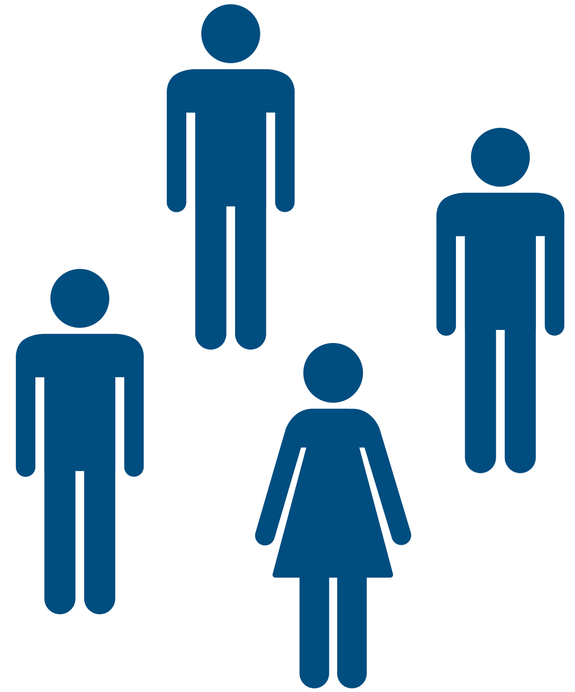
HolyJS, 11–12 ноября 2023

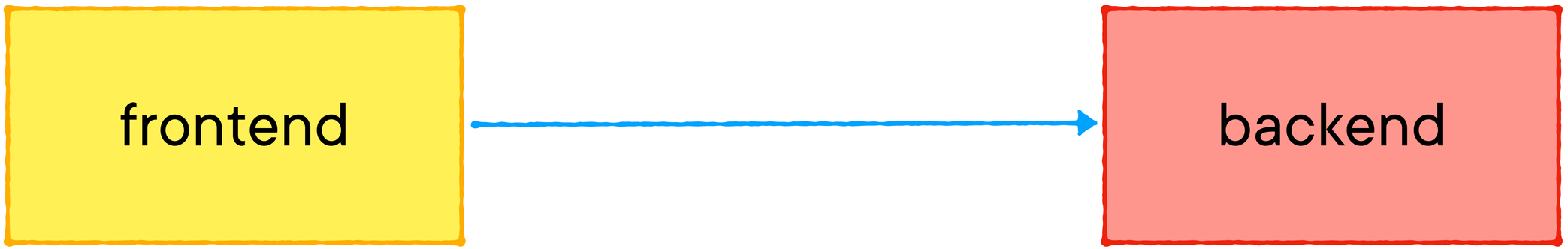
Про фронтенд с точки зрения плюсовика-компиляторщика

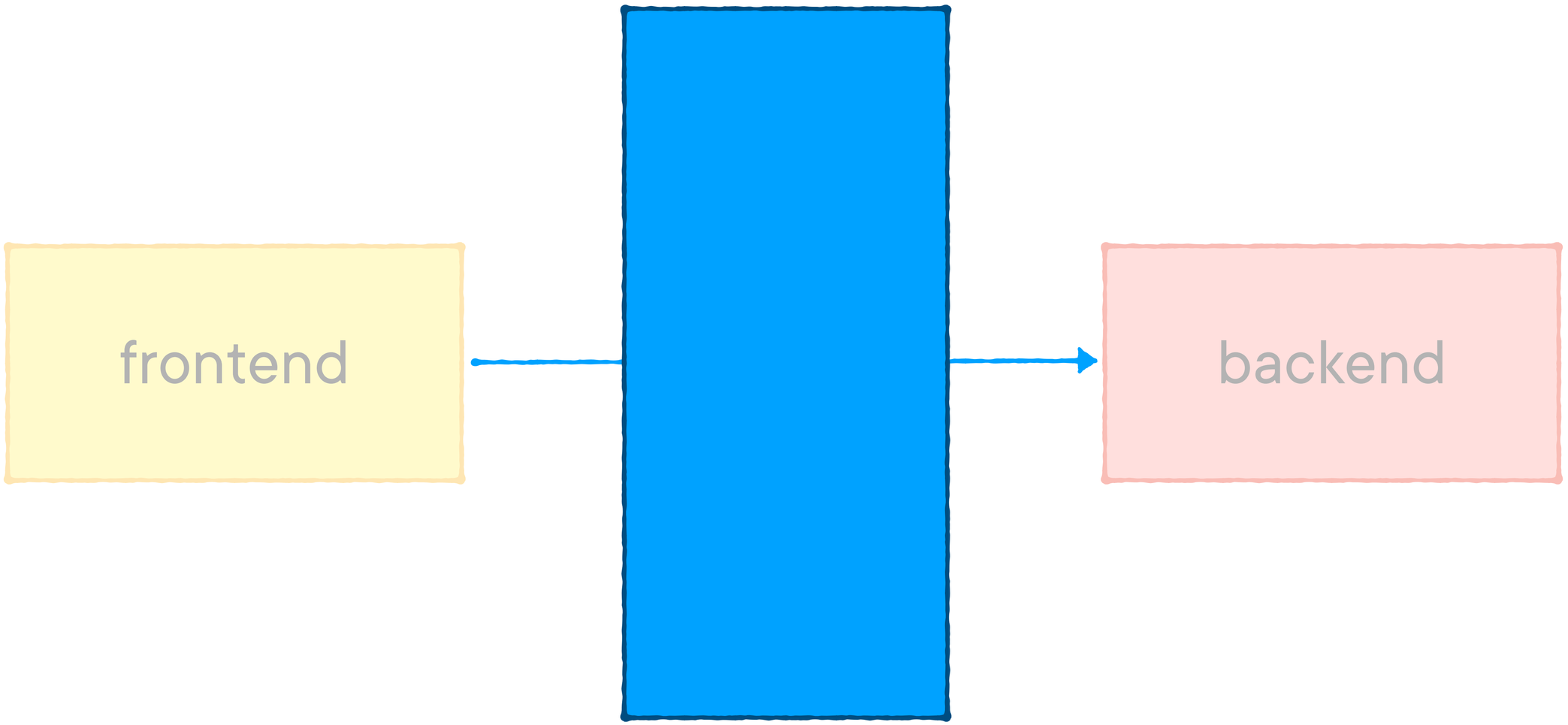
Александр Кирсанов, ВКонтакте

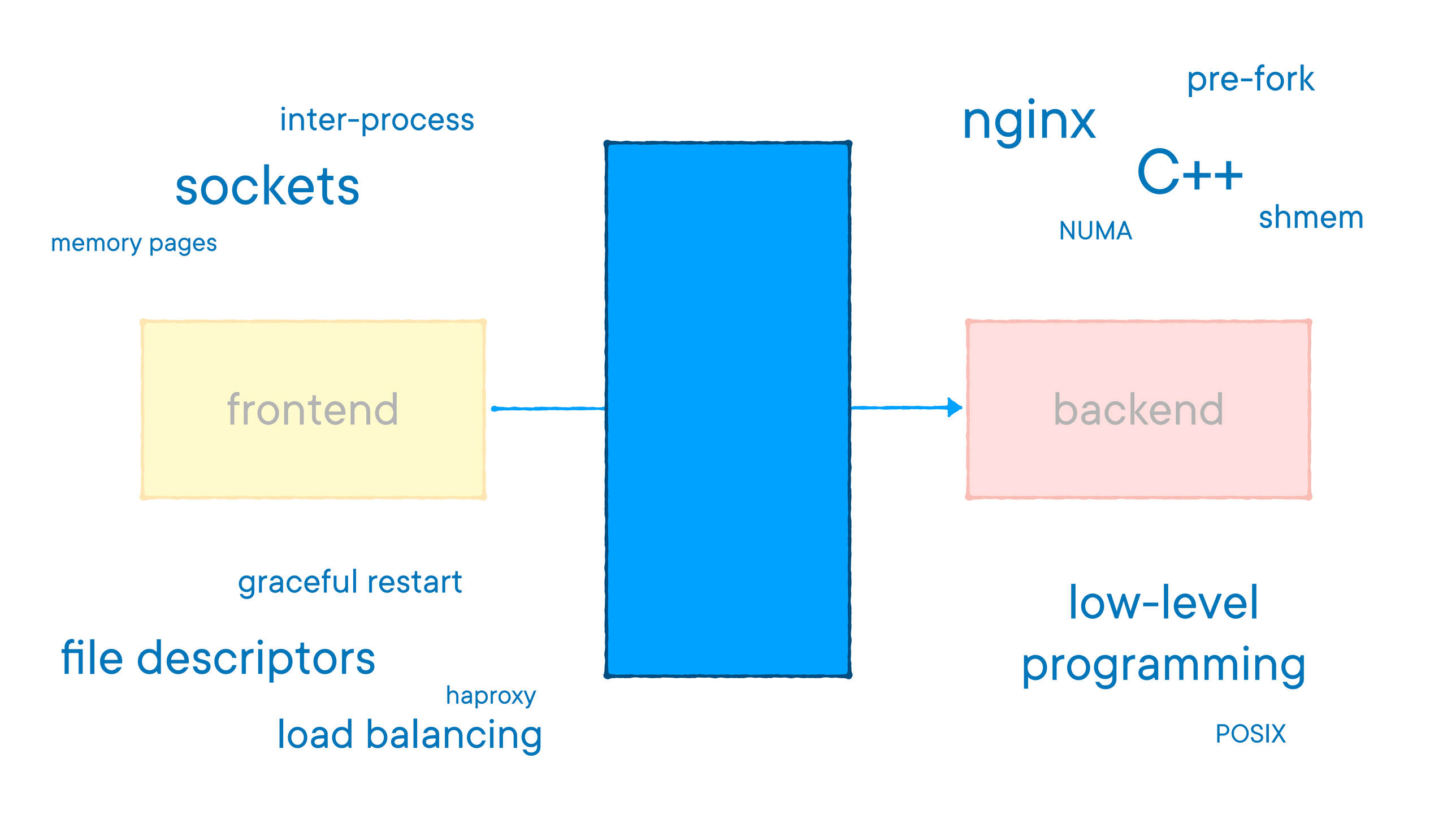


Я

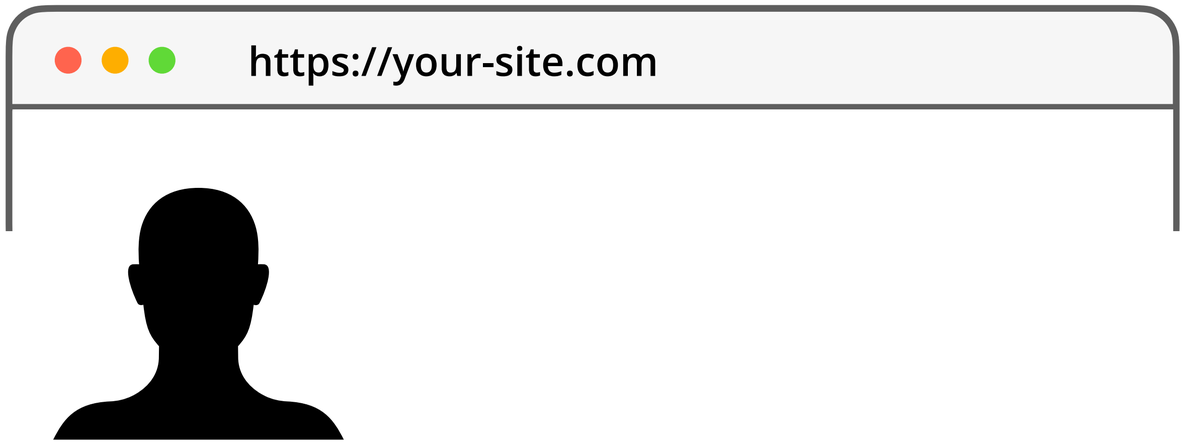
фронтендеры

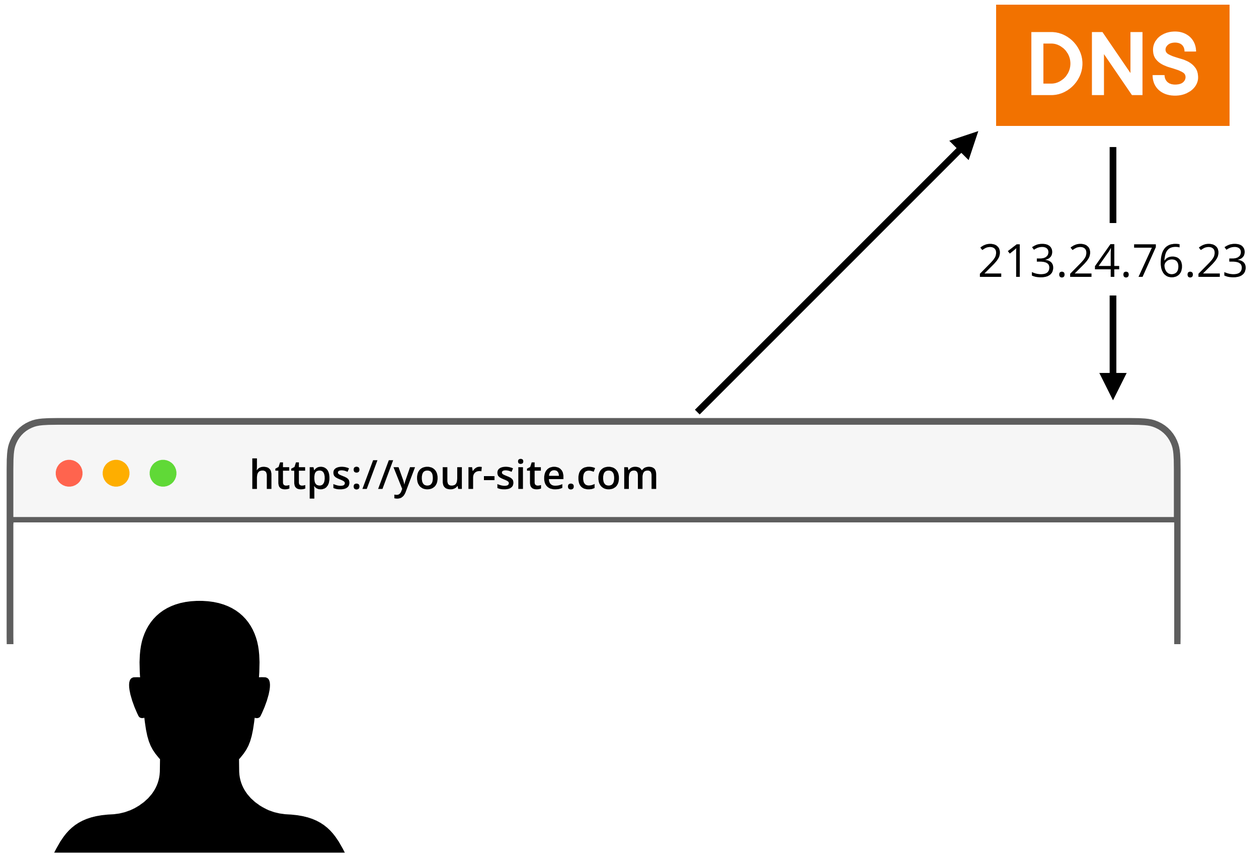

frontend
backend

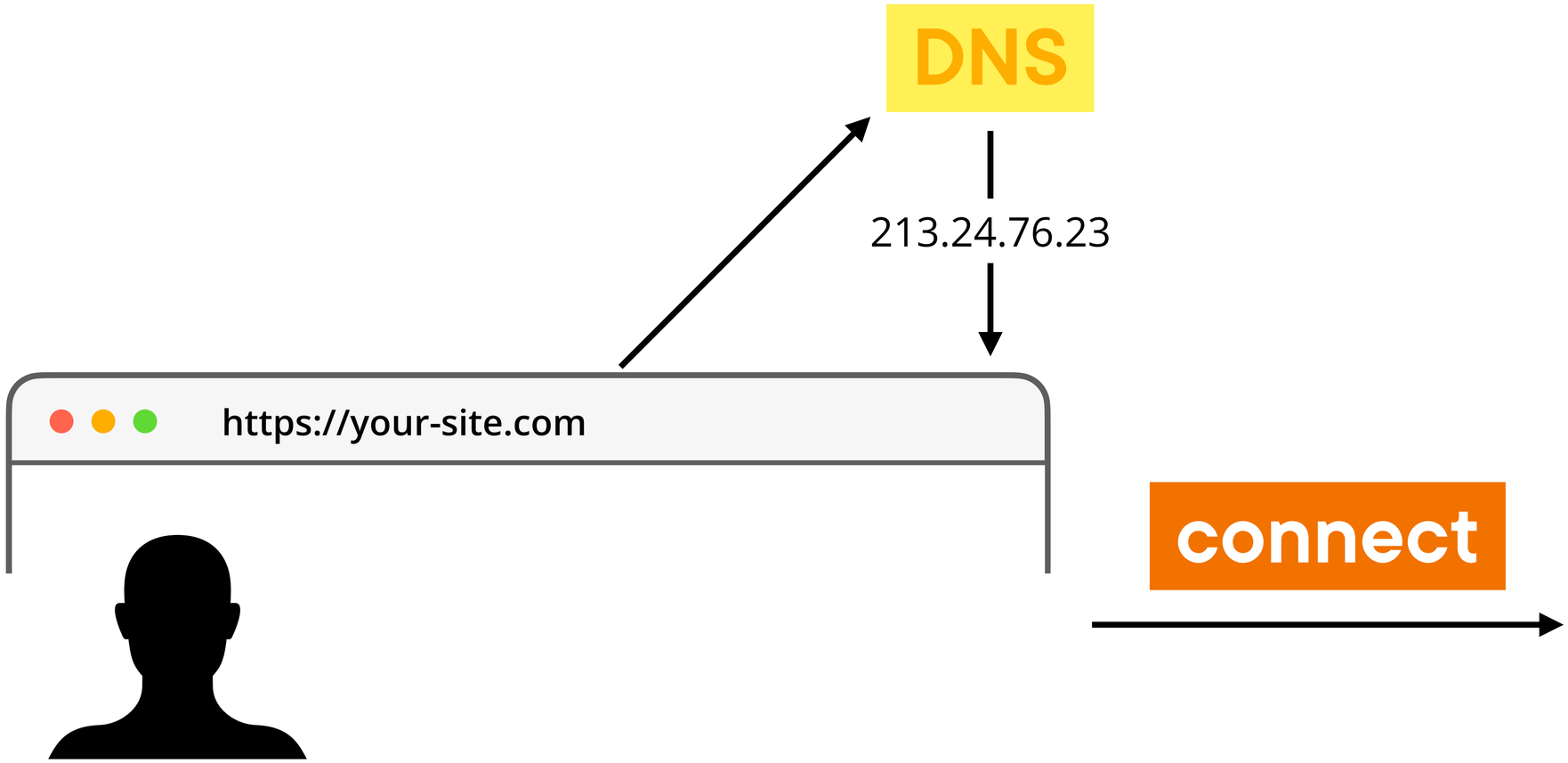

frontend
backend

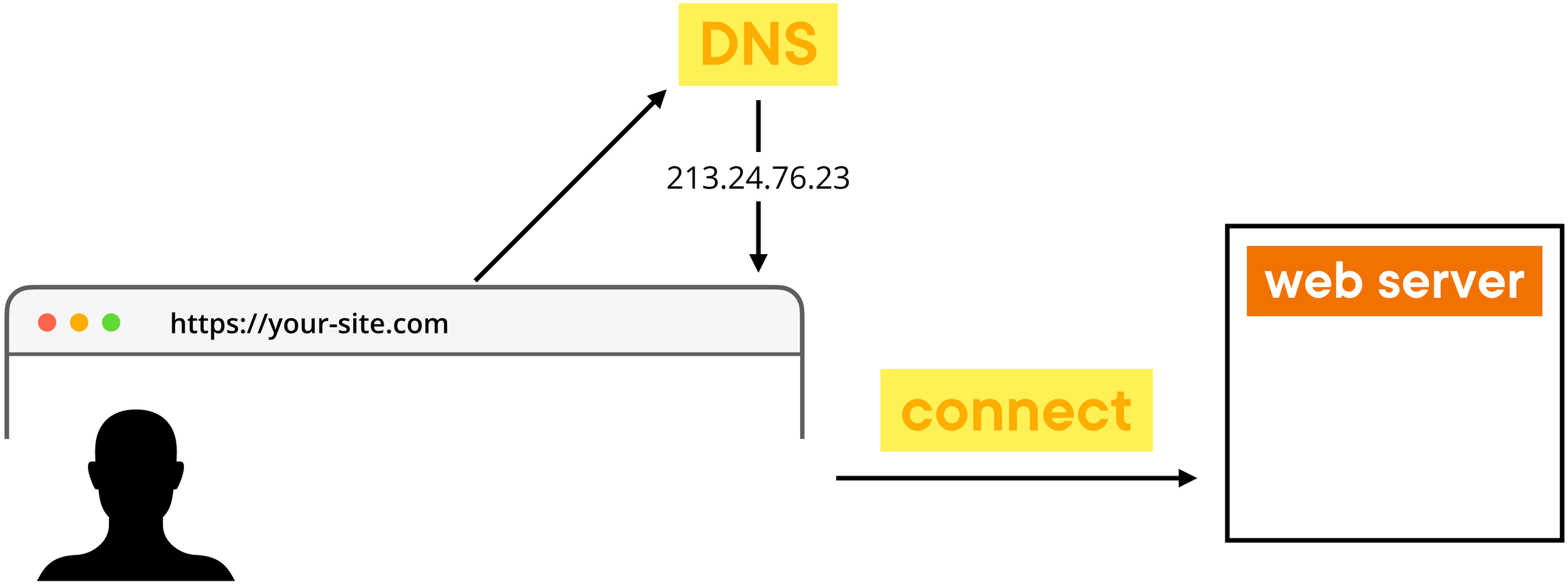

inter-process
sockets
memory pages
pre-fork
nginx
C++
NUMA
shmem
frontend
backend
graceful restart
file descriptors
haproxy
load balancing
low-level
programming
POSIX

