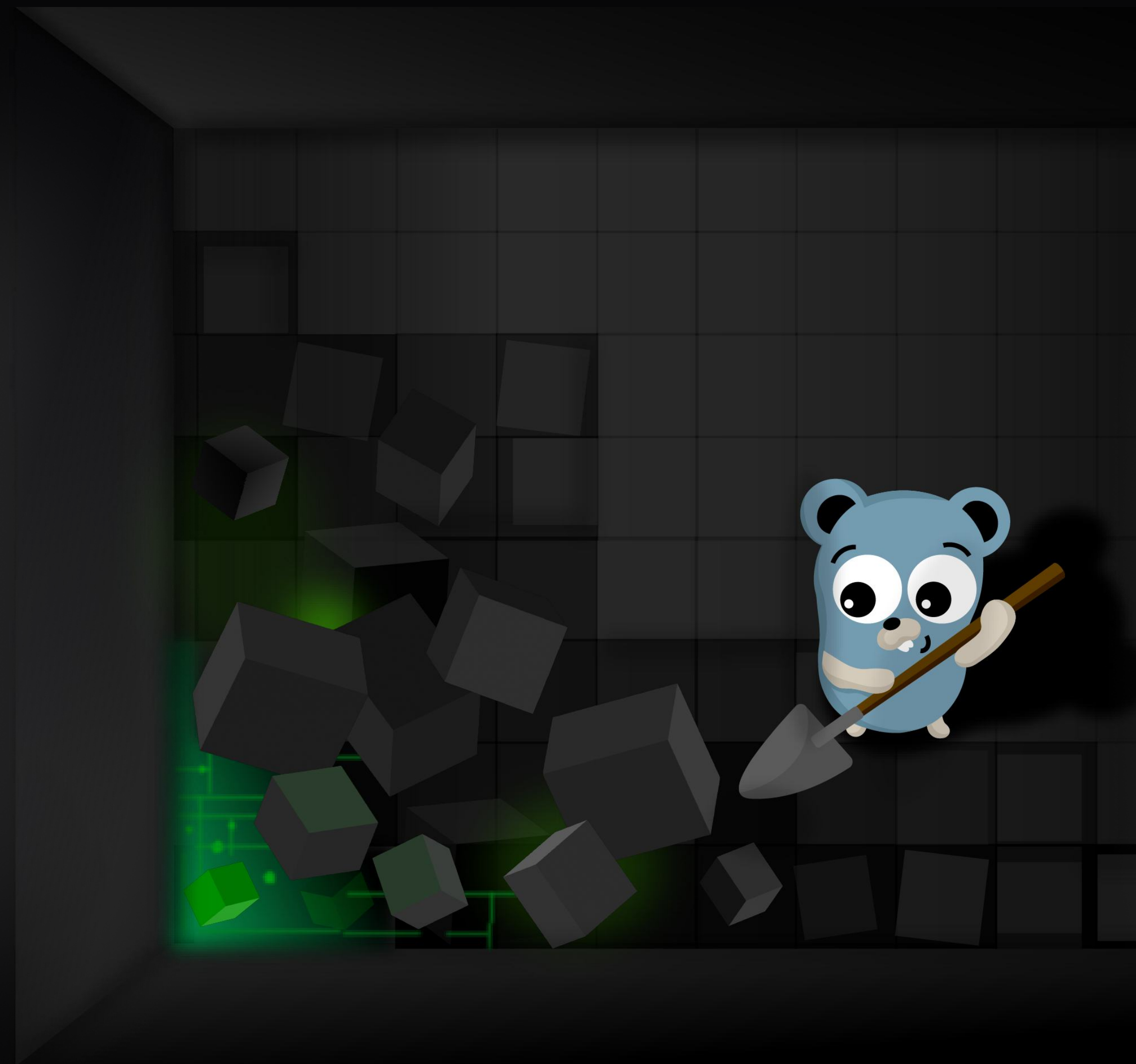
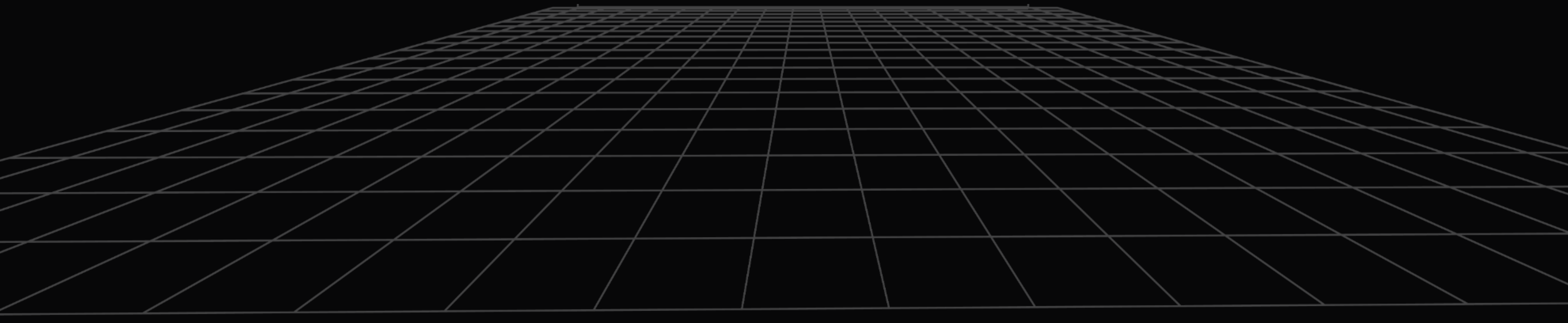


