marketgoo ❤️ C++

José Miguel Pérez — Director Técnico

@twoixter @mktgoo

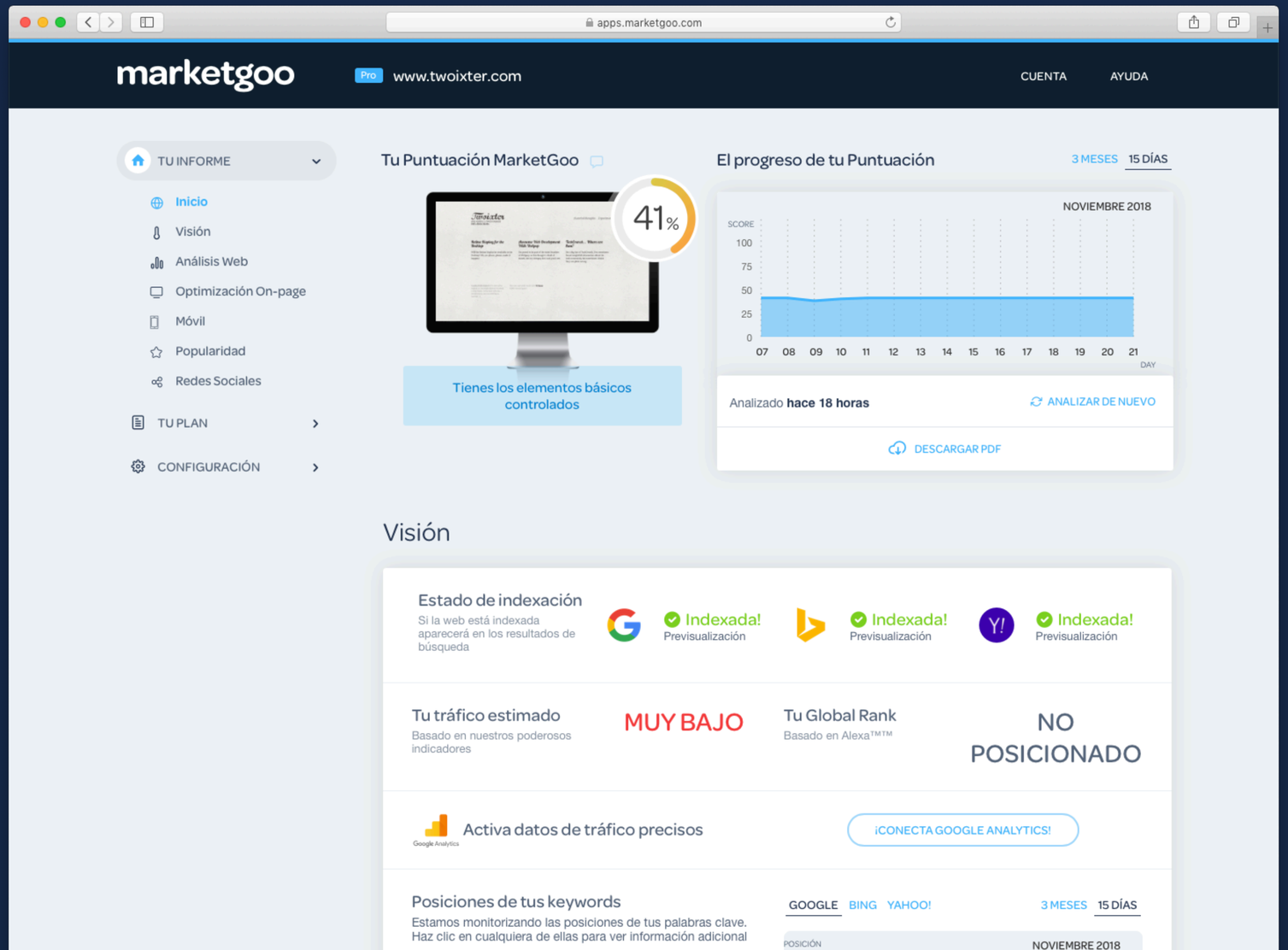
Introducción a MarketGoo

marketgoo

Introducción a MarketGoo

❤️ C++

- Aplicación web para automatizar SEO de una página web
- Análisis diario de una página web (Crawling)
- Medición de resultados (Rankings, enlaces externos, etc)

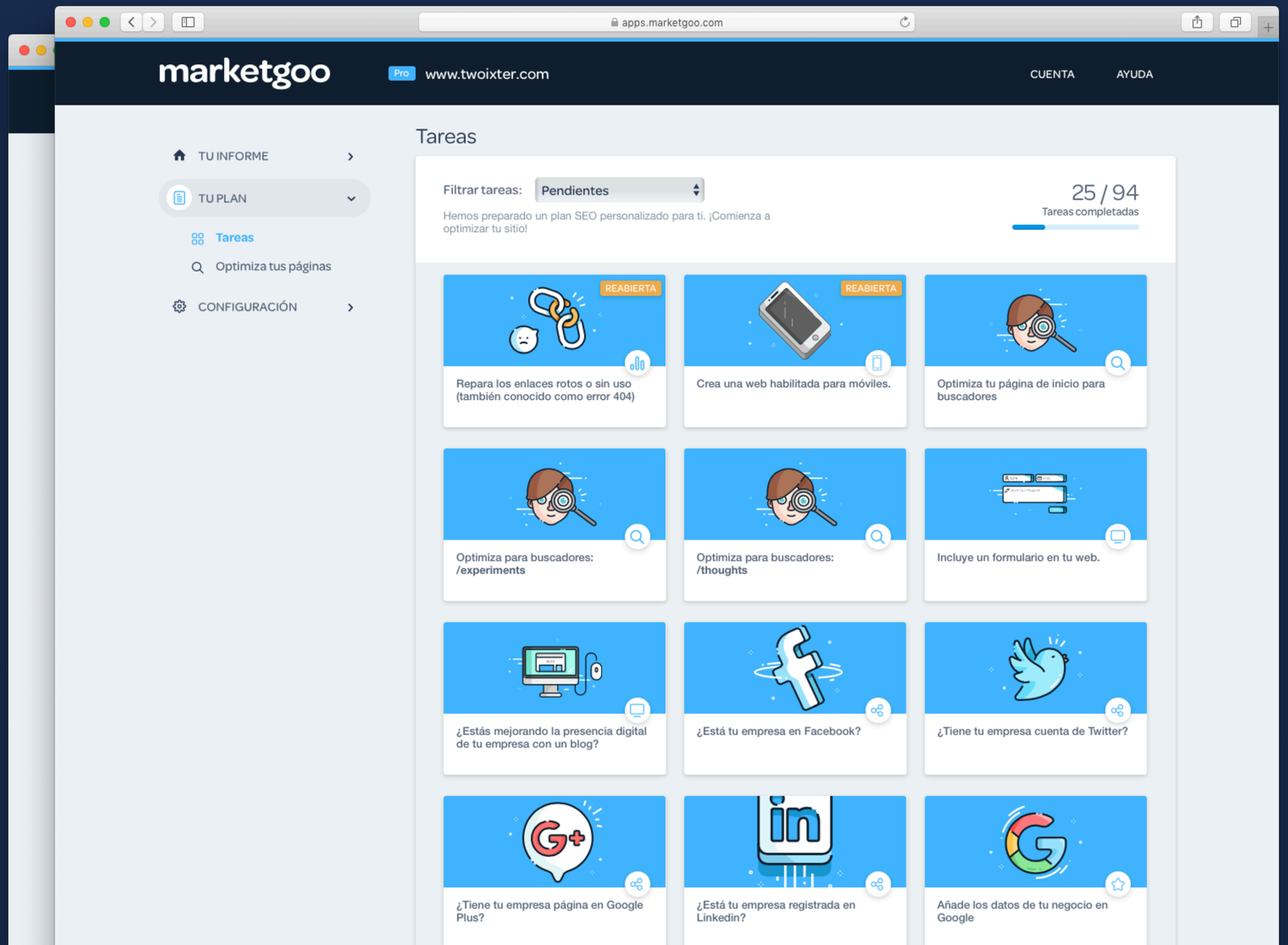


marketgoo

Introducción a MarketGoo

❤ C++

- Aplicación web para automatizar SEO de una página web
- Análisis diario de una página web (Crawling)
- Medición de resultados (Rankings, enlaces externos, etc)
- Generación de sugerencias de mejora (SEO, Social, etc)

