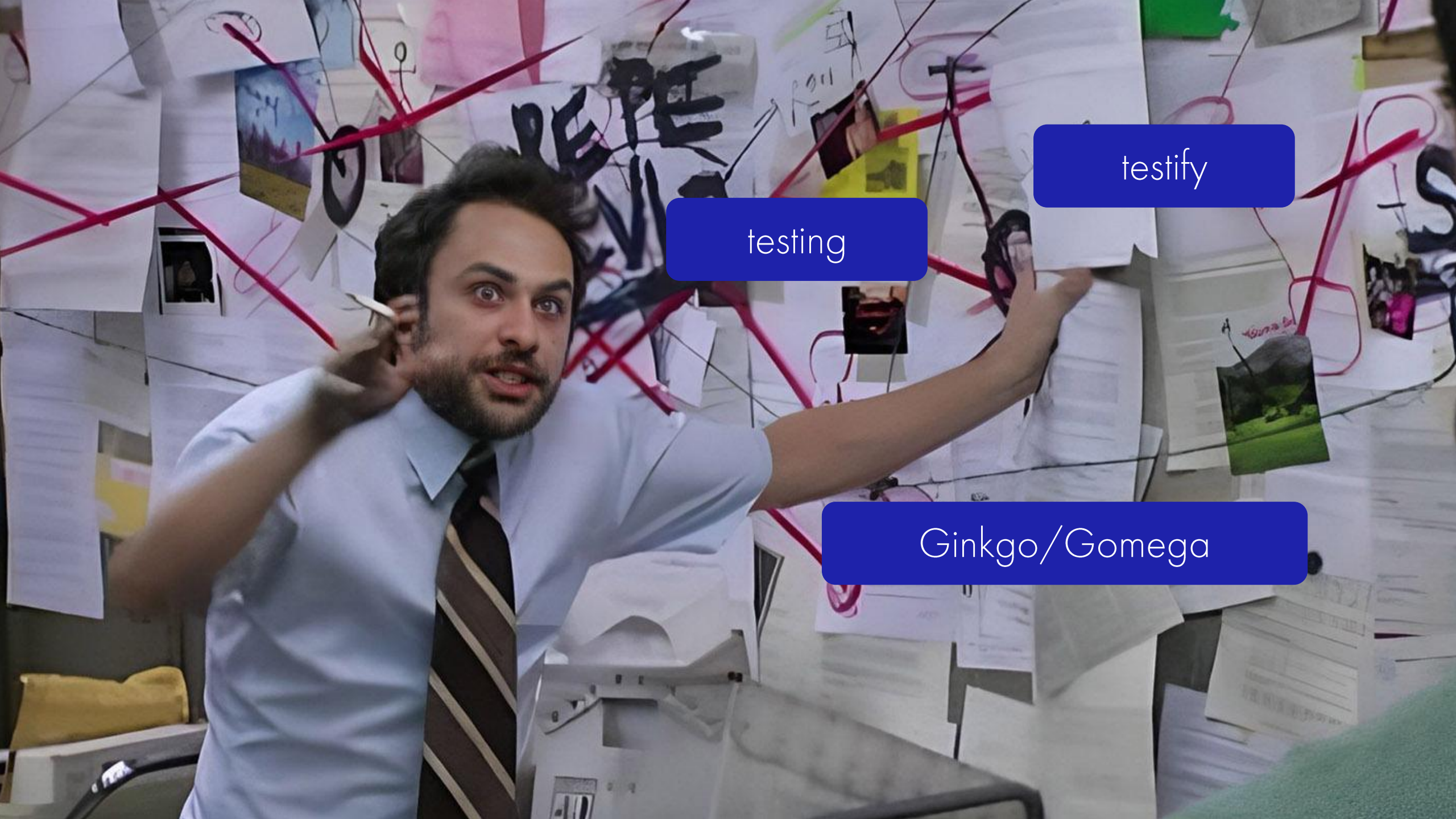
Лучший тестовый фреймворк для Go? Обзор Ginkgo/Gomega

Бондаренко Богдан
TELECOM, OAM, Team R2

Богдан Бондаренко

Инженер-программист

- 2021-2022: Nokia, Digital Automation Cloud (решение для приватной сети 4G/5G)
- 2022-сейчас: YADRO, отдел разработки средств управления базовой станцией


testify
testing
Ginkgo/Gomega

Что это?

Ginkgo — фреймворк для написания выразительных тестов

Что это?

Ginkgo — фреймворк для написания выразительных тестов

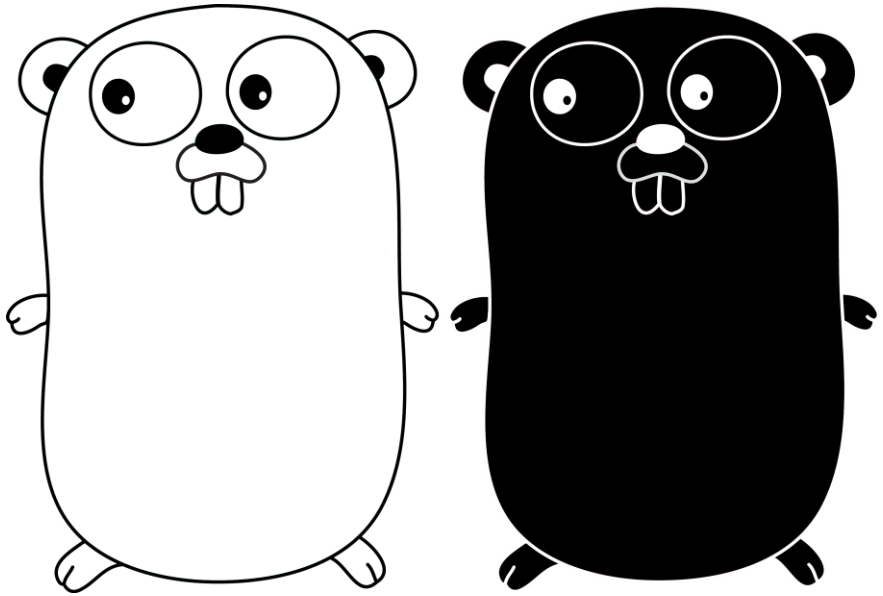
Gomega — библиотека сопоставителей (asserts, matchers, etc.)

Что это?

Ginkgo — фреймворк для написания выразительных тестов

Gomega — библиотека сопоставителей (asserts, matchers, etc.)

Вместе: отдельный язык, специализированный для тестов



TLDR

Плюсы Ginkgo/Gomega:

TLDR

Плюсы Ginkgo/Gomega:

- DSL (обеспечивает хорошую читаемость)

TLDR

Плюсы Ginkgo/Gomega:

- DSL (обеспечивает хорошую читаемость)
- Возможности структурирования тестов (Ginkgo > testify/suite)

TLDR

Плюсы Ginkgo/Gomega:

- DSL (обеспечивает хорошую читаемость)
- Возможности структурирования тестов (Ginkgo > testify/suite)
- Широкие возможности сопоставителей (Gomega > testify/require)

TLDR

Плюсы Ginkgo/Gomega:

- DSL (обеспечивает хорошую читаемость)
- Возможности структурирования тестов (Ginkgo > testify/suite)
- Широкие возможности сопоставителей (Gomega > testify/require)
- Расширяемость и работа со сторонними библиотеками

TLDR

Плюсы Ginkgo/Gomega:

- DSL (обеспечивает хорошую читаемость)
- Возможности структурирования тестов (Ginkgo > testify/suite)
- Широкие возможности сопоставителей (Gomega > testify/require)
- Расширяемость и работа со сторонними библиотеками

Минусы:

TLDR

Плюсы Ginkgo/Gomega:

- DSL (обеспечивает хорошую читаемость)
- Возможности структурирования тестов (Ginkgo > testify/suite)
- Широкие возможности сопоставителей (Gomega > testify/require)
- Расширяемость и работа со сторонними библиотеками

Минусы:

- Трудно залететь с двух ног

TLDR

Плюсы Ginkgo/Gomega:

- DSL (обеспечивает хорошую читаемость)
- Возможности структурирования тестов (Ginkgo > testify/suite)
- Широкие возможности сопоставителей (Gomega > testify/require)
- Расширяемость и работа со сторонними библиотеками

Минусы:

- Трудно залететь с двух ног (попробуем решить эту проблему)

Пример из Telecom может не влезть в тайминг доклада

(кому интересно направление Telecom, подходите на стенд)

Пример из Telecom может не влезть в тайминг доклада
(кому интересно направление Telecom, подходите на стенд)

Поэтому сейчас будут котики

i guess we doin kittens now

Describe cats meowing

- Context - Big cat
- Context - Smaller cat
- Context - Very small cat, wow

Describe cats meowing

- **Before each feed the cat**
- Context – Big cat
- Context - Smaller cat
- Context - Very small cat, wow
  - **Before each take a picture of the kitten**
- **After each put the cat to sleep**

Describe cats meowing

- Before each feed the cat
- Context – Big cat
- Context - Smaller cat
- Context - Very small cat, wow
  - Before each take a picture of the kitten
  - **Specify sunny case**
  - **Specify yawning**
  - **Specify hissing**
- After each put the cat to sleep

Describe cats meowing

- Before each feed the cat
- Context – Big cat
- Context - Smaller cat
- Context - Very small cat, wow
  - Before each take a picture of the kitten
  - Specify sunny case
    - **Expect meowing() to succeed**
  - Specify yawning
  - Specify hissing
- After each put the cat to sleep

```
_ = Describe("cats meowing", func() {
    BeforeEach(func() { ... }) // feed the cat
    Context("Big cat", func() { ... })
    Context("Smaller cat", func() { ... })
    Context("Very small cat, wow", func() {
        BeforeEach(func() { ... }) // take a picture of the kitten
        Specify("sunny case", func() {
            Expect(meowing()).To(Succeed())
        })
        Specify("yawning", func() { ... })
        Specify("hissing", func() { ... })
    })
    AfterEach(func() { ... }) // put the cat to sleep
})
```

```
_ = Describe("cats meowing", func() {
    BeforeEach(func() { ... }) // feed the cat
    Context("Big cat", func() { ... })
    Context("Smaller cat", func() { ... })
    Context("Very small cat, wow", func() {
        BeforeEach(func() { ... }) // take a picture of the kitten
        Specify("sunny case", func() {
            Expect(meowing()).To(Succeed())
        })
        Specify("yawning", func() { ... })
        Specify("hissing", func() { ... })
    })
    AfterEach(func() { ... }) // put the cat to sleep
})
```

Принципы и основные понятия

Дополнительные особенности

Существенные изменения в Ginkgo v2

Gomega

Переход на Ginkgo

```
_ = Describe("cats meowing", func() {
    ...
    Context("Big cat", func() { ... })
    Context("Smaller cat", func() { ... })
    Context("Very small cat, wow", func() { ... })
    ...
})
```

```
_ = Describe("cats meowing", func() {
    ...
    Context("Big cat", func() { ... })
    Context("Smaller cat", func() { ... })
    Context("Very small cat, wow", func() { ... })
    ...
})
```

Container nodes - для иерархической организации различных аспектов кода, который мы тестируем.

Блоки: Describe, Context

```
BeforeEach(func() { ... }) // feed the cat
    ...
    BeforeEach(func() { ... }) // take a picture
AfterEach(func() { ... }) // put the cat to sleep
...
```

```
BeforeEach(func() { ... }) // feed the cat
    ...
    BeforeEach(func() { ... }) // take a picture
AfterEach(func() { ... }) // put the cat to sleep
...
```

Setup nodes - используются для настройки состояний тестового кода.

Для каждого теста в контейнере:
BeforeEach, AfterEach

```
BeforeEach(func() { ... }) // feed the cat
    ...
    BeforeEach(func() { ... }) // take a picture
AfterEach(func() { ... }) // put the cat to sleep
...
```

Setup nodes - используются для настройки состояний тестового кода.