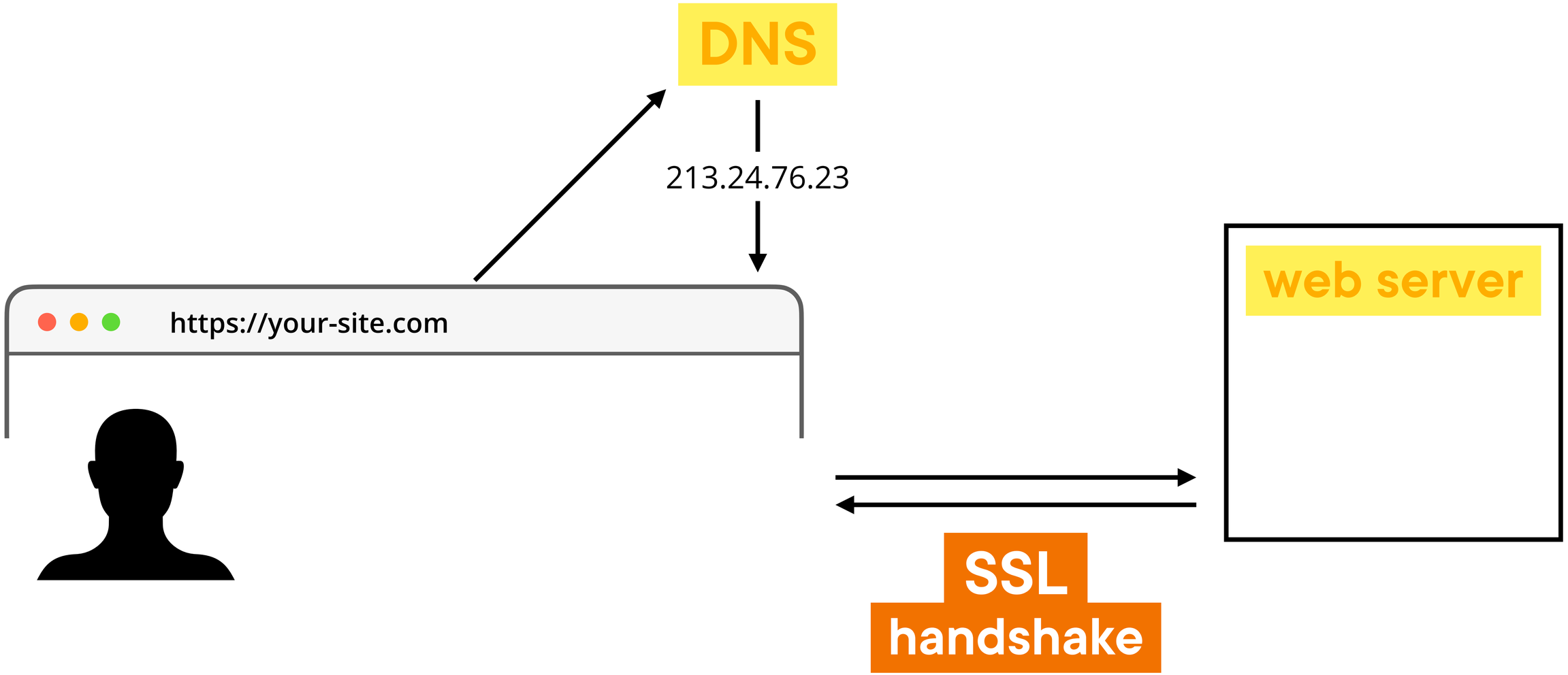
Ликбез по не-фронтовым технологиям

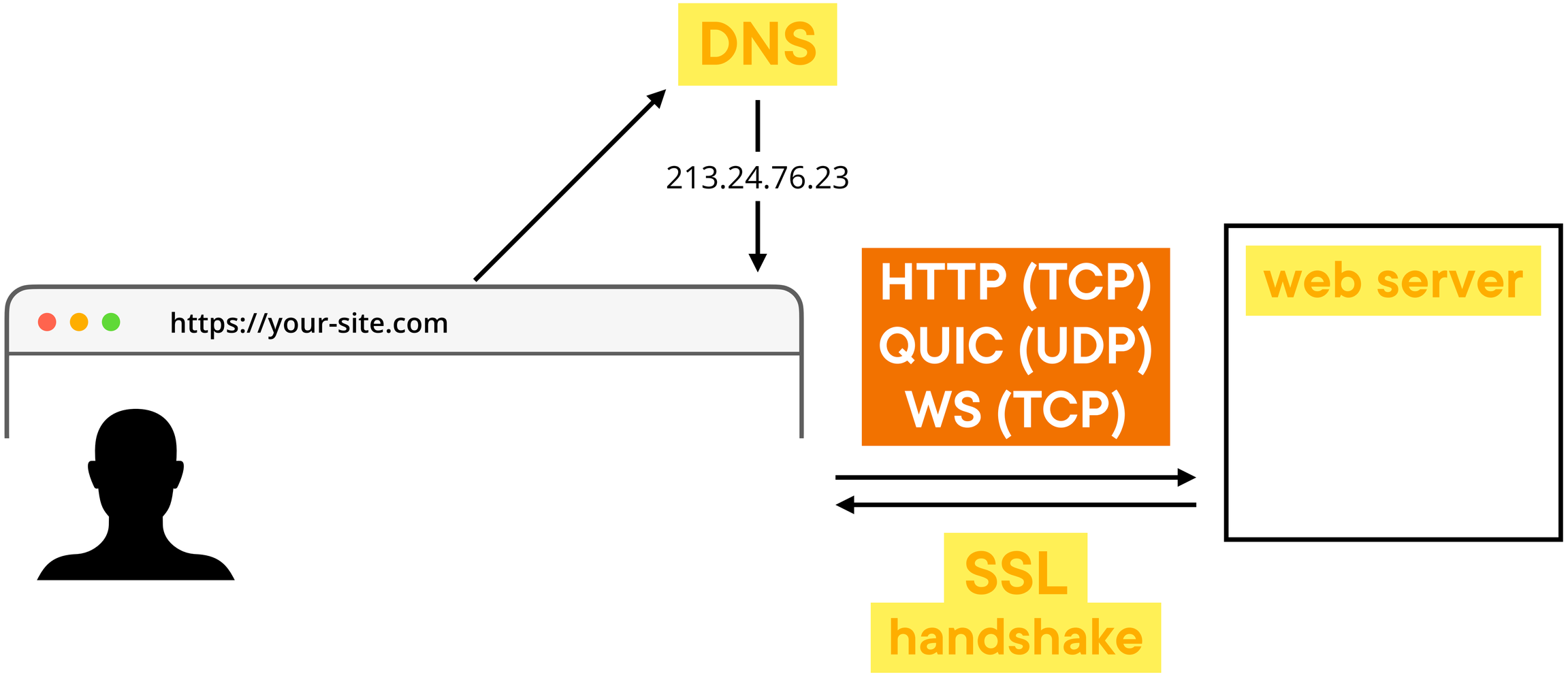

https://your-site.com

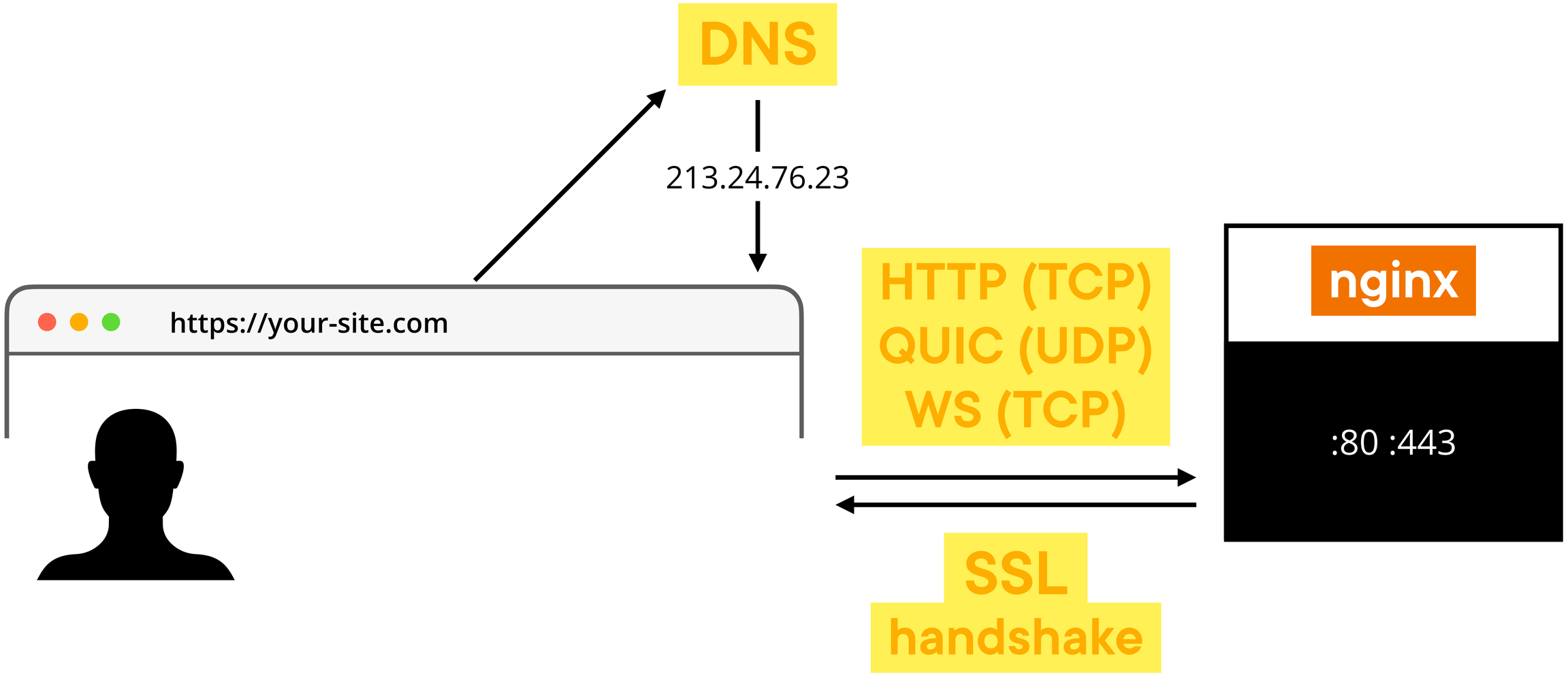

DNS
213.24.76.23
https://your-site.com

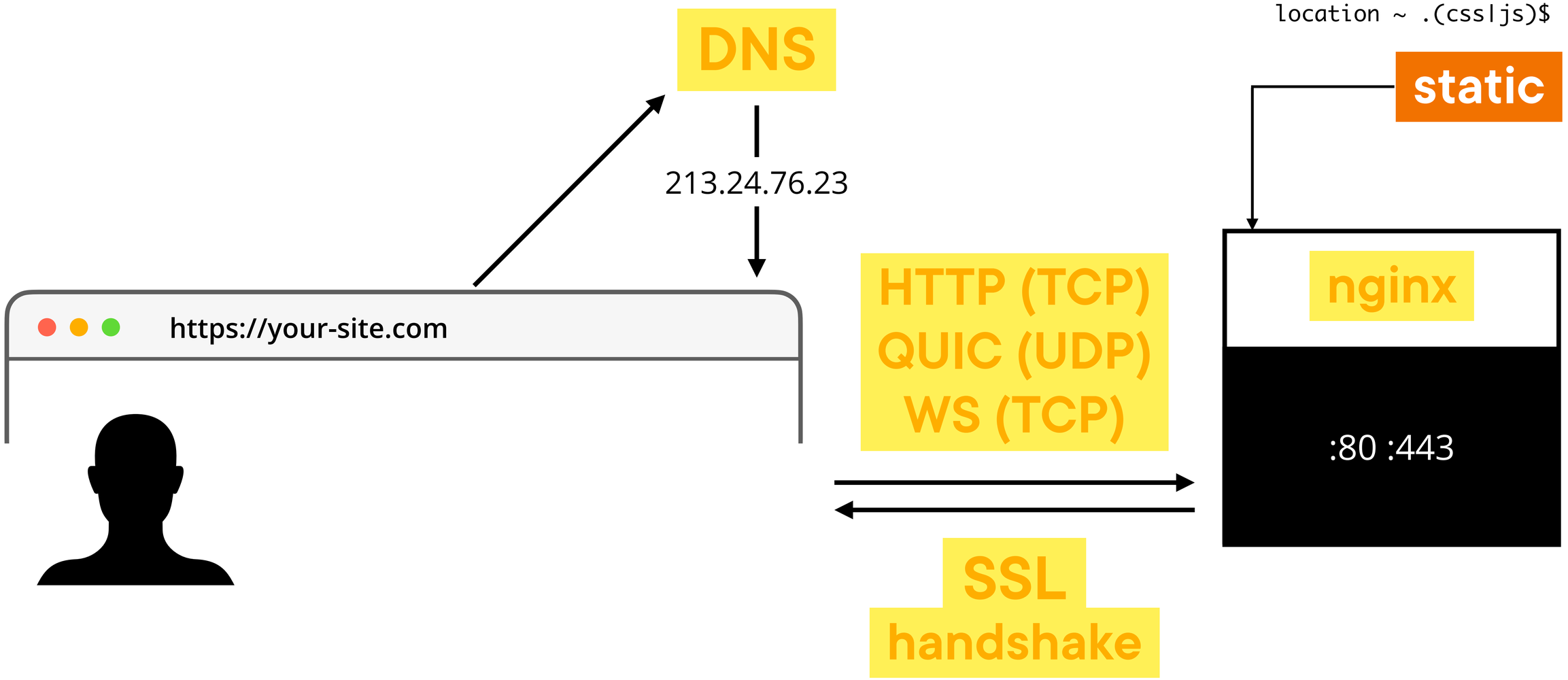

DNS
213.24.76.23
https://your-site.com
connect

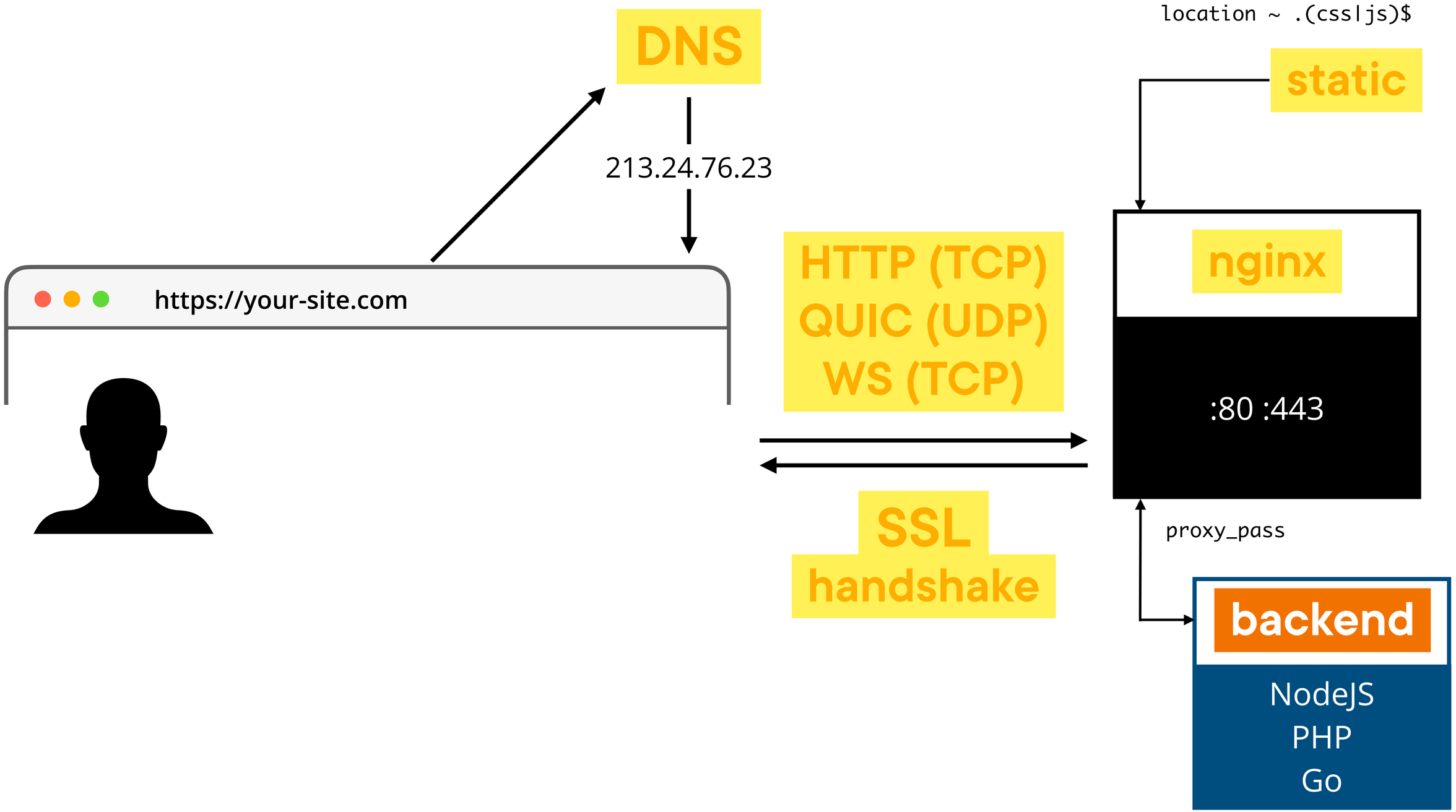

DNS
213.24.76.23
https://your-site.com
connect
web server

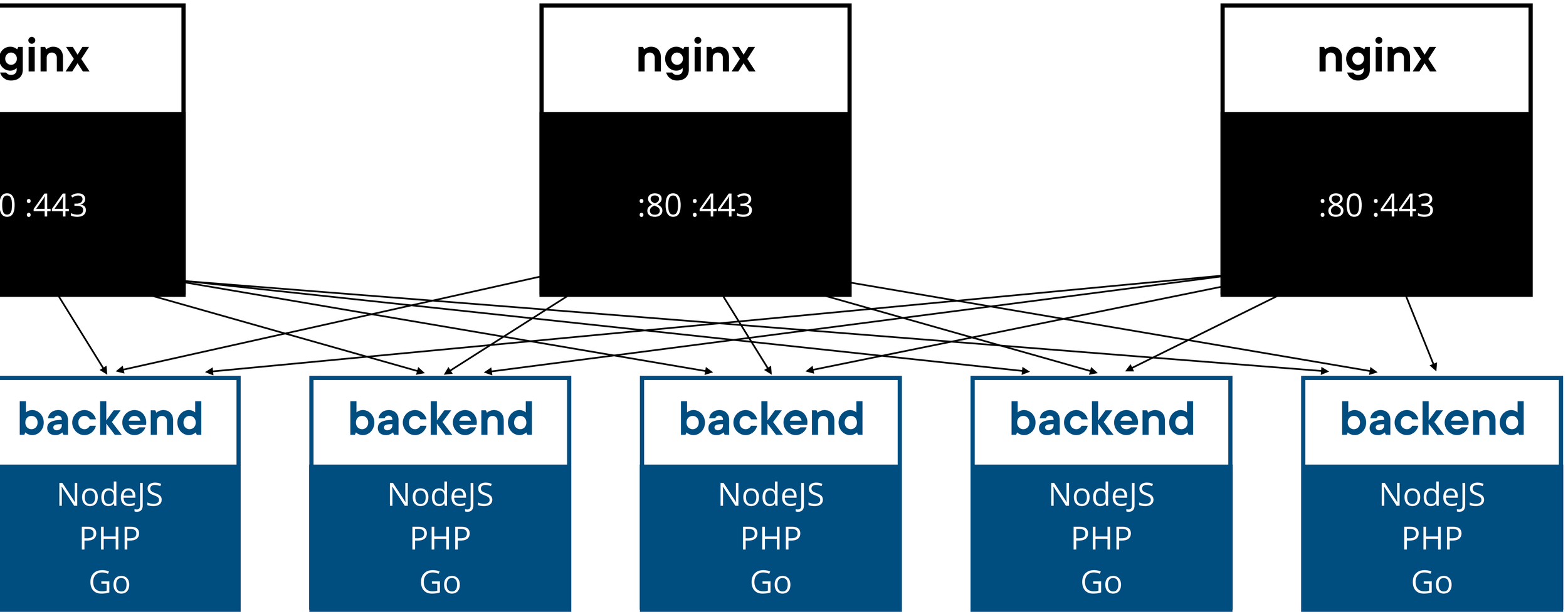

DNS
213.24.76.23
https://your-site.com
web server
SSL
handshake

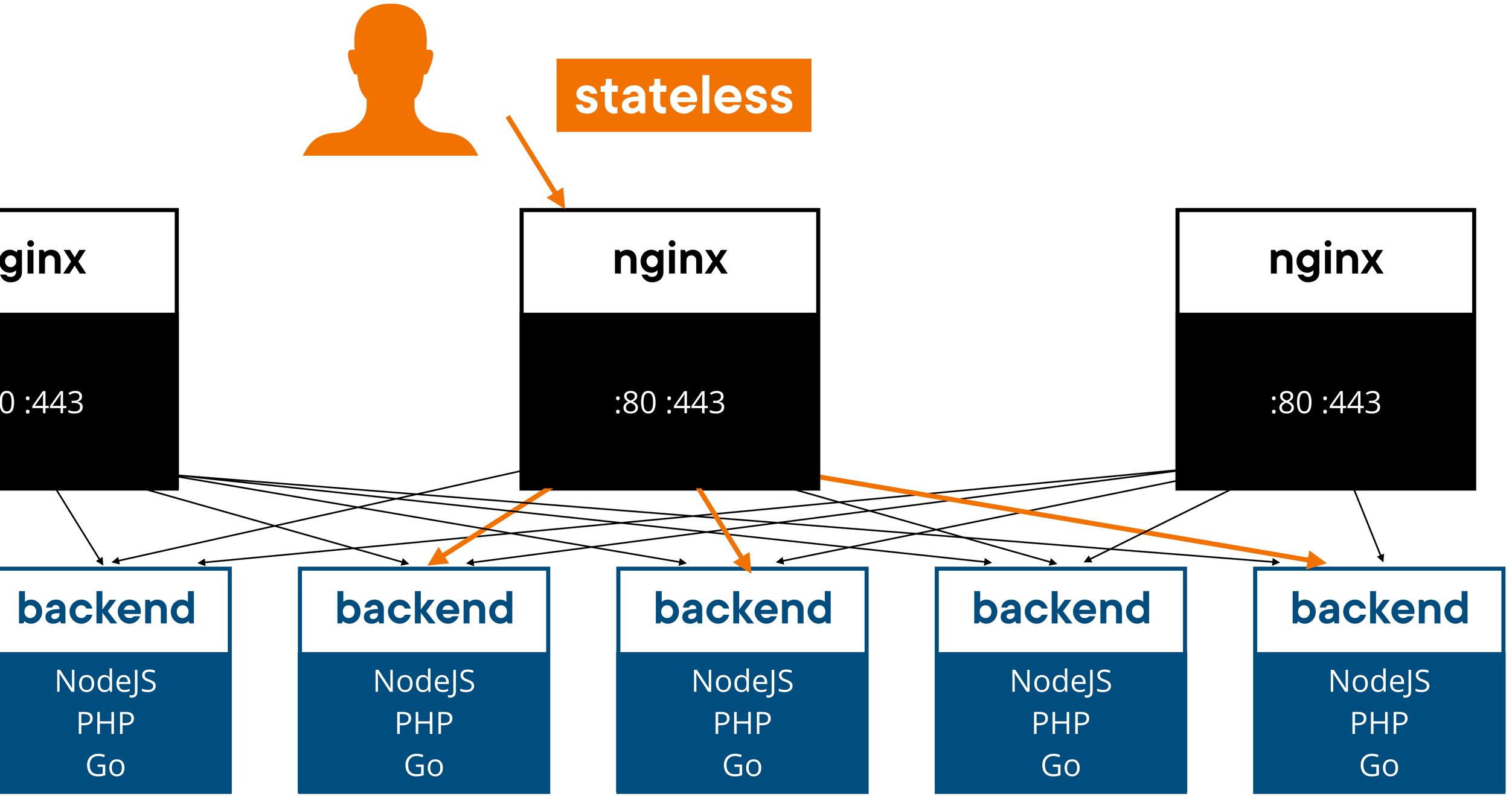

DNS
213.24.76.23
https://your-site.com
HTTP (TCP)
QUIC (UDP)
WS (TCP)
web server
SSL
handshake


DNS
213.24.76.23
https://your-site.com
HTTP (TCP)
QUIC (UDP)
WS (TCP)
nginx
:80 :443
SSL
handshake


location ~ .(css|js)$
DNS
static
213.24.76.23
https://your-site.com
HTTP (TCP)
QUIC (UDP)
WS (TCP)
nginx
:80 :443
SSL
handshake


location ~ .(css|js)$
DNS
213.24.76.23
static
https://your-site.com
HTTP (TCP)
QUIC (UDP)
WS (TCP)
nginx
:80 :443
SSL
handshake
proxy_pass
backend
NodeJS
PHP
Go


ginx
0 :443
nginx
:80 :443
nginx
:80 :443
backend
NodeJS
PHP
Go
backend
NodeJS
PHP
Go
backend
NodeJS
PHP
Go
backend
NodeJS
PHP
Go
backend
NodeJS
PHP
Go


stateless
ginx
0 :443
nginx
:80 :443
nginx
:80 :443
backend
NodeJS
PHP
Go
backend
NodeJS
PHP
Go
backend
NodeJS
PHP
Go
backend
NodeJS
PHP
Go
backend
NodeJS
PHP
Go


frontend
Что мы узнали
backend

NGINX
и другие
веб-сервера

Пишем свой веб-сервер на C++

Алгоритм работы веб-сервера

- создать сокет
- открыть порт
- дождаться коннекта
- считать запрос
- сформировать ответ
- записать ответ в коннекшен

1. Создать сокет

```
int server_fd;
int opt;

server_fd = socket(AF_INET, SOCK_STREAM, 0);
setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));
```

2. Открыть порт

```
sockaddr_in address{};
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(8080);

bind(server_fd, (sockaddr *)&address, sizeof(address));
listen(server_fd, 3);
```

3. Дождаться коннекта

```
int conn_fd;
int addrlen;

conn_fd = accept(server_fd, (sockaddr *)&address, (socklen_t *)&addrlen));
```

4. Считать запрос

```
char buffer[1024];  // считаем, что больше 1 КБ запроса не будет

size_t read_len = recv(conn_fd, buffer, 1024, 0);
buffer[read_len] = '\0';
```

5. Сформировать ответ

```
std::stringstream body;
body
  << "<h1>Request headers</h2>"
  << "<pre>" << buffer << "</pre>"
  << "<i><small>Small C++ HTTP Server</small></i>";

std::stringstream http_response;
http_response
  << "HTTP/1.1 200 OK\r\n"
  << "Version: HTTP/1.1\r\n"
  << "Content-Type: text/html; charset=utf-8\r\n"
  << "Content-Length: " << body.str().size()
  << "\r\n\r\n"
  << body.str();
```

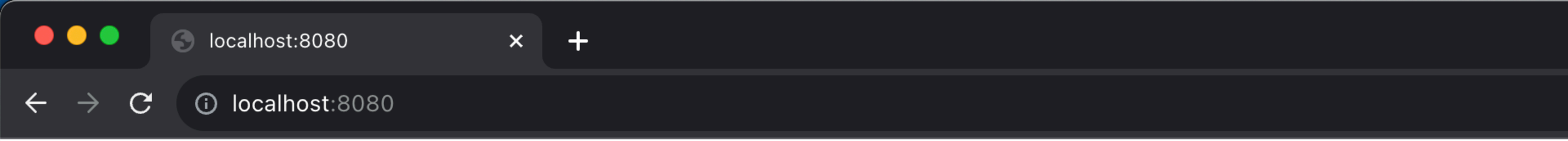
6. Записать ответ в коннекшен