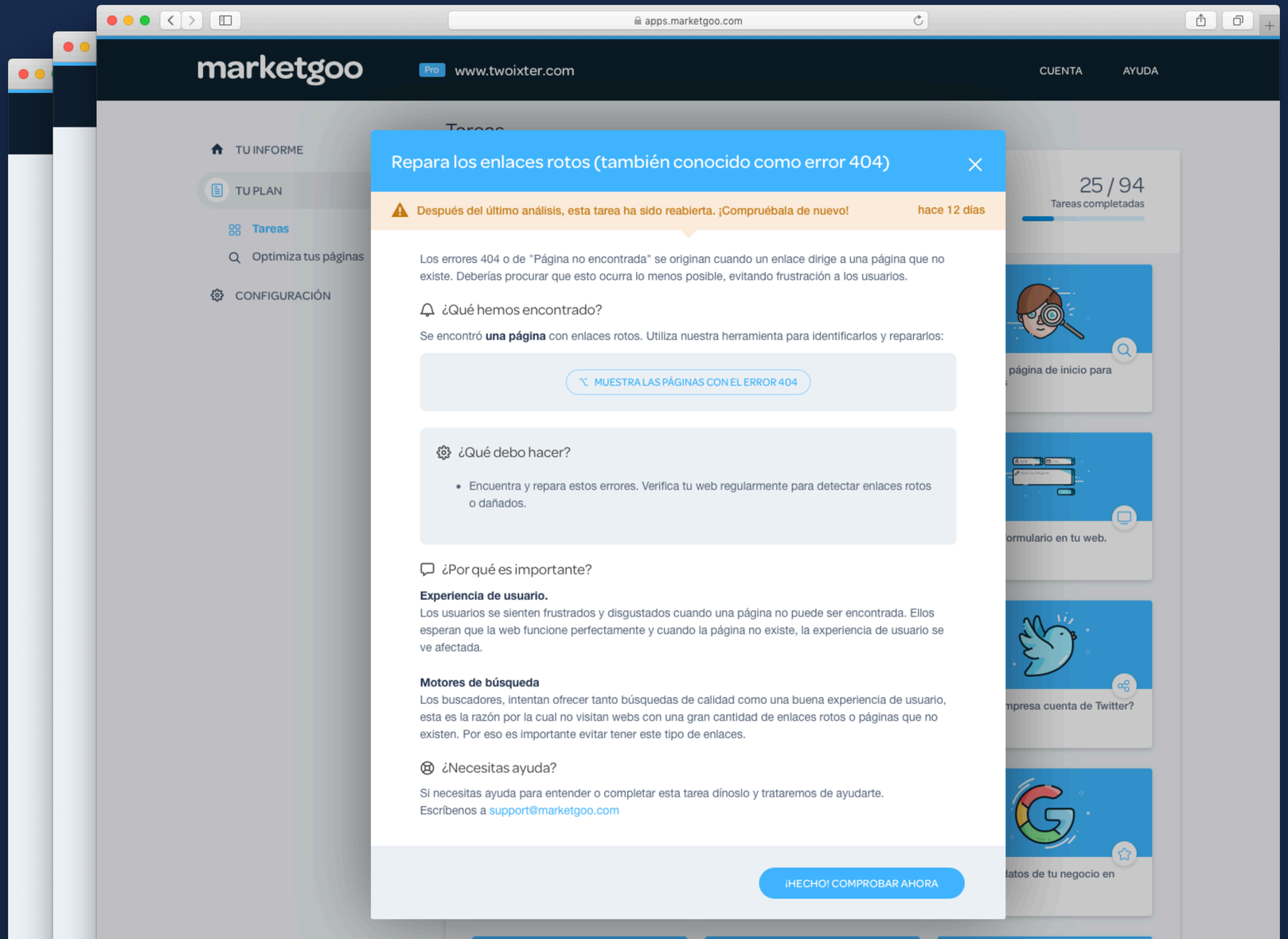


marketgoo

Introducción a MarketGoo

❤️ C++

- Aplicación web para automatizar SEO de una página web
- Análisis diario de una página web (Crawling)
- Medición de resultados (Rankings, enlaces externos, etc)
- Generación de sugerencias de mejora (SEO, Social, etc)
- Herramienta SaaS orientada al usuario final (DIY)