Для каждого теста в контейнере:
BeforeEach, AfterEach

Для всего набора тестов:
BeforeSuite, AfterSuite

```
Specify("sunny case", func() {
    Expect(meowing()).To(Succeed())
})
Specify("yawning", func() { ... })
Specify("hissing", func() { ... })
```

```
Specify("sunny case", func() {
    Expect(meowing()).To(Succeed())
})
Specify("yawning", func() { ... })
Specify("hissing", func() { ... })
```

Subject nodes - описывают операции над тестируемыми объектами

Доступные функции: It, Specify

```
Specify("sunny case", func() {
    Expect(meowing()).To(Succeed())
})
Specify("yawning", func() { ... })
Specify("hissing", func() { ... })
```

Subject nodes - описывают операции над тестируемыми объектами
Доступные функции: It, Specify

Gomega assertions - вызов сопоставителя Gomega
Структура: Expect(<полученное>).To(Equal(<ожидаемое>))

Mental Model: Phases

1. Построение тестового дерева: Tree Construction Phase (Container nodes)

Mental Model: Phases

1. Построение тестового дерева: Tree Construction Phase (Container nodes)
2. Запуск тестового дерева: Run Phase (Setup nodes and Subject nodes code)

Mental Model: Phases

1. Построение тестового дерева: Tree Construction Phase (Container nodes)
2. Запуск тестового дерева: Run Phase (Setup nodes and Subject nodes code)

Правила разработки тестов, которые из этого следуют:

1. Объявляй переменные в контейнерах, работай с ними в тестах

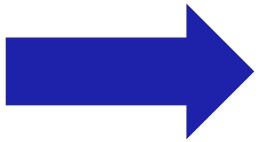
Mental Model: Phases

1. Построение тестового дерева: Tree Construction Phase (Container nodes)
2. Запуск тестового дерева: Run Phase (Setup nodes and Subject nodes code)

Правила разработки тестов, которые из этого следуют:
1. Объявляй переменные в контейнерах, работай с ними в тестах
2. Ginkgo ожидает, что тесты являются независимыми

Tree Construction Phase (easy)

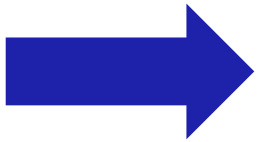
```
Describe:
  BeforeEach:
  Describe:
    Context:
      It: 1
    Context:
      It: 2
```



```
BeforeEach -> It (1)
BeforeEach -> It (2)
```

Tree Construction Phase (easy)

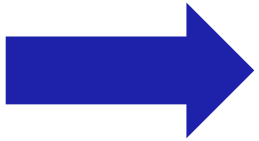
```
Describe:
  BeforeEach:
  Describe:
    Context:
      It: 1
    Context:
      It: 2
```



```
BeforeEach -> It (1)
BeforeEach -> It (2)
```

Tree Construction Phase (easy)

```
Describe:
  BeforeEach:
   Describe:
     Context:
       It: 1
     Context:
       It: 2
```

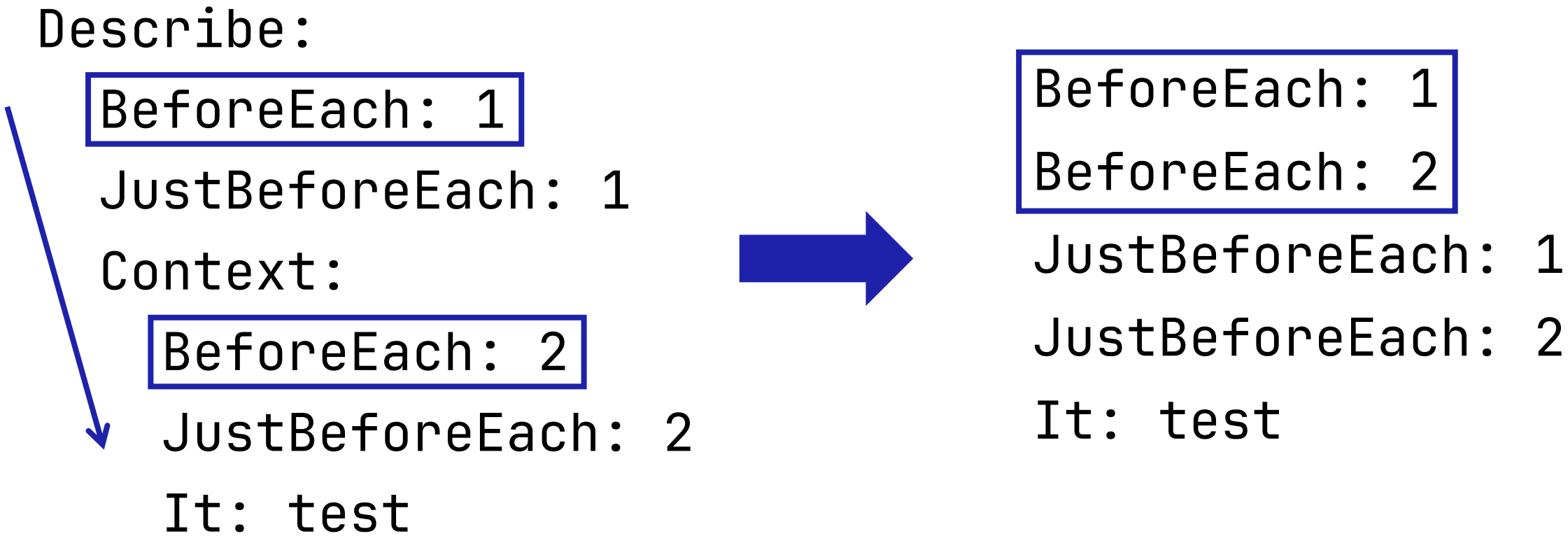


```
BeforeEach -> It (1)
BeforeEach -> It (2)
```

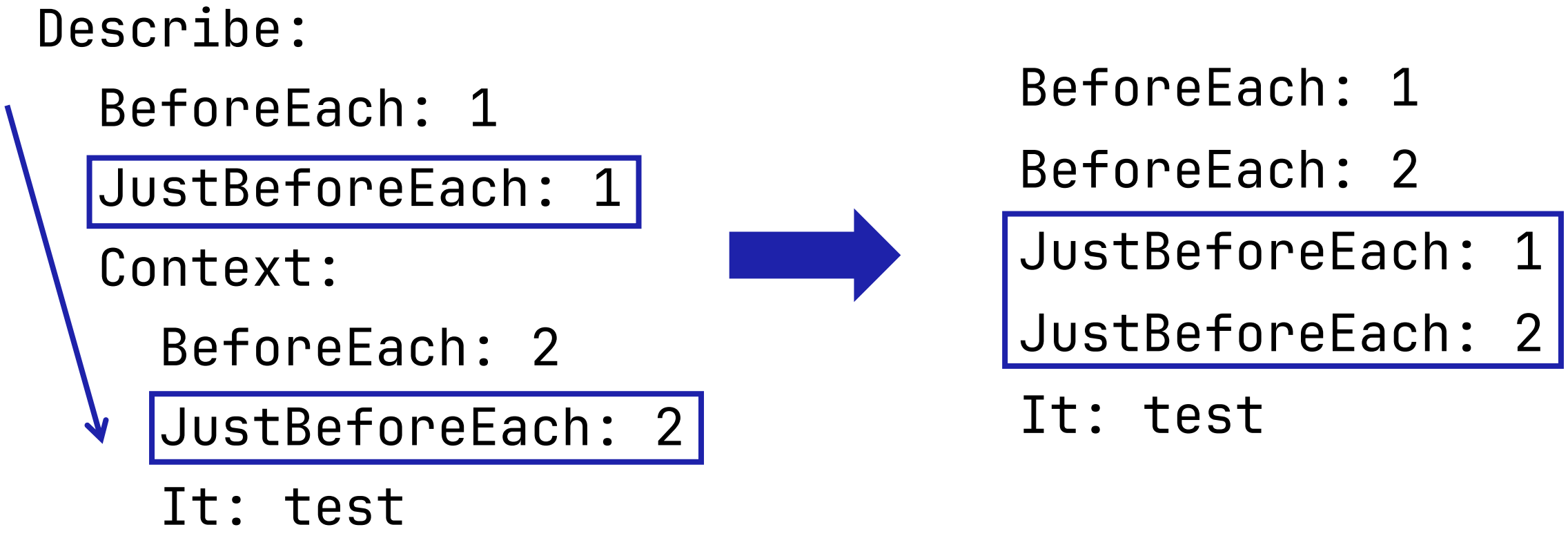
Tree Construction Phase (medium)

```
Describe:
  BeforeEach: 1
  JustBeforeEach: 1
  Context:
    BeforeEach: 2
    JustBeforeEach: 2
    It: test
```

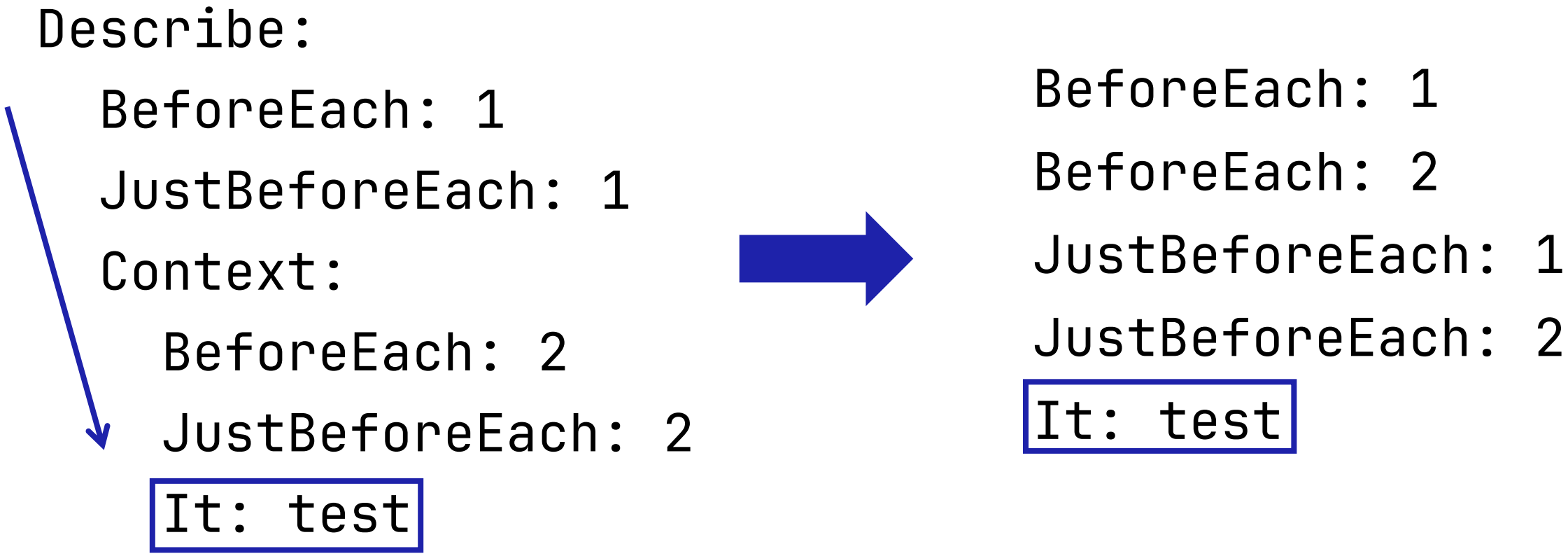
Tree Construction Phase (medium)



Tree Construction Phase (medium)