```
send(
  conn_fd,
   http_response.str().c_str(),
   http_response.str().size(),
   0
);

close(conn_fd);
```

Полный исходник,

чтобы скопировать

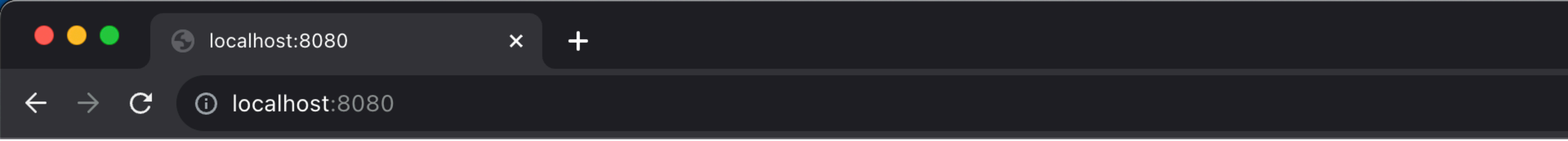
```cpp
#include <netinet/in.h>
#include <cstdlib>
#include <sys/socket.h>
#include <unistd.h>
#include <sstream>

int main(int argc, char const *argv[]) {
  int server_fd, opt;
  if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    exit(1);
  }
  if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt))) {
    exit(1);
  }

  sockaddr_in address{};
  address.sin_family = AF_INET;
  address.sin_addr.s_addr = INADDR_ANY;
  address.sin_port = htons(8080);

  if (bind(server_fd, (sockaddr *)&address, sizeof(address)) < 0) {
    exit(1);
  }
  if (listen(server_fd, 3) < 0) {
    exit(1);
  }

  int conn_fd, addrlen;
  if ((conn_fd = accept(server_fd, (sockaddr *)&address, (socklen_t *)&addrlen)) < 0) {
    exit(1);
  }

  char buffer[1024];
  size_t read_len = recv(conn_fd, buffer, 1024, 0);
  if (read_len <= 0) {
    exit(1);
  }
  buffer[read_len] = '\0';

  std::stringstream body;
  body
    << "<h1>Request headers</h2>"
    << "<pre>" << buffer << "</pre>"
    << "<i><small>Small C++ HTTP Server</small></i>";

  std::stringstream http_response;
  http_response
    << "HTTP/1.1 200 OK\r\n"
    << "Version: HTTP/1.1\r\n"
    << "Content-Type: text/html; charset=utf-8\r\n"
    << "Content-Length: " << body.str().size()
    << "\r\n\r\n"
    << body.str();

  send(conn_fd, http_response.str().c_str(), http_response.str().size(), 0);

  close(conn_fd);
  shutdown(server_fd, SHUT_RDWR);
  return 0;
}
```



Request headers

```
GET / HTTP/1.1
Host: localhost:8080
Connection: keep-alive
sec-ch-ua: "Google Chrome";v="119", "Chromium";v="119", "Not?A_Brand";v="24"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "macOS"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.0.0 Safari
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signe
X-Conference-Name: HolyJS Nov 2023
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: ru-RU,ru;q=0.9
```

Small C++ HTTP Server



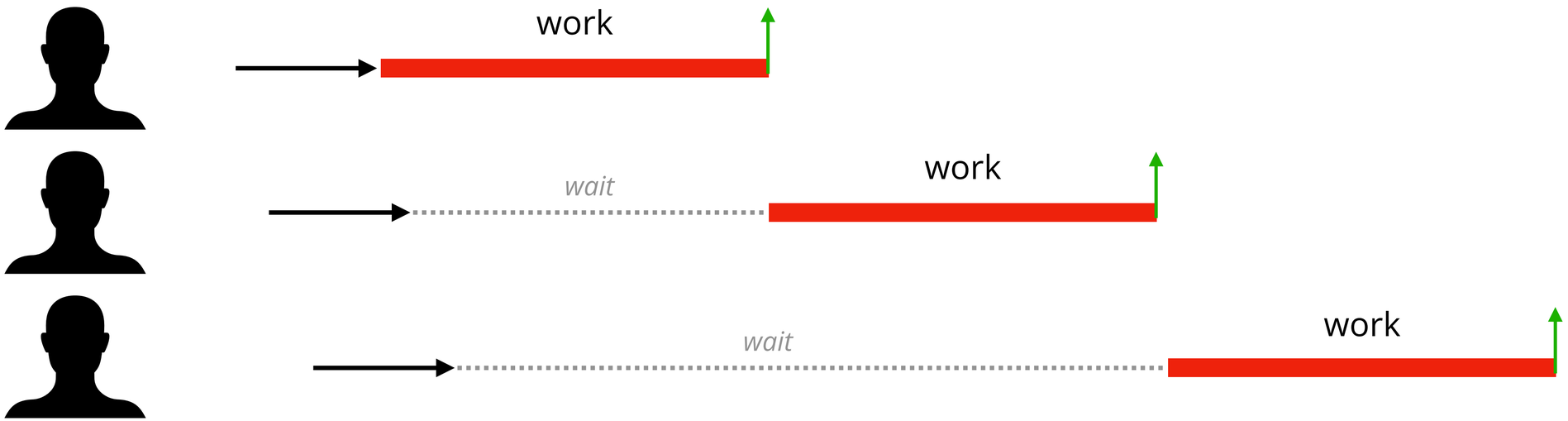
Request headers

```
GET / HTTP/1.1
Host: localhost:8080
Connection: keep-alive
sec-ch-ua: "Google Chrome";v="119", "Chromium";v="119", "Not?A_Brand";v="24"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "macOS"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.0.0 Safari
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signe
X-Conference-Name: HolyJS Nov 2023
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: ru-RU,ru;q=0.9
```

Small C++ HTTP Server

Обрабатывает только 1 запрос.
Везде блокируется.
Нет HTTPS.

Зачем параллелить исполнение



Как обрабатывать запросы параллельно?