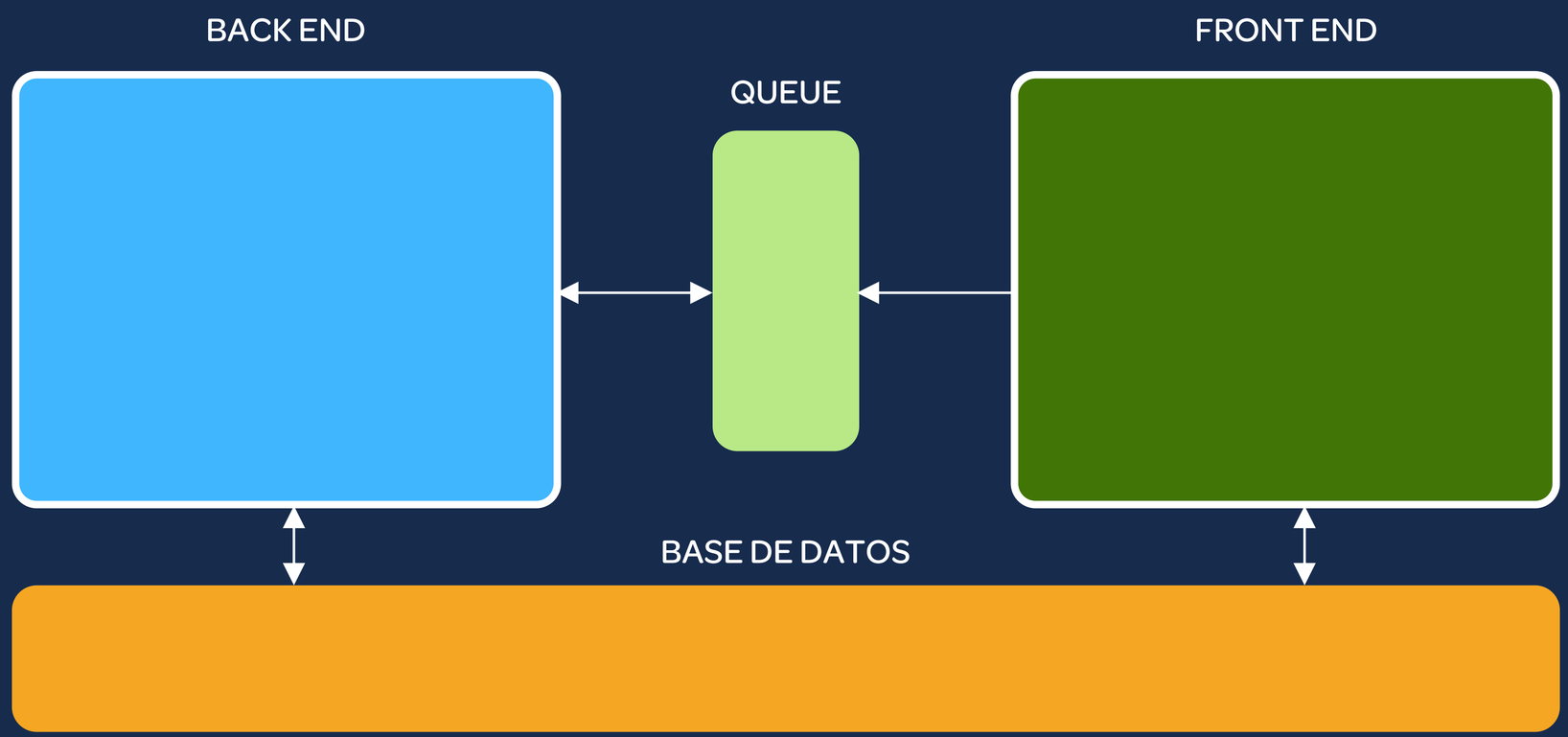


Arquitectura de MarketGoo

marketgoo

Arquitectura de MarketGoo

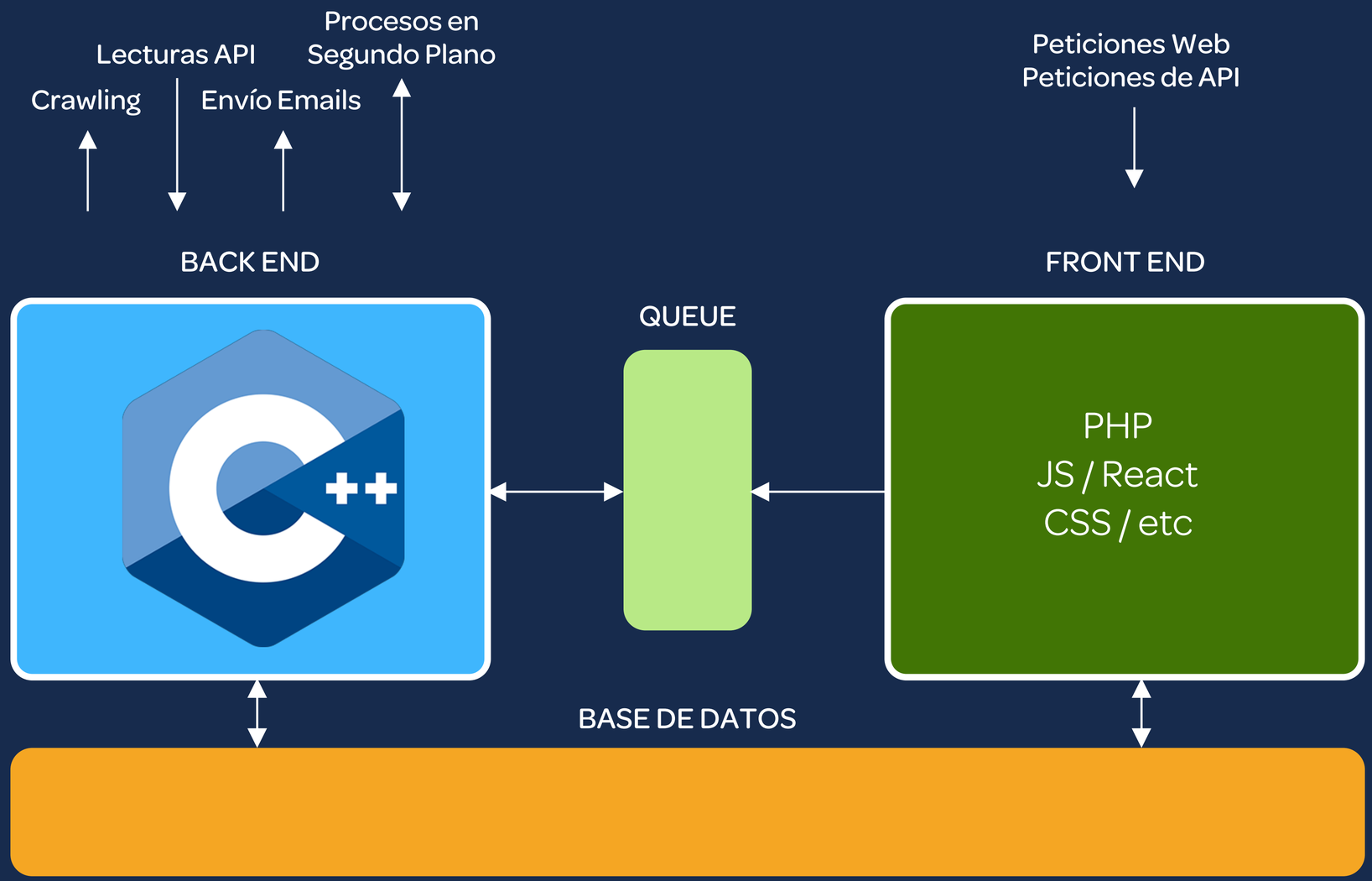
❤️ C++



marketgoo

Arquitectura de MarketGoo

❤️ C++



No usamos C++ en el front

Siento defraudaros :-)

No usamos C++ en el front

Existen algunos frameworks C++ para programación front.
Pero intentan resolver un problema que no existe.

“Al front lo que es de front y al back lo que es de back.”

- **Phalcon** — phalconphp.com (Framework Full-stack como extensión C para PHP)
- **TreeFrog Framework** — treefrogframework.com (High-speed C++ MVC Framework, usa Qt)
- **CppCMS** — cppcms.com (High Performance C++ Web Framework)
- **Wt Toolkit** — webtoolkit.eu (Tiene hasta Widgets en C++)

marketgoo

No usamos C++ en el front

❤️ C++

Ejemplo de Widget en Wt Toolkit

```cpp
#include <Wt/WContainerWidget.h>
#include <Wt/WText.h>

auto container = Wt::cpp14::make_unique<Wt::WContainerWidget>();

container->addWidget(Wt::cpp14::make_unique<Wt::WText>("A first widget"));

for (unsigned int i = 0; i < 3; ++i) {
    // A widget can be added to a container by using addWidget()
    container->addWidget(Wt::cpp14::make_unique<Wt::WText>(Wt::WString("<p>Text {1}</p>").arg(i)));
}
```

(A ver, que no digo yo que no esté chulo para un sistema embebido, ¡ojo!)

Dónde y cómo usamos C++

Dónde y cómo usamos C++

Lo usamos para toda la lógica de negocio de back-end, obviamente.

- **Escalabilidad** — Tenemos que analizar diariamente miles de páginas web. Muchas veces cientos de páginas por segundo.
- **Fiabilidad** — Para grandes volúmenes de datos no podemos permitirnos no tener control sobre los recursos (control sobre el run-time).
- **Flexibilidad** — Estructura de micro-servicios (actores) nos permite tener control granular sobre qué está ejecutándose en cada momento.
- **Herramientas estándar** — Usamos tecnologías probadas y estándar en la industria como libcurl, libxml2, ICU, etc.

