ОСНОВЫ WEB-РАЗРАБОТКИ

Лекция 6. Асинхронное программирование на Golang

Курс читают:

Шульман В.Д. @ShtuzerVD
Пелевина Т.В. @anivelat
Шабанов В.В. @ZeroHug

ПЛАН ЛЕКЦИИ

- Обсудить предстоящий рубежный контроль
- Синхронное и асинхронное программирование
- Конкурентность и параллелизм
- Асинхронное программирование на Golang

РК №1

Рубежный контроль №1. Разработка WEB-сервера на Golang

Рубежный контроль пишется очно на паре лобораторных работ.

В рамках данного РК необходимо реализовать простой сервер на Golang, принимающий клиентские запросы по протоколу HTTP.

Шаблон

Шаблоном для на написания РК является данный репозиторий. В качестве подготовки к РК желательно заблаговременно склонить данный репозиторий и проверить, что сервер корректно запускается, а клиентские запросы обрабатываются в соответствии с заданием в демонстрационном билете.

РК №1

Демонстрационный билет №1. Простой калькулятор

Необходимо написать веб-севрер на GO, реализующий логику простого кулькулятора. Сервер должен запускаться по адресу `127.0.0.1:8081`.

У севрера должна быть ручка (handler) `POST /calculate`. Эта ручка в теле запроса должна принимать JSON с 3 полями: `first_number`, `second_number` и `operator`.

При обработке http-запроса с переданными числами должна производиться соответствующая математическая операция, которая была передана в поле `operator`: `+`, `-`, `*` или `/`.

В качестве ответа сервер должен возвращаться JSON с единственным полем `result`.

РК №1

Примерм запроса (curl):

```
curl --header "Content-Type: application/json" --request POST --data '{"first_number":7,"second_number"
```

Пример ответа:

```
{"result":3.5}
```

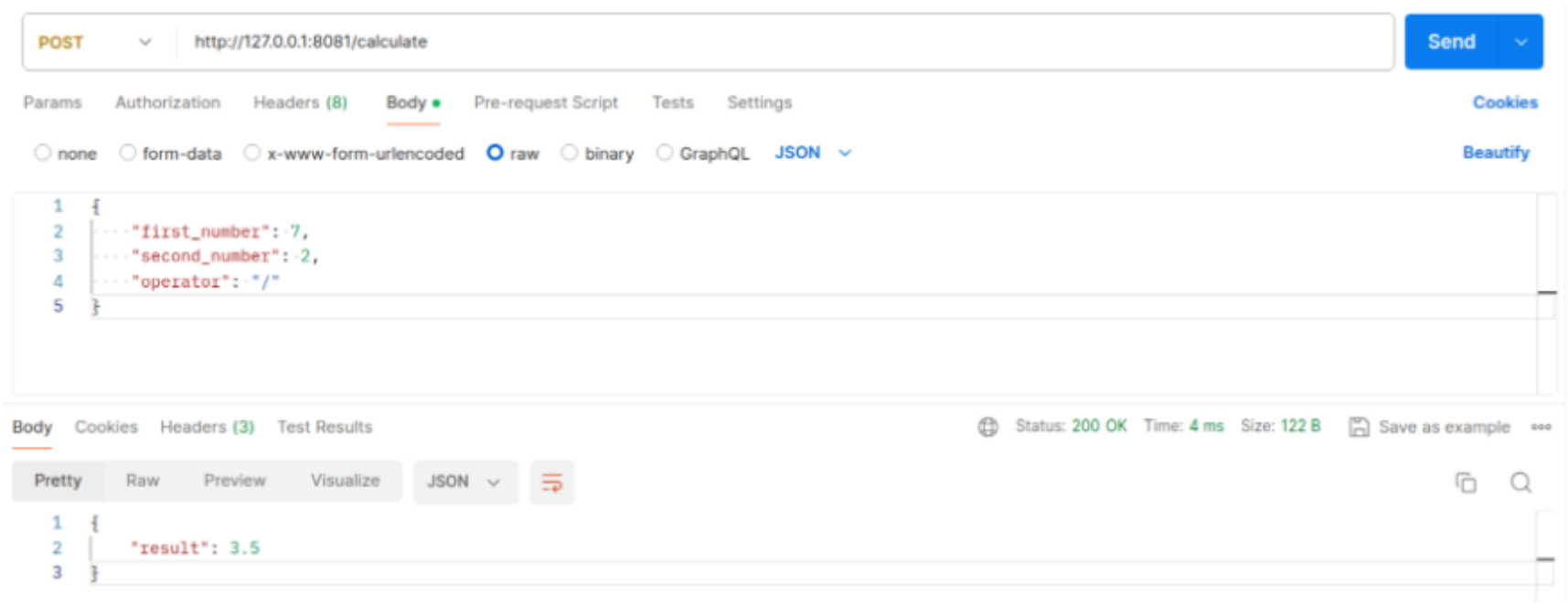
Образец решения данного билета можно посмотреть в репозитории https://github.com/ValeryBMSTU/web-rk1 в файле `main.go`

