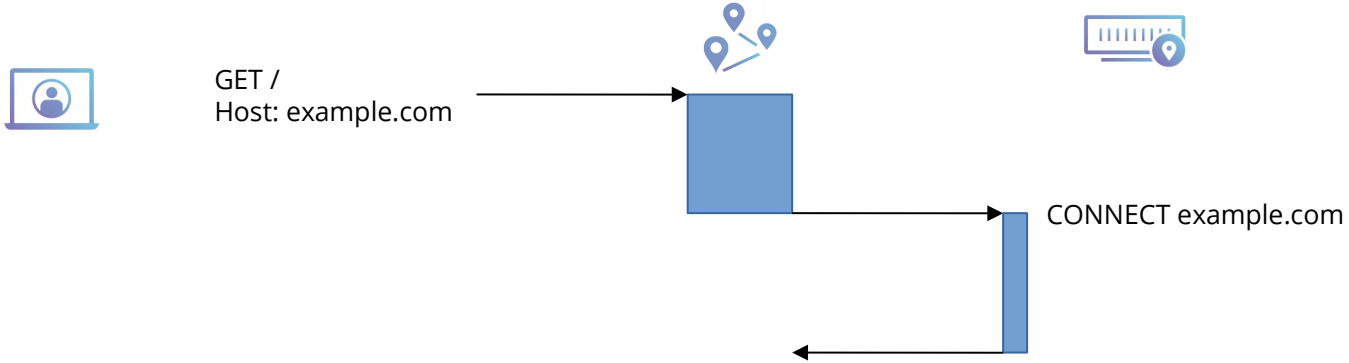
Issues with event loops

Issues with event loops

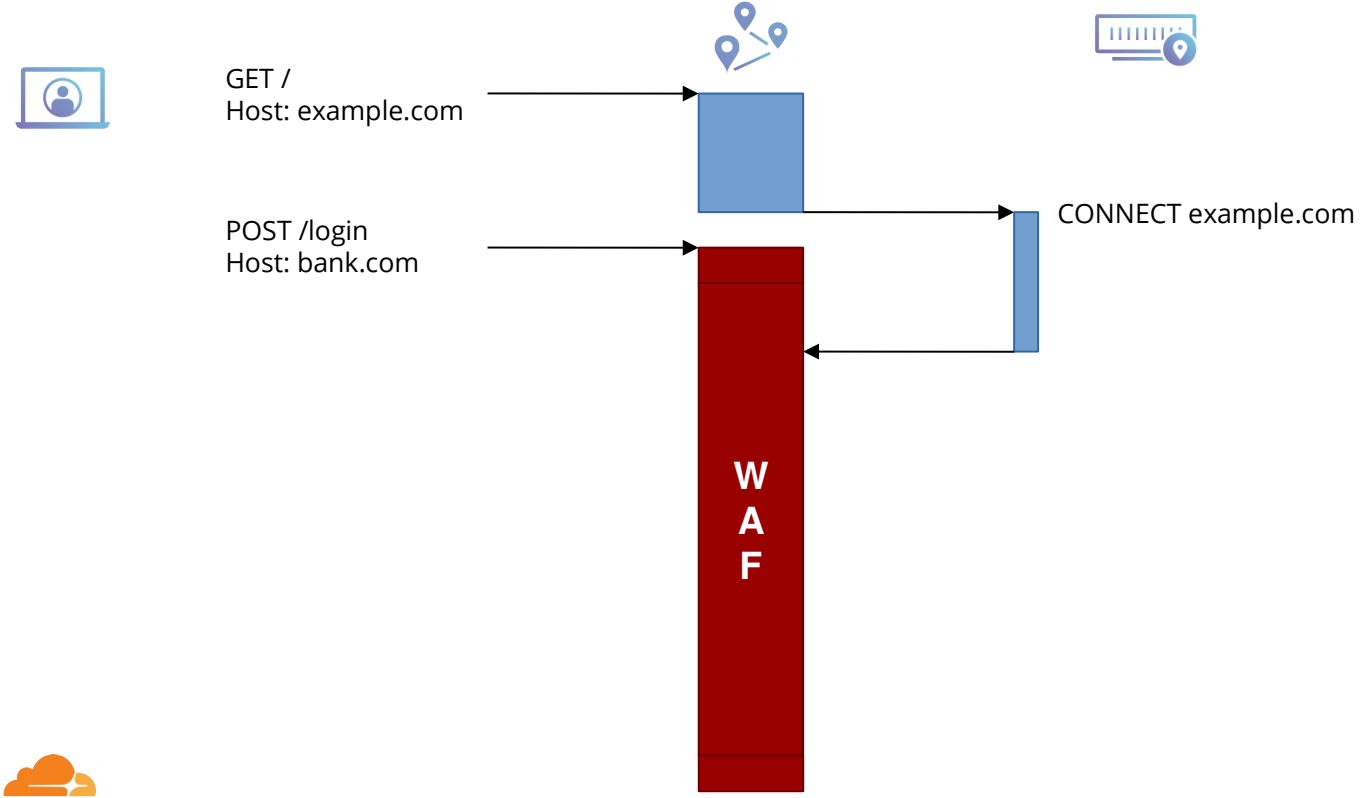


CLOUDFLARE

Issues with event loops

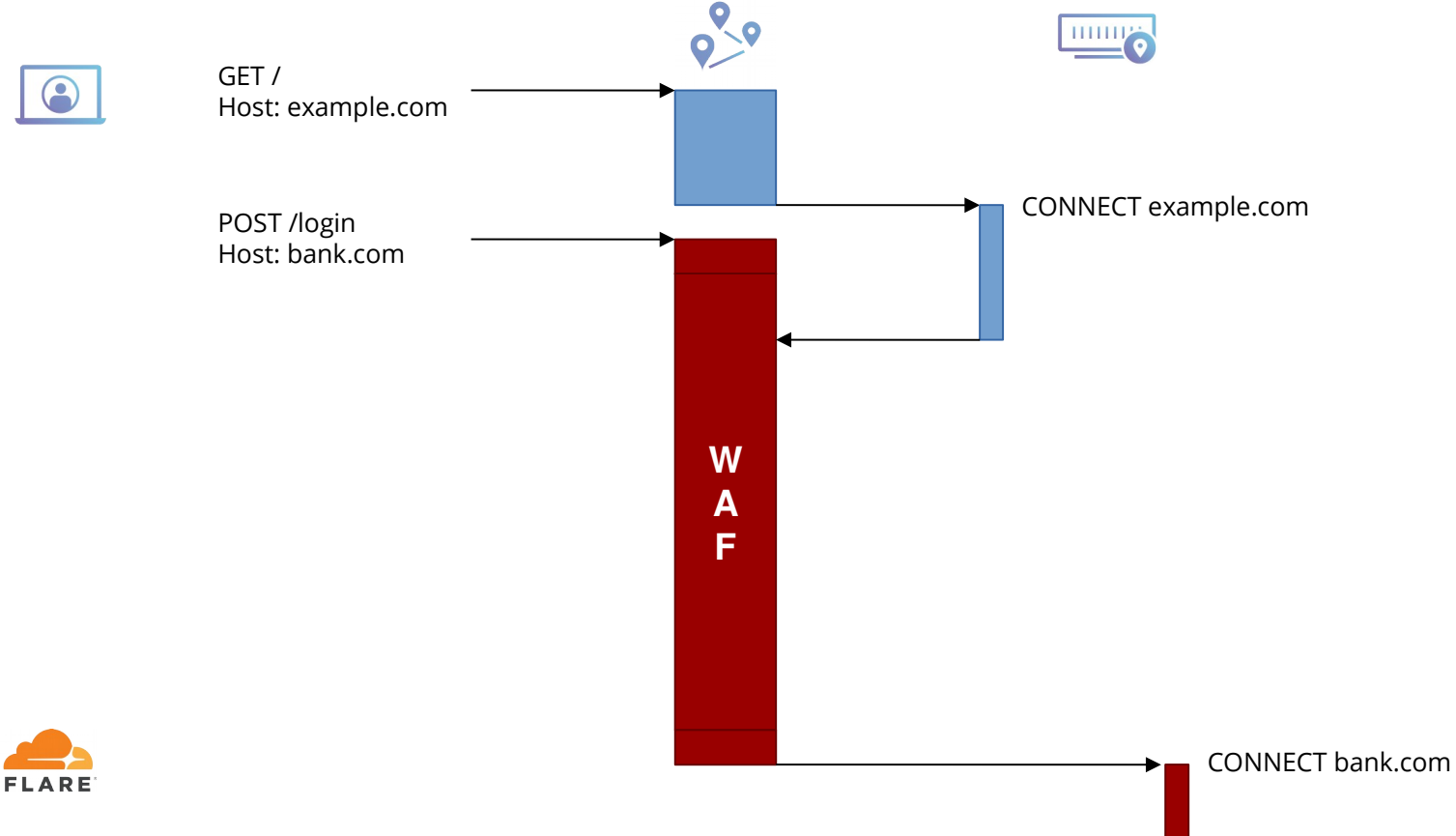


Issues with event loops



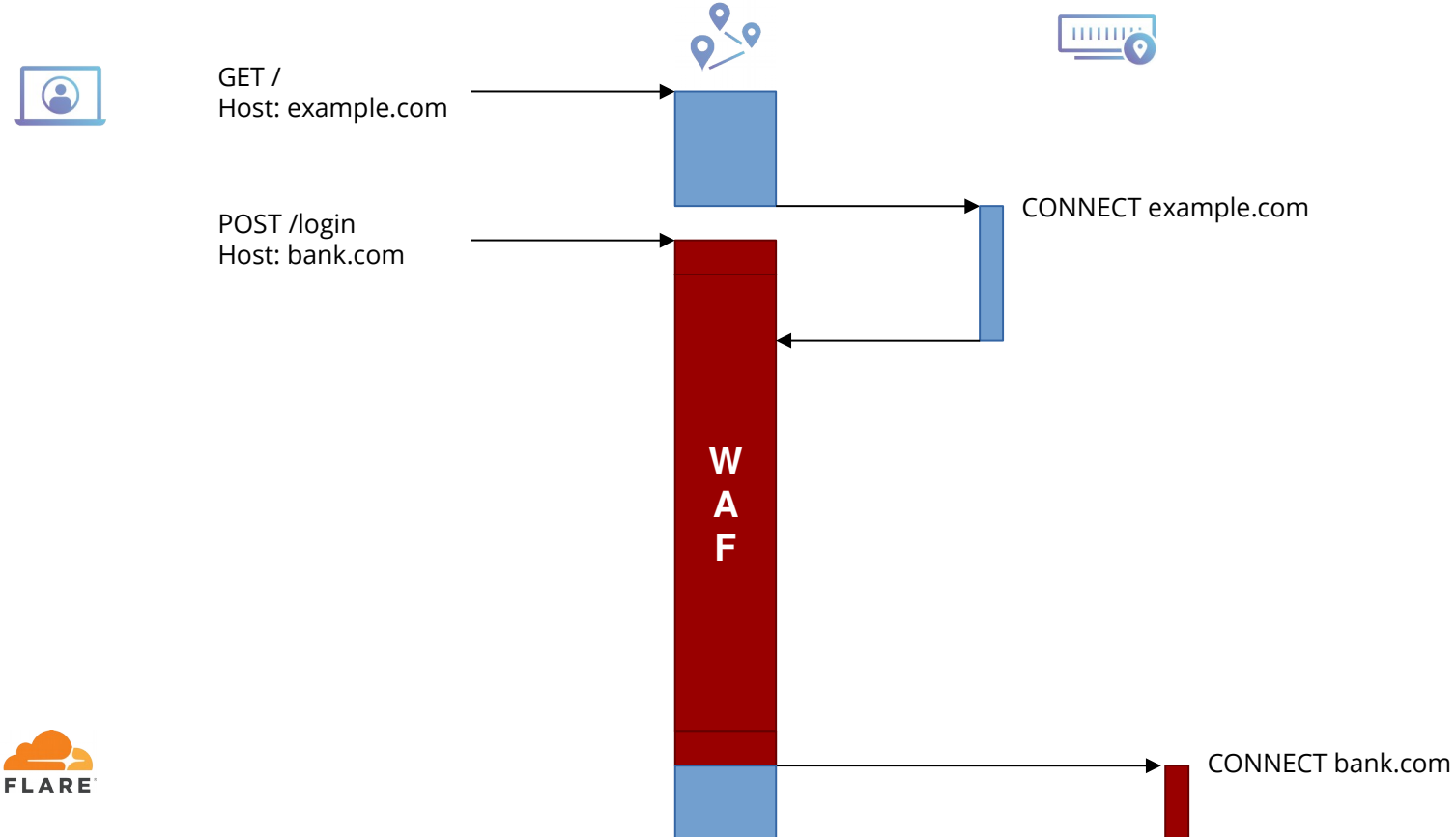
CLOUDFLARE

Issues with event loops



CLOUDFLARE

Issues with event loops



CLOUDFLARE

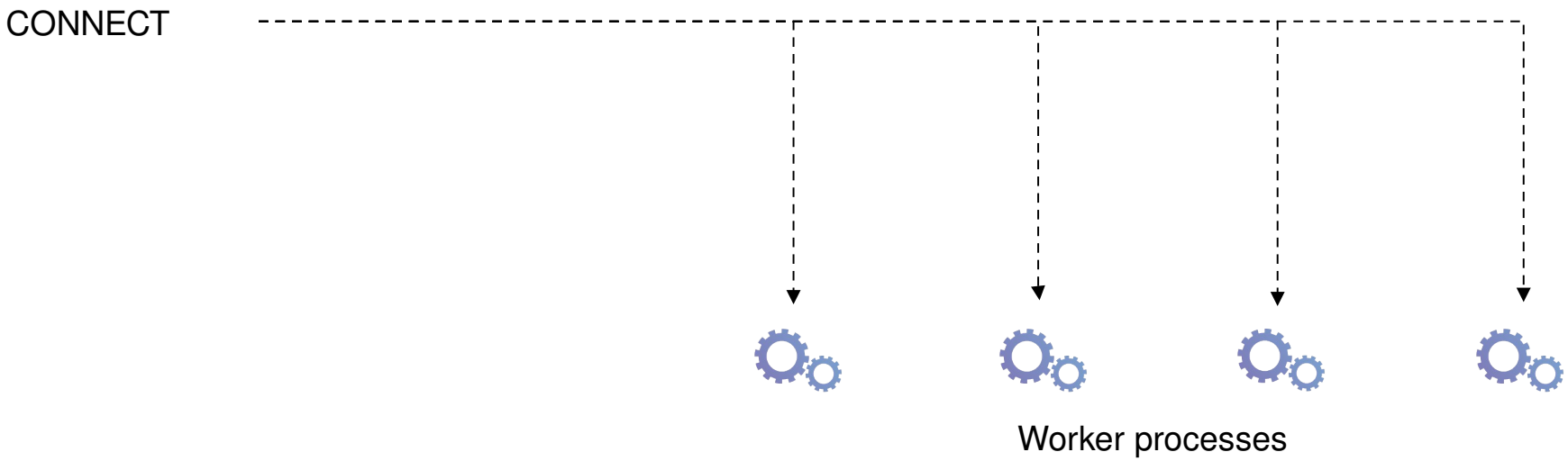
Issues with event loops

CONNECT

Worker processes

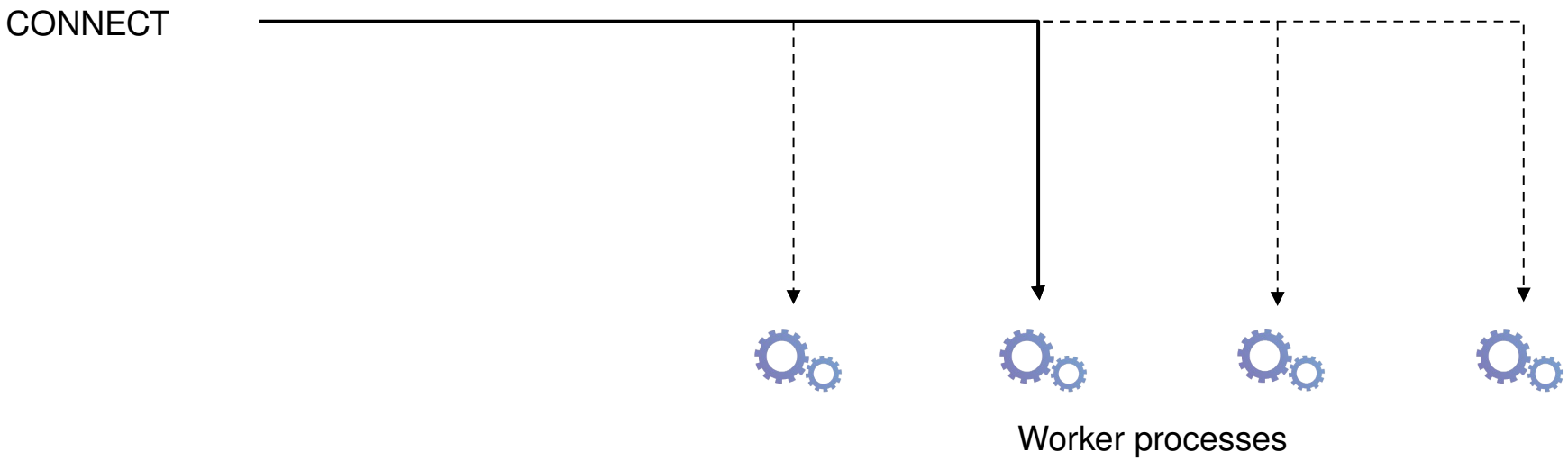
CLOUDFLARE

Issues with event loops



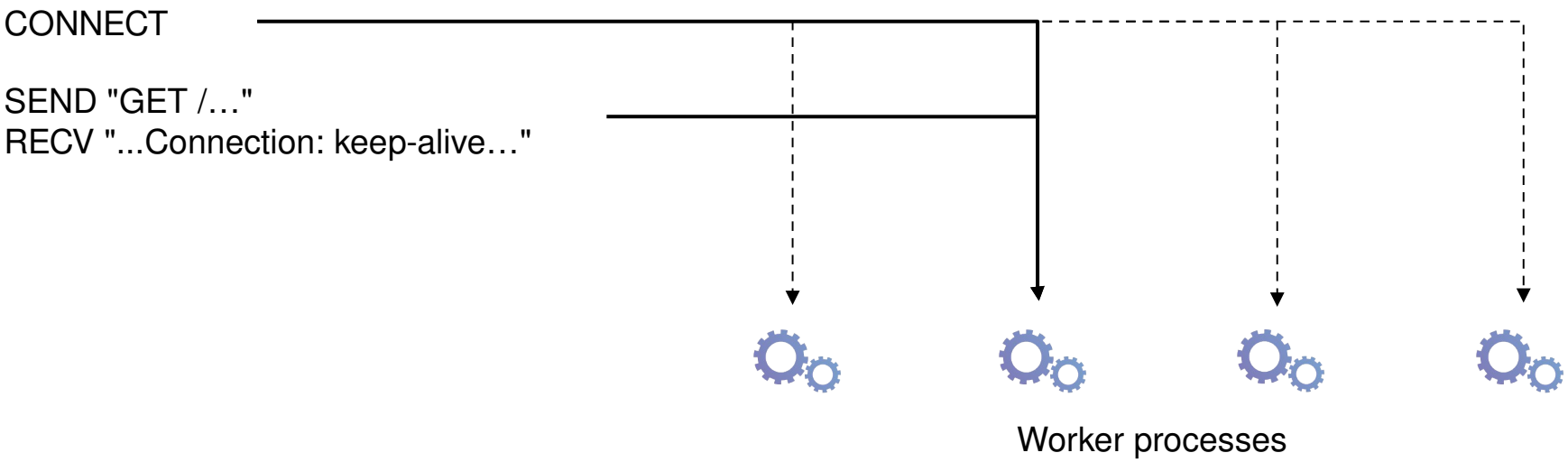
CLOUDFLARE