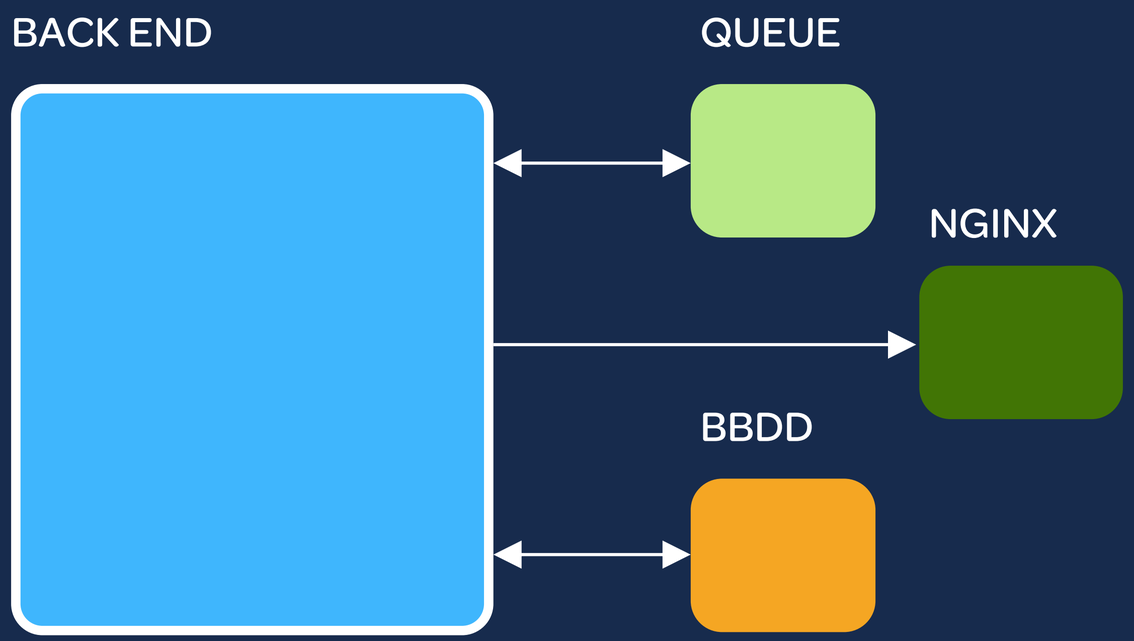
Escalabilidad - Modelo de Actores

Escalabilidad - Modelo de Actores

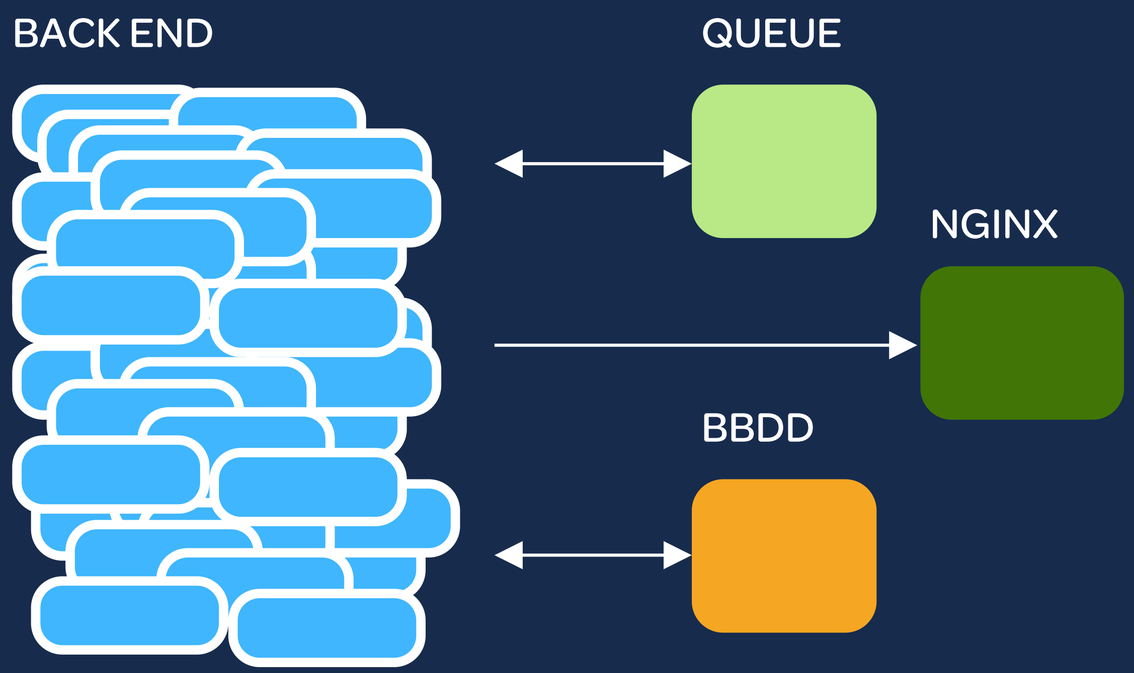
> El modelo de actor en ciencias de la computación es un modelo matemático de computación simultánea que trata a los «actores» como los primitivos universales de la computación concurrente.
>
> — Wikipedia

La idea no es nueva, parte desde 1973 influenciada por ideas de Lisp, Simula y SmallTalk. Usado satisfactoriamente como modelo de computación distribuida por Erlang. También es usado por Scala y Akka en Java (JVM).

Escalabilidad - Modelo de Actores

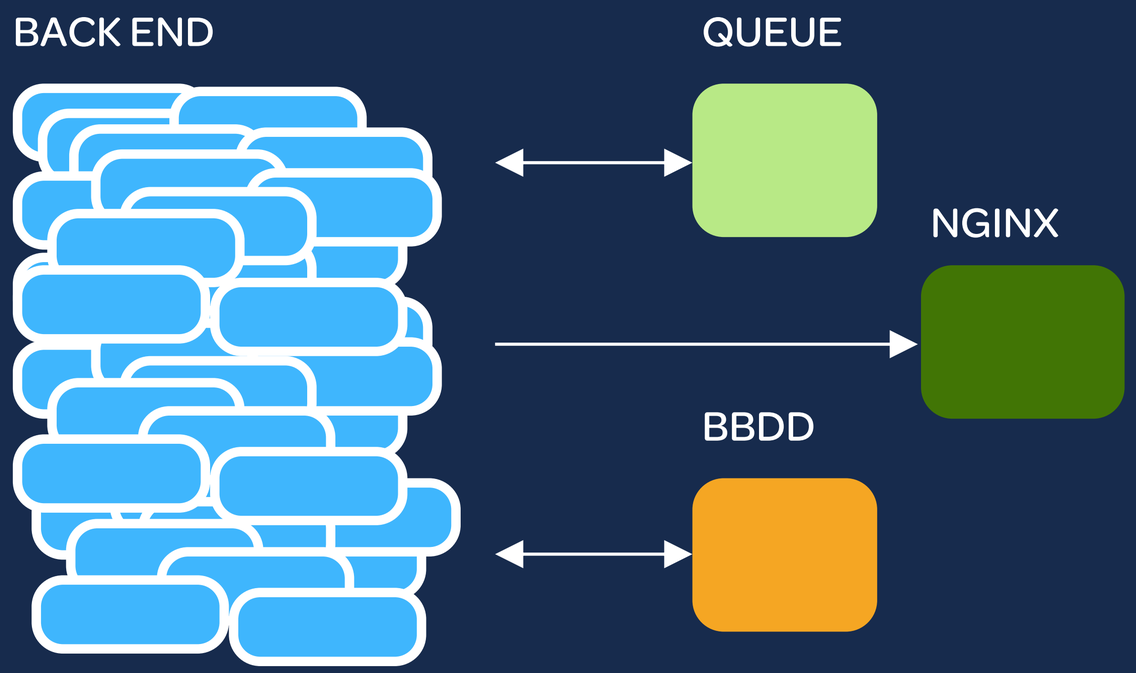


Escalabilidad - Modelo de Actores



Escalabilidad - Modelo de Actores

- Son procesos ágiles que sólo hacen una cosa.
- Su cometido es recibir mensajes de la cola (queue), procesarlo y producir datos u otros mensajes.
- No se comparte memoria entre actores.
- Son ligeros (threads en nuestro caso).
- No hay comunicación entre ellos, a no ser que sea mediante la propia cola de mensajes (serialización).



Fiabilidad — Control de los recursos

Fiabilidad — Control de los recursos

En C++ con el modelo de actores tenemos más control sobre el uso de los recursos

- **Memoria** — Cualquier lenguaje de script (Ruby, PHP, Python) añade un coste de run-time sólo por lanzar el proceso (carga de librerías: gems en Ruby, Composer en PHP).
  - Manejo engorroso o inexistente de threads.
  - Malgasto de memoria por recolección de basura (a veces no se produce, o manual).

- **CPU** — Como alternativa un modelo de eventos (Node.js, RxPHP) no es viable.
  - No se puede limitar el consumo de CPU.

- **Excepciones y librerías** — Al compilar en C++ no tenemos «sustos» con cambios repentinos en las librerías que producen nuevas excepciones o errores que surgen sólo en run-time.

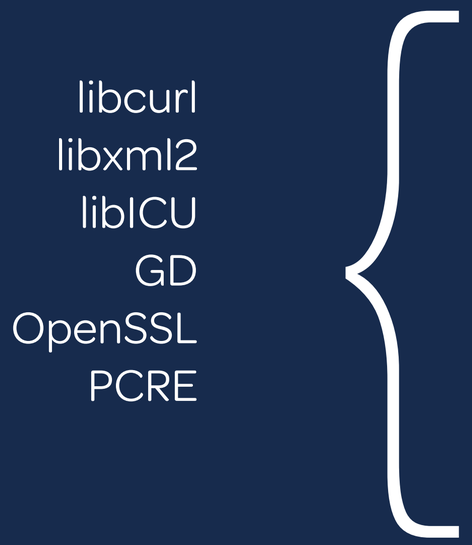
Herramientas Estándar — Librerías

marketgoo

Herramientas Estándar — Librerías

❤️ C++

En C++ tenemos a nuestra disposición las mismas librerías que los gigantes de la Web.
No hay que reinventar la rueda...



Apache
Nginx
Redis
MongoDB
MySQL

Nokogiri (Ruby — Wrapper de «libxml2»)
Guzzle (PHP — Wrapper de «libcurl»)
PHP (PHP-Intl — Wrapper de «libICU»)

Estrategias que usamos

Aquí viene lo bueno :-)

Estrategias que usamos

- **Wrappers** — Intentamos adaptar las librerías del mismo modo que se usarían en lenguajes de script. Nos permite hacer un uso más cómodo y centrarnos en la lógica.
  - Por sistema, evitar usar punteros en la medida de lo posible.
  - Maximizar el uso de RAII.