ОБРАБОТКА ОШИБОК

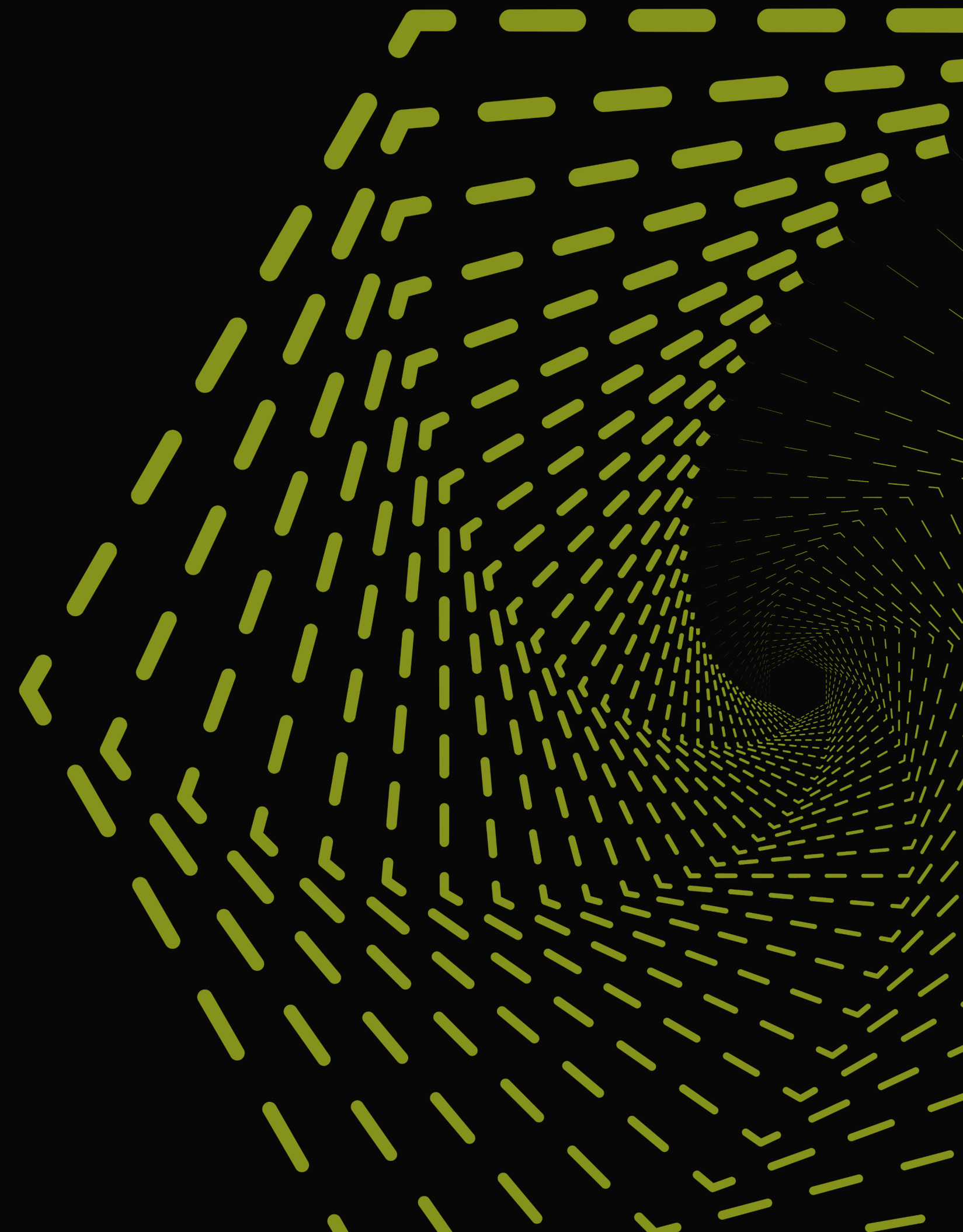


ПРОВЕРЬ ЗАПИСЬ



ПРАВИЛА ЗАНЯТИЯ

1. вопросы в чате можно задавать
в любое время
2. вопросы голосом задаем
по поднятой руке в Zoom
3. ответы на вопросы будут
в запланированных местах