Issues with event loops



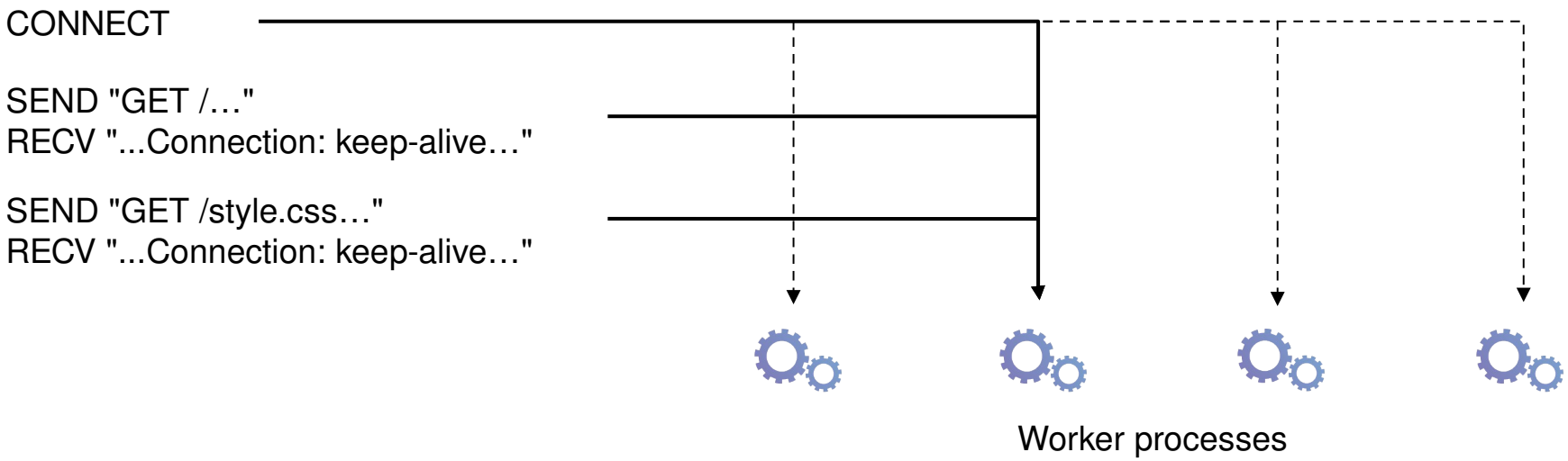
CLOUDFLARE

Issues with event loops



CLOUDFLARE

Issues with event loops



CLOUDFLARE

Here comes Lua

- lua-nginx-module allows to script various parts of the request pipeline
- Use cases:
  - Complex routing
  - Complex ACL
  - Dynamic load balancing
  - ...
  - NOT an application server
- Foundation of OpenResty, a batteries-included nginx bundle
- The event-loop design maps well with Lua coroutines

CLOUDFLARE

Key takeaways

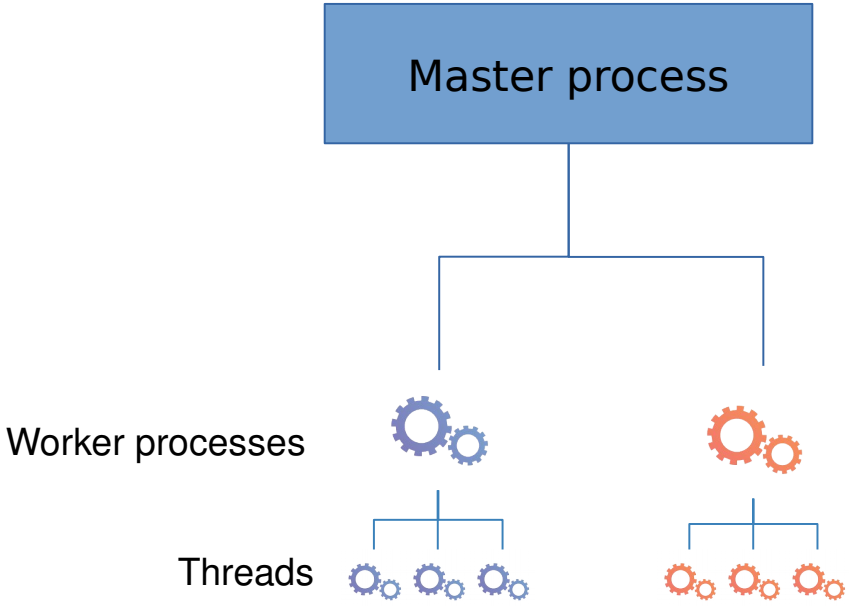
- Event loops have some advantages over threads:
  - Less scheduling overhead
  - Less memory overhead
- They have to be approached as a soft real-time environment
  - Running CPU-heavy work will slow down the entire system

CLOUDFLARE

Blocking calls are already a known issue

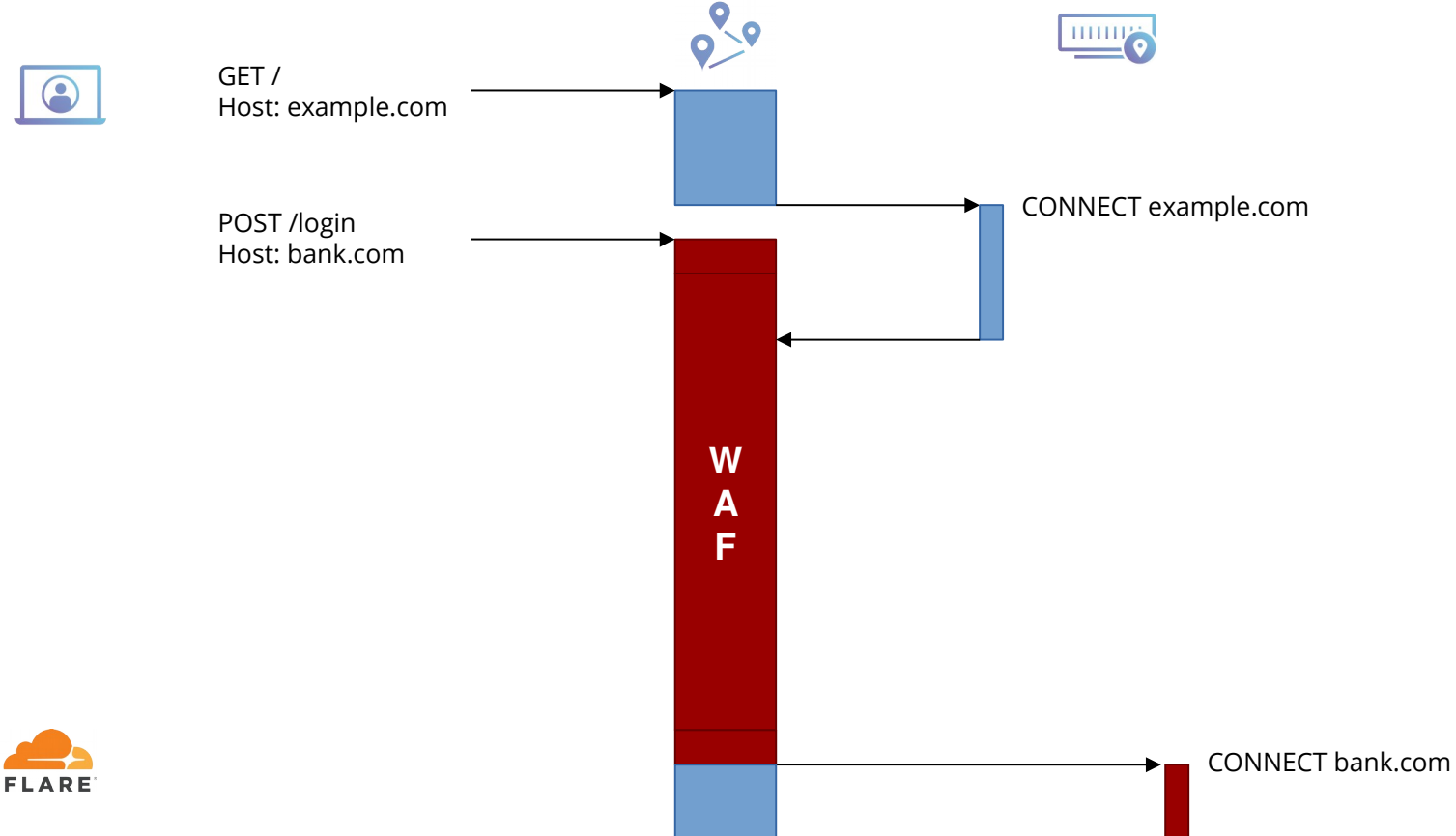
- nginx support thread pools since v1.7.11 (24 March 2015)
  - Used for blocking I/O (read/write/sendfile)
  - Benchmarks report up to 9x gain
- Each worker has its own thread pool
- Threads do not interact directly with the request: accessing the request state is unsafe

Idea: run our CPU-bound work into these threads!