РК №1

Защита

Защита РК (как и написание) проходит очно на паре лабораторных работ. Во время защиты РК необходимо показать код на golang и продемонстрировать правильность обработки http-запросов с использованием какого-нибудь из клиентов из перечня: Postman, Insomnia, Curl

Образец корректно обработанного запроса для демонстрационного билета №1 через Postman:



СИНХРОННОЕ / АСИНХРОННОЕ ПРОГРАММИРОВАНИЕ

Синхронное программирование — это поведение языка программирования, при котором операции (инструкции) в программе выполняются последовательно (синхронно), то есть в порядке их указания в коде

Асинхронное программирование — это поведение, при котором определённые операции в программе выполняются асинхронно, то есть поток не дожидается их завершения и приступает к выполнению других задач

ПРИМЕР СИНХРОННОГО КОДА

main.go

```go
package main

import "fmt"

func main() {
    fmt.Println("Start main...\n")

    fmt.Println("Biba & Boba")
    fmt.Println("Lupa & Pupa")
    fmt.Println("Top_ & Kek_")

    fmt.Println("\nEnd main...")
}
```

```
Start main...

Biba & Boba
Lupa & Pupa
Top_ & Kek_

End main...
```

ЗАПУСК ФУНКЦИИ АСИНХРОННО

Чтобы запустить функцию асинхронно в Golang, нужно поставить перед её вызовом ключевое слово go. Оно создаёт новую горутину, в которой функция будет выполняться асинхронно с вызвавшим её участком кода.

```
1  MyFunc()     // синхронный вызов фукнции
2  go MyFunc() // асинхронный вызов функции
```

ПРИМЕР ПЛОХОЙ АСИНХРОННОСТИ

main.go

```go
package main

import "fmt"

func main() {
    fmt.Println("Start main...\n")

    go fmt.Println("Biba & Boba")
    go fmt.Println("Lupa & Pupa")
    go fmt.Println("Top_ & Kek_")

    fmt.Println("\nEnd main...")
}
```

```
Start main...


End main...
```

ПРИМЕР АСИНХРОННОСТИ

main.go

```go
package main

import (
    "fmt"
    "time"
)

func main() {
    fmt.Println("Start main...\n")

    go fmt.Println("Biba & Boba")
    go fmt.Println("Lupa & Pupa")
    go fmt.Println("Top_ & Kek_")

    time.Sleep(time.Second*1)

    fmt.Println("\nEnd main...")
}
```

```
Start main...

Top_ & Kek_
Biba & Boba
Lupa & Pupa

End main...
```

```
Start main...

Top_ & Kek_
Lupa & Pupa
Biba & Boba

End main...
```

```
Start main...

Lupa & Pupa
Top_ & Kek_
Biba & Boba

End main...
```