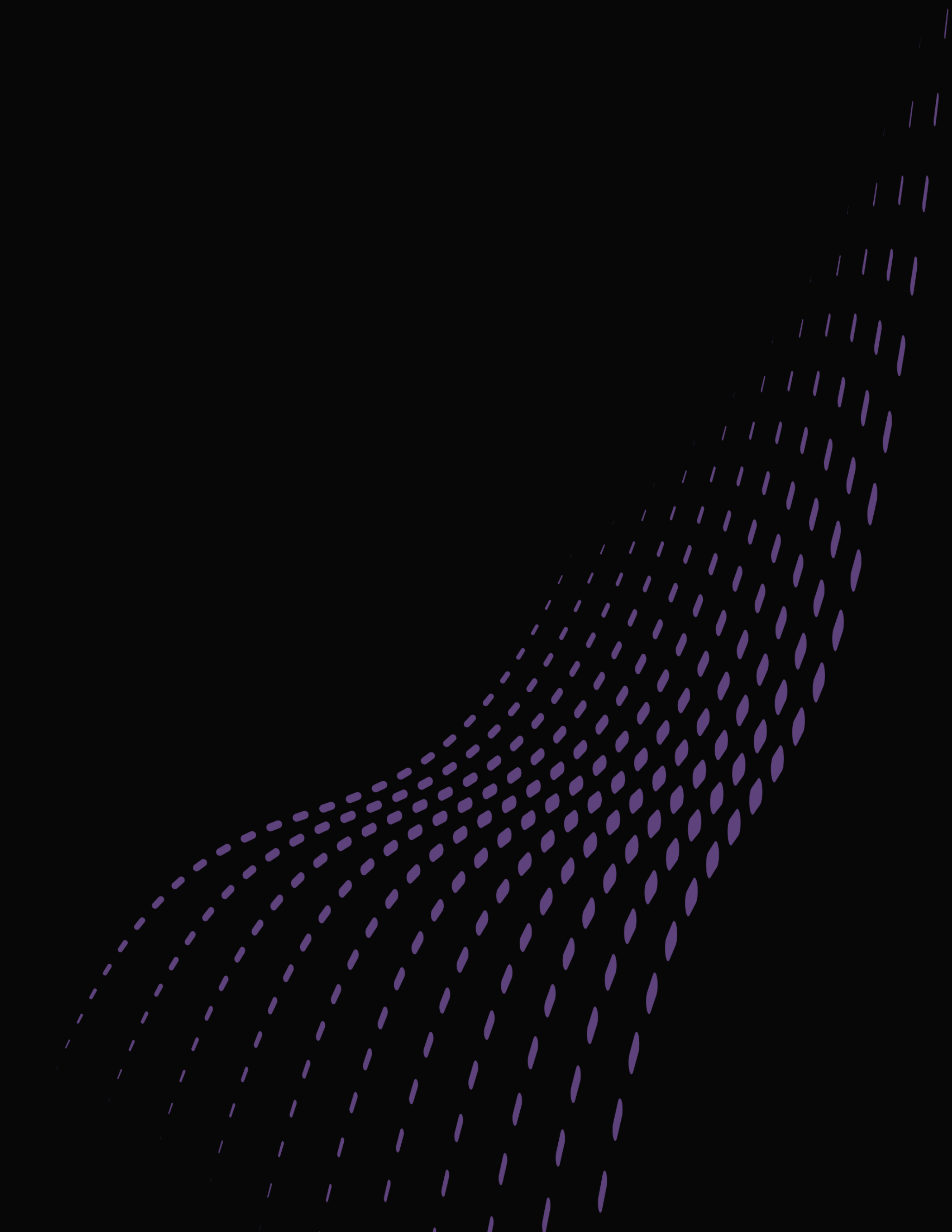


ПЛАН ЛЕКЦИИ

1. Обработка ошибок

2. Ошибки в Go

3. Паника



ОБРАБОТКА ОШИБОК

**КАК МОЖНО СИГНАЛИЗИРОВАТЬ
ОБ ОШИБКАХ?**

EXAMPLE

Error flag

**ЧТО ДЕЛАТЬ, ЕСЛИ ОШИБКА
МОЖЕТ ВОЗНИКНУТЬ ПО
НЕСКОЛЬКИМ ПРИЧИНАМ?**

EXAMPLE

Error status

C++



```
1 std::size_t read(void* buffer, std::size_t size, sys::error_code& ec);  
2 std::size_t write(const void* buffer, std::size_t size, sys::error_code& ec);
```