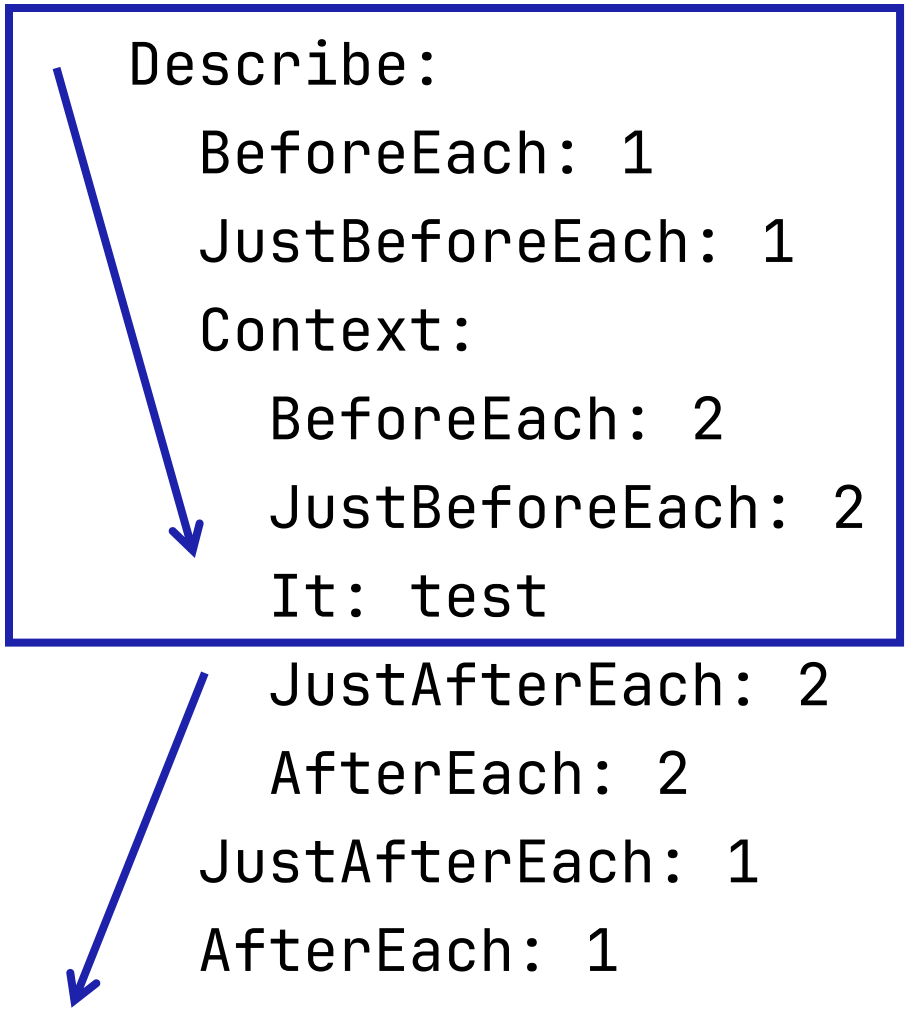
Tree Construction Phase (medium)



Tree Construction Phase (hard)

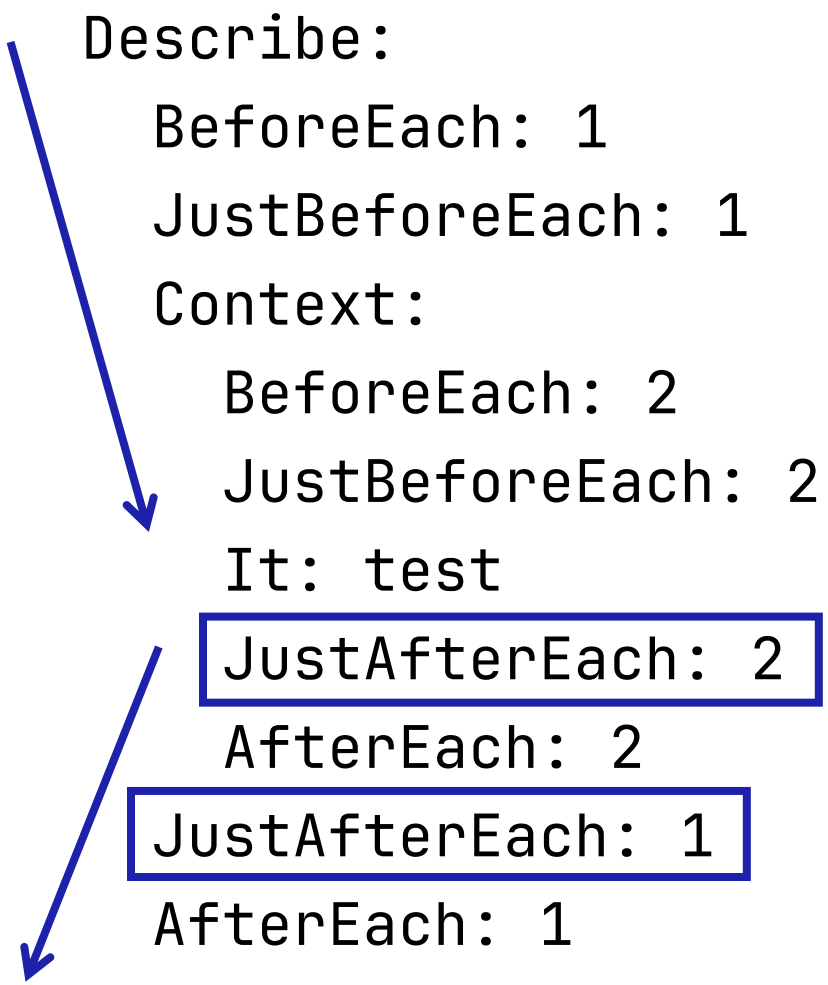
```
Describe:
  BeforeEach: 1
  JustBeforeEach: 1
  Context:
    BeforeEach: 2
    JustBeforeEach: 2
    It: test
    JustAfterEach: 2
    AfterEach: 2
JustAfterEach: 1
AfterEach: 1
```

Tree Construction Phase (hard)



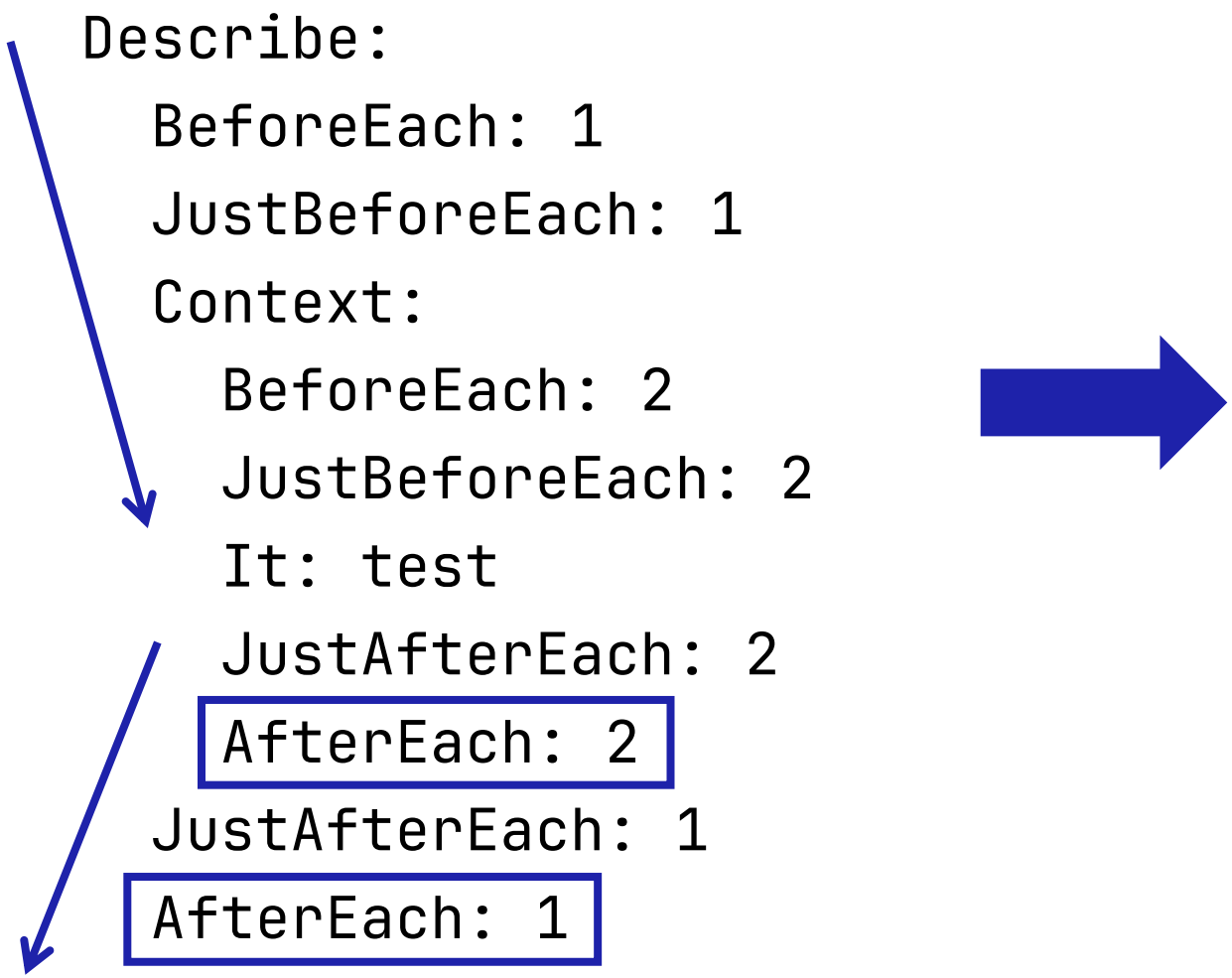

```
BeforeEach: 1
BeforeEach: 2
JustBeforeEach: 1
JustBeforeEach: 2
It: test
JustAfterEach: 2
JustAfterEach: 1
AfterEach: 2
AfterEach: 1
```

Tree Construction Phase (hard)




```
BeforeEach: 1
BeforeEach: 2
JustBeforeEach: 1
JustBeforeEach: 2
It: test
JustAfterEach: 2
JustAfterEach: 1
AfterEach: 2
AfterEach: 1
```

Tree Construction Phase (hard)




```
BeforeEach: 1
BeforeEach: 2
JustBeforeEach: 1
JustBeforeEach: 2
It: test
JustAfterEach: 2
JustAfterEach: 1
AfterEach: 2
AfterEach: 1
```

Закрепляем на реальном примере

```
type Client struct {
      conn      net.Conn
      alarmChan chan
}
```

Закрепляем на реальном примере

```
type Client struct {
      conn      net.Conn
      alarmChan chan
}

// Конструктор
func NewClient(conn net.Conn) (*Client, error) { ... }
```

Закрепляем на реальном примере

```
type Client struct {
      conn       net.Conn
      alarmChan chan
}

// Конструктор
func NewClient(conn net.Conn) (*Client, error) { ... }

// Распределение сообщений по каналам
func (c *Client) readRouter() { ... }
```

Закрепляем на реальном примере

```
type Client struct {
      conn       net.Conn
      alarmChan chan
}

// Конструктор
func NewClient(conn net.Conn) (*Client, error) { ... }

// Распределение сообщений по каналам
func (c *Client) readRouter() { ... }

// Канал с ошибками
func (c *Client) AlarmChan() <-chan string { ... }
```

Закрепляем на реальном примере

```
type Client struct {
      conn       net.Conn
      alarmChan chan
}

// Конструктор
func NewClient(conn net.Conn) (*Client, error) { ... }

// Распределение сообщений по каналам
func (c *Client) readRouter() { ... }

// Канал с ошибками
func (c *Client) AlarmChan() <-chan string { ... }
```

Закрепляем на реальном примере

```
type Client struct {
      conn       net.Conn
      alarmChan chan
}

// Конструктор
func NewClient(conn net.Conn) (*Client, error) { ... }

// Распределение сообщений по каналам
func (c *Client) readRouter() { ... }

// Канал с ошибками
func (c *Client) AlarmChan() <-chan string { ... }
```

Закрепляем на реальном примере

```
var _ = Describe("Client", func() {
	var conn1, conn2 net.Conn

	BeforeEach(func() {
		conn1, conn2 = net.Pipe()
	})

	AfterEach(func() {
		conn1.Close()
		conn2.Close()
	})

	Specify("Sunny case", func() { ... })
})
```

Закрепляем на реальном примере

```
var _ = Describe("Client", func() {
	var conn1, conn2 net.Conn

	BeforeEach(func() {
		conn1, conn2 = net.Pipe()
	})

	AfterEach(func() {
		conn1.Close()
		conn2.Close()
	})

	Specify("Sunny case", func() { ... })
})
```

Закрепляем на реальном примере

```
var _ = Describe("Client", func() {
	var conn1, conn2 net.Conn

	BeforeEach(func() {
		conn1, conn2 = net.Pipe()
	})

	AfterEach(func() {
		conn1.Close()
		conn2.Close()
	})

	Specify("Sunny case", func() { ... })
})
```

Закрепляем на реальном примере

```
var _ = Describe("Client", func() {
      var conn1, conn2 net.Conn

      BeforeEach(func() {
            conn1, conn2 = net.Pipe()
      })

      AfterEach(func() {
            conn1.Close()
            conn2.Close()
      })

      Specify("Sunny case", func() { ... })
})
```