CLOUDFLARE

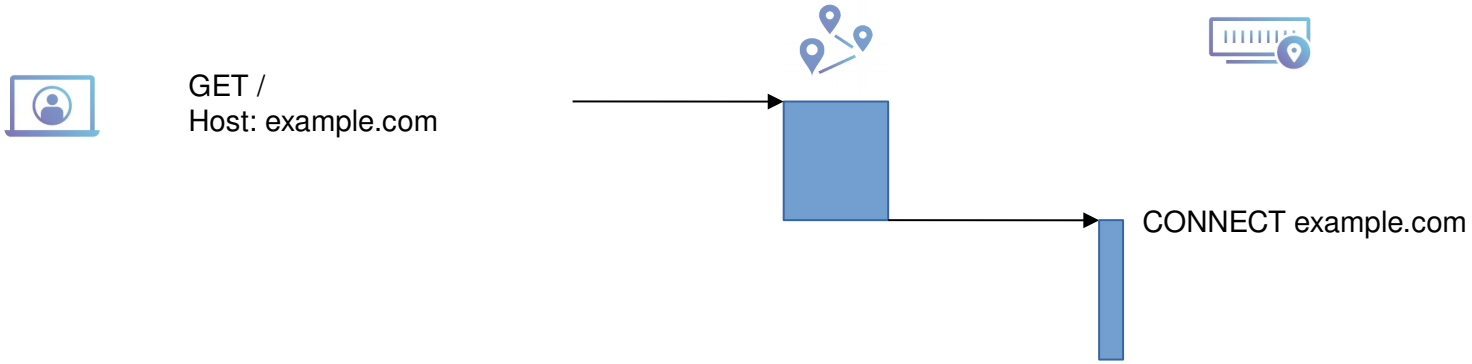
Issues with event loops



CLOUDFLARE

How does it look now?

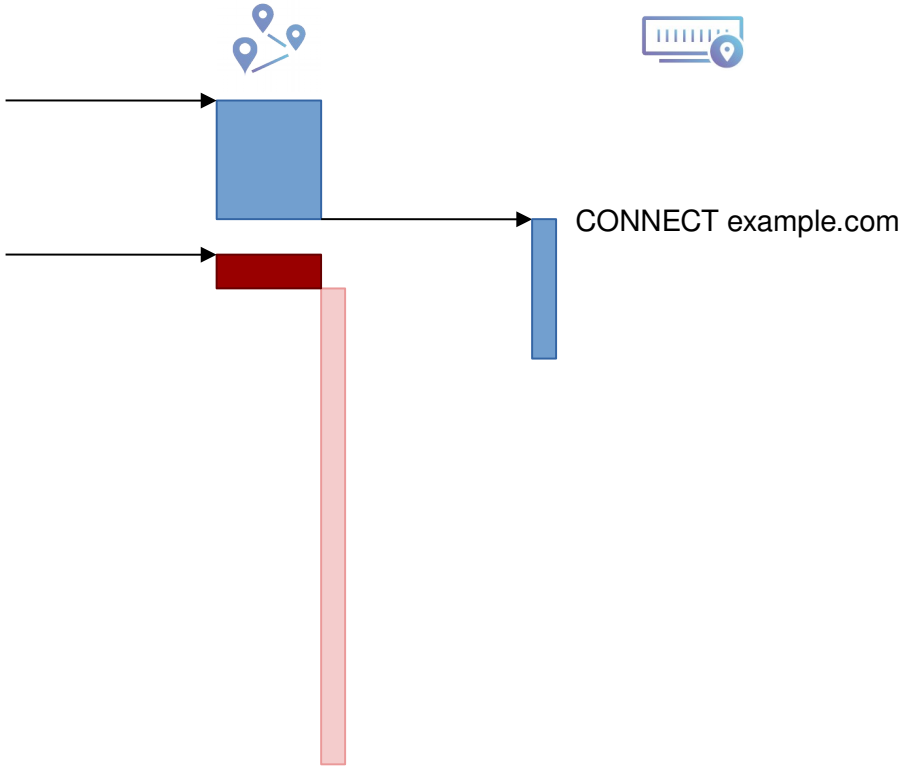
How does it look now?



How does it look now?

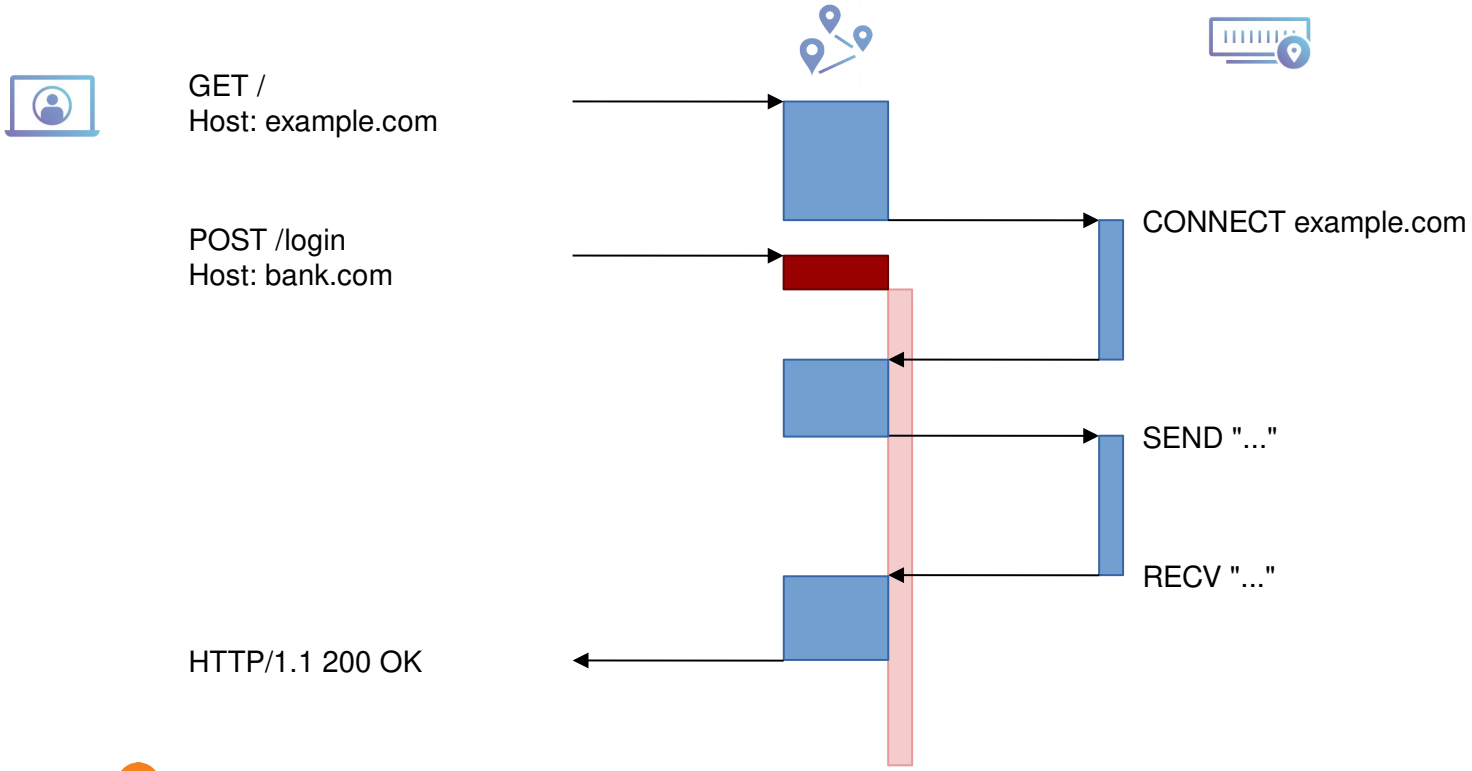
GET /
Host: example.com

POST /login
Host: bank.com



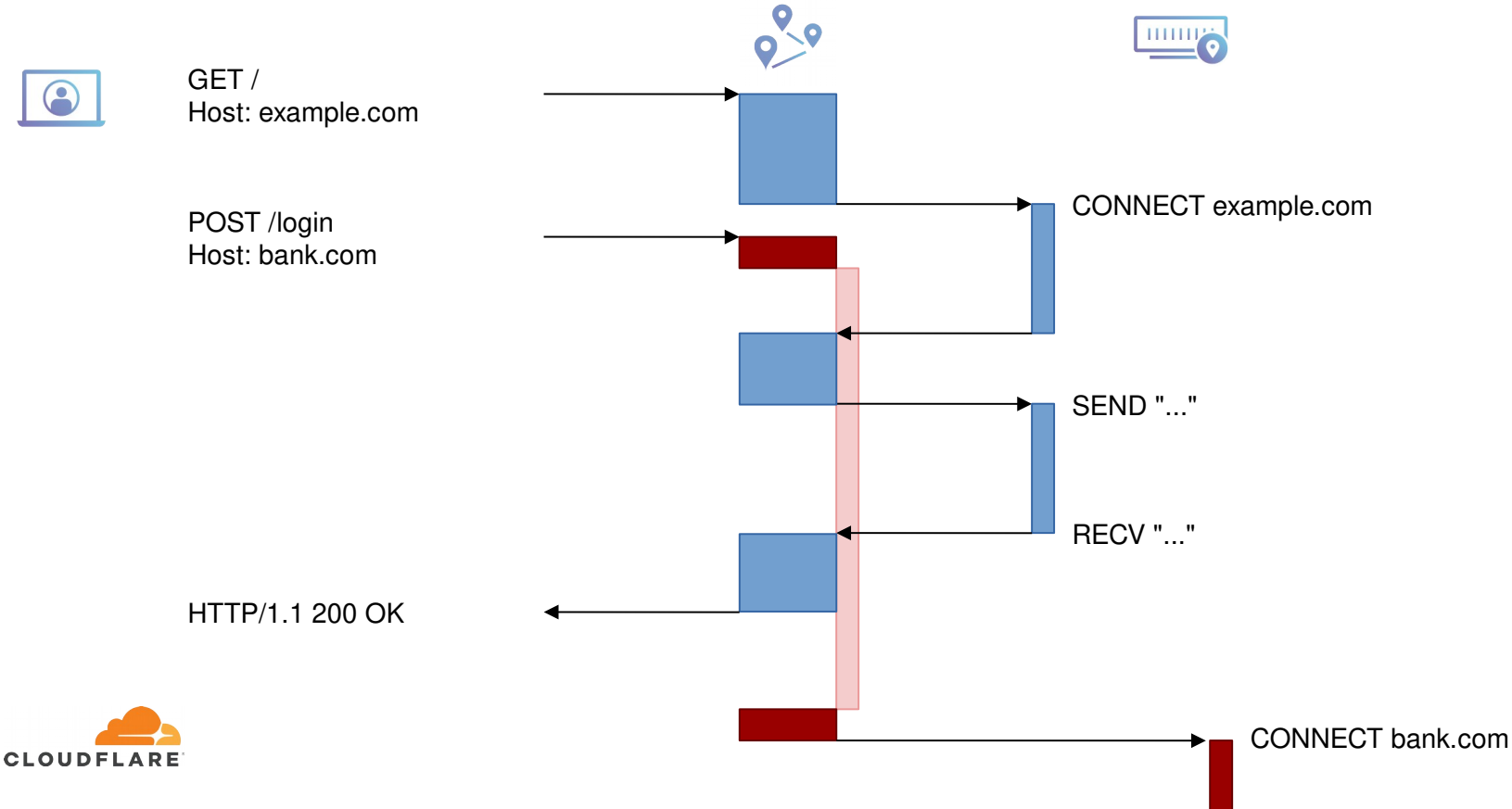
CLOUDFLARE

How does it look now?