EXAMPLE

Optional

**КАК НЕ ЗАСОРЯТЬ
СИГНАТУРЫ ФУНКЦИЙ?**

C



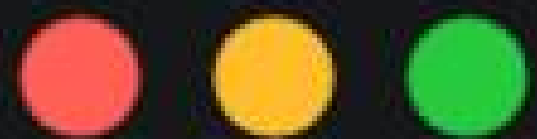
```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <errno.h>
5
6 int main(int argc, const char* argv[])
7 {
8     char* file_name = "text.txt";
9     FILE* file = fopen(file_name, "rb");
10    if (file) {
11        // do something useful...
12        fclose(file);
13    } else {
14        char *errorbuf = strerror(errno);
15        fprintf(stderr, "error message : %s\n", errorbuf);
16    }
17
18    return EXIT_SUCCESS;
19 }
```

EXAMPLE

Errno

**КАК ДОБАВИТЬ ДОПОЛНИТЕЛЬНЫЙ
КОНТЕКСТ ОШИБКЕ?**

Нужно реализовать собственный тип ошибки



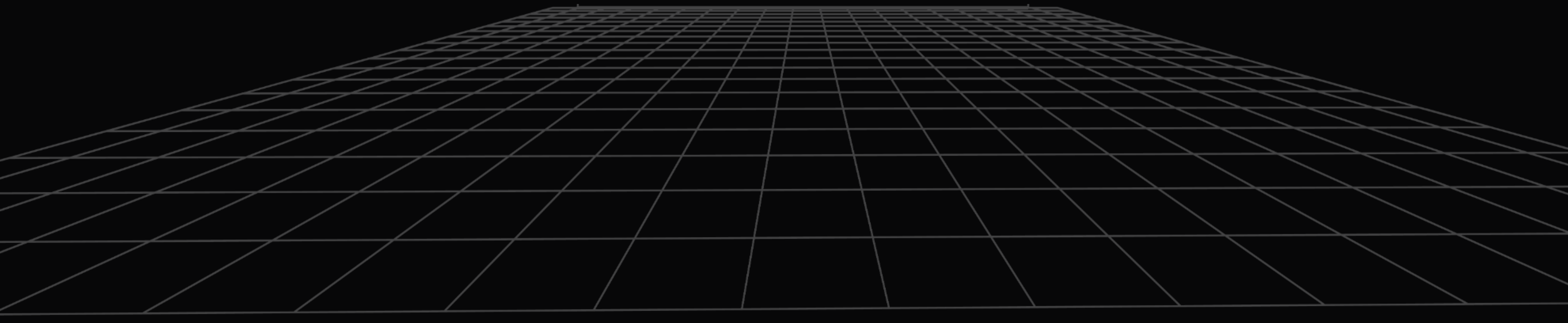
```
1 type error interface {  
2     Error() string  
3 }
```

EXAMPLE

Error

FAQ

Обработка ошибок



ОШИБКИ В GO

ОШИБКА ДОЛЖНА РАССКАЗЫВАТЬ ИСТОРИЮ

Из этого следует несколько
простых правил:

- **Правило №1:** сообщение об ошибке должно быть максимально подробным
- **Правило №2:** сообщение об ошибке должно содержать весь контекст для расследования причин ошибки
- **Правило №3:** сообщение должно однозначно характеризовать место возникновения ошибки

**КАК СОЗДАТЬ СТАНДАРТНУЮ
ОШИБКУ В GO?**

EXAMPLE

Error new

Ошибки в Go принято возвращать
последними из методов и функций!



```
1 func process() (string, error) // correct  
2 func process() (error, string) // incorrect
```

Существует несколько соглашений по именованию ошибок в Go – **error-переменные** начинаются с **err** или **Err**, а **error-типы** заканчиваются на **Error**. Сам текст ошибки принято писать с маленькой буквы.



```
1 var ErrNotFound = errors.New("not found")
2
3 type DatabaseError struct {
4     [...]
5 }
```

EXAMPLE

Errors handling

КАК ПОЛУЧИТЬ СТЕКТРЕЙС ИЗ ОШИБКИ?

GO

Search packages or symbols

/

Q

Why Go

Learn

Docs

Packages

Community

Discover Packages > github.com/pkg/errors

errors

package

module

Version: v0.9.1

Latest

|

Published: Jan 14, 2020

|

License: BSD-2-Clause

|

Imports: 7

|

Imported by: 297,534

Details

⊗ Valid go.mod file

✓ Redistributable license

✓ Tagged version

⊗ Stable version

Learn more about best practices

Repository

github.com/pkg/errors

Links

Open Source Insights

Jump to ...