Закрепляем на реальном примере

```
var _ = Describe("Client", func() {
        ...
        Specify("Sunny case", func() {
                client := NewClient(conn1)

                go func() {
                        _, _ := conn2.Write([]byte{"Alarm!"})
                }()

                go client.readRouter()
                ch := client.AlarmChan()
                Eventually(ch).Should(Receive(Equal("Alarm!")))
        })
})
```

Закрепляем на реальном примере

```
var _ = Describe("Client", func() {
        ...
        Specify("Sunny case", func() {
                client := NewClient(conn1)

                go func() {
                        _, _ := conn2.Write([]byte{"Alarm!"})
                }()

                go client.readRouter()
                ch := client.AlarmChan()
                Eventually(ch).Should(Receive(Equal("Alarm!")))
        })
})
```

Закрепляем на реальном примере

```
var _ = Describe("Client", func() {
        ...
        Specify("Sunny case", func() {
                client := NewClient(conn1)

                go func() {
                        _, _ := conn2.Write([]byte{"Alarm!"})
                }()

                go client.readRouter()
                ch := client.AlarmChan()
                Eventually(ch).Should(Receive(Equal("Alarm!")))
        })
})
```

Закрепляем на реальном примере

```
var _ = Describe("Client", func() {
      ...
      Specify("Sunny case", func() {
            client := NewClient(conn1)

            go func() {
                  _, _ := conn2.Write([]byte{"Alarm!"})
            }()

            go client.readRouter()
            ch := client.AlarmChan()
            Eventually(ch).Should(Receive(Equal("Alarm!")))
      })
})
```

Закрепляем на реальном примере

```
var _ = Describe("Client", func() {
      ...
      Specify("Sunny case", func() {
            client := NewClient(conn1)

            go func() {
                  _, _ := conn2.Write([]byte{"Alarm!"})
            }()

            go client.readRouter()
            ch := client.AlarmChan()
            Eventually(ch).Should(Receive(Equal("Alarm!")))
      })
})
```

Закрепляем на реальном примере

```
var _ = Describe("Client", func() {
      ...
      Specify("Sunny case", func() {
            client := NewClient(conn1)

            go func() {
                  _, _ := conn2.Write([]byte{"Alarm!"})
            }()

            go client.readRouter()
            ch := client.AlarmChan()
            Eventually(ch).Should(Receive(Equal("Alarm!")))
      })
})
```


Внимание!!!
Ginkgo
GΩmega
Спасибо за внимание

Принципы и основные понятия

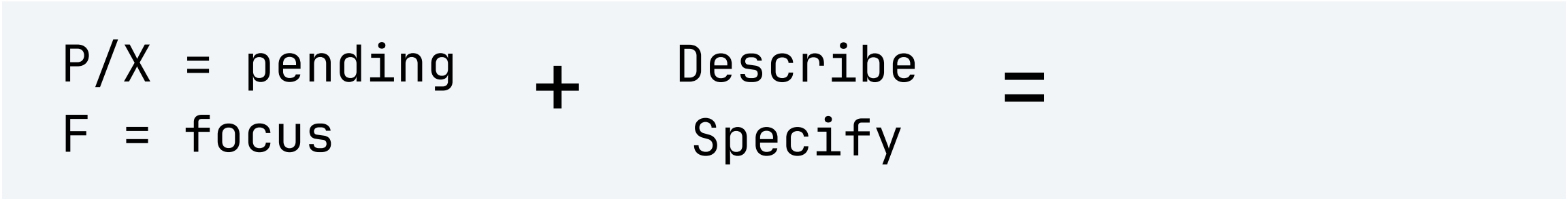
Дополнительные особенности

Существенные изменения в Ginkgo v2

Gomega

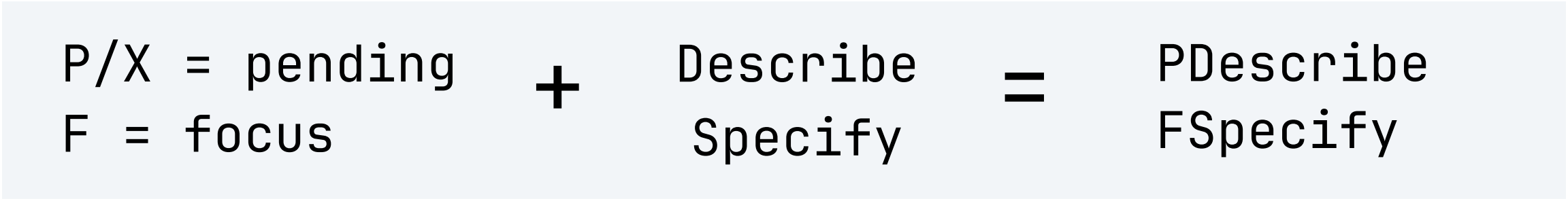
Переход на Ginkgo

Filtering Specs



Filtering Specs

Filtering Specs




```
It("should do something, if it can", func()
{
  if !someCondition {
    Skip("Special condition wasn't met.")
  }
  ...
})
```

Failures

```
gomega.RegisterFailHandler(ginkgo.Fail)
```

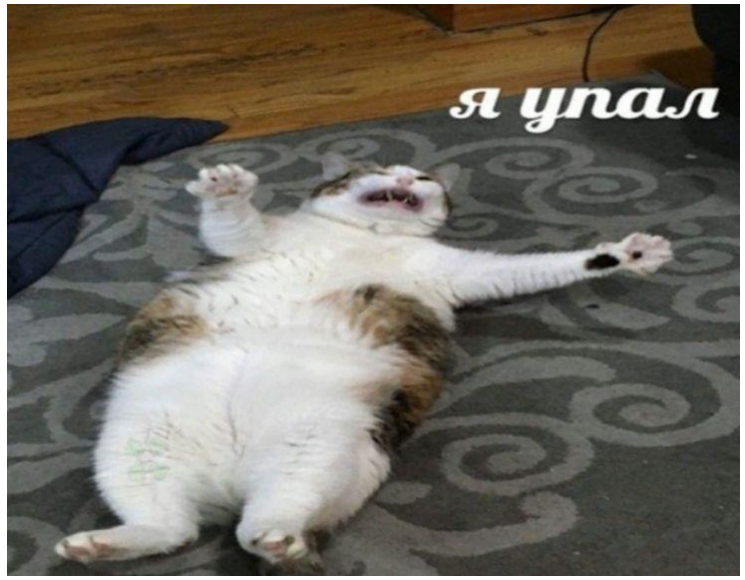
Failures

```
gomega.RegisterFailHandler(ginkgo.Fail)
```

```
It("can read books", func() {
  if book.Title == "War and Peace" && user.Age <= 3 {
    Fail("User is too young for this book")
  }
  user.Read(book)
})
```

goroutines

```
It("panics in a goroutine", func() {
  var c chan interface{}

  go func() {
    ...
    Fail("boom")
    close(c)
  }()

  <-c
})
```



goroutines

```
It("panics in a goroutine", func() {
  var c chan interface{}

  go func() {
    defer GinkgoRecover()
    Fail("boom")
    close(c)
  }()

  <-c
})
```

Принципы и основные понятия

Дополнительные особенности

Существенные изменения в Ginkgo v2

Gomega

Переход на Ginkgo

DeferCleanup

```
var ctl *gomock.Controller

BeforeEach(func() {
      ctl = gomock.NewController(GinkgoT())
      ...
})

AfterEach(func() {
      ctl.Finish()
})
```

DeferCleanup

```
var ctl *gomock.Controller

BeforeEach(func() {
      ctl = gomock.NewController(GinkgoT())
      ...
})

AfterEach(func() {
      ctl.Finish()
})
```

DeferCleanup

```
var ctl *gomock.Controller

BeforeEach(func() {
      ctl = gomock.NewController(GinkgoT())
      ...
})

AfterEach(func() {
      ctl.Finish()
})
```

DeferCleanup

```
BeforeEach(func() {
	ctl := gomock.NewController(GinkgoT())

	DeferCleanup(func() {
		ctl.Finish()
	})
})
```

DescribeTable

```
var _ = Describe("Math", func() {
  DescribeTable("addition",
    func(a, b, c int) {
      Expect(a+b).To(Equal(c))
    },
    Entry("1+2=3", 1, 2, 3),
    Entry("-1+2=1", -1, 2, 1),
    Entry("0+0=0", 0, 0, 0),
    Entry("10+100=110", 10, 100, 110),
  )
})
```

DescribeTable

```
var _ = Describe("Math", func() {
  DescribeTable("addition",
    func(a, b, c int) {
      Expect(a+b).To(Equal(c))
    },
    Entry("1+2=3", 1, 2, 3),
    Entry("-1+2=1", -1, 2, 1),
    Entry("0+0=0", 0, 0, 0),
    Entry("10+100=110", 10, 100, 110),
  )
})
```

DescribeTable

```
var _ = Describe("Math", func() {
  DescribeTable("addition",
    func(a, b, c int) {
      Expect(a+b).To(Equal(c))
    },
    Entry("1+2=3", 1, 2, 3),
    Entry("-1+2=1", -1, 2, 1),
    Entry("0+0=0", 0, 0, 0),
    Entry("10+100=110", 10, 100, 110),
  )
})
```

DescribeTable

```
var _ = Describe("Math", func() {
  DescribeTable("addition",
    func(a, b, c int) {
      Expect(a+b).To(Equal(c))
    },
    Entry("1+2=3", 1, 2, 3),
    Entry("-1+2=1", -1, 2, 1),
    Entry("0+0=0", 0, 0, 0),
    Entry("10+100=110", 10, 100, 110),
  )
})
```

DescribeTable (V2)

```
var _ = Describe("Math", func() {
  DescribeTable("addition",
    func(a, b, c int) {
      Expect(a+b).To(Equal(c))
    },
    EntryDescription("%d + %d = %d"),
    Entry(nil, 1, 2, 3),
    Entry(nil, -1, 2, 1),
    Entry(nil, 0, 0, 0),
    Entry(nil, 10, 100, 110),
  )
})
```

DescribeTable (V2)

```
var _ = Describe("Math", func() {
  DescribeTable("addition",
    func(a, b, c int) {
      Expect(a+b).To(Equal(c))
    },
    EntryDescription("%d + %d = %d"),
    Entry(nil, 1, 2, 3),
    Entry(nil, -1, 2, 1),
    Entry(nil, 0, 0, 0),
    Entry(nil, 10, 100, 110),
  )
})
```

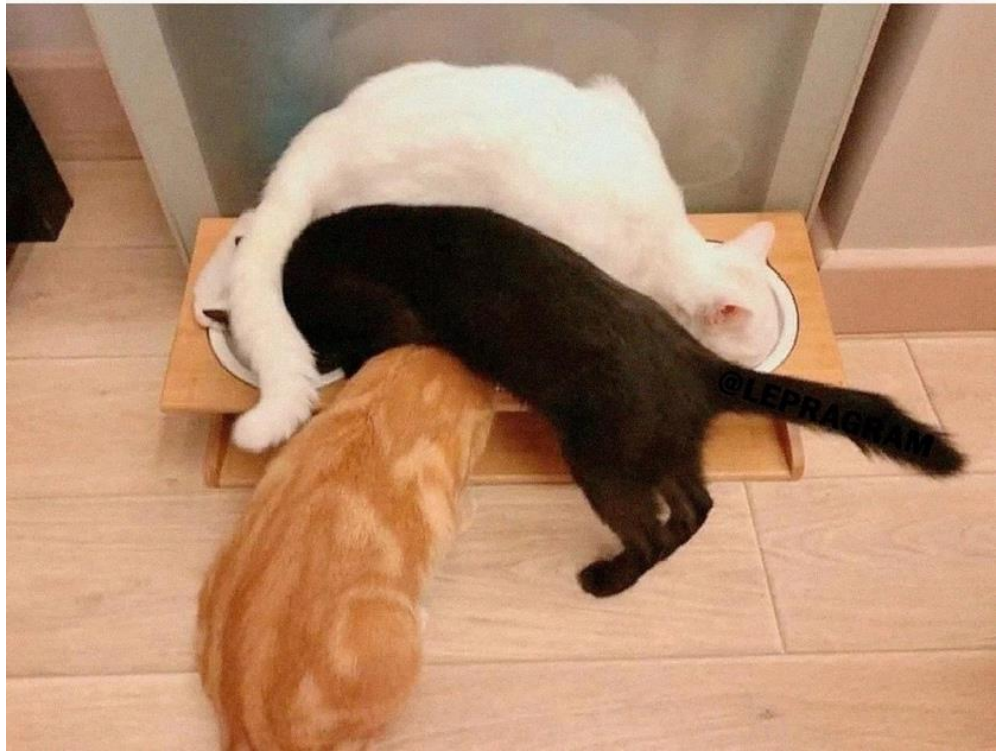
Mental Model: Phases (again)