Apache

VS

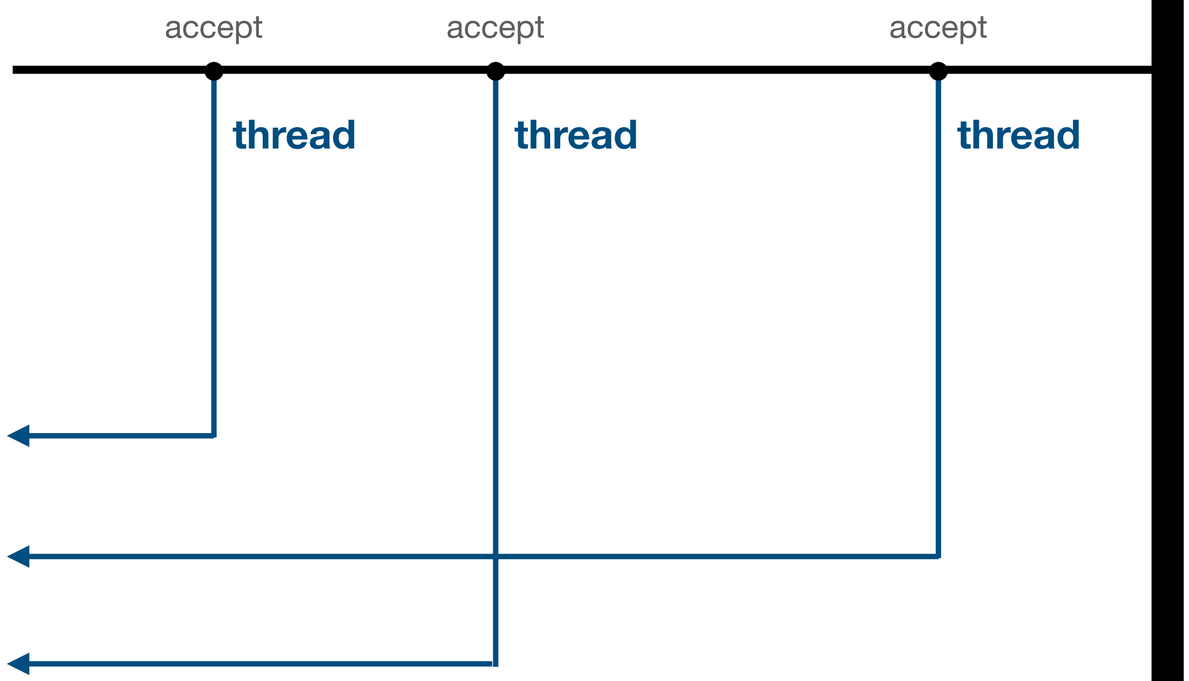
NGINX

Apache

VS

NGINX

thread per connection



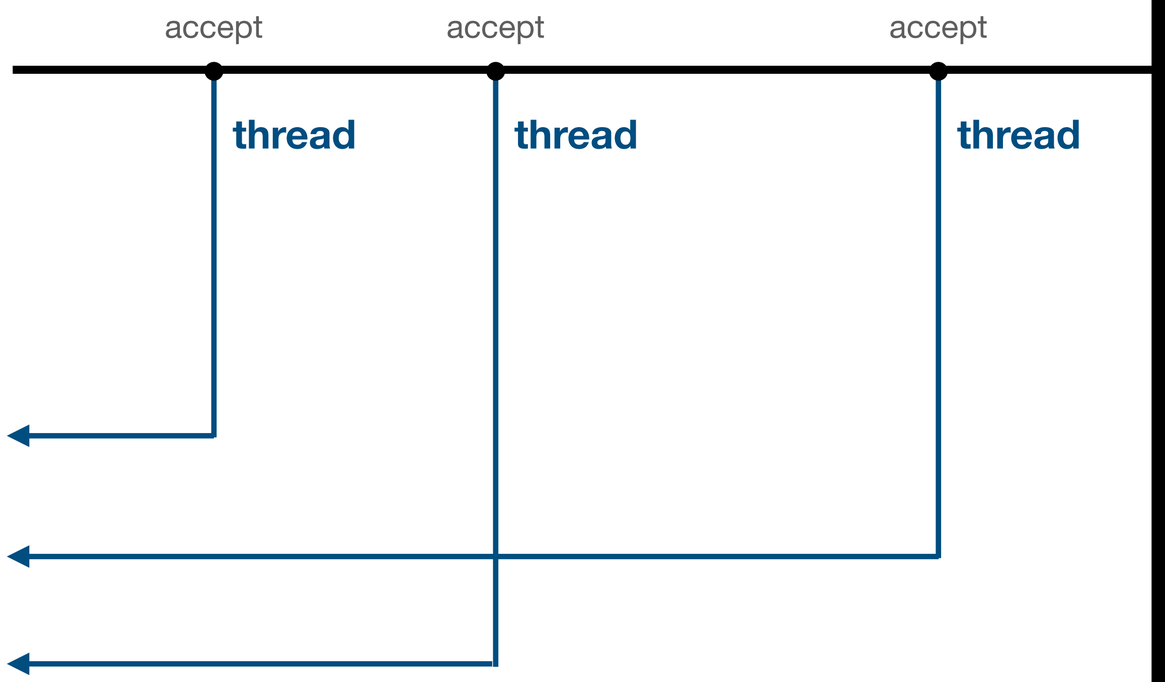
Apache

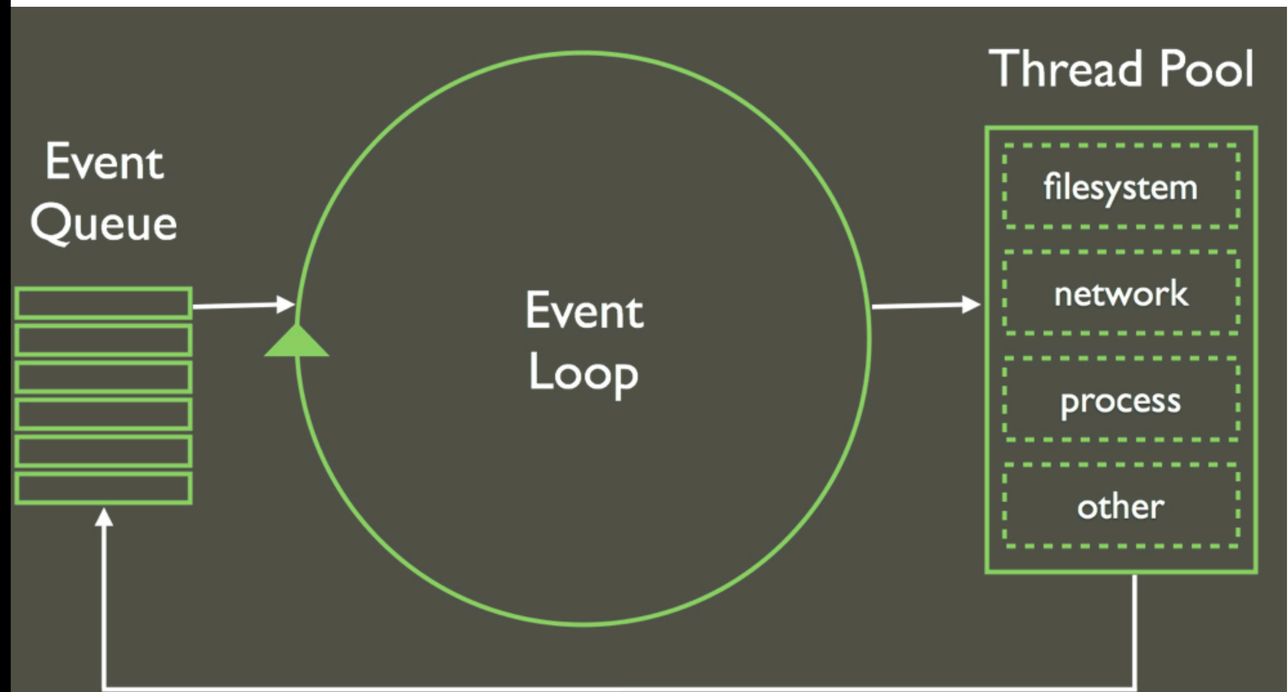
VS

NGINX

thread per connection

non-blocking event loop



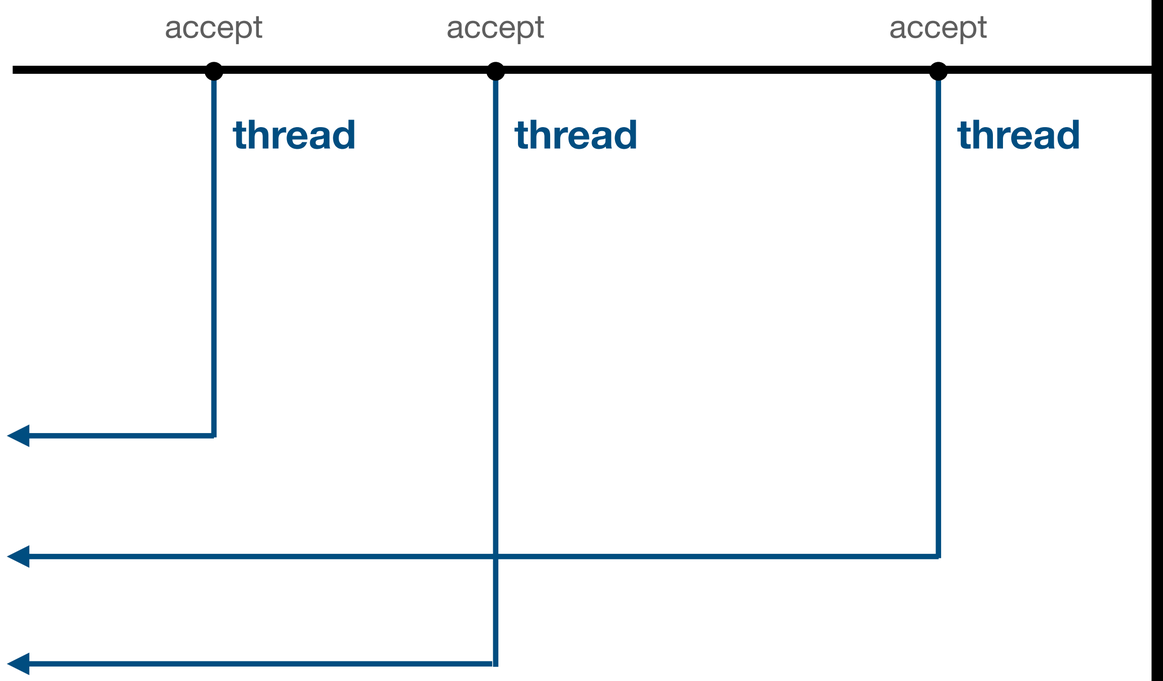


Apache

VS

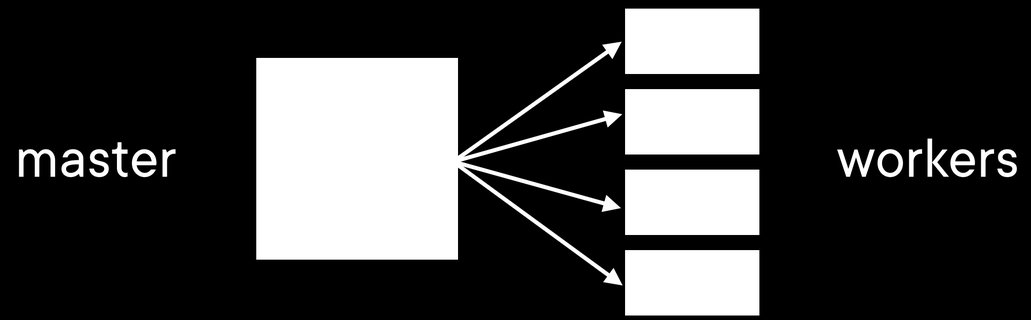
NGINX

thread per
connection



non-blocking
event loop

pre-fork model



Чем занимается nginx

Чем занимается nginx

- парсит и патчит http-запросы

Чем занимается nginx

- парсит и патчит http-запросы
- терминирует https

Чем занимается nginx

- парсит и патчит http-запросы
- терминирует https
- раздаёт статический контент

Чем занимается nginx

- парсит и патчит http-запросы
- терминирует https
- раздаёт статический контент
- проксирует на бэкенд

Чем занимается nginx

- парсит и патчит http-запросы
- терминирует https
- раздаёт статический контент
- проксирует на бэкенд
- кэширующий сервер

Чем занимается nginx

- парсит и патчит http-запросы
- терминирует https
- раздаёт статический контент
- проксирует на бэкенд
- кэширующий сервер
- сжимает данные на лету (gzip, zstd)

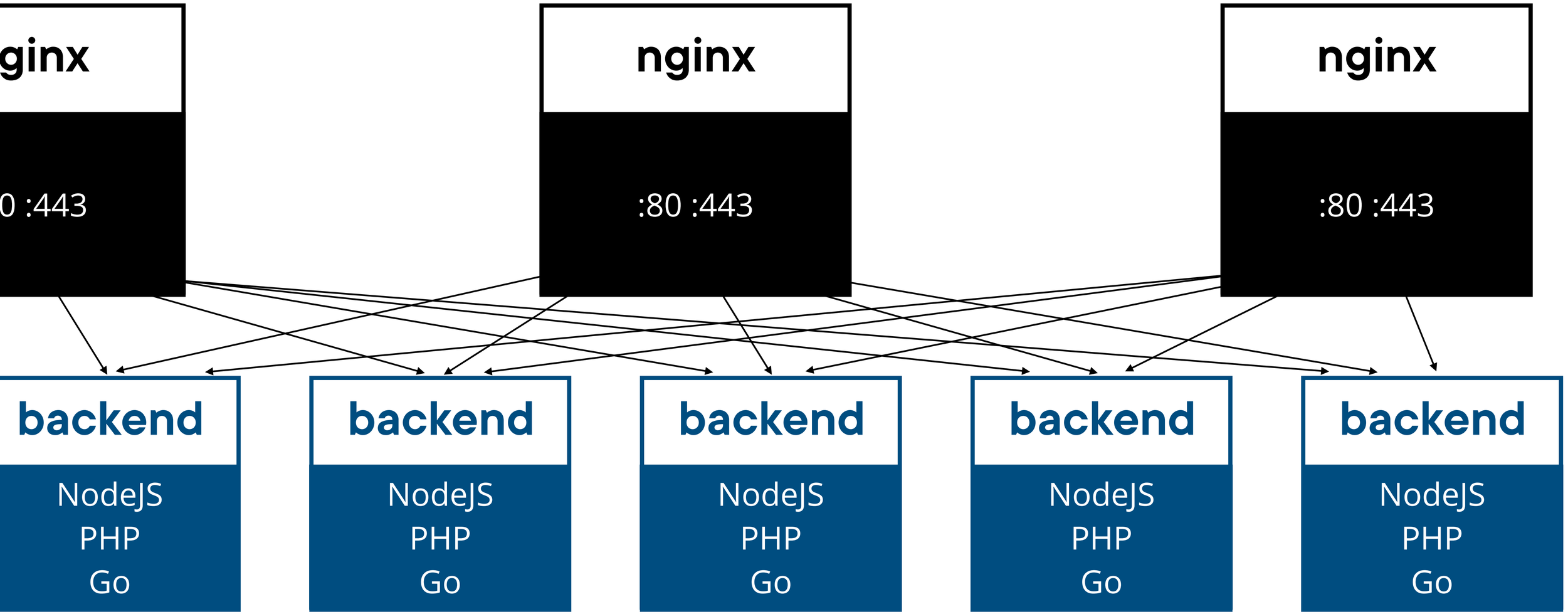
Чем занимается nginx

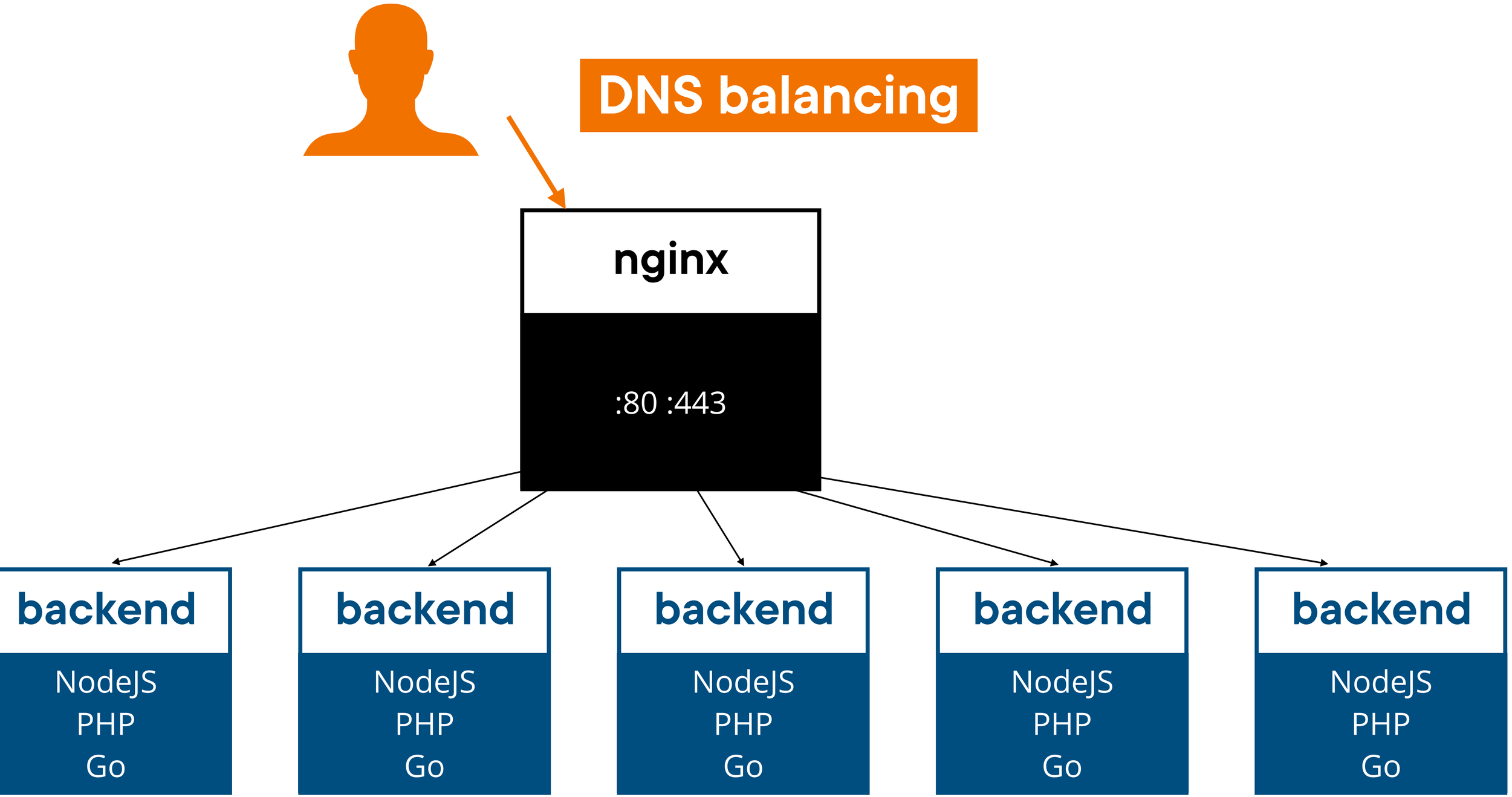
- парсит и патчит http-запросы
- терминирует https
- раздаёт статический контент
- проксирует на бэкенд
- кэширующий сервер
- сжимает данные на лету (gzip, zstd)
- защита от DDoS

Чем занимается nginx

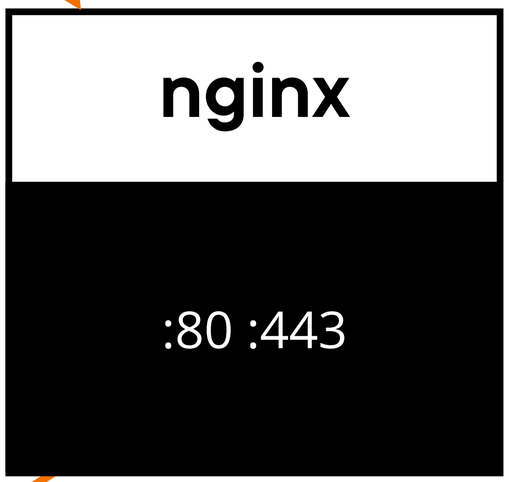
- парсит и патчит http-запросы
- терминирует https
- раздаёт статический контент
- проксирует на бэкенд
- кэширующий сервер
- сжимает данные на лету (gzip, zstd)
- защита от DDoS
- балансировка нагрузки

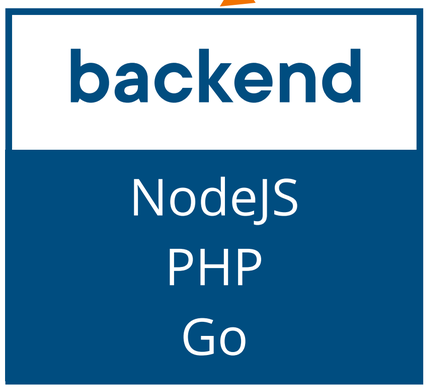
Балансировка нагрузки


nginx
0 :443
nginx
:80 :443
nginx
:80 :443
backend
NodeJS
PHP
Go
backend
NodeJS
PHP
Go
backend
NodeJS
PHP
Go
backend
NodeJS
PHP
Go
backend
NodeJS
PHP
Go


DNS balancing
nginx
:80 :443
backend
NodeJS
PHP
Go
backend
NodeJS
PHP
Go
backend
NodeJS
PHP
Go
backend
NodeJS
PHP
Go
backend
NodeJS
PHP
Go