README

Adding context to an error

Retrieving the cause of an error

Roadmap

Contributing

License

README

errors

Travis-CI

build passing

GO reference

go report B

* used by 12.7k projects

Package errors provides simple error handling primitives.

go get github.com/pkg/errors

The traditional error handling idiom in Go is roughly akin to

if err != nil {



```
1 // New returns an error with the supplied message.
2 // New also records the stack trace at the point it was called.
3 func New(message string) error {
4     return &fundamental{
5         msg:    message,
6         stack: callers(),
7     }
8 }
```



```
1 func (f *fundamental) Format(s fmt.State, verb rune) {  
2     switch verb {  
3     case 'v':  
4         if s.Flag('+') {  
5             io.WriteString(s, f.msg)  
6             f.stack.Format(s, verb)  
7             return  
8         }  
9         fallthrough  
10    case 's':  
11        io.WriteString(s, f.msg)  
12    case 'q':  
13        fmt.Fprintf(s, "%q", f.msg)  
14    }  
15 }
```

EXAMPLE

Error with stacktrace

EXAMPLE

Errors performance

**СКОЛЬКО РАЗ НУЖНО
ОБРАБАТЫВАТЬ ОШИБКУ?**

НЕ НУЖНО ОБРАБАТЫВАТЬ ОШИБКИ НЕСКОЛЬКО РАЗ

зачастую функция передаёт ошибку вверх
по стеку вызовов, добавляя к ней
дополнительную информацию

Если метод возвращает ошибку, значит, потенциально в его работе может возникнуть проблема, которую нужно обработать.

В качестве реализации обработчика может выступать логирование ошибки или более сложные сценарии (например, переоткрытие установленного сетевого соединения, повторный вызов метода и тому подобные операции)

EXAMPLE

Many times handling 1

Если метод возвращает разные ошибки, то нужно проверить, к чему они относятся и для каждой ошибки написать свой обработчик, но

КАК ПРОВЕРИТЬ К ЧЕМУ ОТНОСИТСЯ ОШИБКА?

Сигнальная / дозорная ошибка (sentinel error) – ошибка, определенная как глобальная переменная нужна для того, чтобы понять, не возникла ли конкретная ошибка



```
1 import "errors"
2
3 var ErrNotFound = errors.New("not found")
```

EXAMPLE

Signal errors

ЧТО ДЕЛАТЬ

если ошибку при пробрасывании нужно пометить
дополнительное сигнальной ошибки, либо добавить
ошибке дополнительную информацию?

Можно использовать `fmt.Errorf()` для добавления
ошибке дополнительной информации, но подойдет
ли такой вариант?



```
1 if err != nil {  
2     return fmt.Errorf("decompress error: %v", err)  
3 }
```

EXAMPLE

Errorf

С Go 1.13 поддерживается команда %w, которая возвращает новую ошибку с методом Unwrap(). Такая упаковка ошибки делает ее доступной для функций **errors.Is()** и **errors.As()**



```
1 if err != nil {  
2     return fmt.Errorf("decompress error: %w", err)  
3 }
```