1. Построение тестового дерева: Tree Construction Phase (Container nodes)
2. Запуск тестового дерева: Run Phase (Setup nodes and Subject nodes code)

Правила разработки тестов, которые из этого следуют:
1. Объявляй переменные в контейнерах, работай с ними в тестах
2. Ginkgo ожидает, что тесты являются независимыми

В каком порядке
тесты написаны

В каком порядке
тесты запустились



Mental Model: Phases (again)

1. Построение тестового дерева: Tree Construction Phase (Container nodes)
2. Запуск тестового дерева: Run Phase (Setup nodes and Subject nodes code)

Правила разработки тестов, которые из этого следуют:

1. Объявляй переменные в контейнерах, работай с ними в тестах
2. Ginkgo ожидает, что тесты являются независимыми
   - поэтому Ginkgo запускает их **в случайном порядке** (гарант независимости)

Mental Model: Phases (again)

1. Построение тестового дерева: Tree Construction Phase (Container nodes)
2. Запуск тестового дерева: Run Phase (Setup nodes and Subject nodes code)

Правила разработки тестов, которые из этого следуют:

1. Объявляй переменные в контейнерах, работай с ними в тестах
2. Ginkgo ожидает, что тесты являются независимыми
   - поэтому Ginkgo запускает их в случайном порядке (гарант независимости)
   - если у вас всё так — легко параллелизуйте запуск (`ginkgo -p`)

Mental Model: Phases (again)

1. Построение тестового дерева: Tree Construction Phase (Container nodes)
2. Запуск тестового дерева: Run Phase (Setup nodes and Subject nodes code)

Правила разработки тестов, которые из этого следуют:

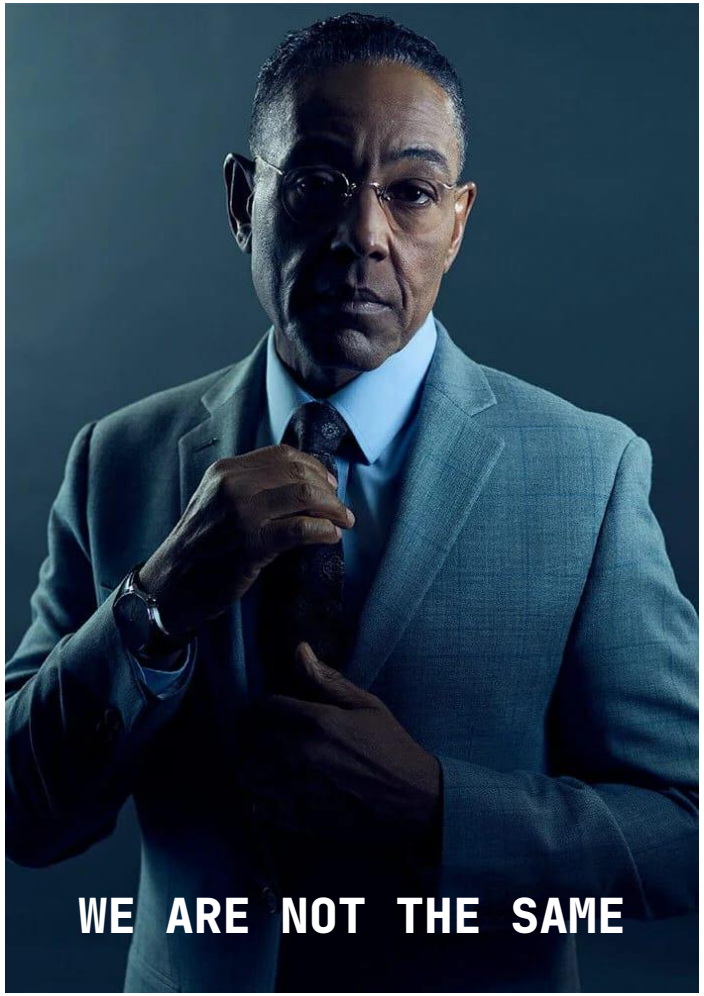
1. Объявляй переменные в контейнерах, работай с ними в тестах
2. Ginkgo ожидает, что тесты являются независимыми
   - поэтому Ginkgo запускает их в случайном порядке (гарант независимости)
   - если у вас всё так — легко параллелизуйте запуск (`ginkgo -p`)
   - если у вас не так — вам помогут декораторы

Decorators

```
Describe("a focused container", Focus, func() {
   ....
})
```

Python decorators

```
def stars(input_func):
    def output_func():
        print("********************")
        input_func()
    return output_func

@stars
def hello():
    print("YADRO Golang Meetup")

if __name__ == "__main__":
    hello()
```



Decorators

```
Describe("a focused container", Focus, func() {
   ....
})
```

Focus/Pending — аналогично использованию вариантов FX или PX

Decorators

```
Describe("a focused container", Focus, func() {
   ....
})
```

Focus/Pending — аналогично использованию вариантов FX или PX

Ordered — тесты будут выполняться в порядке появления в коде

Decorators

```
Describe("a focused container", Focus, func() {
   ....
})
```

Focus/Pending — аналогично использованию вариантов FX или PX

Ordered — тесты будут выполняться в порядке появления в коде

Serial — не запускать тесты в параллель с другими

Decorators

```
Describe("a focused container", Focus, func() {
   ....
})
```

Focus/Pending — аналогично использованию вариантов FX или PX

Ordered — тесты будут выполняться в порядке появления в коде

Serial — не запускать тесты в параллель с другими

FlakeAttempts(N) — запускать тест N раз, пока не пройдёт успешно

Decorators

```
Describe("a focused container", Focus, func() {
   ....
})
```

Focus/Pending — аналогично использованию вариантов FX или PX

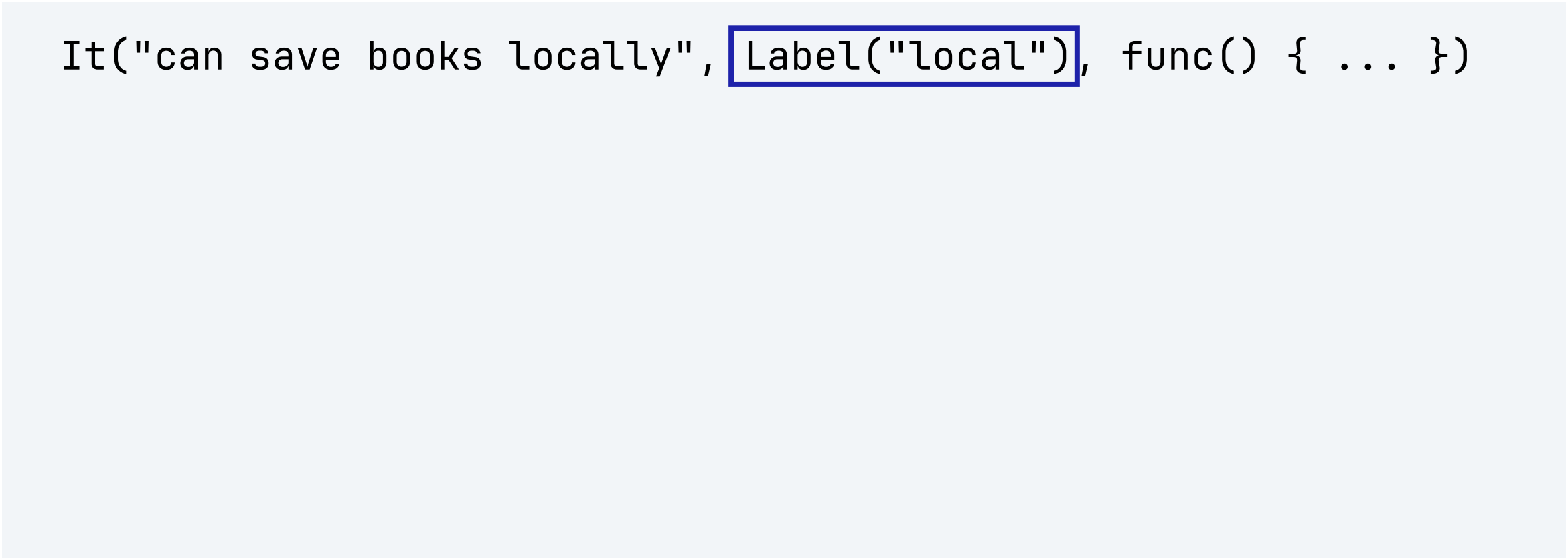
Ordered — тесты будут выполняться в порядке появления в коде

Serial — не запускать тесты в параллель с другими

FlakeAttempts(N) — запускать тест N раз, пока не пройдёт успешно

Label — поставить метку на блок

Decorators (Label)



Decorators (Label)

```
It("can save books locally", Label("local"), func() { ... })

It("is labelled", Label("first label", "second label"),
      func() { ... })
It("is labelled", Label("first label"), Label("second label"),
      func() { ... })
```

Decorators (Label)

```
It("can save books locally", Label("local"), func() { ... })

It("is labelled", Label("first label", "second label"),
      func() { ... })
It("is labelled", Label("first label"), Label("second label"),
      func() { ... })

It("can fetch a list of books by shelf",
      Label("API:Shelf", "Readiness:Alpha"), func() { ... })
```

Reporting Infrastructure and Profiling

```
> ginkgo outline <файл> // описание тестов
```

Reporting Infrastructure and Profiling

```
> ginkgo outline <файл> // описание тестов

Name,Text,Start,End,Spec,Focused,Pending,Labels
Describe,Formulas,795,11183,false,false,false,""
Context,Computation,1562,3819,false,false,false,""
It,RuPortID,1603,2484,true,false,false,""
...
```

Reporting Infrastructure and Profiling

```
> ginkgo --json-report=report.json --junit-report=report.xml
```

Reporting Infrastructure and Profiling

```
> ginkgo --json-report=report.json --junit-report=report.xml

<?xml ...>
  <testsuites tests="79" failures="0" time="0.009159434">
      ...
```

Reporting Infrastructure and Profiling

```
> ginkgo --json-report=report.json --junit-report=report.xml

<?xml ...>
  <testsuites ...>
      <testsuite name="Formulas Suite"
package="/home/bondarenko/Yadro/libs/utils/formulas" tests="79"
failures="0" time="0.0091594" timestamp="2024-09-16T02:24:40">
          ...
```

Reporting Infrastructure and Profiling

```
> ginkgo --json-report=report.json --junit-report=report.xml

<?xml ...>
  <testsuites ...>
      <testsuite ...>
          <properties>
              <property name="SuiteSucceeded" value="true">
               </property>
              <property name="ParallelTotal" value="1">
              </property>
              ...
          </properties>
          ...
```

Reporting Infrastructure and Profiling

```
> ginkgo --json-report=report.json --junit-report=report.xml

<?xml ...>
  <testsuites ...>
      <testsuite ...>
          <properties>...</properties>
          <testcase ...>
          </testcase>
          ...
```

Reporting Infrastructure and Profiling

```
> ginkgo --json-report=report.json --junit-report=report.xml

<?xml ...>
  <testsuites ...>
      <testsuite ...>
          <properties>...</properties>
          <testcase name="[It] Formulas computation RuPortID"
classname="Formulas Suite" status="passed" time="0.000159822">
              <system-err>...</system-err>
          </testcase>
          ...
```

Reporting Infrastructure and Profiling

```
> ginkgo --json-report=report.json --junit-report=report.xml

<?xml ...>
  <testsuites ...>
      <testsuite ...>
          <properties>...</properties>
          <testcase name="[It] Formulas computation RuPortID"
classname="Formulas Suite" status="passed" time="0.000159822">
              <system-err>...</system-err>
          </testcase>
          ...
```

Reporting Infrastructure and Profiling

```
> ginkgo --json-report=report.json --junit-report=report.xml

<?xml ...>
  <testsuites ...>
      <testsuite ...>
          <properties>...</properties>
          <testcase ...>
              <system-err>
Enter [It] RuPortID - formulas_test.go:54 @ 09/16/24 02:24:40.902
Exit [It] RuPortID - formulas_test.go:54 @ 09/16/24 02:24:40.902 (0s)
              </system-err>
          </testcase>
          ...
```