CLOUDFLARE

How does it look now?



CLOUDFLARE

Let's run Lua into threads

```
thread_pool upper_pool threads=1;
...
http {
    thread_pool_init_by_lua upper_pool "
        ngx.log(ngx.INFO, 'optional code run once when the thread is spawned...')
    ";
    thread_pool_process_by_lua upper_pool "
        local arg = ...
        ngx.log(ngx.INFO, 'got a request to uppercase \"', arg, '\"')
        return arg:upper()
    ";
    ...
    server {
        listen       8080;
        location / {
            content_by_lua_block {
                local tp = require "resty.threadpool"
                ngx.say(assert(tp.upper_pool:push_task("foo")))
            }
        }
    }
}
```

CLOUDFLARE

Let's run Lua into threads

- Each thread will have its own persistent Lua VM
- Thread pools are focused on doing one task
- Expose the same(ish) API as the regular Lua callbacks
  - Functions are reused when possible
  - Otherwise, emulate their behaviour (ngx.re.*)
- Some APIs are not exposed at all:
  - Asynchronous calls (ngx.socket.*, ngx.sleep, ...)
  - APIs that access the request or response
- Values are serialized to move between event loop and threads

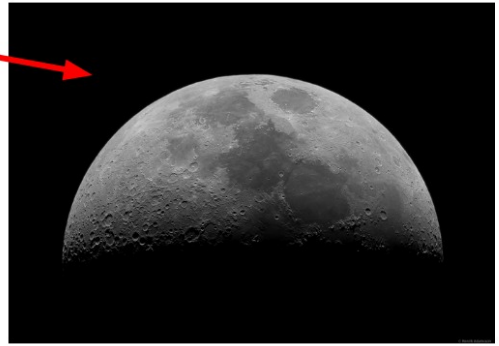
CLOUDFLARE

Benchmark: image resizing proxy

172.17.0.2:8080

Hello, Lua in Moscow

Actual size:
2835 × 1984



```
<html>
 <head>
   <title>Hello, Lua in Moscow</title>
   <link rel="stylesheet" href="style.css">
 </head>
 <body>
   ...
     <img src="/thumb/moon.jpg" />
   ...
     <img src="moon.jpg" width="50%" />
   ...
 </body>
</html>
```

CLOUDFLARE

Event loop implementation

```lua
local magick = require "magick"

-- load watermark
local watermark = assert(magick.load_image(
   ngx.config.prefix() .. "/root/lua_moscow_logo.jpg"))
watermark:resize(25, 25)

return function()
 local path = string.format("%s/root/%s",
     ngx.config.prefix(),
     ngx.var.uri:match("^/thumb/(.+)"))
 ngx.log(ngx.DEBUG, "resizing ", path)

 local img = assert(magick.load_image(path))
 img:resize(300, 200)
 img:composite(watermark, 0, 0)

 ngx.print(img:get_blob())
end
```

Name	Status	Protocol	Remote Address	Type	Size	Time	Waterfall
172.17.0.2	200 OK	http/1.1	172.17.0.2:8080	document	578 B 340 B	2 ms 1 ms	
style.css	200 OK	http/1.1	172.17.0.2:8080	stylesheet	286 B 51 B	2 ms 2 ms	
moon.jpg /thumb	200 OK	http/1.1	172.17.0.2:8080	jpeg	30.9 KB 30.7 KB	149 ms 148 ms	
moon.jpg	200 OK	http/1.1	172.17.0.2:8080	jpeg	935 KB 935 KB	159 ms 148 ms	

CLOUDFLARE

Event loop implementation

```lua
local magick = require "magick"

-- load watermark
local watermark = assert(magick.load_image(
   ngx.config.prefix() .. "/root/lua_moscow_logo.jpg"))
watermark:resize(25, 25)

return function()
 local path = string.format("%s/root/%s",
     ngx.config.prefix(),
     ngx.var.uri:match("^/thumb/(.+)"))
 ngx.log(ngx.DEBUG, "resizing ", path)

 local img = assert(magick.load_image(path))
 img:resize(300, 200)
 img:composite(watermark, 0, 0)

 ngx.print(img:get_blob())
end
```

The (fast) static image has to wait

Name	Status	Protocol	Remote Address	Type	Size	Time	Waterfall
172.17.0.2	200 OK	http/1.1	172.17.0.2:8080	document	578 B 340 B	2 ms 1 ms	
style.css	200 OK	http/1.1	172.17.0.2:8080	stylesheet	286 B 51 B	2 ms 2 ms	
moon.jpg /thumb	200 OK	http/1.1	172.17.0.2:8080	jpeg	30.9 KB 30.7 KB	149 ms 148 ms	
moon.jpg	200 OK	http/1.1	172.17.0.2:8080	jpeg	935 KB 935 KB	159 ms 148 ms	

CLOUDFLARE

Thread implementation

```lua
-- content_by_lua
return function()
 local threadpool = require("resty.threadpool")
 local path = string.format("%s/root/%s",
     ngx.config.prefix(),
     ngx.var.uri:match("^/thumb/(.+)"))

 ngx.print(assert(threadpool.resize:push_task(path)))
end
```

```lua
-- thread_pool_process_by_lua
local magick = require "magick"

-- load watermark
local watermark = assert(magick.load_image(
   ngx.config.prefix() ..
   "/root/lua_moscow_logo.jpg"))
watermark:resize(25, 25)

return function(path)
 ngx.log(ngx.DEBUG, "resizing ", path)

 local img = assert(magick.load_image(path))
 img:resize(300, 200)
 img:composite(watermark, 0, 0)

 return assert(img:get_blob())
end
```

Name	Status	Protocol	Remote Address	Type	Size	Time	Waterfall
172.17.0.2	200 OK	http/1.1	172.17.0.2:8080	document	578 B 340 B	9 ms 3 ms	
style.css	200 OK	http/1.1	172.17.0.2:8080	stylesheet	286 B 51 B	27 ms 26 ms	
moon.jpg /thumb	200 OK	http/1.1	172.17.0.2:8080	jpeg	30.9 KB 30.7 KB	168 ms 167 ms	
moon.jpg	200 OK	http/1.1	172.17.0.2:8080	jpeg	935 KB 935 KB	40 ms 27 ms	

CLOUDFLARE

Thread implementation

```lua
-- content_by_lua
return function()
 local threadpool = require("resty.threadpool")
 local path = string.format("%s/root/%s",
     ngx.config.prefix(),
     ngx.var.uri:match("^/thumb/(.+)"))

 ngx.print(assert(threadpool.resize:push_task(path)))
end
```

```lua
-- thread_pool_process_by_lua
local magick = require "magick"

-- load watermark
local watermark = assert(magick.load_image(
   ngx.config.prefix() ..
   "/root/lua_moscow_logo.jpg"))
watermark:resize(25, 25)

return function(path)
 ngx.log(ngx.DEBUG, "resizing ", path)

 local img = assert(magick.load_image(path))
 img:resize(300, 200)
 img:composite(watermark, 0, 0)

 return assert(img:get_blob())
end
```

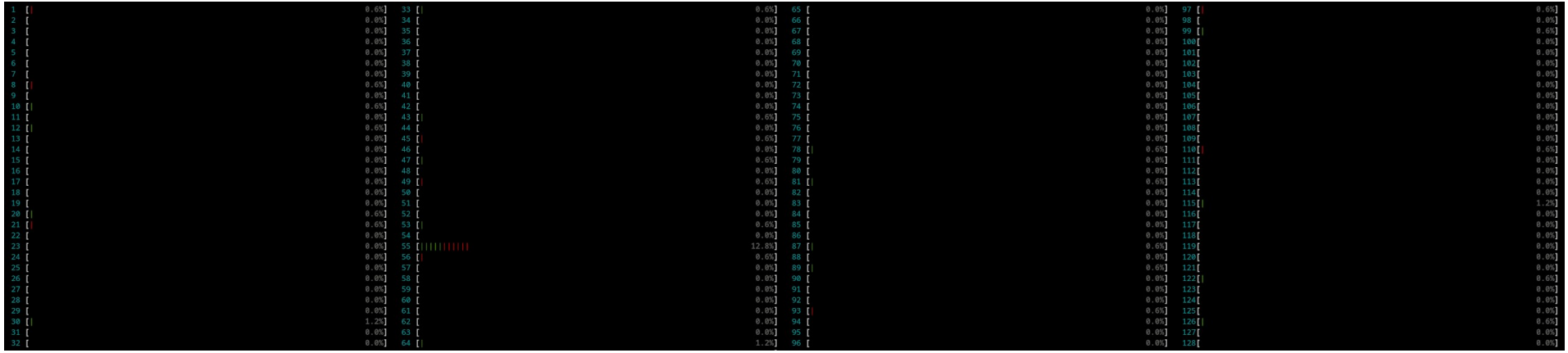
Name	Status	Protocol	Remote Address	Type	Size	Time	Waterfall
172.17.0.2	200 OK	http/1.1	172.17.0.2:8080	document	578 B 340 B	9 ms 3 ms	
style.css	200 OK	http/1.1	172.17.0.2:8080	stylesheet	286 B 51 B	27 ms 26 ms	
moon.jpg /thumb	200 OK	http/1.1	172.17.0.2:8080	jpeg	30.9 KB 30.7 KB	168 ms 167 ms	
moon.jpg	200 OK	http/1.1	172.17.0.2:8080	jpeg	935 KB 935 KB	40 ms 27 ms	