EXAMPLE

Errorf with packing

EXAMPLE

Incorrect wrapping

EXAMPLE

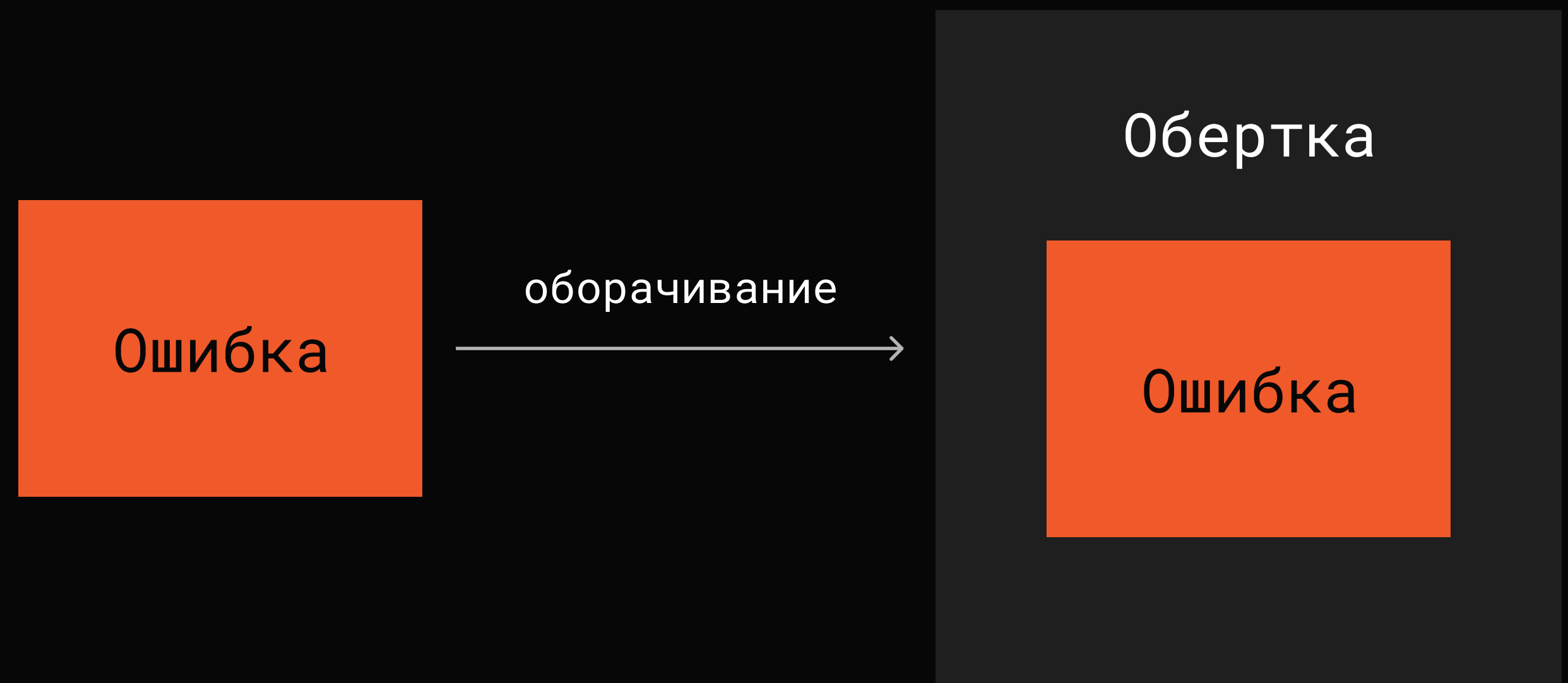
Many times handling 2

ОБОРАЧИВАНИЕ

Упаковка ошибки в контейнеры-обертки, которые делают доступной и исходную ошибку.

Используется при:

- добавлении дополнительного контекста к ошибке;
- маркировке ошибки.





EXAMPLE

Errors type checking

Для того, чтобы проверить, относится ли обернутая ошибка к некоторому типу или нет – **нужно использовать `errors.As()`** (эта функция рекурсивно разворачивает ошибку)

EXAMPLE

`errors.As()`

EXAMPLE

Errors value checking

НИКОГДА НЕ ПРОВЕРЯЙТЕ ВЫВОД `ERR.ERROR()`

В первую очередь он предназначен для людей, а не для кода.

-  Содержимое этой строки должно находиться в файле журнала или отображаться на экране (не стоит пытаться изменить поведение программы, в зависимости от значения этой строки)

Для того, чтобы проверить, относится ли значение обернутой ошибки к определенному значению – нужно использовать **errors.Is()** (эта функция рекурсивно разворачивает ошибку)

EXAMPLE

`errors.Is()`

Функция **errors.Is()** рекурсивно сравнивает ошибку со значением



```
1 // Similar to:
2 //     if err == ErrNotFound { ... }
3 if errors.Is(err, ErrNotFound) {
4     // something wasn't found
5 }
```

Функция **errors.As()** рекурсивно проверяет, относится ли ошибка к конкретному типу



```
1 // Similar to:
2 //     if e, ok := err.(*QueryError); ok { ... }
3 var e *QueryError
4 if errors.As(err, &e) {
5     // err is a *QueryError
6 }
```

Лучше использовать **errors.Is** и **errors.As** всегда,
так как код постепенно меняется и со временем
кто-то может обернуть ошибку

Иногда сигнальные ошибки специально оборачивают,
чтобы пользователи с самого начала не могли
сравниваться со значением ошибки



```
1 var ErrPermission = errors.New("permission denied")
2
3 func ProcessUser() error {
4     if !HasPermission() {
5         return fmt.Errorf("%w", ErrPermission)
6     }
7
8     // implementation...
9 }
```

Какие могут возникать проблемы
с сигнальными ошибками?



```
1 import "errors"  
2  
3 var ErrNotFound = errors.New("not found")
```

Такая ошибка является **публичной**
переменной, поэтому ее можно изменить
в том числе и из другого пакета

EXAMPLE

Change global error

EXAMPLE

Constant error

Когда использовать сигнальные ошибки,
а когда отдельные типы ошибок?



```
1 if errors.Is(err, ErrNotFound) { ... }
```



```
1 if errors.As(err, &DBError{}) { ... }
```