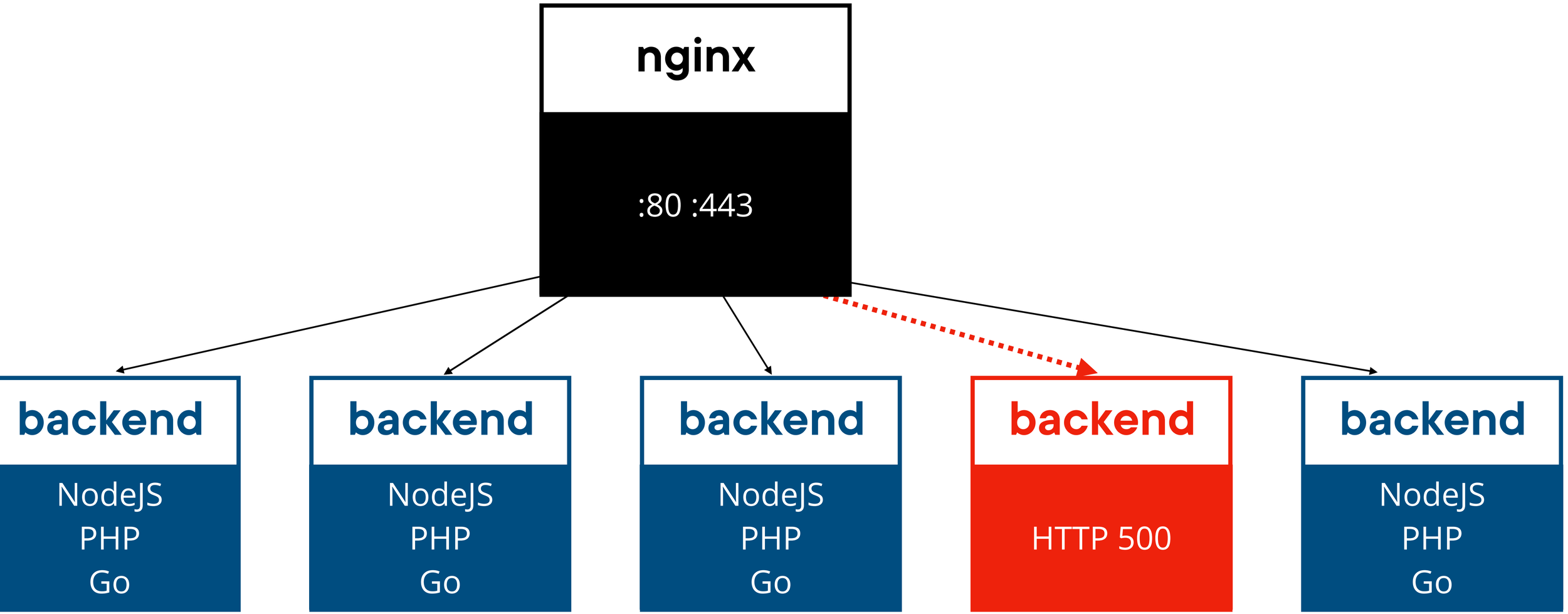
DNS balancing



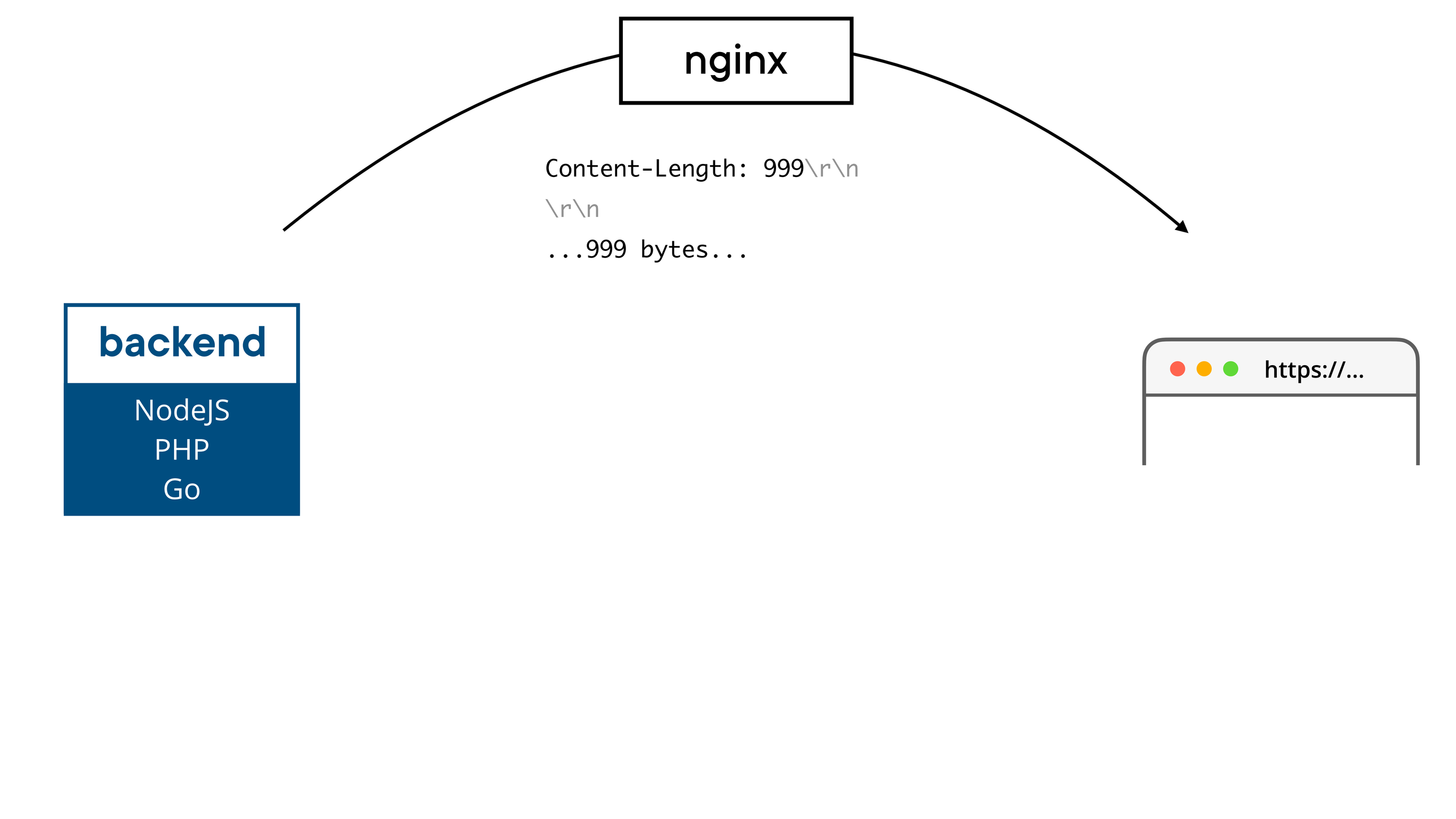


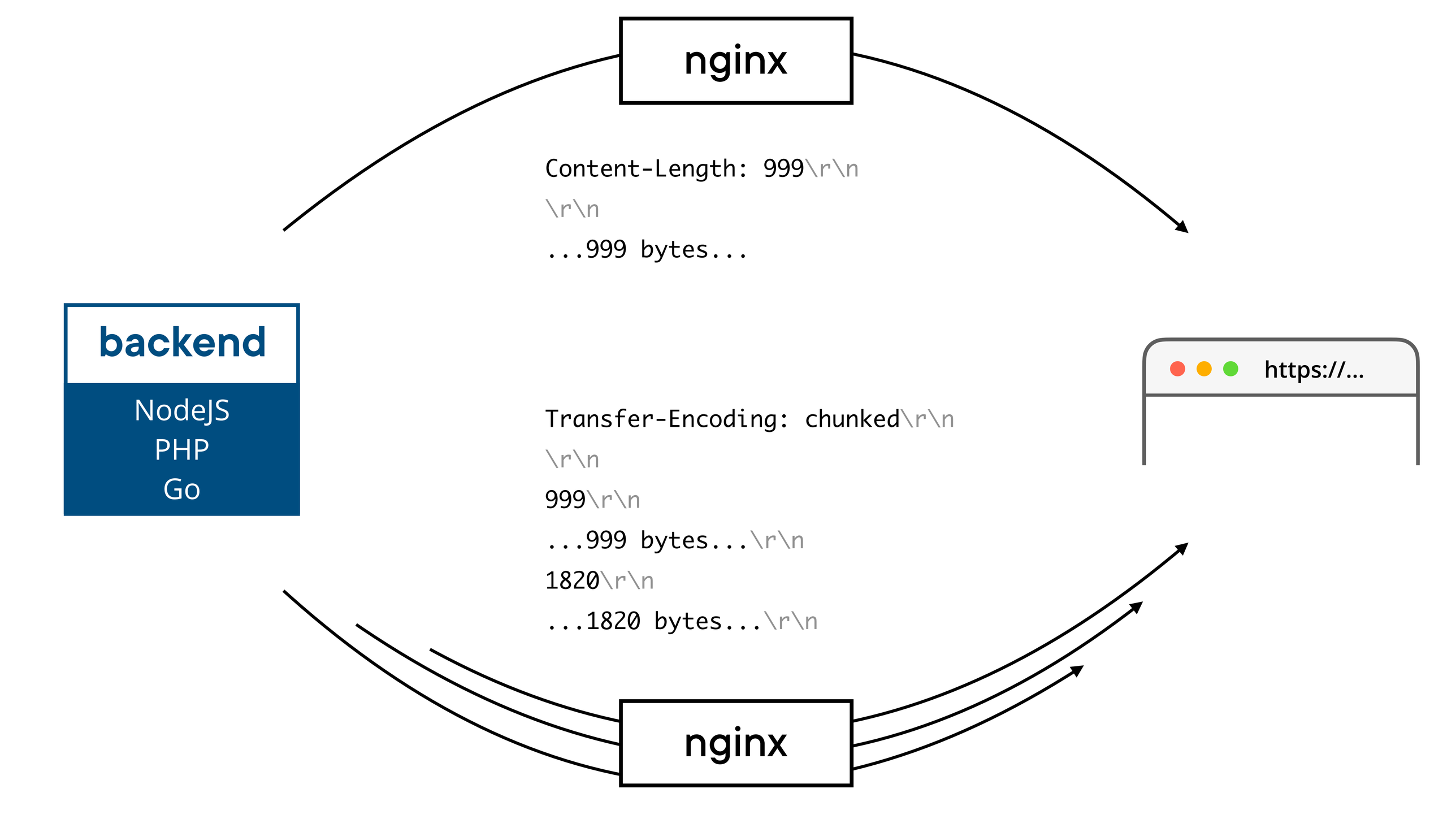
backend balancing

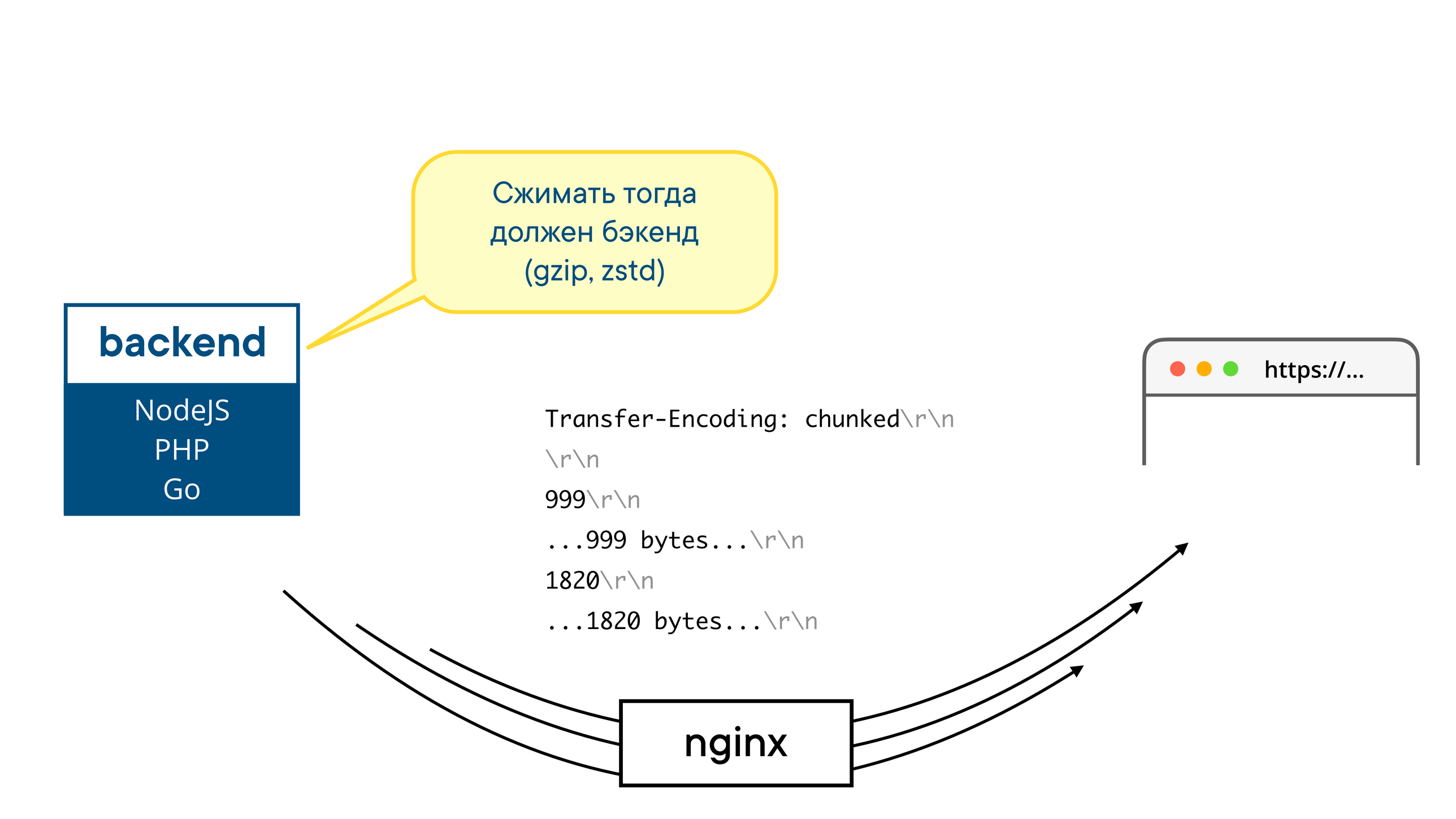
- фронты знают список бэкендов
- у которых есть веса
- простейшее: взвешенный рандом
- сложнее: от статы нагрузки


nginx
:80 :443
backend
NodeJS
PHP
Go
backend
NodeJS
PHP
Go
backend
NodeJS
PHP
Go
backend
HTTP 500
backend
NodeJS
PHP
Go

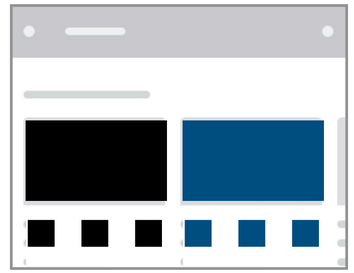
Отдача HTTP-ответа частями


nginx
Content-Length: 999\r\n
\r\n
...999 bytes...
backend
NodeJS
PHP
Go
https://...


nginx
Content-Length: 999\r\n
\r\n
...999 bytes...
backend
NodeJS
PHP
Go
https://...
Transfer-Encoding: chunked\r\n
\r\n
999\r\n
...999 bytes...\r\n
1820\r\n
...1820 bytes...\r\n
nginx

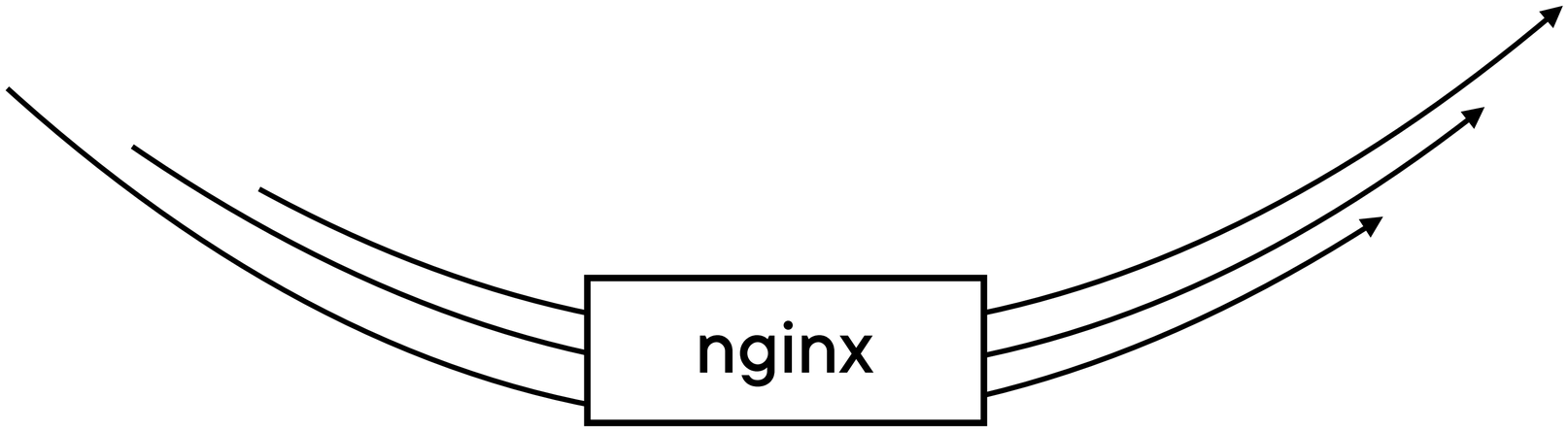

Сжимать тогда должен бэкенд (gzip, zstd)
backend
NodeJS
PHP
Go
Transfer-Encoding: chunked\r\n
\r\n
999\r\n
...999 bytes...\r\n
1820\r\n
...1820 bytes...\r\n
https://...
nginx

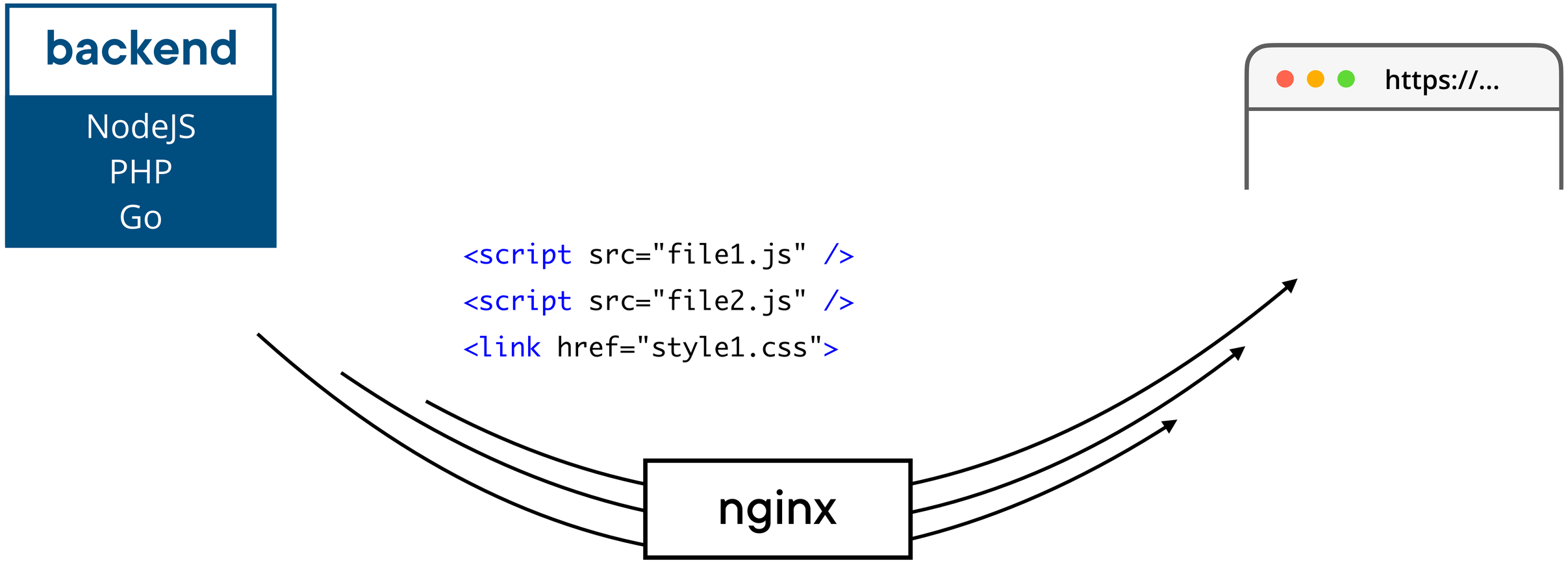
```
<body>
    ... skeleton ...
</body>
// -------- wait
<script>f({some: data})</script>
// -------- wait
<script>f({another: data})</script>
```

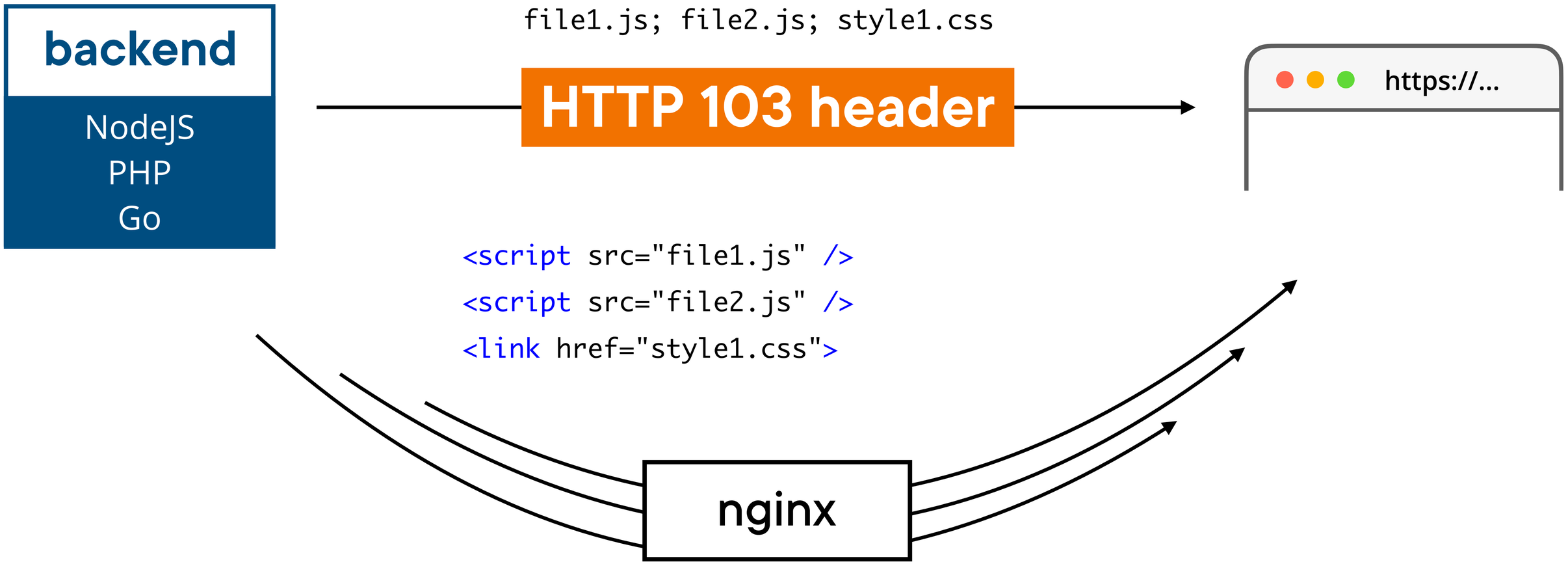


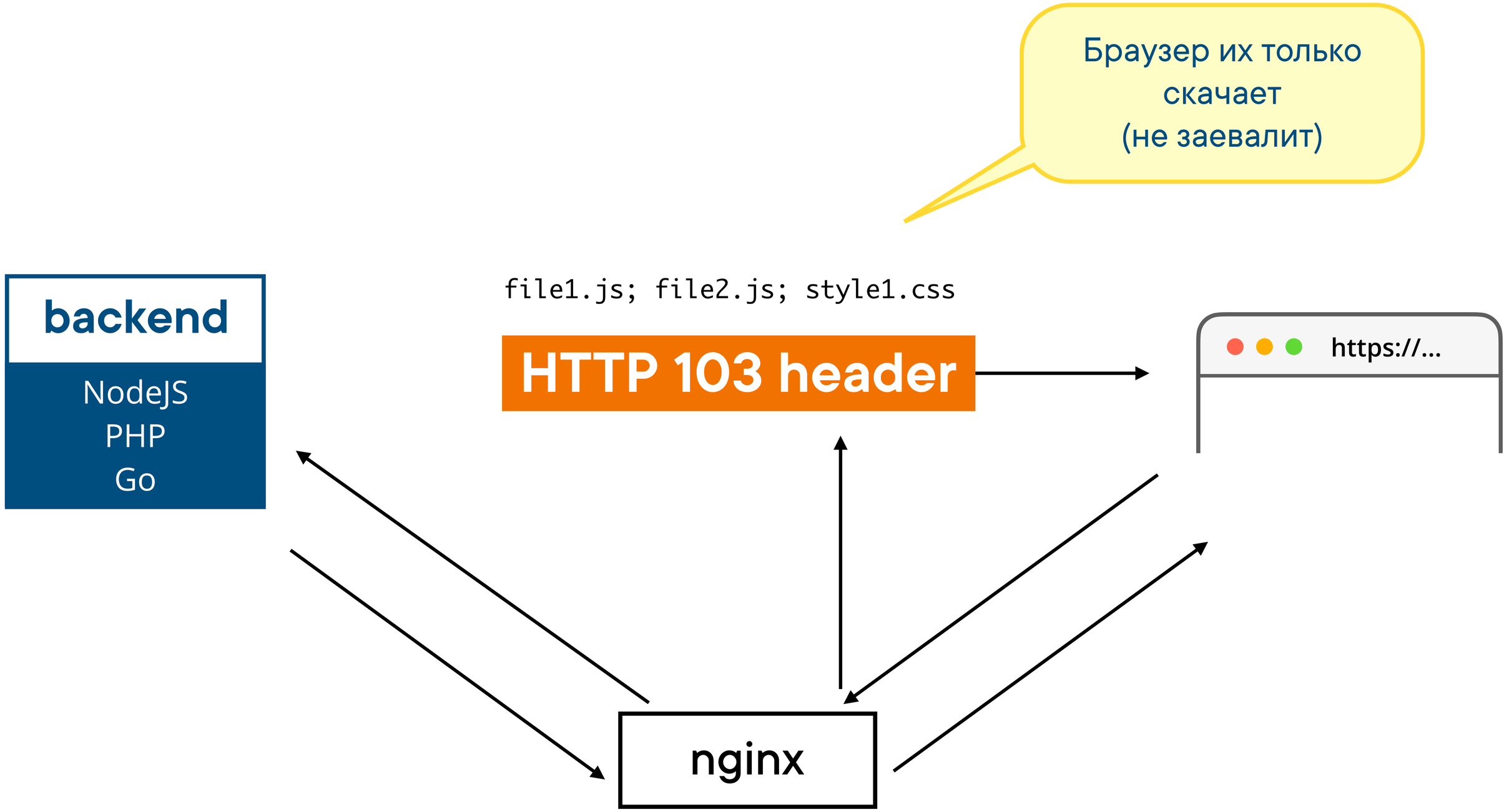
backend

NodeJS
PHP
Go




backend
NodeJS
PHP
Go
https://...
<script src="file1.js" />
<script src="file2.js" />
<link href="style1.css">
nginx


backend
NodeJS
PHP
Go
file1.js; file2.js; style1.css
HTTP 103 header
https://...
<script src="file1.js" />
<script src="file2.js" />
<link href="style1.css">
nginx


Браузер их только
скачает
(не заевалит)
backend
NodeJS
PHP
Go
file1.js; file2.js; style1.css
HTTP 103 header
https://...
nginx


frontend
Что мы узнали
backend

3/3

Язык, компилятор и веб-сервер ВКонтакте


PHP
МОНОЛИТ

25к .php

9 млн строк

25к .php

PHP

монолит

KPHP

9 млн строк

25к .php

9 млн строк

25к .php

200к .cpp

9 млн строк

1.5 GB бинарь

25к .php

200к .cpp

9 млн строк

1.5 GB бинарь

Чем C++ отличается от JS/PHP/... и других высокоуровневых

Возьмём JS-объект

```
let o = {
  f: 0.5,
  b: true,
  i: 10,
}

console.log(o);
// {f: 0.5, b: true, i: 10}
```

Возьмём JS-объект

```
let o = {
  f: 0.5,
  b: true,
  i: 10,
}

console.log(o);
// {f: 0.5, b: true, i: 10}
```

Что можно:

- читать и менять поля
  `o.f = 1.5`
- создавать новые поля
  `o.s = 's'`
- присваивать другие типы
  `o.b = []`
- обращаться по имени
  ```
  let n = 'i'
  o[n]  // 10
  ```