Reporting Infrastructure and Profiling

```
> ginkgo --json-report=report.json --junit-report=report.xml

<?xml ...>
  <testsuites ...>
      <testsuite ...>
          <properties>...</properties>
          <testcase ...>
              <system-err>
Enter [It] RuPortID - formulas_test.go:54 @ 09/16/24 02:24:40.902
Exit [It] RuPortID - formulas_test.go:54 @ 09/16/24 02:24:40.902 (0s)
              </system-err>
          </testcase>
          ...
```

Reporting Infrastructure and Profiling

```
Profiling:
ginkgo --race // to analyze race conditions
ginkgo --cover // to compute code coverage
ginkgo --vet // to evaluate and vet your code
ginkgo --cpuprofile // to profile CPU performance
ginkgo --memprofile // to profile memory usage
ginkgo --blockprofile // to profile blocking goroutines
ginkgo --mutexprofile // to profile locking around mutexes
```

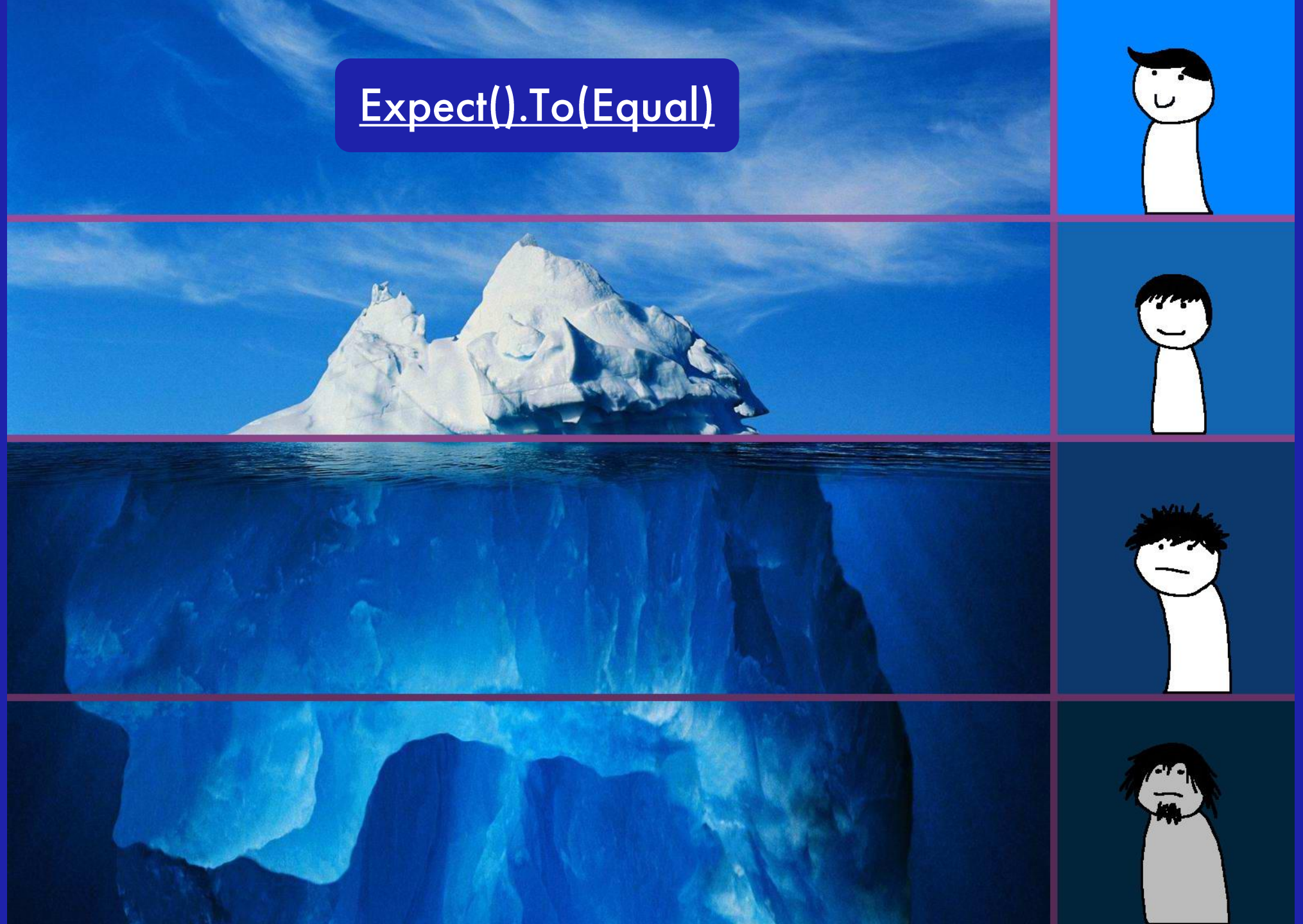
Принципы и основные понятия

Дополнительные особенности

Существенные изменения в Ginkgo v2

Gomega

Переход на Ginkgo


Expect().To(Equal)

Gomega usage

```
Ω(ACTUAL).Should(MATCHER)
Ω(ACTUAL).ShouldNot(MATCHER)

Expect(ACTUAL).To(MATCHER)
Expect(ACTUAL).NotTo(MATCHER)
Expect(ACTUAL).ToNot(MATCHER)
```

Gomega usage

```
Ω(ACTUAL).Should(MATCHER)
Ω(ACTUAL).ShouldNot(MATCHER)

Expect(ACTUAL).To(MATCHER)
Expect(ACTUAL).NotTo(MATCHER)
Expect(ACTUAL).ToNot(MATCHER)
```

Gomega usage

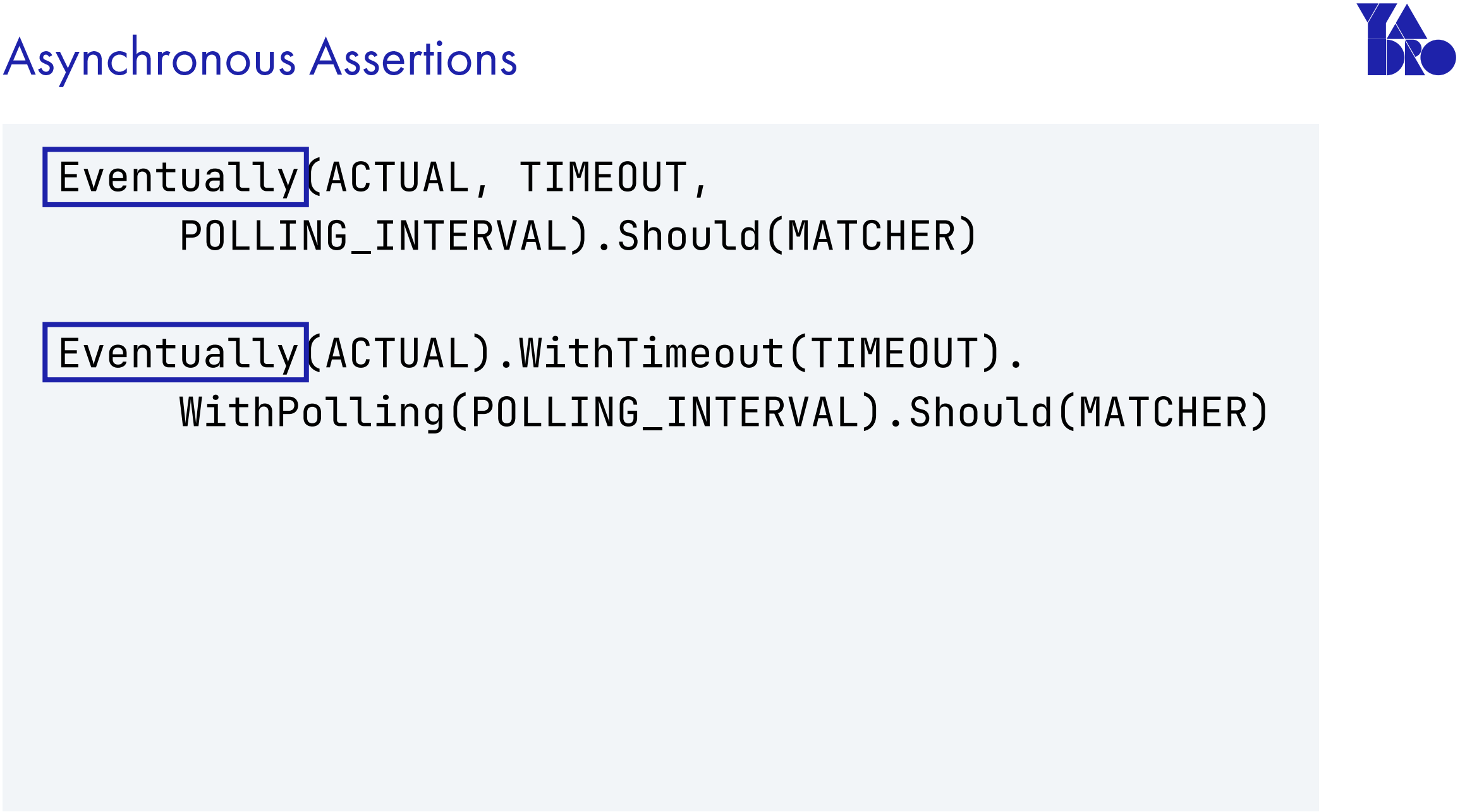
```
Ω(ACTUAL).Should(MATCHER)
Ω(ACTUAL).ShouldNot(MATCHER)

Expect(ACTUAL).To(MATCHER)
Expect(ACTUAL).NotTo(MATCHER)
Expect(ACTUAL).ToNot(MATCHER)
```

Gomega usage

```
Ω(ACTUAL).Should(MATCHER)
Ω(ACTUAL).ShouldNot(MATCHER)

Expect(ACTUAL).To(MATCHER)
Expect(ACTUAL).NotTo(MATCHER)
Expect(ACTUAL).ToNot(MATCHER)
```


Expect().To(Equal)
Asynchronous Assertions

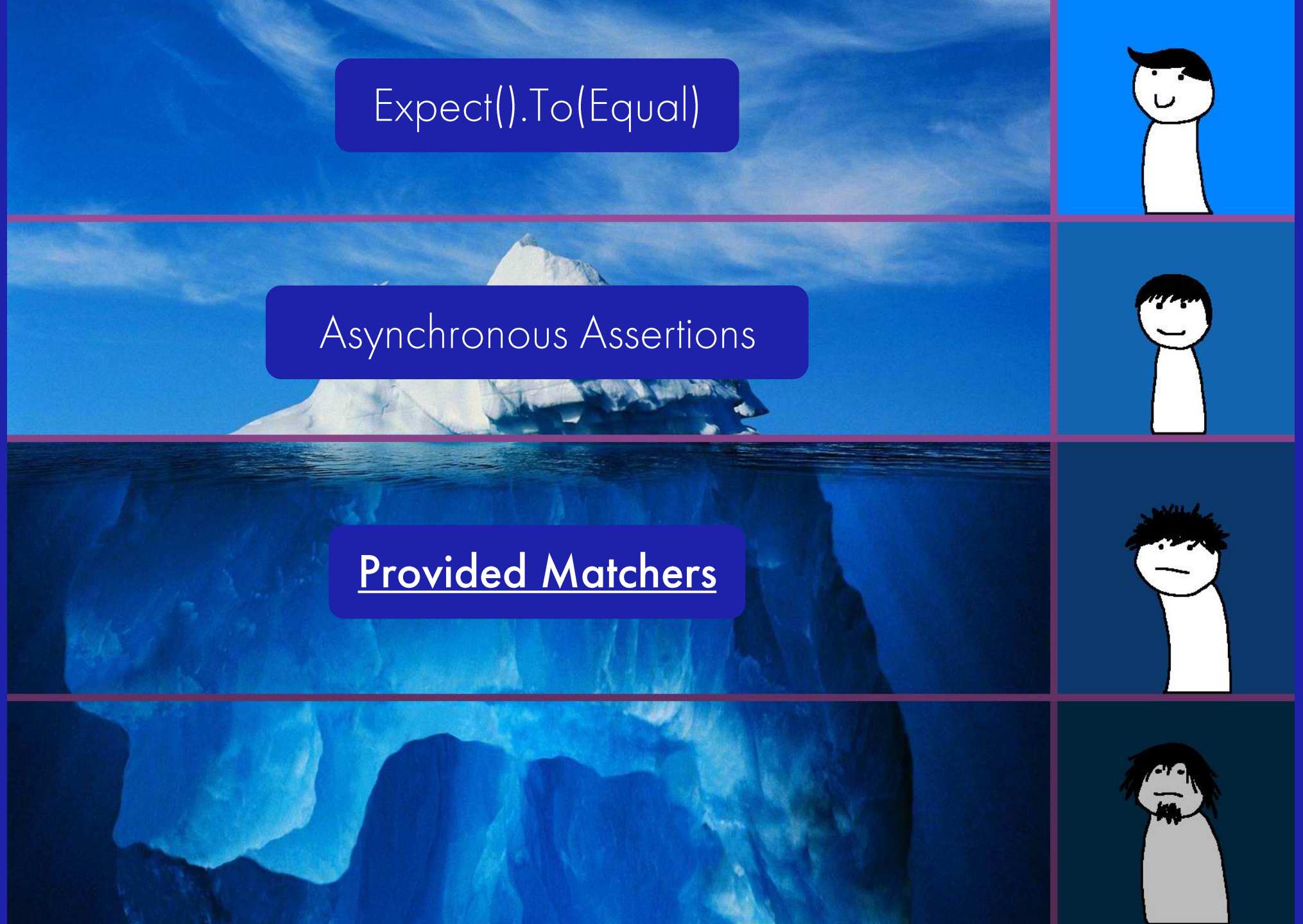
Asynchronous Assertions



```
Eventually(ACTUAL, TIMEOUT,
      POLLING_INTERVAL).Should(MATCHER)

Eventually(ACTUAL).WithTimeout(TIMEOUT).
      WithPolling(POLLING_INTERVAL).Should(MATCHER)
```