- **Excepciones** — Evitamos las excepciones como transmisión de errores.
  - Obviamente atendemos excepciones pero no para errores comunes.

- **Tipos de datos dinámicos** — Mediante un tipo de dato «dinámico» **`goo::var`** simulamos ciertas estrategias de lenguajes no tipados (Ej: para JSON).
  - **`goo::var`** es parecido a **`std::any`** pero con conversiones automáticas.

Estrategias que usamos

Ejemplo de uso de **`goo::var`**

- Uso para variables dinámicas parecido a **`std::any`** pero más «liberal»
- Conversiones de tipo mediante métodos **`asInteger()`**, **`asString()`**, **`asDouble()`**
- Conversión desde / hacia JSON mediante streams
- Acepta tipos básicos (int, std::string, double) más array (en realidad es un «std::map») o vector (en realidad «std::vector»)

```cpp
goo::var precio = 123.4;

// Selección de tipo al acceder
std::cout << precio.asInteger() << std::endl;   //  123
std::cout << precio.asString() << std::endl;    //  123.4

// Acceso transparente como array
precio["sub_total"] = 100;  // Se crean automáticamente índices
precio["iva"] = 21;
precio["total"] = 121;

// No se producen excepciones al acceder a un índice inexistente
std::cout << precio["oferta"].asDouble() << std::endl;  //  0.0

// Podemos testear índices inexistentes
if (precio["oferta"].exists()) {
    ...
}

// Conversión a JSON hacia a streams
std::cout << precio << std::endl;
// {"sub_total": 100.0, "iva": 21, "total": 121.0}

// Parseado desde streams
std::cin >> precio;     // Admite JSON
```

Estrategias que usamos

Ejemplo de uso de **`goo::http::request`**

- Lectura de alto nivel de recursos HTTP
- Wrapper de «libcurl»
- Maneja automáticamente errores más comunes encontrados en páginas web (Ej: timeouts, redirecciones, etc).
- Compatible con todos los métodos REST: GET, POST, PUT, DELETE...
- Envío de datos de POST mediante variables o JSON.

```cpp
goo::http::request query;

if (query.get("https://www.marketgoo.com")) {
    std::cout << "Página leída con status: " << query.status << std::endl;

    if (query.status == 200) {
        std::cout << "Bytes leídos: " << query.buffer().size() << std::endl;

        // Ejemplo de acceso a cabeceras (es un goo::var)
        if (query.received_headers["content-type"].exists()) {
            std::cout << "Tenemos Content-Type: "
                << query.received_headers["content-type"].asString()
                << std::endl;
        }

        // Hacemos buen uso de goo::var...
        goo::var payload = query.as_json();
    }
} else {
    // En realidad, ha sido un error de red. Resultado de lectura
    // de libcurl ≠ CURL_OK
    std::cerr << "Oops! No se pudo leer..." << std::endl;
}
```

Cola de mensajes — Actores

Cola de mensajes — Actores

Usamos «Beanstalkd» como cola de mensajes.

Otras alternativas pueden ser « ØMQ», «RabbitMQ», «Gearman», etc.

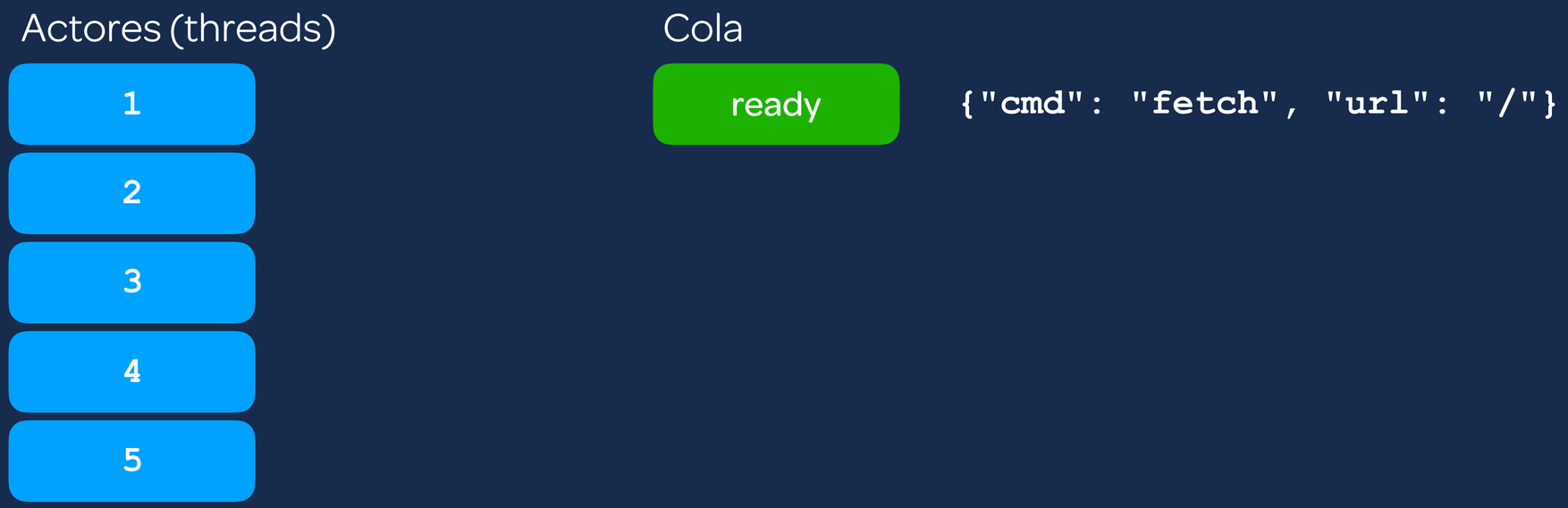
- **Jobs** — Está orientada a ser usada como cola de trabajos (jobs).
  - Cada mensaje puede estar «ready», «reserved» o «delayed».
  - Cada actor que procesa un mensaje debe «confirmar» que ha terminado.
  - Admite un time-out en el cual si un actor no ha terminado el mensaje pasa automáticamente de «reserved» a «ready» para que otro actor lo termine.

- **Lightweight** — Es un broker de mensajes muy ligero y fácil de usar.

- **Tipo de datos** — El formato de los mensajes es un simple «string», podemos fácilmente usar JSON como payload.