И сравним с C++ структуркой

```
struct MyStruct {
  float f;
  bool b;
  int i;
};

MyStruct o = {
  0.5,
  true,
  10,
};
```

И сравним с C++ структуркой

```
struct MyStruct {
  float f;
  bool b;
  int i;
};

MyStruct o = {
  0.5,
  true,
  10,
};
```

А что с ней:

- читать и менять можно
  `o.f = 1.5;`
- создавать новые поля нельзя
- присваивать другие типы нельзя
- обращаться по имени нельзя
- аналога console.log нет

И сравним с C++ структуркой

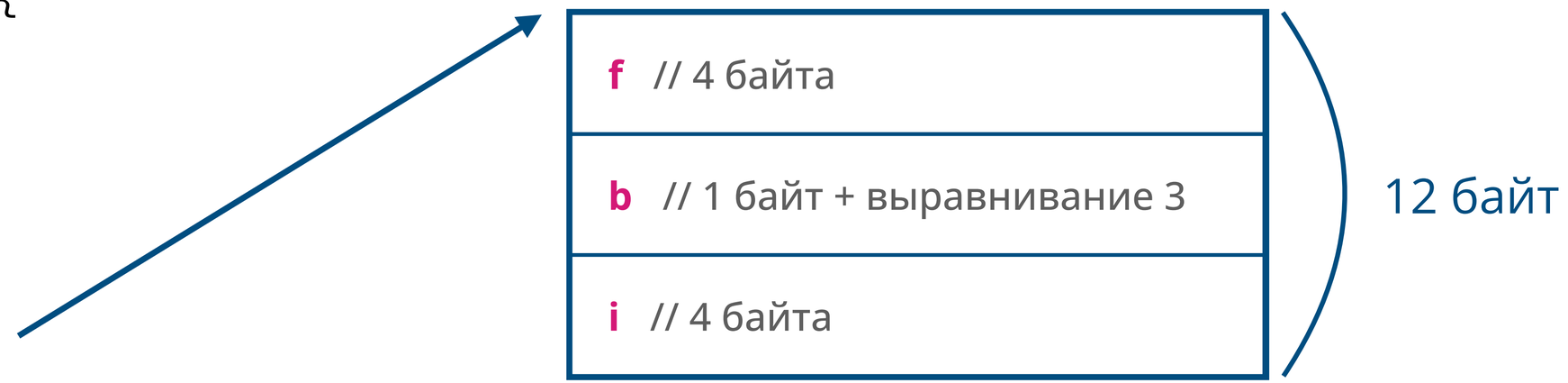
```
struct MyStruct {
  float f;
  bool b;
  int i;
};

MyStruct o = {
  0.5,
  true,
  10,
};
```

А что с ней:

- читать и менять можно
  `o.f = 1.5;`
- создавать новые поля нельзя
- присваивать другие типы нельзя
- обращаться по имени нельзя
- аналога console.log нет

Поля — это лишь смещения в памяти

```
struct MyStruct {
  float f;
  bool b;
  int i;
};

MyStruct o = {
  0.5,
  true,
  10,
};
```



`o.i` это "смещение переменной `o` + 8 байт"

Так, можно изменить поле без обращения

```
MyStruct o = {
  0.5,
  true,
  10,
};

*(int *)((char *)&o + 8) = 999;     // 8 байт от адреса `o`

printf("%d", o.i);                  // 999
```

Или одну строку через другую

```
struct HasStrings {
  char s1[16] = "hello";
  char s2[16] = "world";
};

HasStrings h;

h.s1[17] = '!';      // потому что s1[N] это *(s1+N)

printf("%s", h.s2);  // w!rld
```

Или вообще не в свою память

```
h.s1[100500] = '!';
```

Или вообще не в свою память

```
h.s1[100500] = '!';
```

```
Program terminated with signal 11, Segmentation fault.
#0  0x88207fc2 in memcpy () from /usr/lib/libc.so.6
(gdb) bt
#0  0x88207fc2 in memcpy () from /usr/lib/libc.so.6
#1  0x88205eb6 in __sfvwrite () from /usr/lib/libc.so.6
#2  0x881fbc95 in strchr () from /usr/lib/libc.so.6
#3  0xbfbe6c14 in ?? ()
#4  0xbfbe69d8 in ?? ()
#5  0x881ed91e in localeconv () from /usr/lib/libc.so.6
#6  0x881fec05 in __vfprintf () from /usr/lib/libc.so.6
#7  0x881f7d80 in snprintf () from /usr/lib/libc.so.6
#8  0x08052b64 in my_function (files=0xbfbed710, filename=<value
#9  0x08053bfb in main (argc=4, argv=0xbfbedd90) at myfile.c:225
```

На практике: fast inverse square root

$$\frac{1}{\sqrt{x}}$$

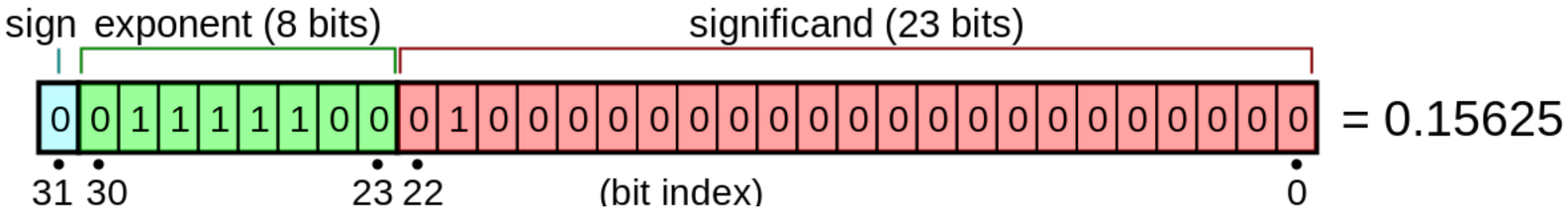
```
float Q_rsqrt(float number) {
  long i;
  float x2, y;

  x2 = number * 0.5F;
  y = number;
  i = *(long *)&y;                  // wtf?
  i = 0x5f3759df - (i >> 1);        // wtf???
  y = *(float *)&i;
  y = y * (1.5F - (x2 * y * y));

  return y;
}
```

На практике: fast inverse square root

$$\frac{1}{\sqrt{x}}$$

```
float Q_rsqrt(float number) {
  long i;
  float x2, y;

  x2 = number * 0.5F;
  y = number;
  i = *(long *)&y;                  // wtf?
  i = 0x5f3759df - (i >> 1);        // wtf???
  y = *(float *)&i;
  y = y * (1.5F - (x2 * y * y));

  return y;
}
```

0.1 + 0.2 != 0.3 ***(IEEE 754)***



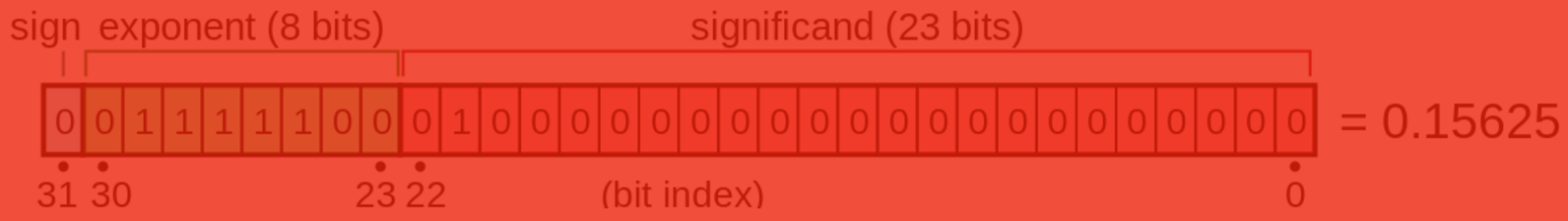
На практике: fast inverse square root

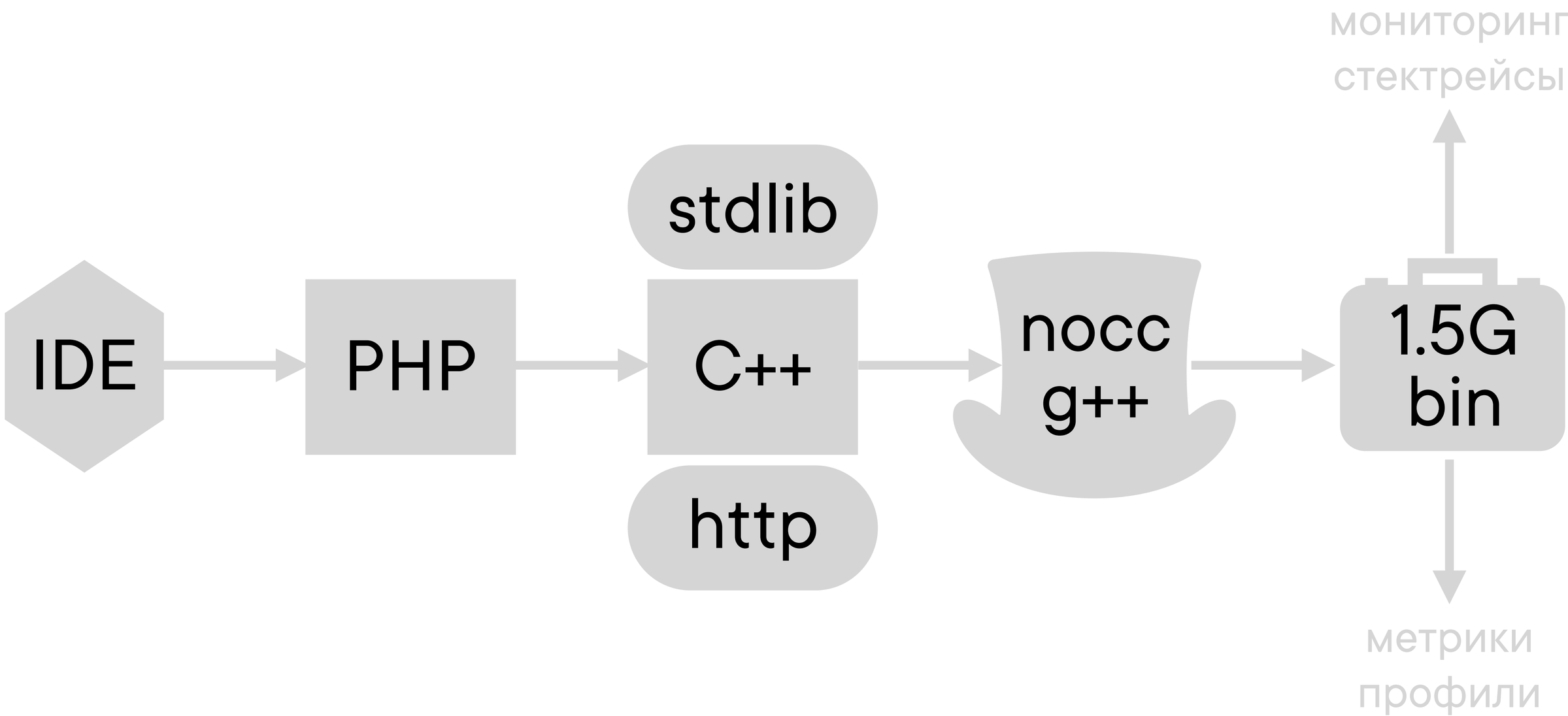
$$\frac{1}{\sqrt{x}}$$

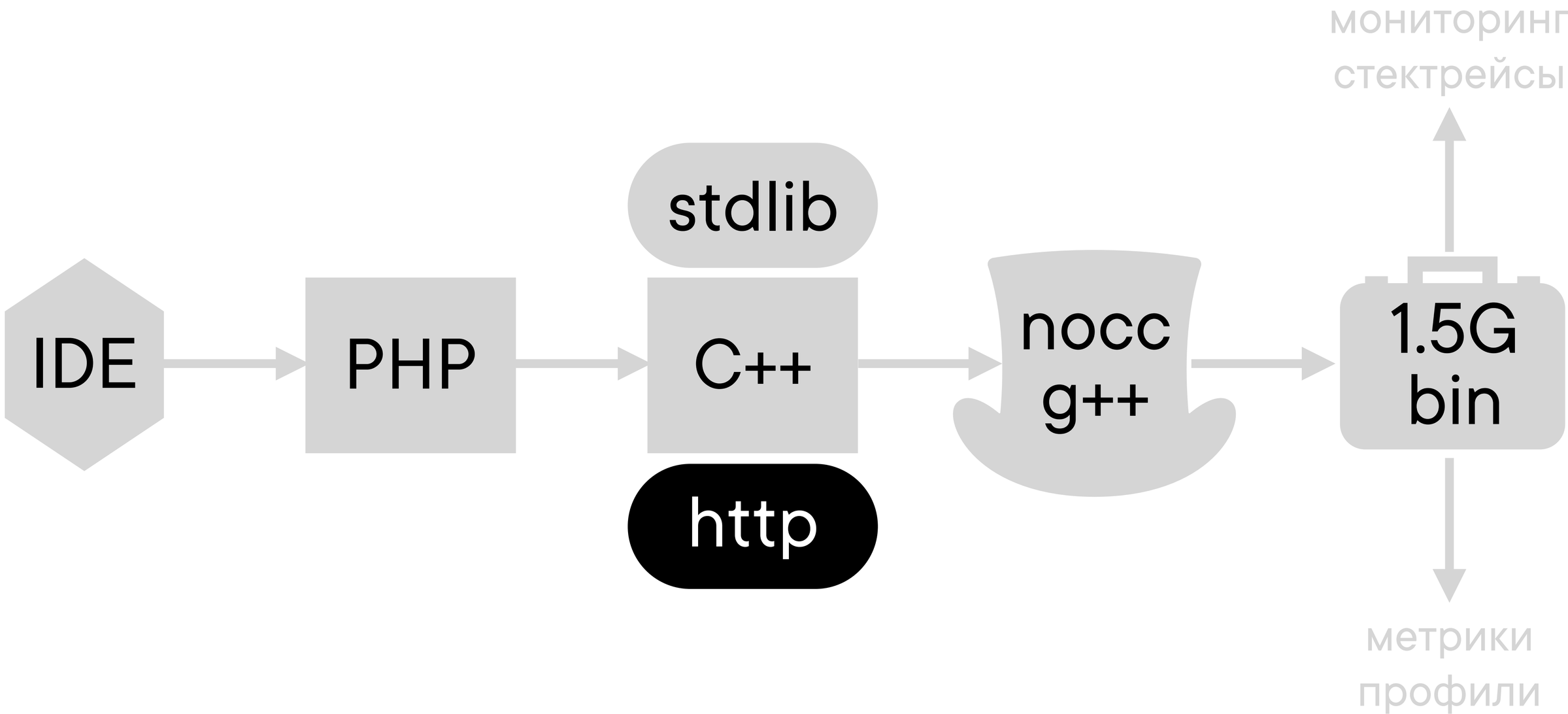
```
float Q_rsqrt(float number) {
  long i;
  float x2, y;

  x2 = number * 0.5F;
  y  = number;
  i  = *(long *) &y;                    // wtf?
  i = 0x5f3759df - (i >> 1);            // wtf???
  y = *(float *)&i;
  y = y * (1.5F - (x2 * y * y));

  return y;
}
```

ПОЧЕМУ

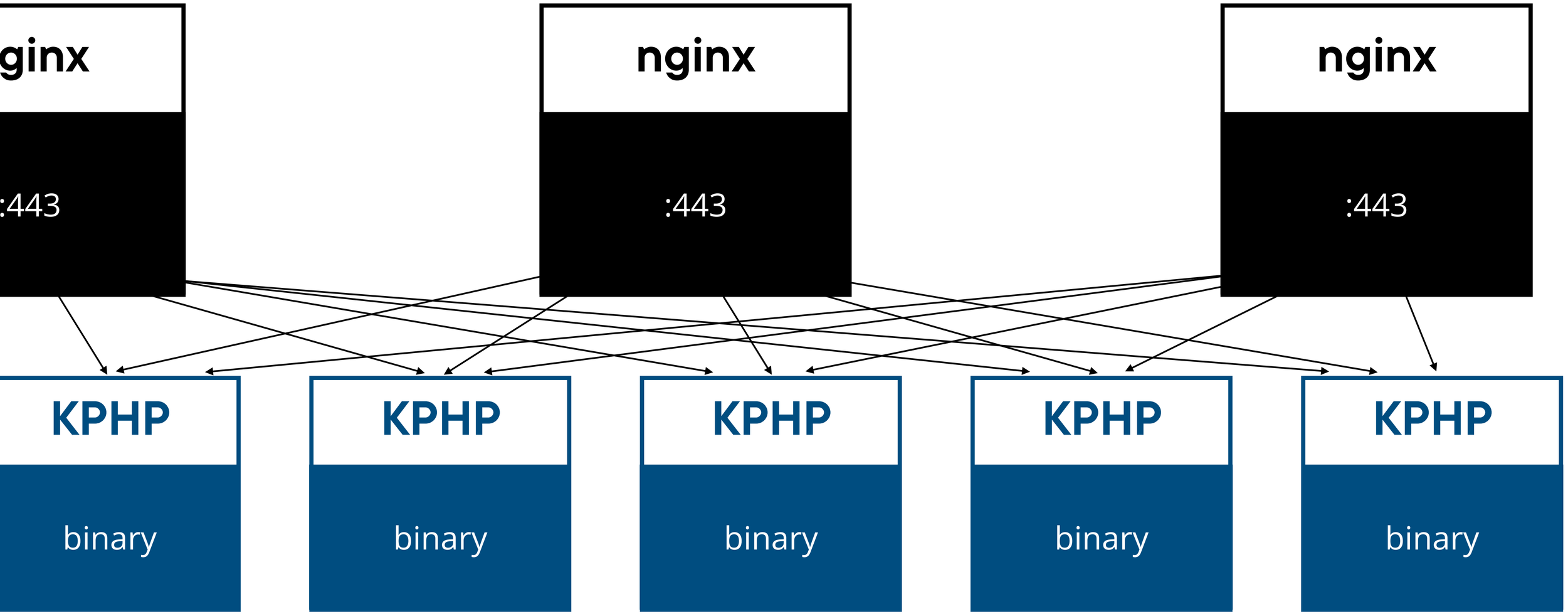
0.1 + 0.2 != 0.3 *(IEEE 754)*




мониторинг
стектрейсы
stdlib
IDE
PHP
C++
nocc
g++
1.5G
bin
http
метрики
профили


мониторинг
стектрейсы
stdlib
IDE
PHP
C++
nocc
g++
1.5G
bin
http
метрики
профили

их называют "фронты"



Как устроен KPHP HTTP server

- prefork-модель, как и в nginx

  мастер инициализирует окружение, открывает порт, форкается, воркеры слушают коннекты через epoll

Как устроен KPHP HTTP server

- prefork-модель, как и в nginx

  мастер инициализирует окружение, открывает порт, форкается, воркеры слушают коннекты через epoll

- что вообще такое fork

  копируется полное состояние процесса, открытые дескрипторы и страницы памяти на чтение

Как устроен KPHP HTTP server

- prefork-модель, как и в nginx

  мастер инициализирует окружение, открывает порт, форкается, воркеры слушают коннекты через epoll

- что вообще такое fork

  копируется полное состояние процесса, открытые дескрипторы и страницы памяти на чтение

- каждый воркер обрабатывает запрос

  там исполняется бэкенд-логика, то есть "скомпилированный PHP"

Как устроен KPHP HTTP server

- prefork-модель, как и в nginx

  мастер инициализирует окружение, открывает порт, форкается, воркеры слушают коннекты через epoll

- что вообще такое fork

  копируется полное состояние процесса, открытые дескрипторы и страницы памяти на чтение