Asynchronous Assertions

```
Eventually(ACTUAL, TIMEOUT,
     POLLING_INTERVAL).Should(MATCHER)

Eventually(ACTUAL).WithTimeout(TIMEOUT).
     WithPolling(POLLING_INTERVAL).Should(MATCHER)

Consistently(ACTUAL, DURATION,
     POLLING_INTERVAL).Should(MATCHER)

Consistently(ACTUAL).WithTimeout(DURATION).
     WithPolling(POLLING_INTERVAL).Should(MATCHER)
```


Expect().To(Equal)
Asynchronous Assertions
Provided Matchers

Provided Matchers (common)

Equivalence:

```
Equal(EXPECTED)
and similar
```

Provided Matchers (common)

Equivalence:

```
Equal(EXPECTED)
and similar
```

Errors:

```
MatchError(EXPECTED)
Succeed()
HaveOccurred()
```

Provided Matchers (common)

Equivalence:

```
Equal(EXPECTED)
and similar
```

Errors:

```
MatchError(EXPECTED)
Succeed()
HaveOccurred()
```

Presence:

```
BeNil()
BeZero()
```

Provided Matchers (common)

Equivalence:

```
Equal(EXPECTED)
and similar
```

Errors:

```
MatchError(EXPECTED)
Succeed()
HaveOccurred()
```

Presence:

```
BeNil()
BeZero()
```

Panic:

```
Panic()
PanicWith(VALUE)
```

Provided Matchers (common)

Equivalence:

```
Equal(EXPECTED)
and similar
```

Errors:

```
MatchError(EXPECTED)
Succeed()
HaveOccurred()
```

Presence:

```
BeNil()
BeZero()
```

Panic:

```
Panic()
PanicWith(VALUE)
```

Truthiness:

```
BeTrue()
BeFalse()
```

Provided Matchers (common)

Equivalence:

```
Equal(EXPECTED)
and similar
```

Errors:

```
MatchError(EXPECTED)
Succeed()
HaveOccurred()
```

Presence:

```
BeNil()
BeZero()
```

Panic:

```
Panic()
PanicWith(VALUE)
```

Truthiness:

```
BeTrue()
BeFalse()
```

Channels:

```
BeClosed()
Receive()
BeSent(VALUE)
```

Provided Matchers (composite types and strings)

Collections:

```
BeEmpty()
HaveLen(INT)
HaveCap(INT)
ContainElement(ELEMENT)
ContainElements(ELEMENT1, ...)
BeElementOf(ELEMENT1, ...)
ConsistOf(ELEMENT1, ...)
HaveEach(ELEMENT)
```

Provided Matchers (composite types and strings)

Collections:

```
BeEmpty()
HaveLen(INT)
HaveCap(INT)
ContainElement(ELEMENT)
ContainElements(ELEMENT1, ...)
BeElementOf(ELEMENT1, ...)
ConsistOf(ELEMENT1, ...)
HaveEach(ELEMENT)
```

Strings:

```
ContainSubstring(STRING, ARGS...)
HavePrefix(STRING, ARGS...)
HaveSuffix(STRING, ARGS...)
MatchRegexp(STRING, ARGS...)
MatchJSON(EXPECTED)
MatchXML(EXPECTED)
MatchYAML(EXPECTED)
```

Provided Matchers (composite types and strings)

Collections:

```
BeEmpty()
HaveLen(INT)
HaveCap(INT)
ContainElement(ELEMENT)
ContainElements(ELEMENT1, ...)
BeElementOf(ELEMENT1, ...)
ConsistOf(ELEMENT1, ...)
HaveEach(ELEMENT)
```

Strings:

```
ContainSubstring(STRING, ARGS...)
HavePrefix(STRING, ARGS...)
HaveSuffix(STRING, ARGS...)
MatchRegexp(STRING, ARGS...)
MatchJSON(EXPECTED)
MatchXML(EXPECTED)
MatchYAML(EXPECTED)
```

Structs:

```
HaveField(KEY, VALUE)
```

Provided Matchers (composite types and strings)

Collections:

```
BeEmpty()
HaveLen(INT)
HaveCap(INT)
ContainElement(ELEMENT)
ContainElements(ELEMENT1, ...)
BeElementOf(ELEMENT1, ...)
ConsistOf(ELEMENT1, ...)
HaveEach(ELEMENT)
```

Strings:

```
ContainSubstring(STRING, ARGS...)
HavePrefix(STRING, ARGS...)
HaveSuffix(STRING, ARGS...)
MatchRegexp(STRING, ARGS...)
MatchJSON(EXPECTED)
MatchXML(EXPECTED)
MatchYAML(EXPECTED)
```

Structs:

```
HaveField(KEY, VALUE)
```

Maps:

```
HaveKey(KEY)
HaveKeyWithValue(KEY, VALUE)
```

Provided Matchers (specific)

Numbers and Times:

```
BeNumerically(COMPARATOR, EXPECTED, <THRESHOLD>)
BeTemporally(COMPARATOR, EXPECTED_TIME, <THRESHOLD_DURATION>)

COMPARATOR = "=="|"~"|">"|">="|"<"|"<="
```

Provided Matchers (specific)

Numbers and Times:

```
BeNumerically(COMPARATOR, EXPECTED, <THRESHOLD>)
BeTemporally(COMPARATOR, EXPECTED_TIME, <THRESHOLD_DURATION>)

COMPARATOR = "=="|"~"|">"|">="|"<"|"<="
```

HTTP:

```
HaveHTTPStatus(EXPECTED1, EXPECTED2, ...)
HaveHTTPBody(EXPECTED)
HaveHTTPHeaderWithValue(EXPECTED, VALUE)
```

Provided Matchers (specific)

Numbers and Times:

```
BeNumerically(COMPARATOR, EXPECTED, <THRESHOLD>)
BeTemporally(COMPARATOR, EXPECTED_TIME, <THRESHOLD_DURATION>)

COMPARATOR = "=="|"~"|">"|">="|"<"|"<="
```

HTTP:

```
HaveHTTPStatus(EXPECTED1, EXPECTED2, ...)
HaveHTTPBody(EXPECTED)
HaveHTTPHeaderWithValue(EXPECTED, VALUE)
```

Files:

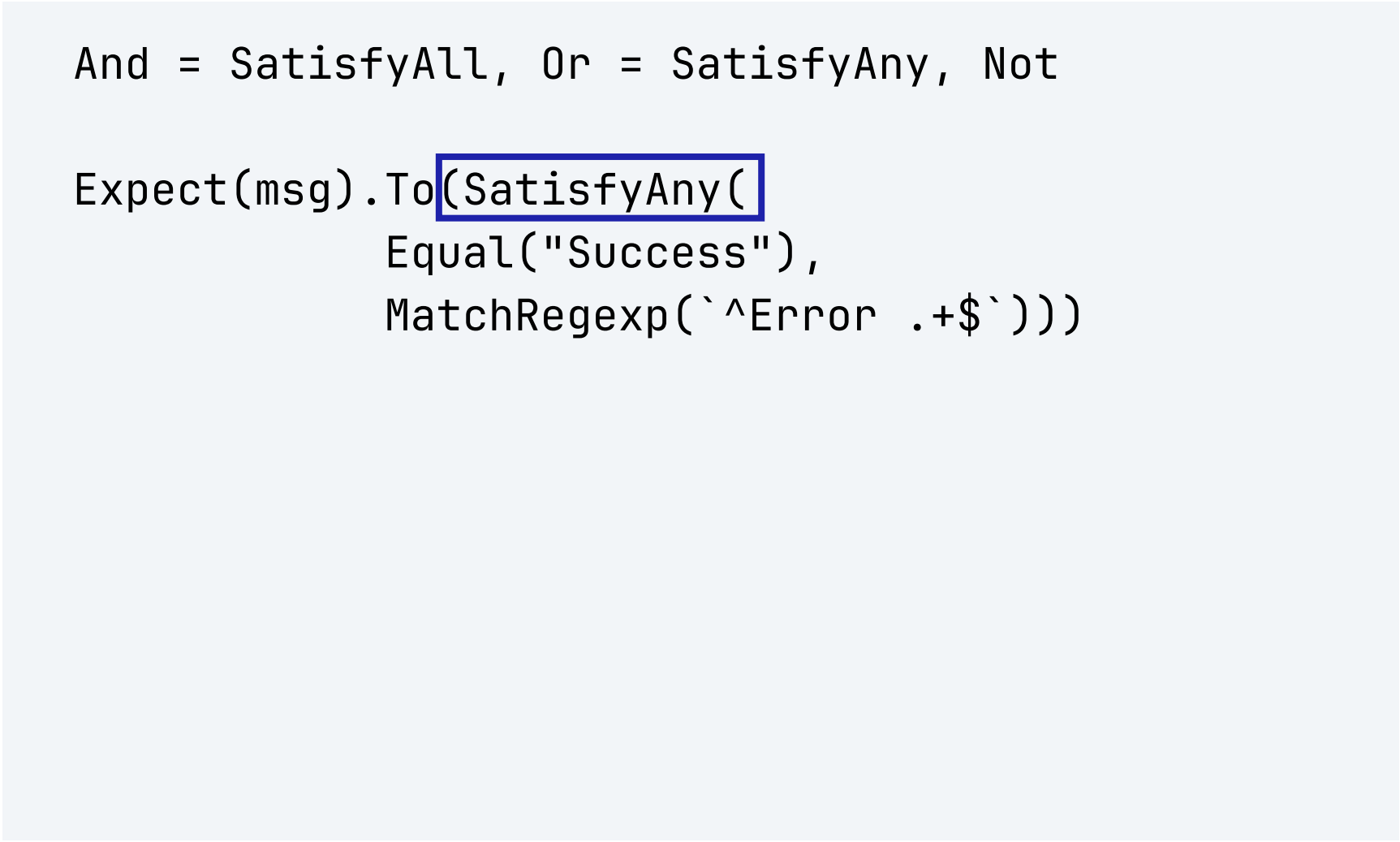
```
BeAnExistingFile()
BeARegularFile()
BeADirectory()
```


Expect().To(Equal)
Asynchronous Assertions
Provided Matchers
Your Own Matchers

Your Own Matchers (from existing)

```
And = SatisfyAll, Or = SatisfyAny, Not
```

Your Own Matchers (from existing)



Your Own Matchers (from existing)

```
And = SatisfyAll, Or = SatisfyAny, Not

Expect(msg).To(SatisfyAny(
            Equal("Success"),
            MatchRegexp(`^Error .+$`)))

func BeBetween(min, max int) GomegaMatcher {
  return SatisfyAll(
    BeNumerically(">", min),
    BeNumerically("<", max))
}

Ω(number).Should(BeBetween(0, 10))
```