No more blocking

CLOUDFLARE

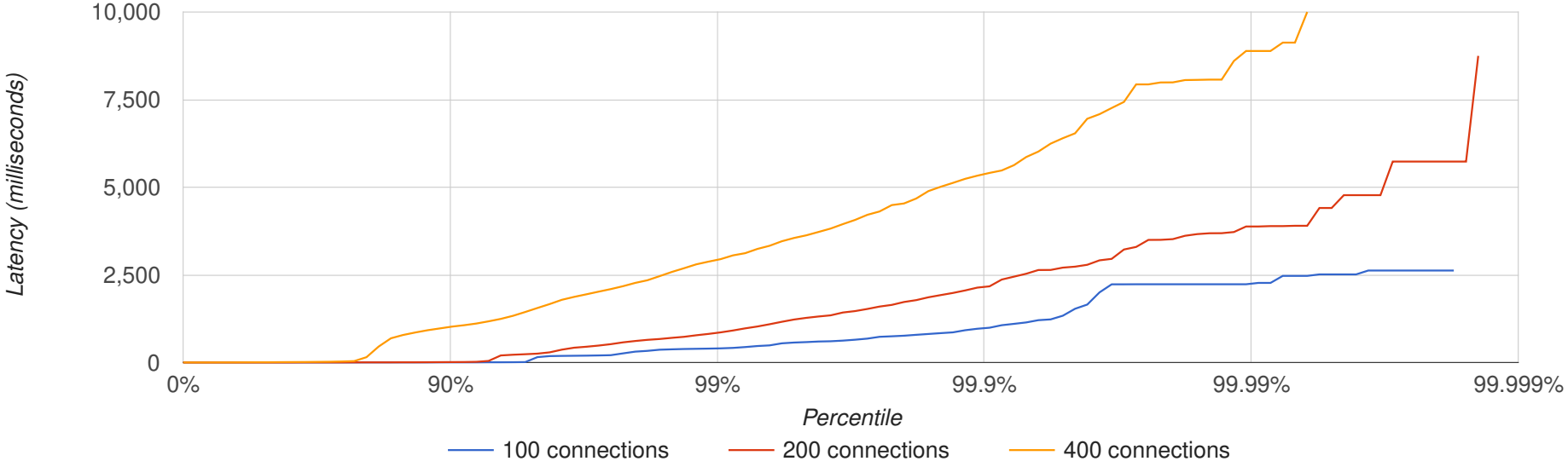
Benchmark time!

- Direct resizing runs 120 workers
- Threaded resizing runs 20 workers with 6 threads each
- Run with 100, 200, 400 simultaneous connections for 180 seconds
- Timeout is 10 seconds