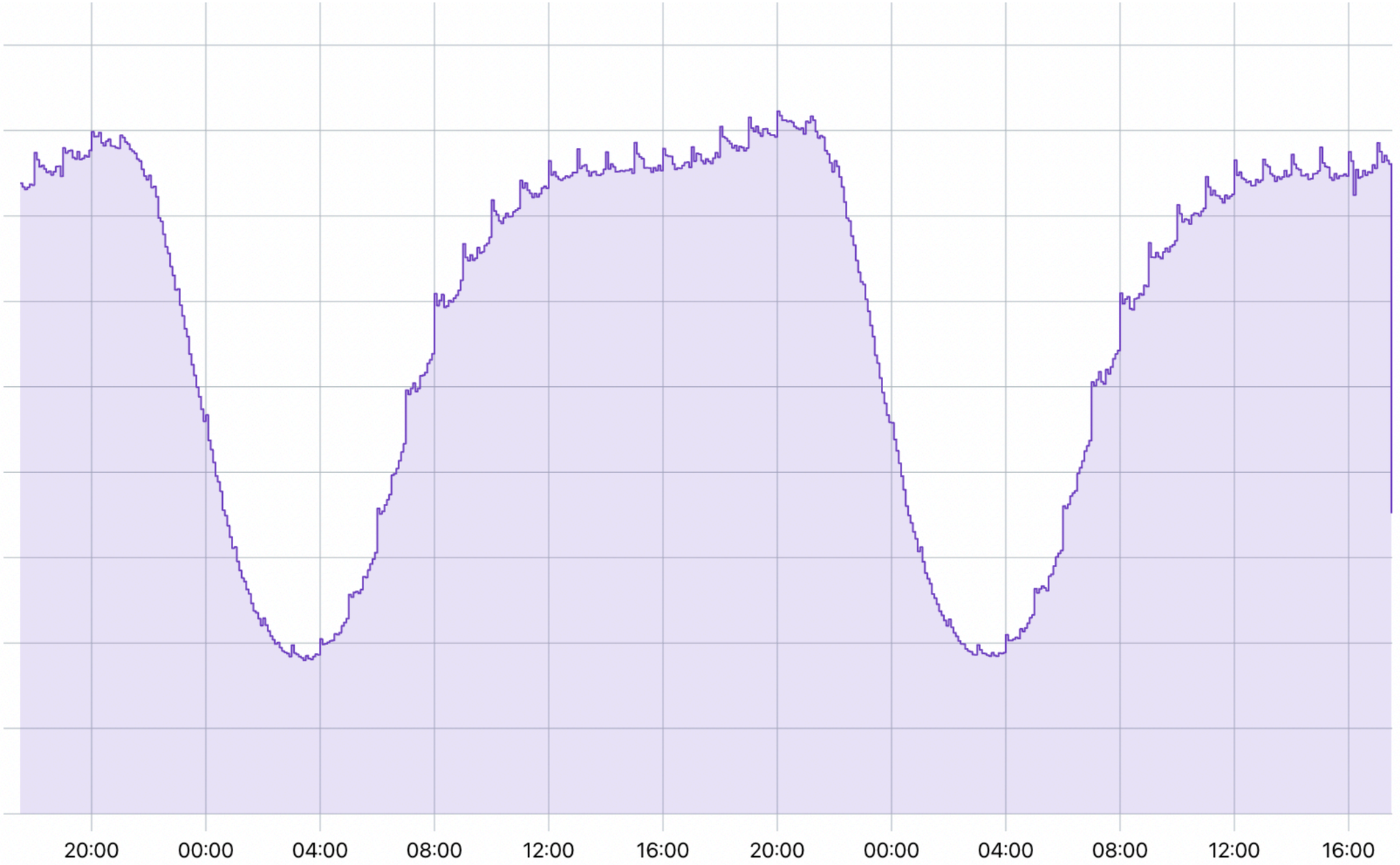
- каждый воркер обрабатывает запрос

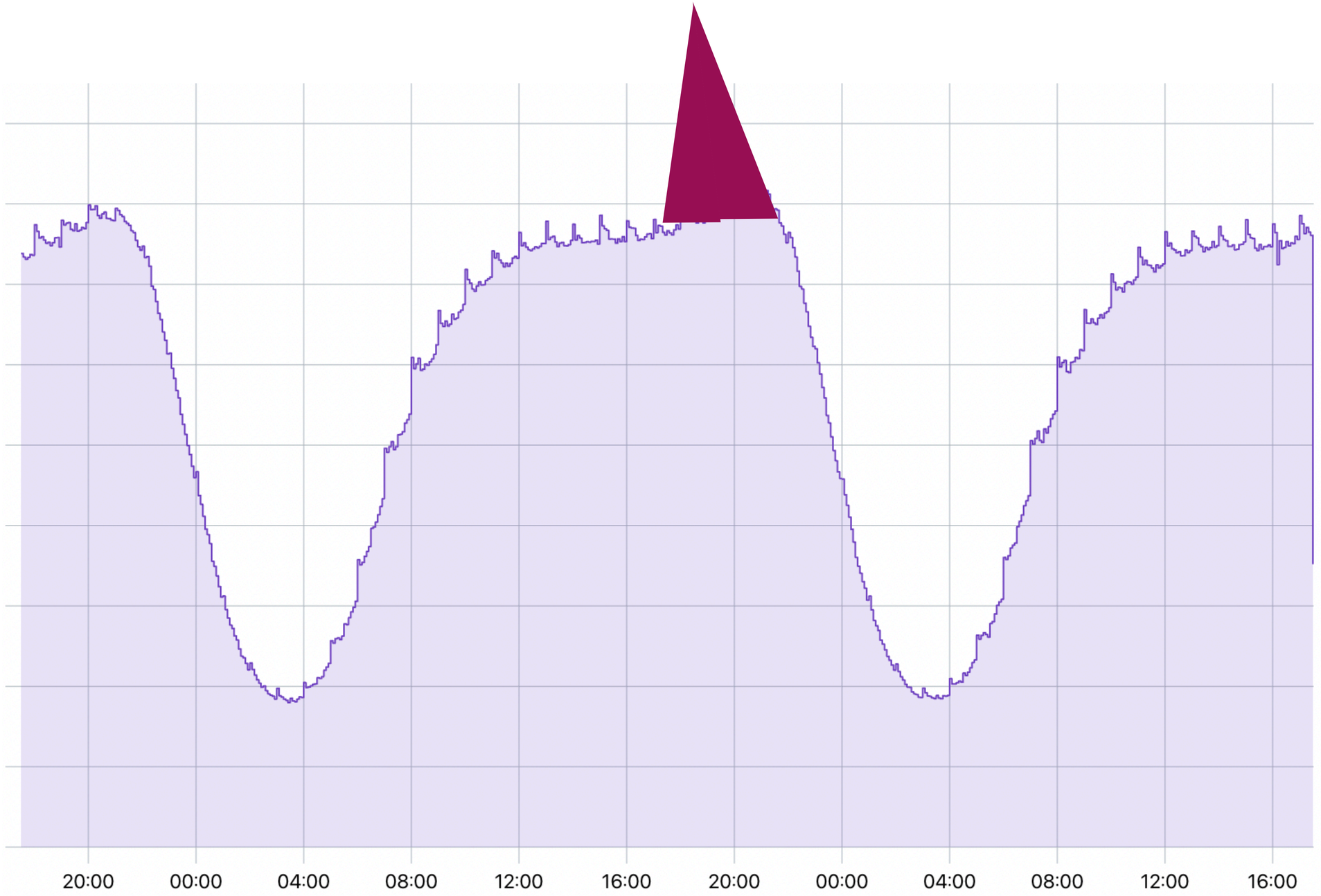
  там исполняется бэкенд-логика, то есть "скомпилированный PHP"

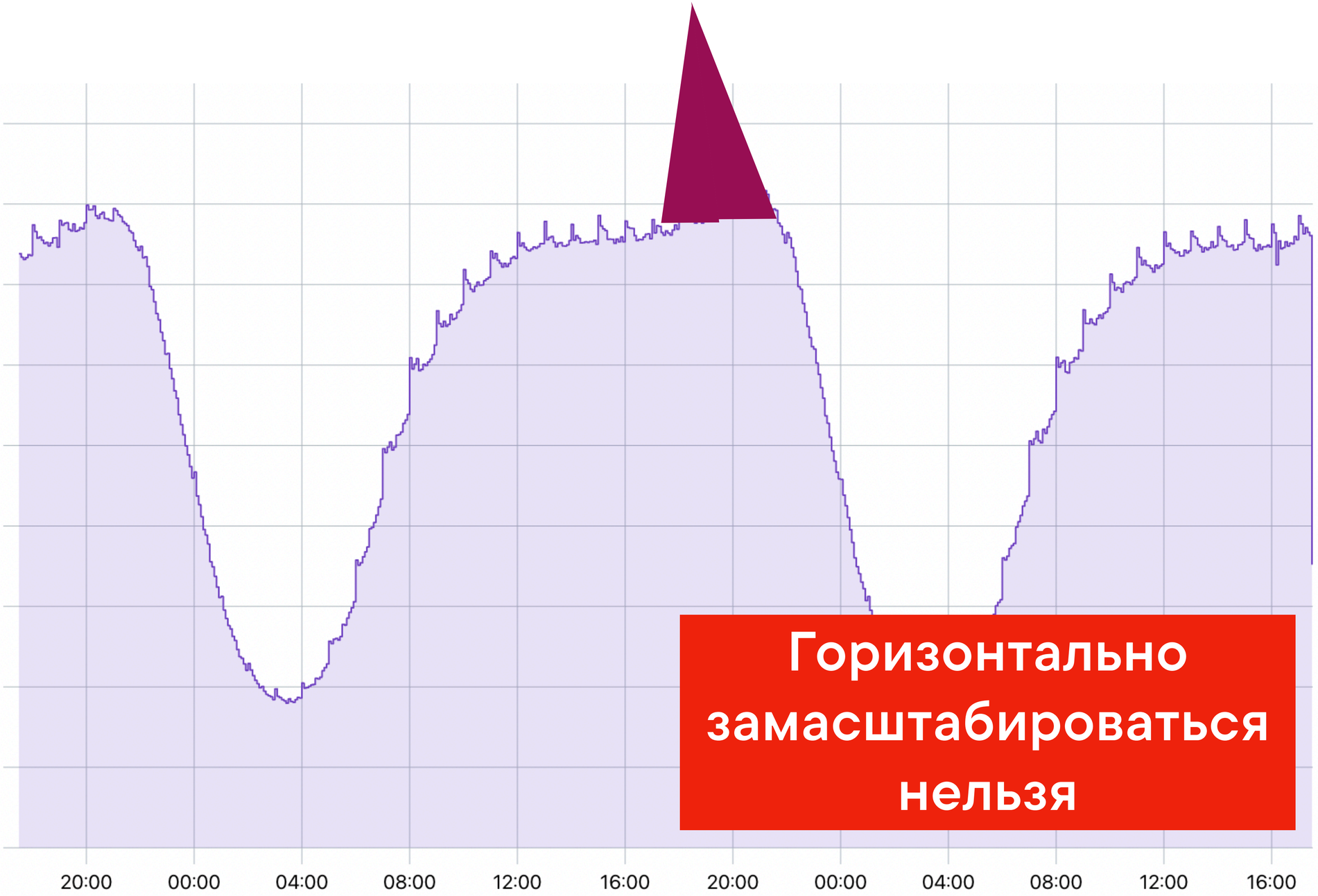
- однопоточность + epoll-очередь

  каждый воркер принципиально однопоточен, а их количество больше чем CPU-ядер

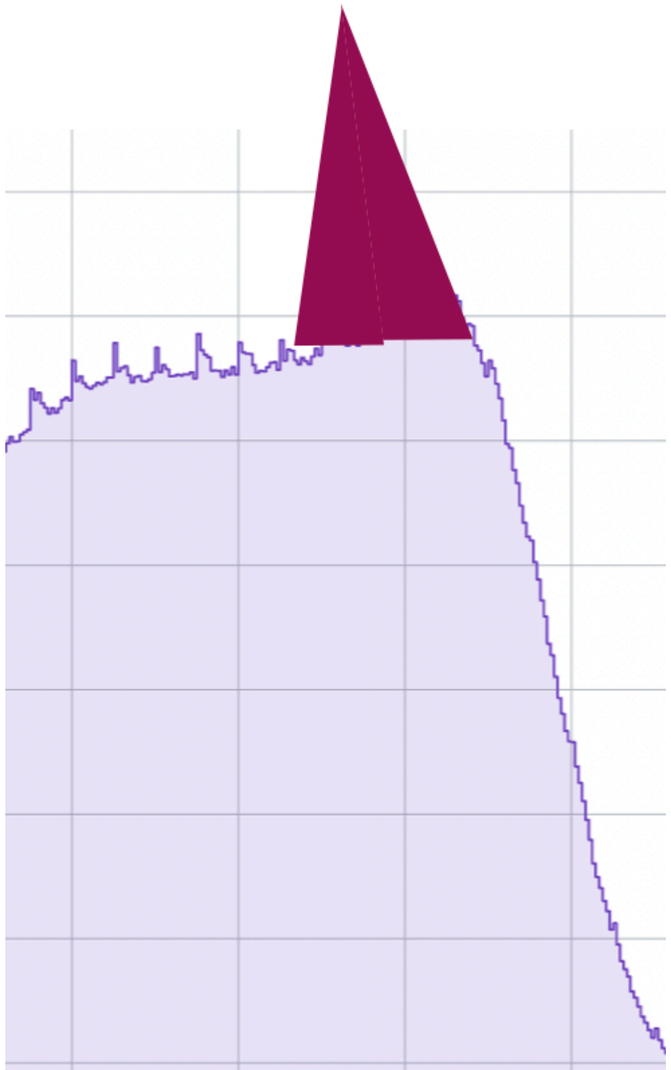
Деградация при пиковых нагрузках


20:00
00:00
04:00
08:00
12:00
16:00
20:00
00:00
04:00
08:00
12:00
16:00

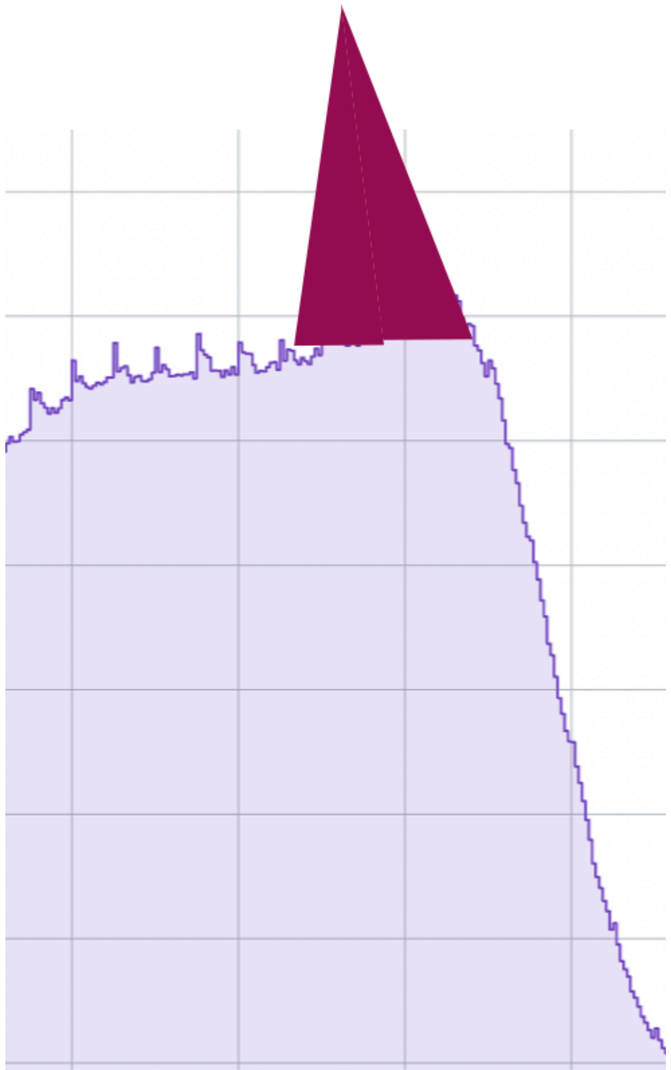

20:00
00:00
04:00
08:00
12:00
16:00
20:00
00:00
04:00
08:00
12:00
16:00


20:00
00:00
04:00
08:00
12:00
16:00
20:00
00:00
04:00
08:00
12:00
16:00
Горизонтально замасштабироваться нельзя

Направления борьбы с деградацией



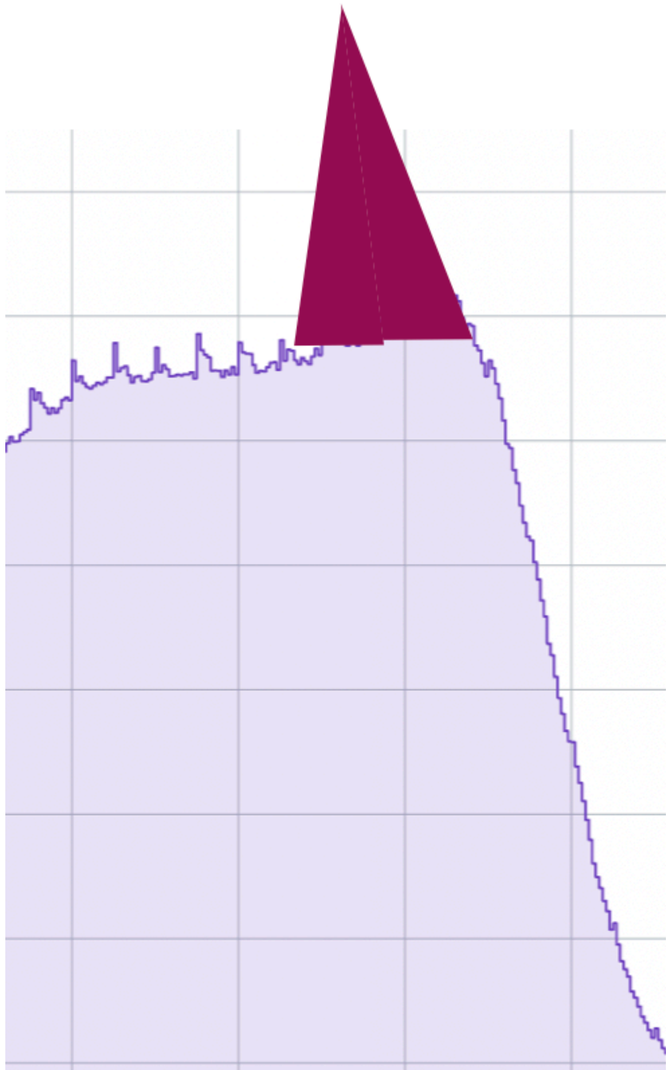
Направления борьбы с деградацией



- тюнинг системного шедулера

  чем больше процессов, тем больше можем в параллель, НО становится хуже шедулеру

Направления борьбы с деградацией

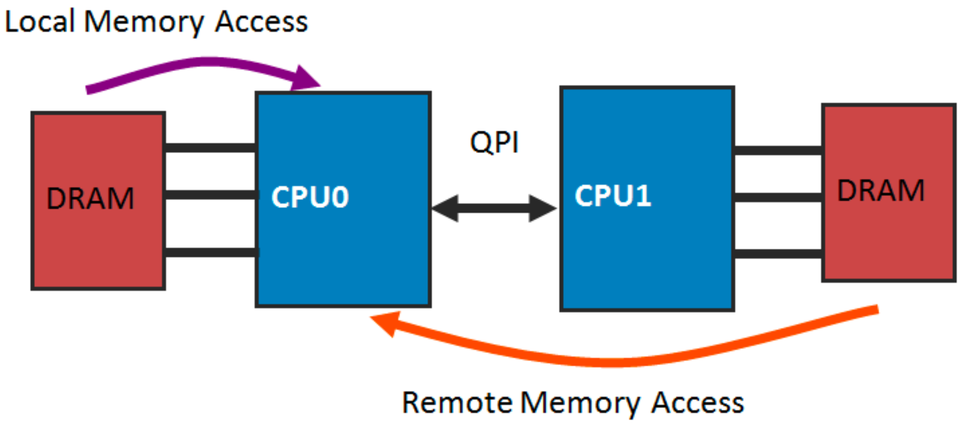


- тюнинг системного шедулера

  чем больше процессов, тем больше можем в параллель, НО становится хуже шедулеру

- NUMA nodes и CPU affinity

  физический доступ к памяти зависит от расположения процессора на плате



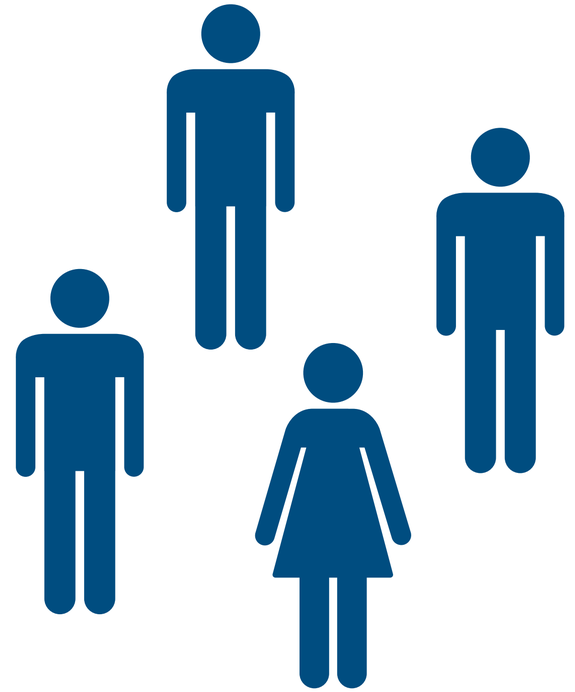
PHP

Что мы узнали

C++

Заключение

Ещё рано.
Я не закончил.