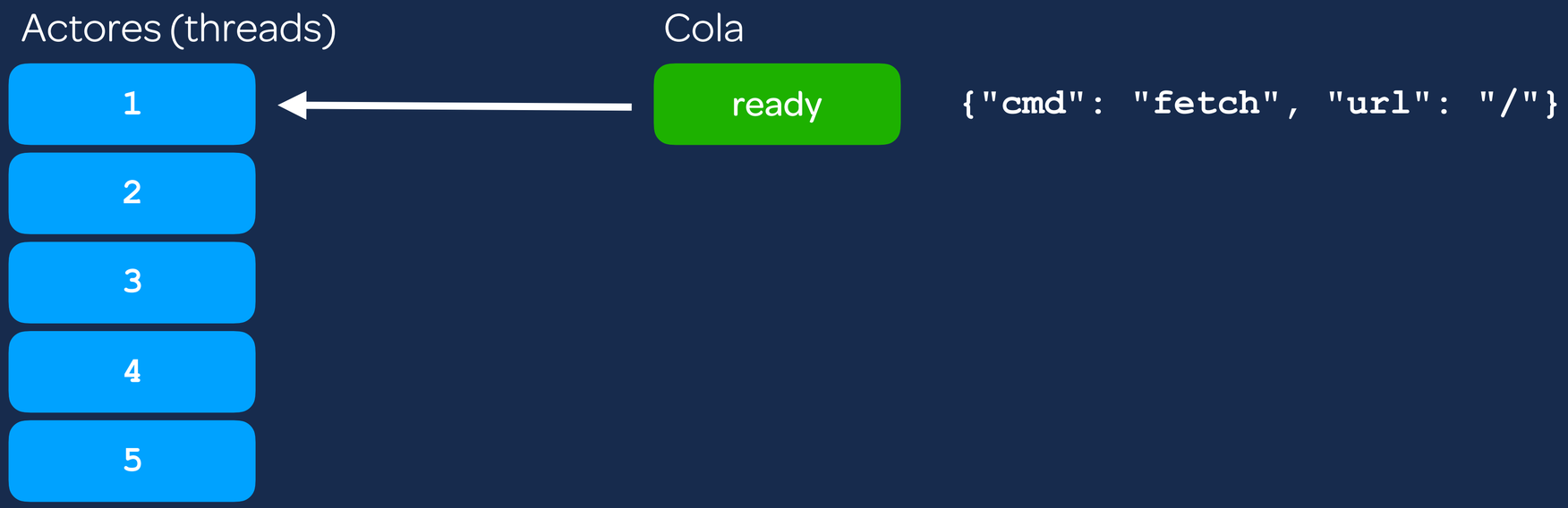
Cola de mensajes — Actores

Ejemplo de flujo de mensajes



Cola de mensajes — Actores

Ejemplo de flujo de mensajes

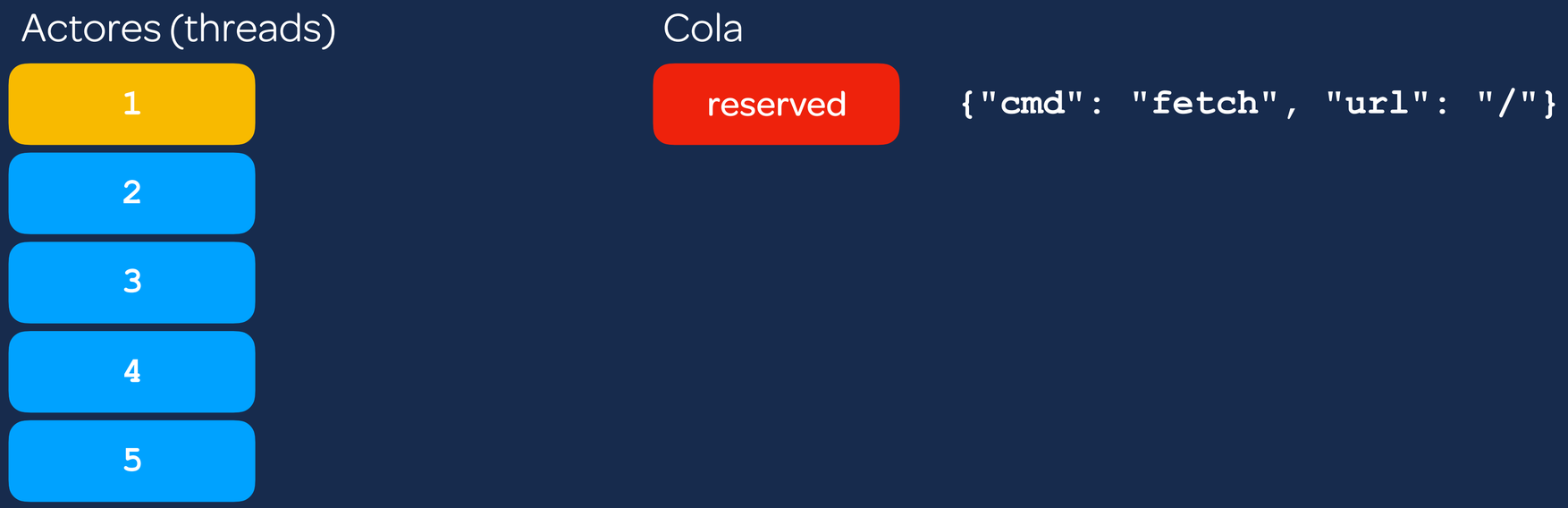


marketgoo

Cola de mensajes — Actores

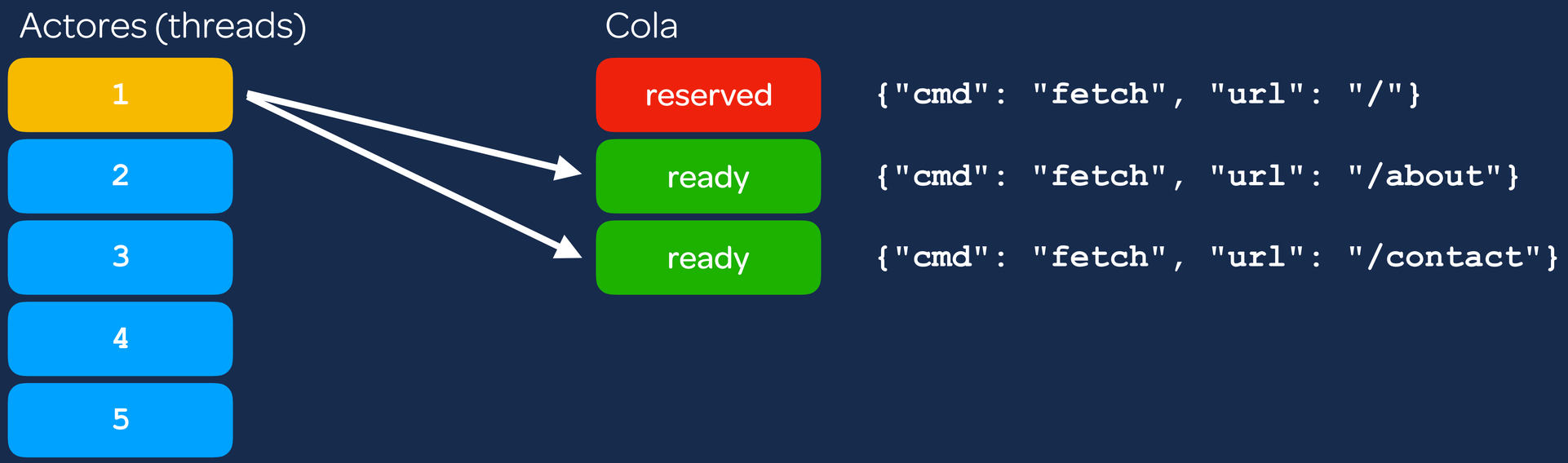
❤️ C++

Ejemplo de flujo de mensajes



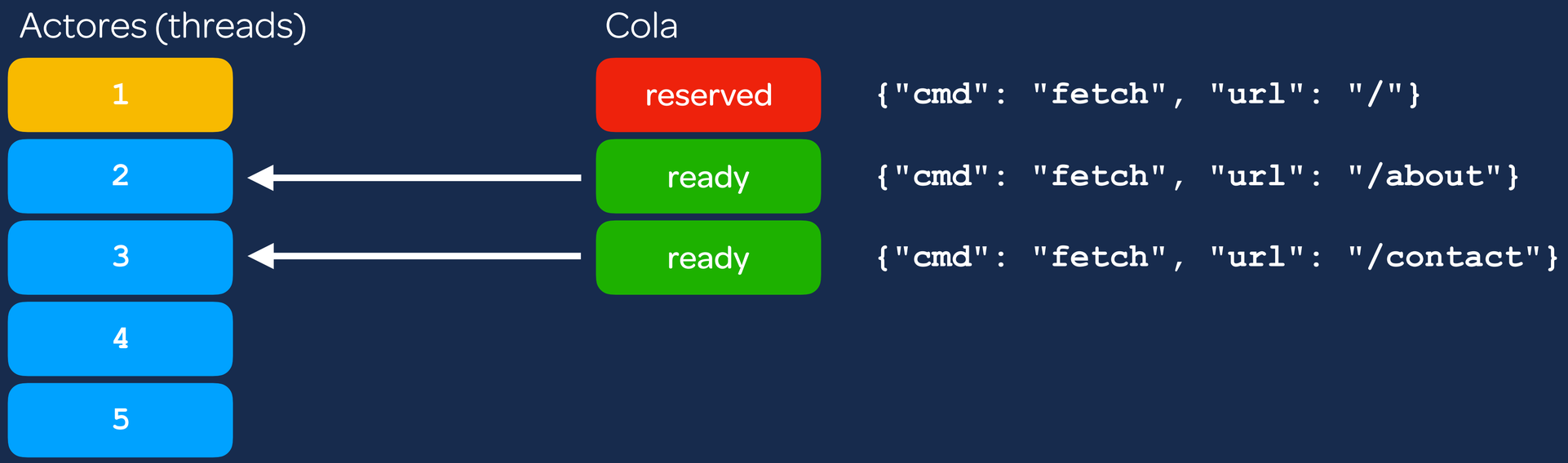
Cola de mensajes — Actores

Ejemplo de flujo de mensajes



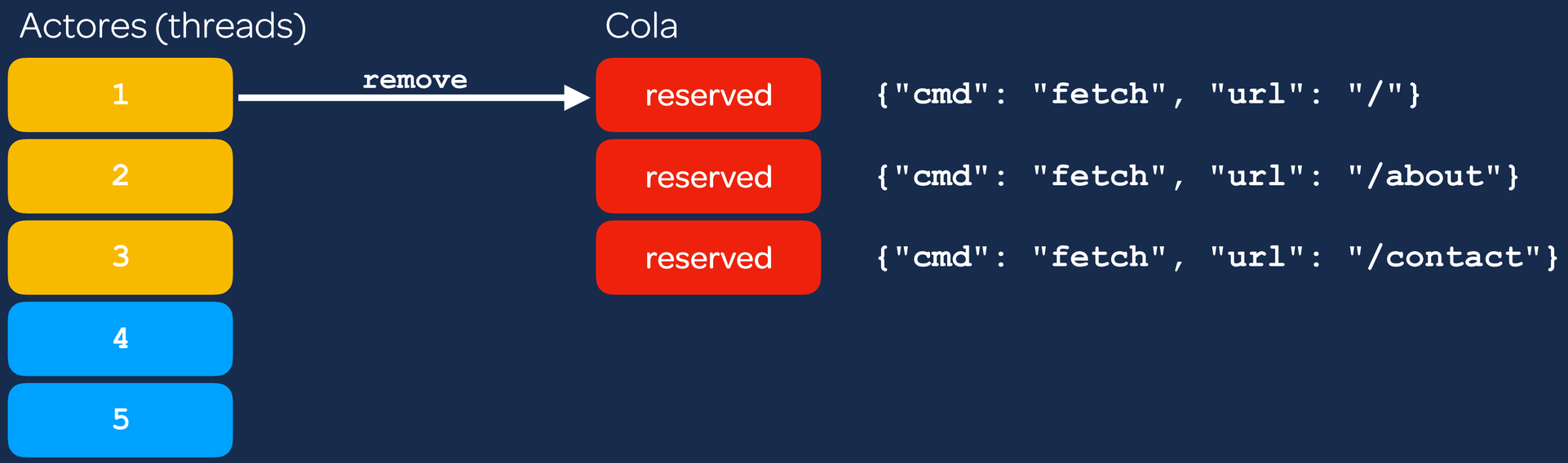
Cola de mensajes — Actores

Ejemplo de flujo de mensajes



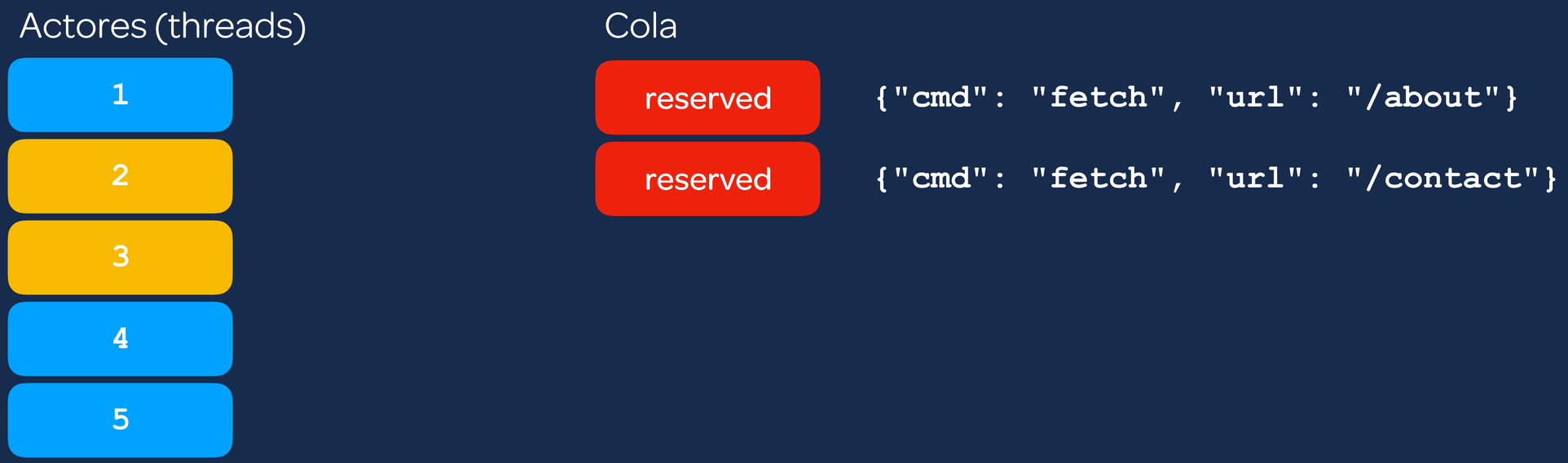
Cola de mensajes — Actores

Ejemplo de flujo de mensajes