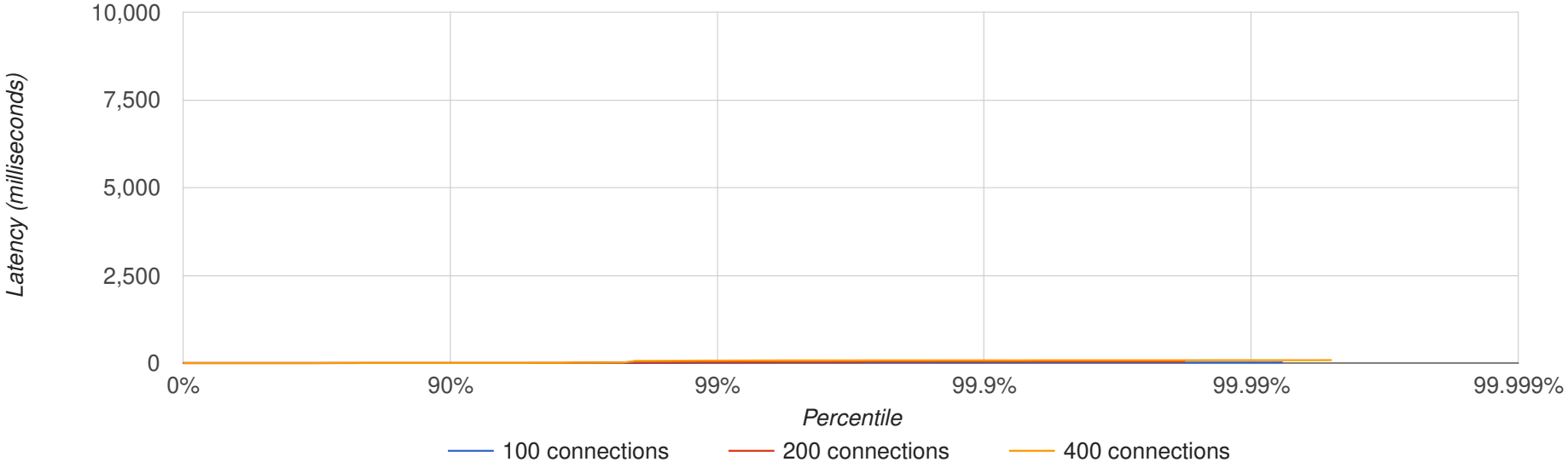
CLOUDFLARE

TTFB for small files using direct mode



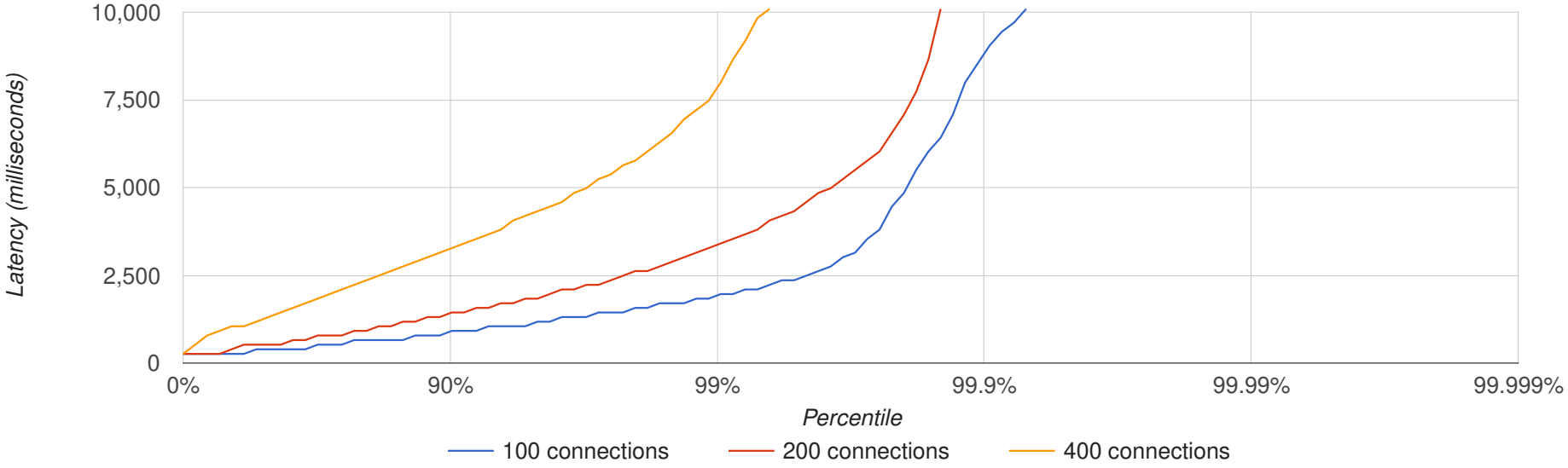
CLOUDFLARE

TTFB for small files using threads



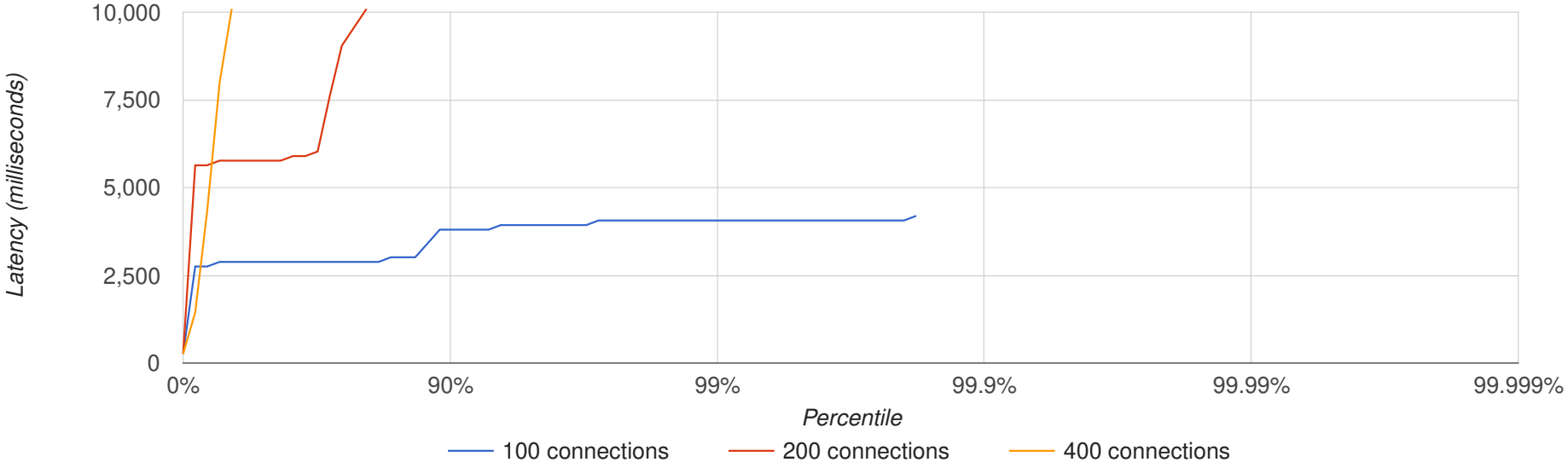
CLOUDFLARE

TTFB for thumbnails using direct mode



CLOUDFLARE

TTFB for thumbnails using threads



CLOUDFLARE

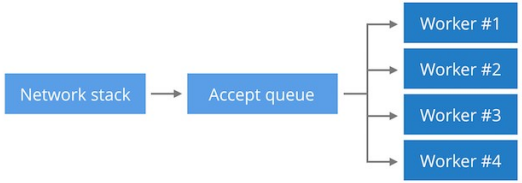
Timeout rate

Mode	Type	100	200	400
Direct	Static	0.03%	0.05%	0.18%
	Thumb	0.07%	0.15%	0.69%
Thread	Static	0.00%	0.00%	0.00%
	Thumb	0.00%	20.83%	67.51%

CLOUDFLARE

Why???

- ngnix event loop uses epoll
- Connections are not distributed fairly
  - Some even consider it is "fundamentally broken"
- Worker processes are now idle most of the time
- They will happily accept new connections

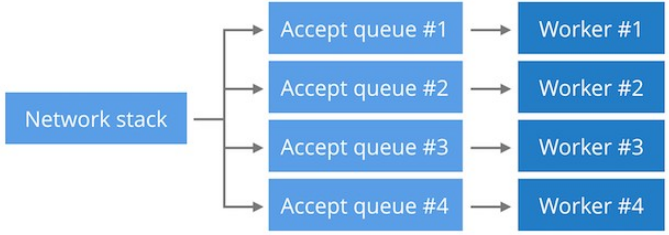


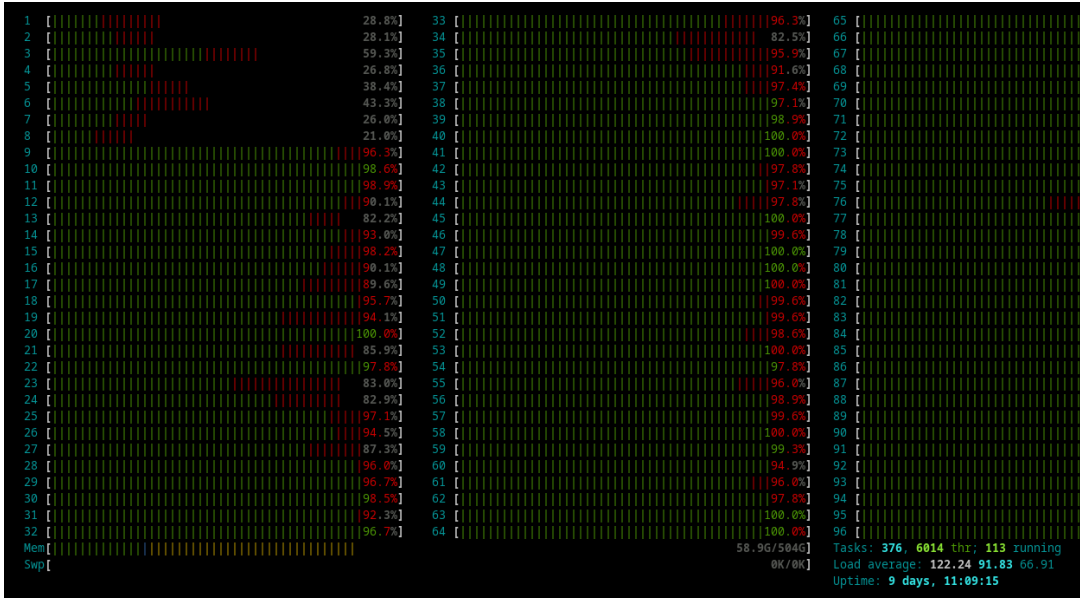

```
239529 julien    20   0 72892  5828  1904 S  0.0  0.0  0:00.00
239528 julien    20   0 72892  5828  1904 S  0.0  0.0  0:00.00
239526 julien    20   0 72892  5828  1904 S  0.0  0.0  0:00.00
239525 julien    20   0 72892  5828  1904 S  0.0  0.0  0:00.00
239478 julien    20   0 72892  5828  1904 S  0.0  0.0  0:00.00
239517 julien    20   0 72892  5828  1904 S  0.0  0.0  0:00.00
239515 julien    20   0 72892  5828  1904 S  0.0  0.0  0:00.00
239513 julien    20   0 72892  5828  1904 S  0.0  0.0  0:00.00
239511 julien    20   0 72892  5828  1904 S  0.0  0.0  0:00.00
239509 julien    20   0 72892  5828  1904 S  0.0  0.0  0:00.00
239507 julien    20   0 72892  5828  1904 S  0.0  0.0  0:00.00
239477 julien    20   0  558M  305M  8716 S 40.4  0.1  1:15.90
239496 julien    20   0  558M  305M  8716 S  8.5  0.1  0:12.42
239494 julien    20   0  558M  305M  8716 S  8.5  0.1  0:13.09
239492 julien    20   0  558M  305M  8716 R  5.1  0.1  0:12.01
239491 julien    20   0  558M  305M  8716 S  9.1  0.1  0:12.71
239489 julien    20   0  558M  305M  8716 S  0.0  0.1  0:12.35
239487 julien    20   0  558M  305M  8716 S  9.7  0.1  0:12.82
239476 julien    20   0  193M 15348  8656 S  0.0  0.0  0:00.64
239514 julien    20   0  193M 15348  8656 S  0.0  0.0  0:00.63
239512 julien    20   0  193M 15348  8656 S  0.0  0.0  0:00.00
239510 julien    20   0  193M 15348  8656 S  0.0  0.0  0:00.00
239508 julien    20   0  193M 15348  8656 S  0.0  0.0  0:00.00
239506 julien    20   0  193M 15348  8656 S  0.0  0.0  0:00.00
239505 julien    20   0  193M 15348  8656 S  0.0  0.0  0:00.00
239475 julien    20   0  515M 64632  8688 S  0.0  0.0  0:11.61
239501 julien    20   0  515M 64632  8688 S  0.0  0.0  0:01.81
239500 julien    20   0  515M 64632  8688 S  0.0  0.0  0:01.81
239498 julien    20   0  515M 64632  8688 S  0.0  0.0  0:01.97
239497 julien    20   0  515M 64632  8688 S  0.0  0.0  0:02.06
239495 julien    20   0  515M 64632  8688 S  0.0  0.0  0:02.06
239493 julien    20   0  515M 64632  8688 S  0.0  0.0  0:01.80
239474 julien    20   0 1262M 1014M  325M S 585.  0.2 30:03.94
239490 julien    20   0 1262M 1014M  325M R 96.3  0.2  4:57.91
239488 julien    20   0 1262M 1014M  325M R 97.4  0.2  4:58.47
239486 julien    20   0 1262M 1014M  325M R 96.3  0.2  4:59.45
239485 julien    20   0 1262M 1014M  325M R 98.6  0.2  5:00.03
239484 julien    20   0 1262M 1014M  325M R 95.1  0.2  4:59.29
239483 julien    20   0 1262M 1014M  325M R 97.4  0.2  4:58.63
```

CLOUDFLARE

Band-aid: reuseport

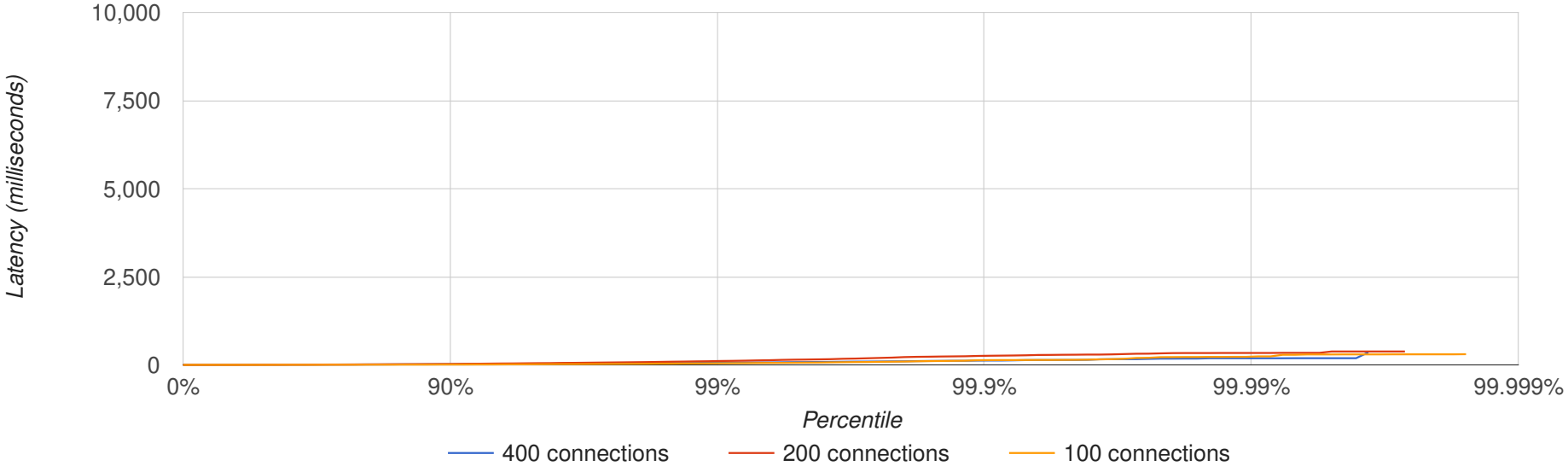
- TCP option originally meant to allow multiple listen on the same port
- Internally reuseport create a separate accept queue for each listening socket
- Once a new connection is queued somewhere, it is stuck there
- Only support inet sockets