Большим преимуществом отдельных типов ошибок является их способность обеспечения большего контекста, например **os.PathError**



```
1 type PathError struct {  
2     Op    string  
3     Path  string  
4     Err   error  
5 }
```

Сигнальные ошибки и отдельные типы ошибок создают зависимость между пакетами, например, чтобы проверить, равна ли ошибка значению `io.EOF`, ваш код должен импортировать пакет `io`

Можно еще анализировать поведение ошибки,
а не тип или значение, чтобы избавиться
от зависимостей между пакетами

EXAMPLE

Error behavior

ОШИБКИ - ЭТО ЧАСТЬ ПУБЛИЧНОГО API ВАШЕГО ПАКЕТА

относиться к ним нужно так же
бережно, как и к любой другой
части вашего публичного API

**ЧТО ДЕЛАТЬ, ЕСЛИ ИЗ ФУНКЦИИ
НУЖНО ВЕРНУТЬ СПИСОК ОШИБОК?**

Screenshot



Search packages or symbols

/

Q

Why Go ▾LearnDocs ▾PackagesCommunity ▾

Discover Packages > [github.com/hashicorp/go-multierror](#)

multierror

package

module

Version: [v1.1.1](#) **Latest** | Published: Mar 11, 2021 | License: [MPL-2.0](#) | Imports: [5](#) | Imported by: [14,130](#)

Details

Repository

Links

Valid [go.mod](#) file ?

Redistributable license ?

Tagged version ?

Stable version ?

[Learn more about best practices](#)

github.com/hashicorp/go-multierror

[Open Source Insights](#)

Jump to ...

f

README

Installation and Docs

Usage

Documentation

Source Files

README

 **reference**

`go-multierror` is a package for Go that provides a mechanism for representing a list of `error` values as a single `error`.

This allows a function in Go to return an `error` that might actually be a list of errors. If the caller knows this, they can unwrap the list and access the errors. If the caller doesn't know, the error formats to a nice human-readable format.

Google Chrome

EXAMPLE

Multierror

**МОЖНО ЛИ ИГНОРИРОВАТЬ
ОШИБКИ?**

EXAMPLE

Errors ignoring

ИГНОРИРОВАНИЕ ОШИБКИ ДОЛЖНО ВЫПОЛНЯТЬСЯ ЯВНО

иначе читатели кода не поймут –
игнорируете ли вы ошибку или
забыли ее обработать

EXAMPLE

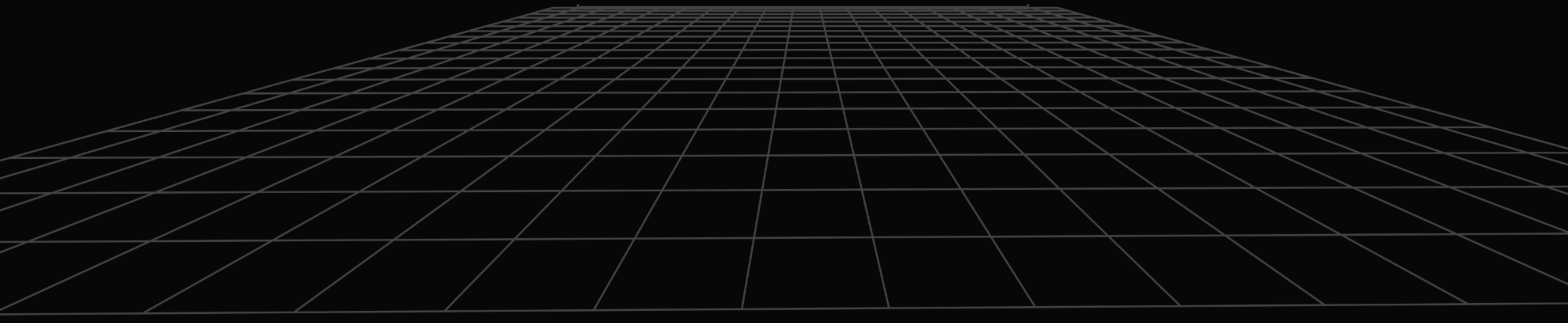
Errors defer ignoring

EXAMPLE

Error from defer

FAQ

Ошибки в Go



ПЕРЕРЫВ 5 МИНУТ

ПАНИКА

GO НЕ ПОДДЕРЖИВАЕТ ИСКЛЮЧЕНИЯ

Вместо этого в программировании на Go предпочтительнее использовать явную обработку ошибок. Но есть одна особенность – Go неявно поддерживает механизм исключений и этот механизм называется **паника/восстановление**

Если присмотреться, то можно подумать, что panic – это то же самое, что и throw, но это семантически не совсем так.