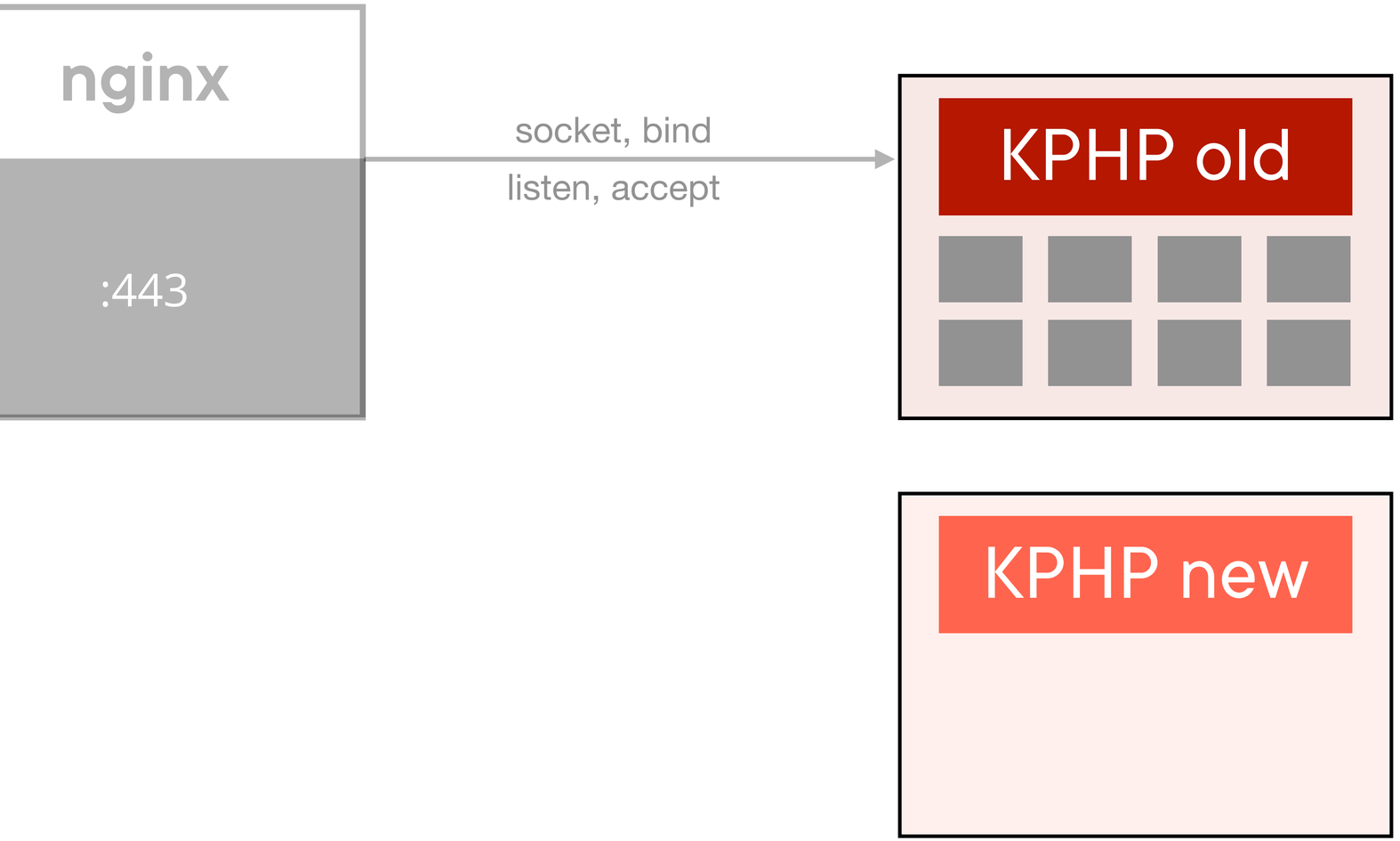


Я

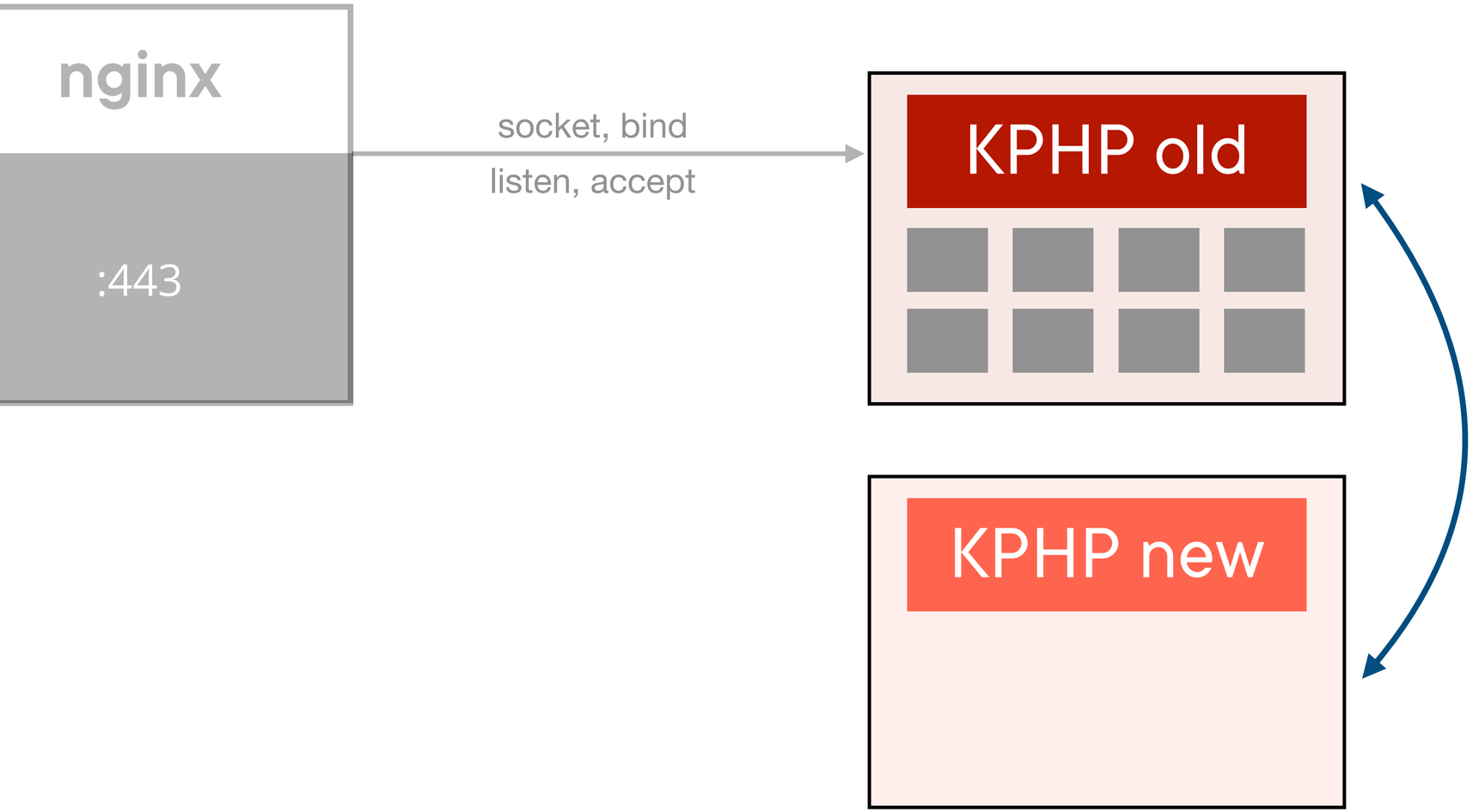
фронтендеры

Graceful restart

Graceful restart



Graceful restart

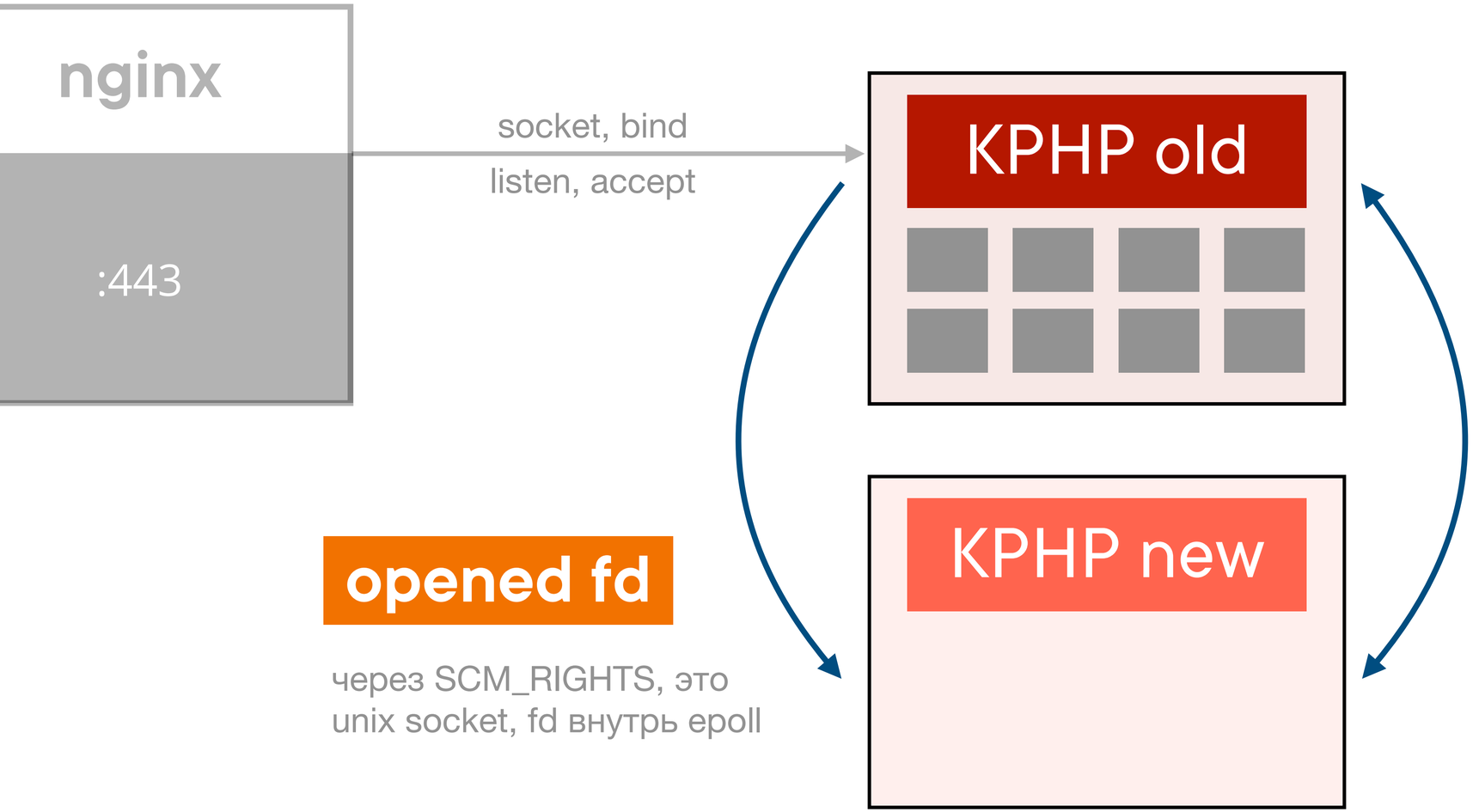


shm_open() маппит в
адресное пространство процесса

/tmpfs/kphp_shm

```
struct ShmStaticLayout {
  std::mutex mutex;

  int state;
  int n_old_workers_alive;
  ...
};
```

Graceful restart



opened fd

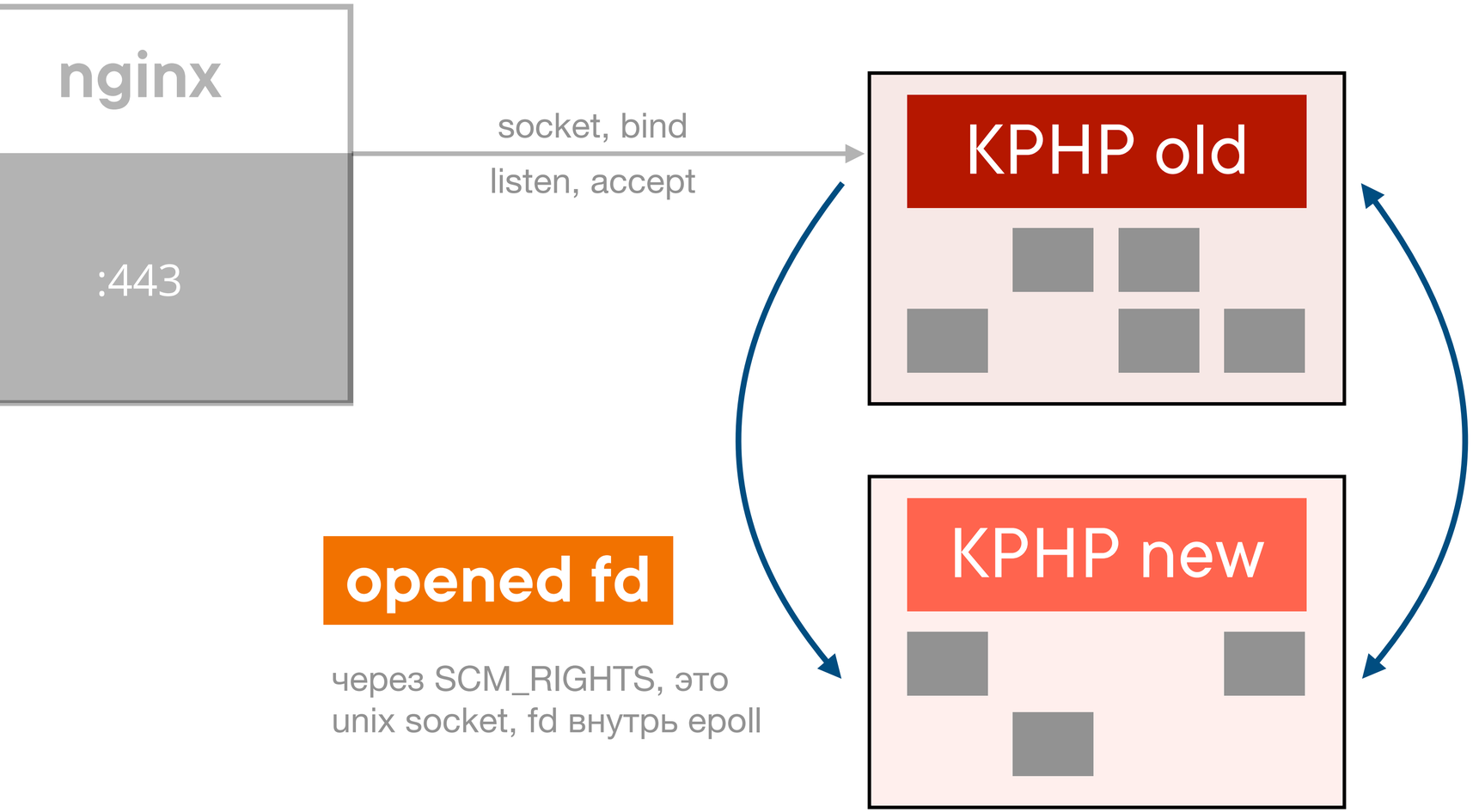
через SCM_RIGHTS, это
unix socket, fd внутрь epoll

shm_open() маппит в
адресное пространство процесса

/tmpfs/kphp_shm

```
struct ShmStaticLayout {
  std::mutex mutex;

  int state;
  int n_old_workers_alive;
  ...
};
```

Graceful restart



opened fd

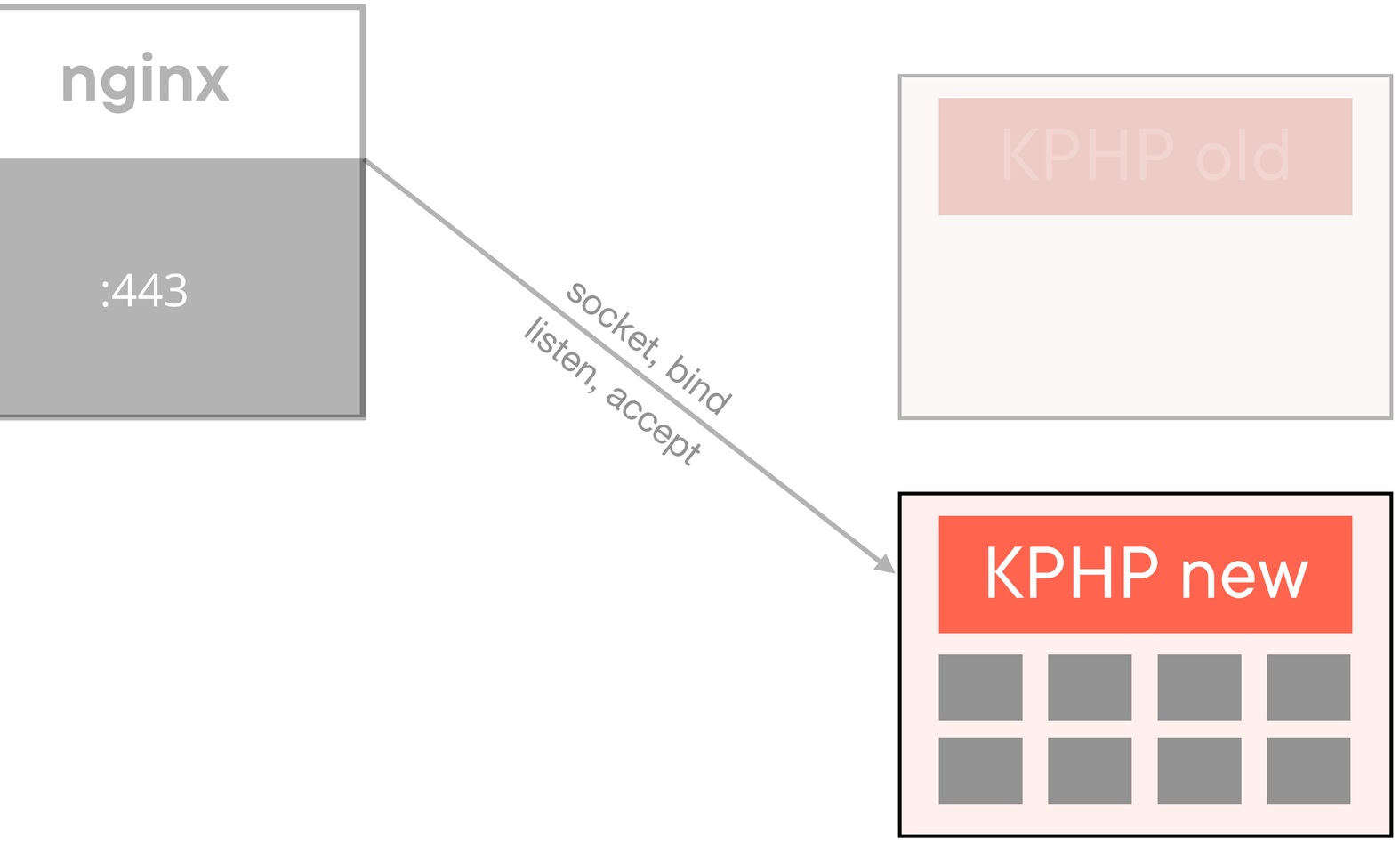
через SCM_RIGHTS, это
unix socket, fd внутрь epoll

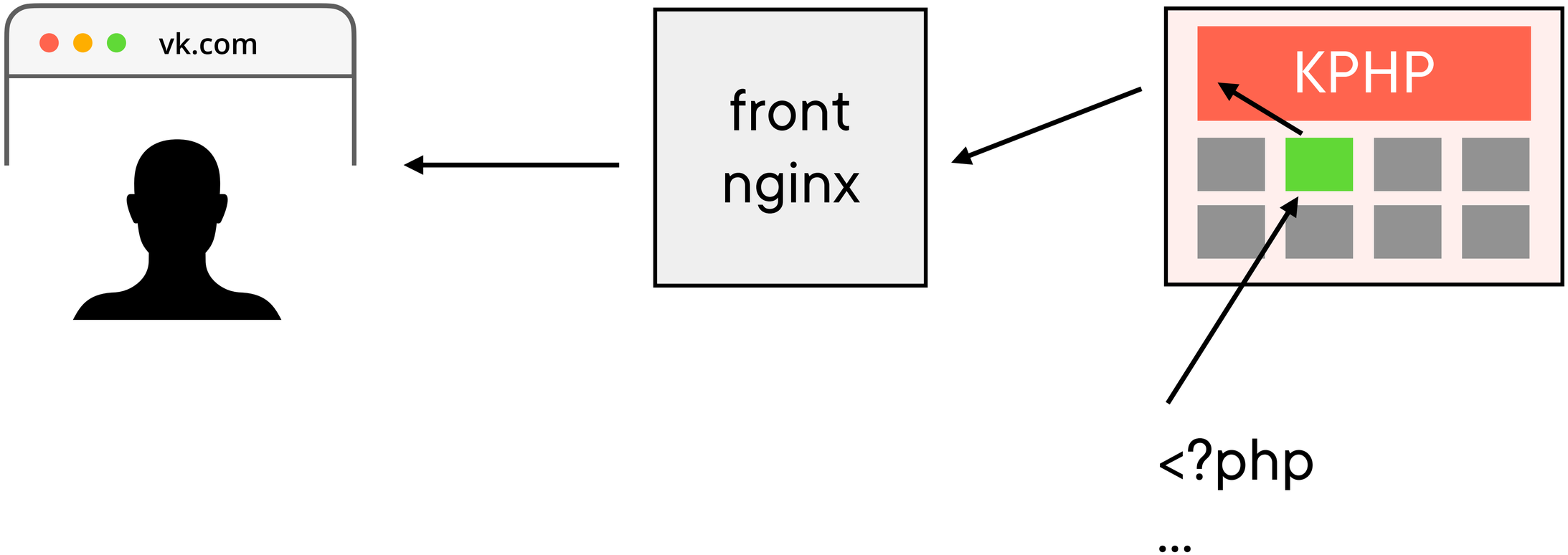
shm_open() маппит в
адресное пространство процесса

/tmpfs/kphp_shm

```
struct ShmStaticLayout {
  std::mutex mutex;

  int state;
  int n_old_workers_alive;
  ...
};
```

Graceful restart




vk.com
front
nginx
KPHP
<?php
...

Вот теперь
почти всё

HolyJS, 11–12 ноября 2023

Пр енд с то ия плюсовика яторщика

Александр Кирсанов, ВКонтакте