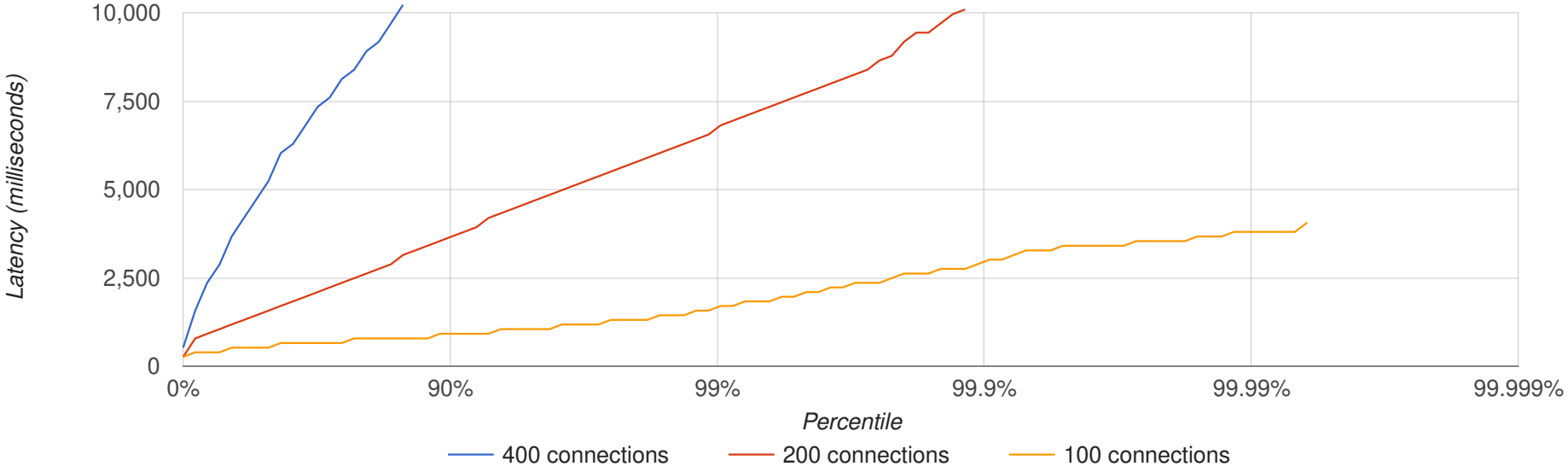
CLOUDFLARE

Small files with reuseport



CLOUDFLARE

Thumbnails with reuseport



CLOUDFLARE

Timeout rate

Mode	Type	100	200	400
Direct	Static	0.03%	0.05%	0.18%
	Thumb	0.07%	0.15%	0.69%
Thread	Static	0.00%	0.00%	0.00%
	Thumb	0.00%	20.83%	67.51%
Thread + reuseport	Static	0.00%	0.00%	0.00%
	Thumb	0.00%	0.11%	14.87%

CLOUDFLARE

Key takeaways

- Threads don't magically bring you more compute power
  - CPU-bound work still have to run
  - Might make things worse for overloaded servers
- epoll unbalance is very tricky
- Reuseport might not be great either
- Used carefully, thread pools will free the event loop
  - More consistent latencies
  - Lightweight request stay fast

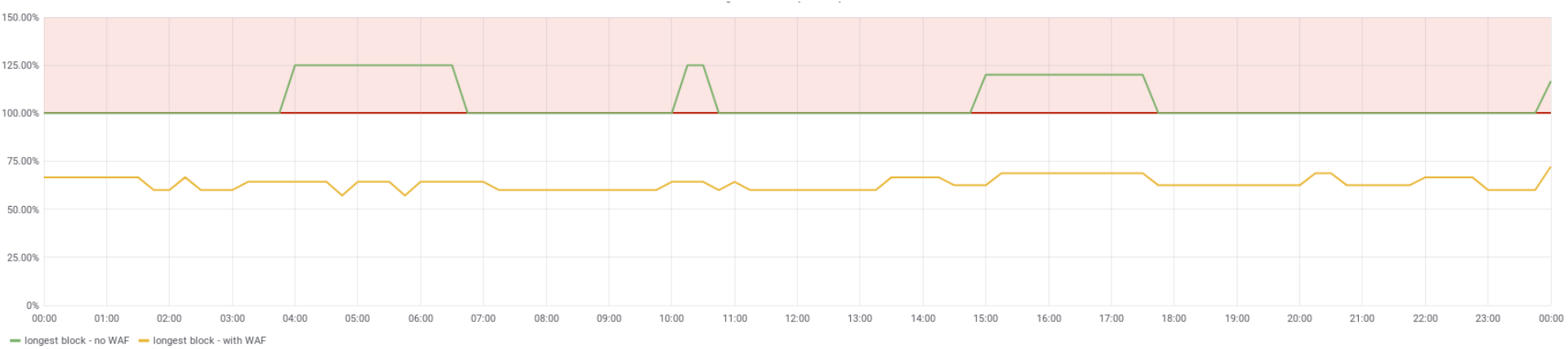
CLOUDFLARE

So... what about the WAF?

- Each nginx worker now have multiple WAF threads
- If the task queue is full, we fall back to the event loop (like before)
- Argument passing:
  - All the needed data is packed into a table
  - The small request bodies are passed as strings
  - Bigger ones in temp files, only the path is passed to the thread
- Cloudflare runs a custom kernel that overcomes the epoll issues

CLOUDFLARE

Production results - event loop block

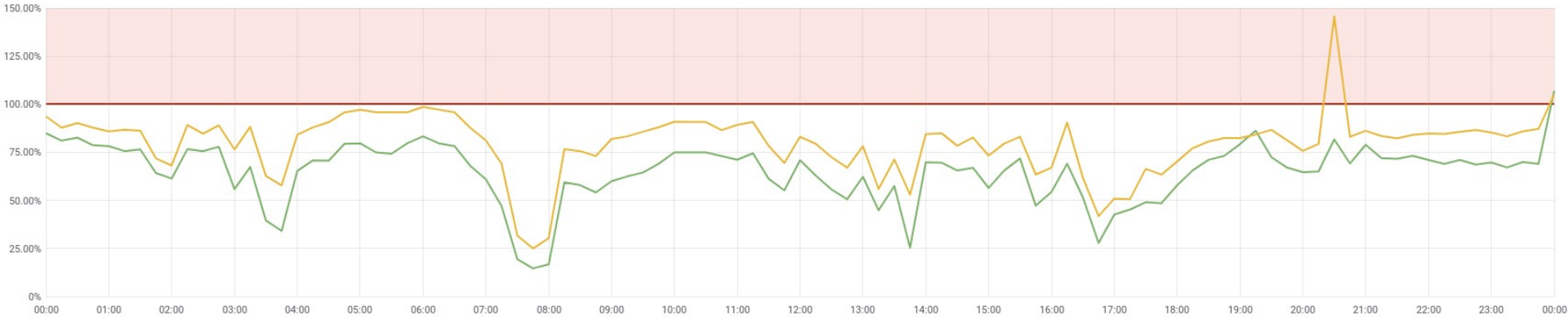


Without WAF

With WAF

CLOUDFLARE

Production results - TTFB

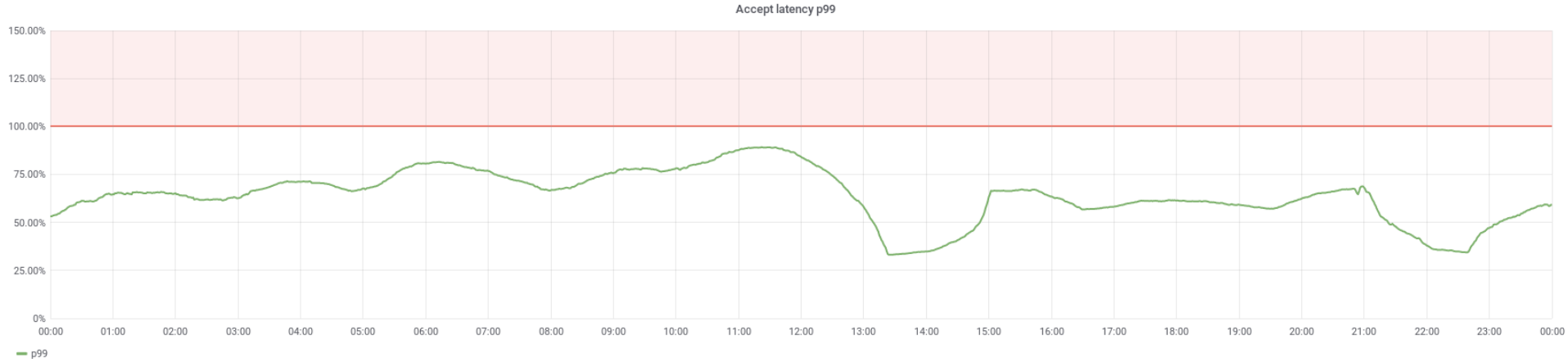


Without WAF

With WAF

CLOUDFLARE

Production results - accept latency



CLOUDFLARE

Current state

- Open-sourcing is in progress
  - Most of the relevant ngx.* APIs are available
  - (Ab)use private APIs at the moment
- Only the request/response pattern is implemented
- Data serialization creates a lot of duplication
- Tested only on Linux
- Issues with epoll unbalance

CLOUDFLARE

That's all folks!

- Links: http://tiny.cc/resty-threadpool

CLOUDFLARE