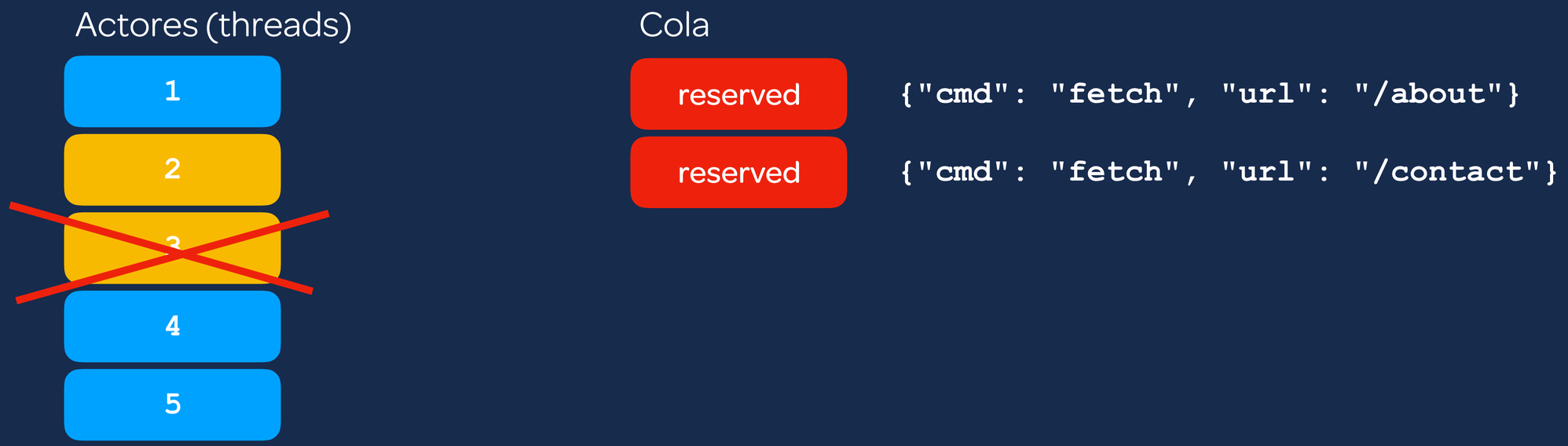
Cola de mensajes — Actores

Ejemplo de flujo de mensajes



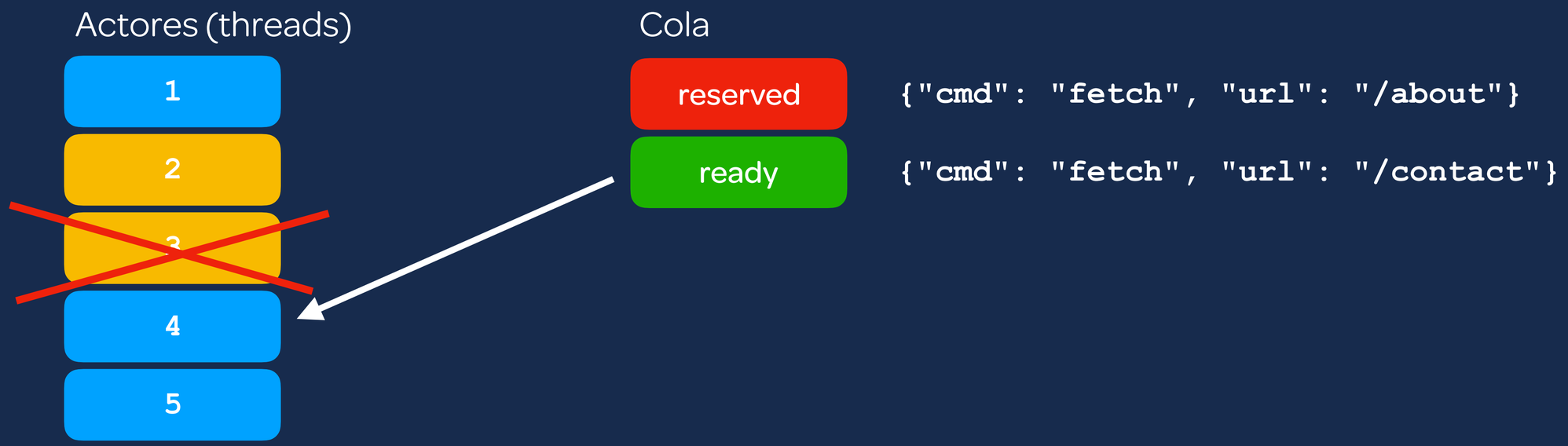
Cola de mensajes — Actores

Ejemplo de flujo de mensajes



Cola de mensajes — Actores

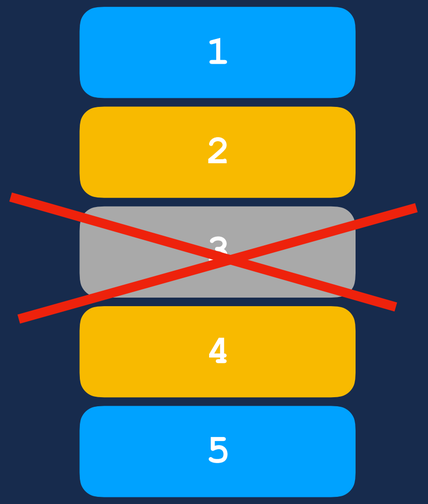
Ejemplo de flujo de mensajes



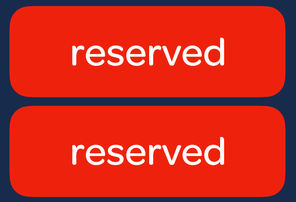
Cola de mensajes — Actores

Ejemplo de flujo de mensajes

Actores (threads)



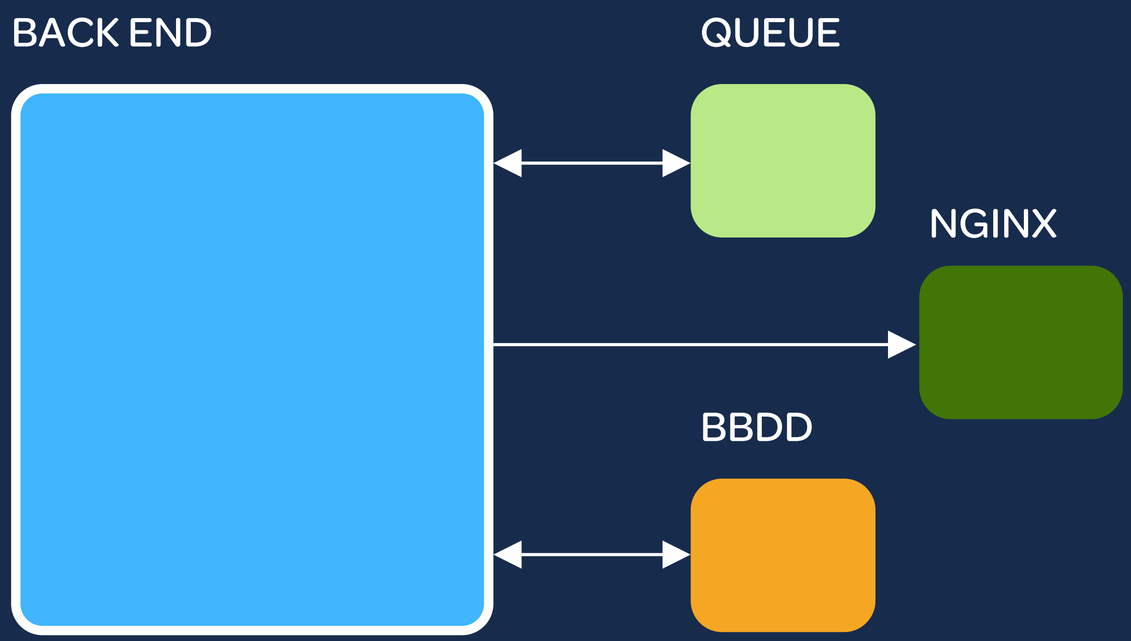
Cola



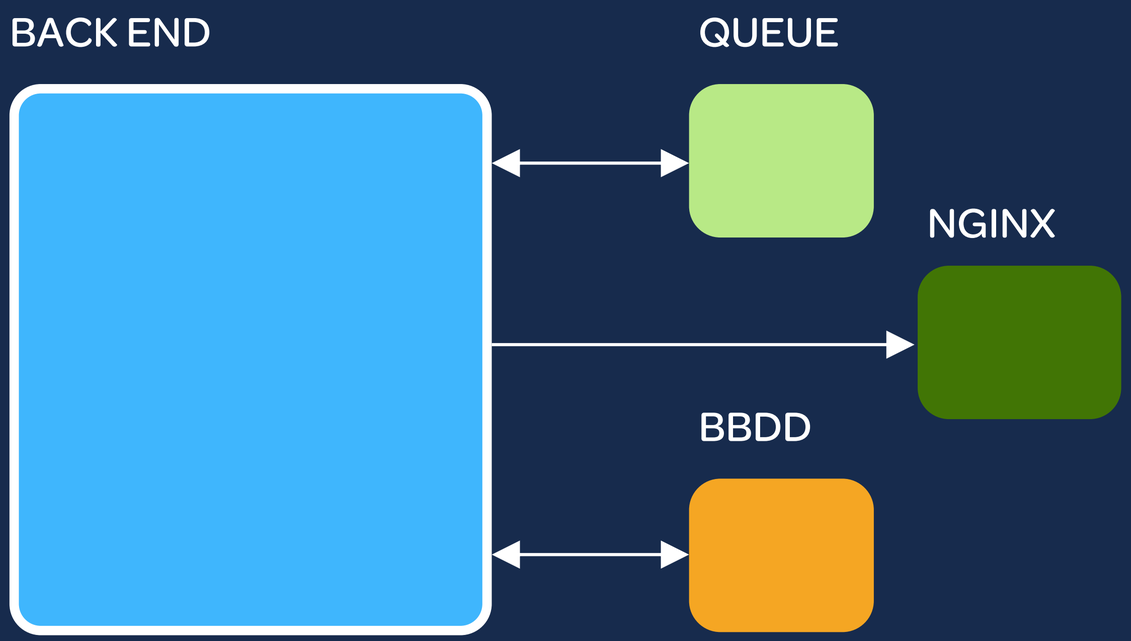

```
{"cmd": "fetch", "url": "/about"}
{"cmd": "fetch", "url": "/contact"}
```

Conexión con el front — Websockets

Conexión con el front — Websockets



Conexión con el front — Websockets



- **PHP** — Desde front (PHP) se envían mensajes a la cola para que se procesen.
  - Enviar cierto email.
  - Empezar crawling de una web.
  - Renderizar un informe en PDF.
  - PHP NO ESCUCHA EN LA COLA.

- **Websockets** — Desde el backend enviamos mensajes a un canal de Websockets.
  - Plugin: Nginx Push Stream.
  - **`POST /pub/<nombre_del_canal>`**
  - Enviamos mensajes con formato Redux:
  - **`{"type": "SCAN_STARTED", "payload": {…}}`**

Ejemplos de código

marketgoo

Ejemplos de código

❤️ C++

Manos a la obra... :-)

marketgoo ❤️ C++

¡¡ No olvidéis probar www.marketgoo.com !!