Когда вы бросаете исключение, вы делаете его проблемой вызывающей стороны, а при panic вы никогда не предполагаете, что вызывающий код сможет решить проблему



Поэтому panic используется только в исключительных обстоятельствах, когда продолжение работы вашего приложения невозможно

**ПОЧЕМУ GO НЕ ПОДДЕРЖИВАЕТ
ИСКЛЮЧЕНИЯ?**

ИСПОЛЬЗОВАНИЕ

- Используется для обозначения по-настоящему исключительных ситуаций, например ошибки программиста
- Используется, когда приложение требует зависимость, но ее не удастся инициализировать

EXAMPLE

Panic

При возникновении паники Go приостанавливает выполнение функции и начинает раскручивать стек (stack unwinding) вызовов до тех пор, пока не завершит работу приложения или не найдёт функцию обработки паники `recover()`

**Вызов функции `recover()` полезен
только внутри функции `defer`**

в противном случае, `recover()`
просто вернет `nil` и больше не
будет ни на что влиять

EXAMPLE

Recover without defer

**Когда некоторая функция `func` вызывает `panic`,
в программе происходит следующее:**

- Останавливается выполнение `func`
- Вызываются все её внутренние `defer`-функции
- Запускаются `defer`-функции, связанные с `func` вплоть до верхнего уровня в исполняемой горутине
- Программа заканчивает выполнение и выводит ошибку, включая значение аргумента `panic`

EXAMPLE

Panic with defer 1

EXAMPLE

Panic with defer 2

EXAMPLE

Runtime exit

`runtime.Goexit()` terminates the goroutine that calls it.

No other goroutine is affected. `runtime.Goexit()` runs all deferred calls before terminating the goroutine.

Because **`runtime.Goexit()` is not a panic**, any **recover calls in those deferred functions will return nil.**

EXAMPLE

0S exit

**ПОСЛЕ ВСЕХ ЛИ ОШИБОК МОЖНО
«ВОССТАНАВЛИВАТЬСЯ»?**

EXAMPLE

Division by zero

EXAMPLE

Division by zero float

EXAMPLE

Nil pointer dereference

EXAMPLE

Stack overflow

EXAMPLE

OOM

Для Go некоторые ошибки, такие как переполнение стека и нехватка памяти, не подлежат «восстановлению». Как только они произойдут, программа выйдет из строя

EXAMPLE

Panic nil

EXAMPLE

Panic rethrow

**МОЖНО ЛИ ПОДМЕНИТЬ
ПАНИКУ?**

EXAMPLE

Replace panic

В любой момент времени функция может
быть ассоциирована **не более, чем с одной**
необработанной паникой

EXAMPLE

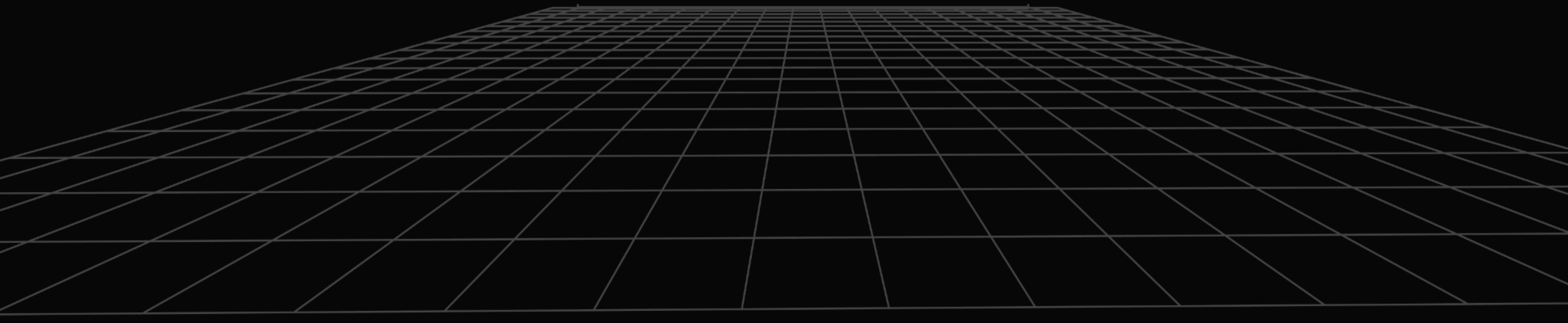
Panic jump

EXAMPLE

Panic task

FAQ

Паника



ПОЖАЛУЙСТА, ЗАПОЛНИ ОПРОС О ЗАНЯТИИ

Ссылка в чате и в группе участников