Your Own Matchers (transform actual value)

```
WithTransform(TRANSFORM, MATCHER)
```

Your Own Matchers (transform actual value)

```
WithTransform(TRANSFORM, MATCHER)

func HaveColor(c Color) GomegaMatcher {
    return WithTransform(func(e Element) Color {
        return e.Color
    }, Equal(c))
}

Ω(element).Should(HaveColor(BLUE)))
```

Your Own Matchers (transform actual value)

```
WithTransform(TRANSFORM, MATCHER)

func HaveColor(c Color) GomegaMatcher {
    return WithTransform(func(e Element) Color {
        return e.Color
    }, Equal(c))
}

Ω(element).Should(HaveColor(BLUE))
```

Your Own Matchers (custom)

```
type GomegaMatcher interface {
    Match(actual interface{}) (success bool, err error)
    FailureMessage(actual interface{}) (message string)
    NegatedFailureMessage(actual interface{}) (
        message string,
    )
}
```

Your Own Matchers (custom)

```
type GomegaMatcher interface {
    Match(actual interface{}) (success bool, err error)
    FailureMessage(actual interface{}) (message string)
    NegatedFailureMessage(actual interface{}) (
        message string,
    )
}
```

Your Own Matchers (custom)

```
type GomegaMatcher interface {
    Match(actual interface{}) (success bool, err error)
    FailureMessage(actual interface{}) (message string)
    NegatedFailureMessage(actual interface{}) (
        message string,
    )
}
```

Your Own Matchers (custom)

```
type Colored interface {
    Color() Color
}
```

Your Own Matchers (custom)

```
type Colored interface {
    Color() Color
}

type colorMatcher struct {
    wantColor Color
}
```

Your Own Matchers (custom)

```
type Colored interface {
    Color() Color
}

type colorMatcher struct {
    wantColor Color
}

func Color(expected interface{}) types.GomegaMatcher {
    return &colorMatcher{wantColor: expected}
}
```

Your Own Matchers (custom)

```
type Colored interface {
    Color() Color
}

type colorMatcher struct {
    wantColor Color
}

func Color(expected interface{}) types.GomegaMatcher {
    return &colorMatcher{wantColor: expected}
}
```

Match 

FailureMessage 

NegatedFailureMessage 

Your Own Matchers (custom)

```
func (m colorMatcher) Match(actual interface{}) (
    success bool,
    err error,
) {
    got, ok := actual.(Colored)
    if !ok {
        return false,
            fmt.Errorf("expected to have Colored interface")
    }

    return Equal(m.wantColor).Match(got.Color())
}
```

Match 

FailureMessage 

NegatedFailureMessage 

Your Own Matchers (custom)

```
func (m colorMatcher) Match(actual interface{}) (
    success bool,
    err error,
) {
    got, ok := actual.(Colored)
    if !ok {
        return false,
            fmt.Errorf("expected to have Colored interface")
    }

    return Equal(m.wantColor).Match(got.Color())
}
```

Match 

FailureMessage 

NegatedFailureMessage 

Your Own Matchers (custom)

```
func (m colorMatcher) FailureMessage(actual interface{}) (
    message string
) {
    return format.Message(actual, "to be colored in", m.wantColor)
}

func (m colorMatcher) NegatedFailureMessage(actual interface{}) (
    message string
) {
    return format.Message(actual, "not to be colored in", m.wantColor)
}
```

Match ✚ | FailureMessage ✚ | NegatedFailureMessage ⊘

Your Own Matchers (custom)

```
func (m colorMatcher) FailureMessage(actual interface{}) (
    message string
) {
    return format.Message(actual, "to be colored in", m.wantColor)
}

func (m colorMatcher) NegatedFailureMessage(actual interface{}) (
    message string
) {
    return format.Message(actual, "not to be colored in", m.wantColor)
}
```

Match ✚ FailureMessage ✚ NegatedFailureMessage ✚


THE
GΩmega

Gomega packages

ghttp: Testing HTTP Clients

Gomega packages

ghttp: Testing HTTP Clients

gbytes: Testing Streaming Buffers

Gomega packages

ghttp: Testing HTTP Clients

gbytes: Testing Streaming Buffers

gexec: Testing External Processes

Gomega packages

ghttp: Testing HTTP Clients

gbytes: Testing Streaming Buffers

gexec: Testing External Processes

gstruct: Testing Complex Data Types

Gomega packages

ghttp: Testing HTTP Clients

gbytes: Testing Streaming Buffers

gexec: Testing External Processes

gstruct: Testing Complex Data Types

gmeasure: Benchmarking Code

Gomega packages

ghttp: Testing HTTP Clients

gbytes: Testing Streaming Buffers

gexec: Testing External Processes

gstruct: Testing Complex Data Types

gmeasure: Benchmarking Code

gleak: Finding Leaked Goroutines

Принципы и основные понятия

Дополнительные особенности

Существенные изменения в Ginkgo v2

Gomega

Переход на Ginkgo

Установка и создание тестового набора

```
go install -mod=mod github.com/onsi/ginkgo/v2/ginkgo
go get github.com/onsi/gomega/...
```

Установка и создание тестового набора

```
go install -mod=mod github.com/onsi/ginkgo/v2/ginkgo
go get github.com/onsi/gomega/...

ginkgo bootstrap
ginkgo generate <test_suite_name>
ginkgo
```

Точка входа

```
// library_suite_test.go
// Generated by ginkgo bootstrap

package library_test

import (
    "testing"

    . "github.com/onsi/ginkgo/v2"
    . "github.com/onsi/gomega"
)

func TestGinkgo(t *testing.T) {
    RegisterFailHandler(Fail)
    RunSpecs(t, "Library Suite")
}
```

Точка входа

```
// library_suite_test.go
// Generated by ginkgo bootstrap

package library_test

import (
    "testing"

    . "github.com/onsi/ginkgo/v2"
    . "github.com/onsi/gomega"
)

func TestGinkgo(t *testing.T) {
    RegisterFailHandler(Fail)
    RunSpecs(t, "Library Suite")
}
```

Точка входа

```
// library_suite_test.go
// Generated by ginkgo bootstrap

package library_test

import (
    "testing"

    . "github.com/onsi/ginkgo/v2"
    . "github.com/onsi/gomega"
)

func TestGinkgo(t *testing.T) {
    RegisterFailHandler(Fail)
    RunSpecs(t, "Library Suite")
}
```

Точка входа

```
// library_suite_test.go
// Generated by ginkgo bootstrap

package library_test

import (
    "testing"

    . "github.com/onsi/ginkgo/v2"
    . "github.com/onsi/gomega"
)

func TestGinkgo(t *testing.T) {
    RegisterFailHandler(Fail)
    RunSpecs(t, "Library Suite")
}
```

```
// book_test.go
// Generated by ginkgo generate book

package library_test

import (
    . "github.com/onsi/ginkgo/v2"
    . "github.com/onsi/gomega"

    <path to book package>
)

var _ = Describe("Book", func() {
    // Пишите тесты здесь!
})
```

Сопоставление testify/suite и Ginkgo

testify/suite interfaces	Ginkgo nodes
SetupSuite	BeforeSuite
TearDownSuite	AfterSuite
SetupTest/SetupSubTest	BeforeEach
BeforeTest	JustBeforeEach
AfterTest	JustAfterEach
TearDownTest/TearDownSubTest	AfterEach
Test...	It/Specify

Сопоставление testify/suite и Ginkgo

testify/suite interfaces	Ginkgo nodes
SetupSuite	BeforeSuite
TearDownSuite	AfterSuite
SetupTest/SetupSubTest	BeforeEach
BeforeTest	JustBeforeEach
AfterTest	JustAfterEach
TearDownTest/TearDownSubTest	AfterEach
Test...	It/Specify

Сопоставление testify/suite и Ginkgo

testify/suite interfaces	Ginkgo nodes
SetupSuite	BeforeSuite
TearDownSuite	AfterSuite
SetupTest/SetupSubTest	BeforeEach
BeforeTest	JustBeforeEach
AfterTest	JustAfterEach
TearDownTest/TearDownSubTest	AfterEach
Test...	It/Specify

Ginkgo + testing/testify

```
package foo_test

import (
  . "github.com/onsi/ginkgo/v2"

  "github.com/stretchr/testify/assert"
)

var _ = Describe(func("foo") {
  It("should testify to its correctness", func(){
    assert.Equal(GinkgoT(), foo{}.Name(), "foo")
  })
})
```

Gomega + testing/testify

```
import (
   . "github.com/onsi/gomega"
)

func TestFarmHasCow(t *testing.T) {
  g := NewWithT(t)

  f := farm.New([]string{"Cow", "Horse"})
  g.Expect(f.HasCow()).To(BeTrue(), "Farm should have cow")
}
```

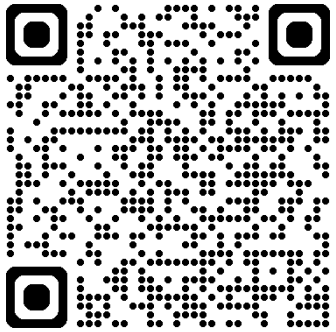
Gomega + testing/testify

```
import (
  . "github.com/onsi/gomega"
)

func TestFarmHasCow(t *testing.T) {
  g := NewWithT(t)

  f := farm.New([]string{"Cow", "Horse"})
  g.Expect(f.HasCow()).To(BeTrue(), "Farm should have cow")
}
```

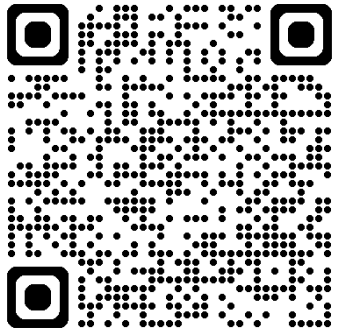
Как упростить работу? (IDE Integration)

Для CLI: перезапуск тестов при изменениях — `ginkgo watch`

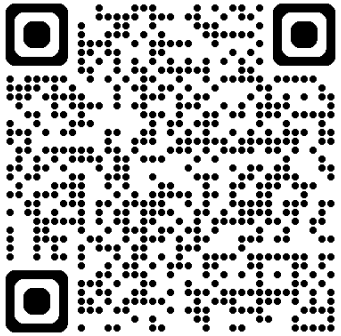
VSCode (неофициальный)



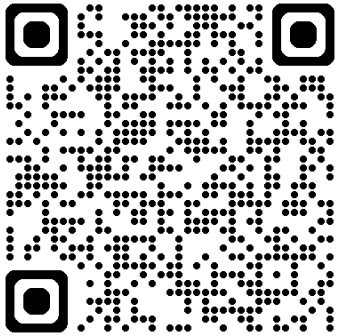
GoLand



VSCode (офиц., но не обновляется)



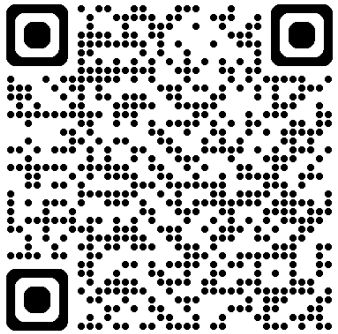
Sublime



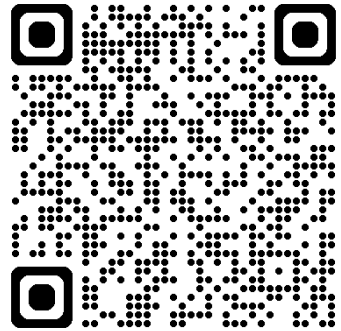
Спасибо за внимание!

Monospaced font: **JetBrains Mono** (OFL-1.1)

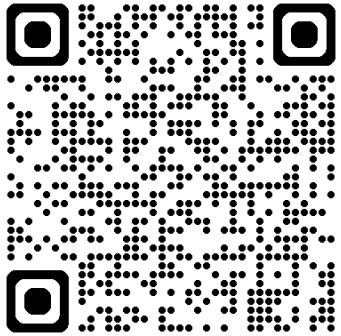
Официальная документация Ginkgo



Миграция с Ginkgo v1 на Ginkgo v2



Официальная документация Gomega



Контакты:
b.bondarenko@yadro.com
TG: @somebodyonestoldme