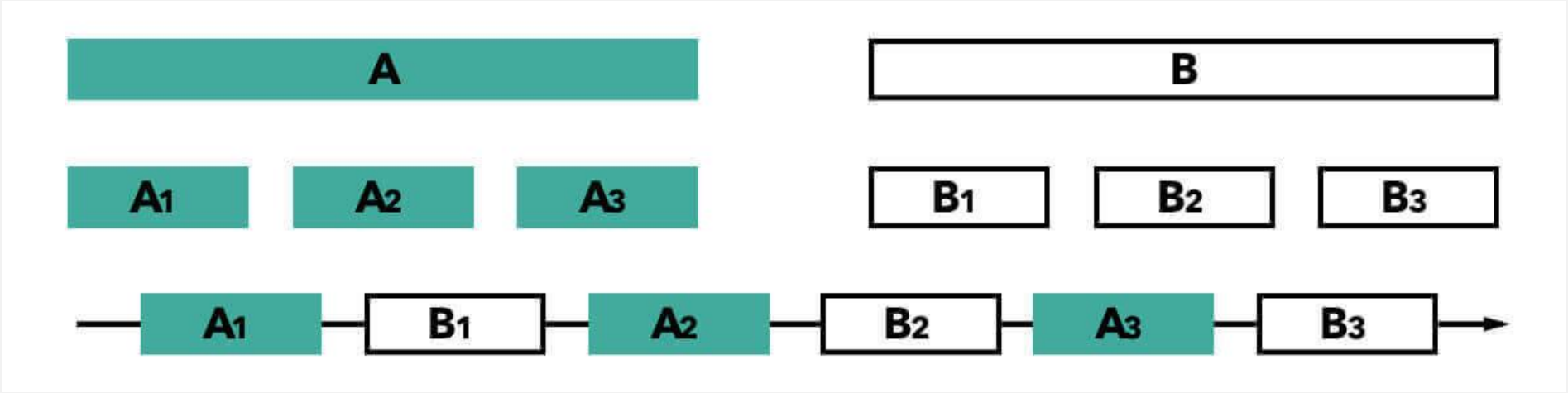
КОНКУРЕНТНОСТЬ И ПАРАЛЛЕЛИЗМ

Конкурентность — это выполнение нескольких задач за определённое время. Задачи необязательно выполняются одновременно, поэтому их можно разделить на более мелкие и чередующиеся.

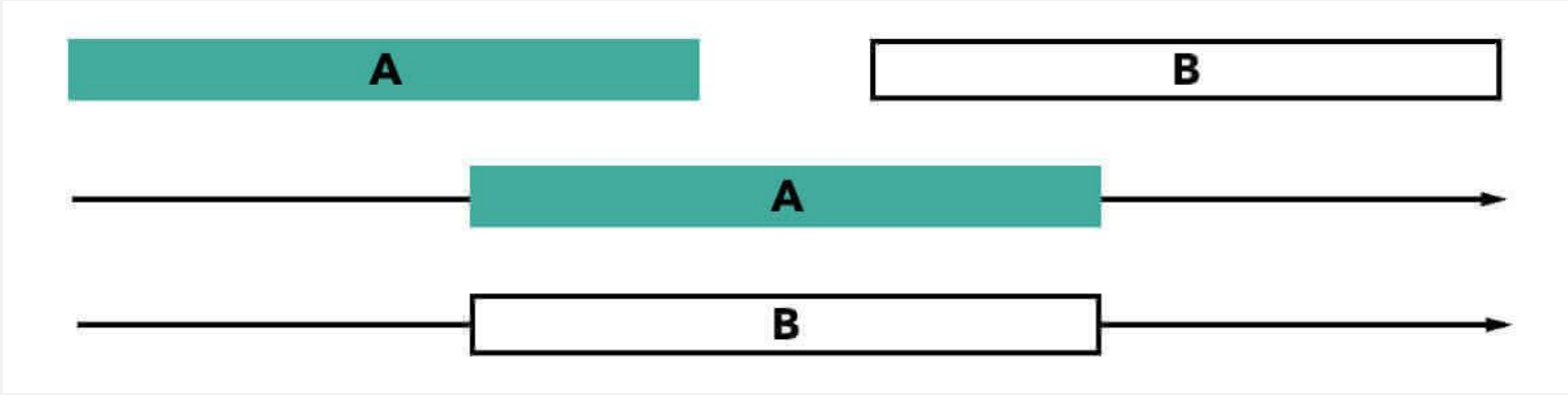
Параллелизм — это выполнение задач в одно и то же время. Подразумевается, что эти задачи будут действительно выполняться на разных физических ядрах процессора параллельно

КОНКУРЕНТНОСТЬ И ПАРАЛЛЕЛИЗМ

Конкурентность:

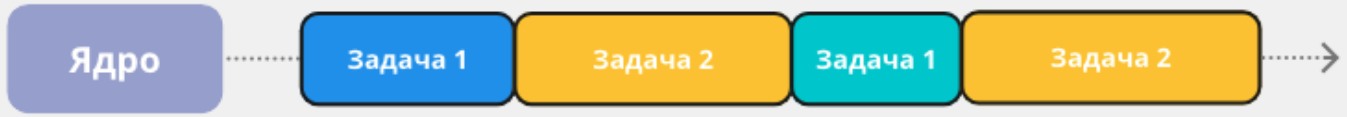


Параллелизм:

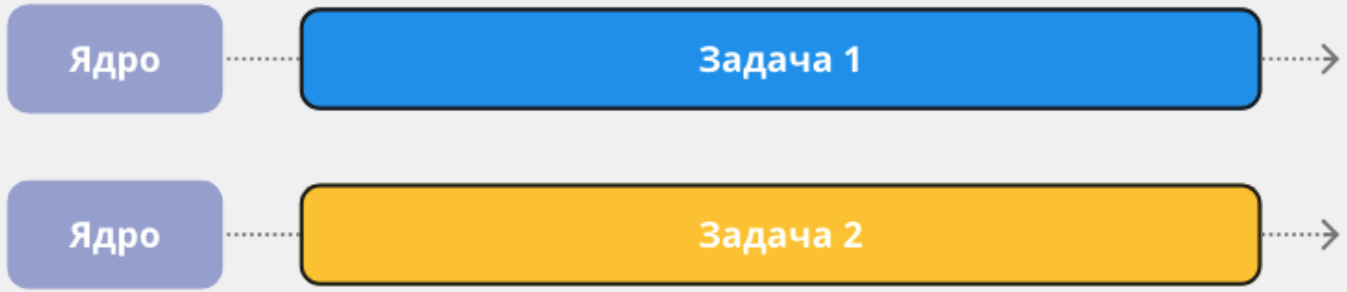


КОНКУРЕНТНОСТЬ И ПАРАЛЛЕЛИЗМ

Конкурентное выполнение



Параллельное выполнение

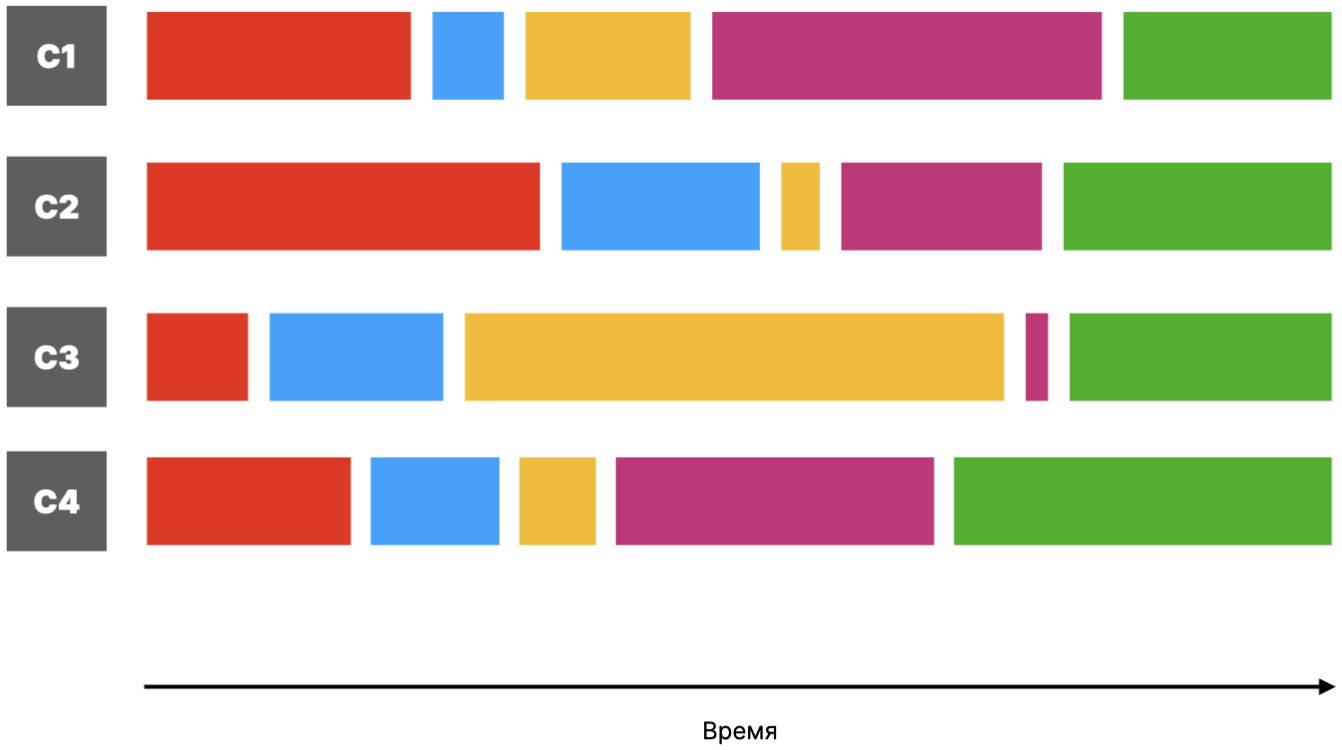


Конкурентный параллелизм

КОНКУРЕНТНОСТЬ В ОДНОЯДЕРНОЙ СИСТЕМЕ

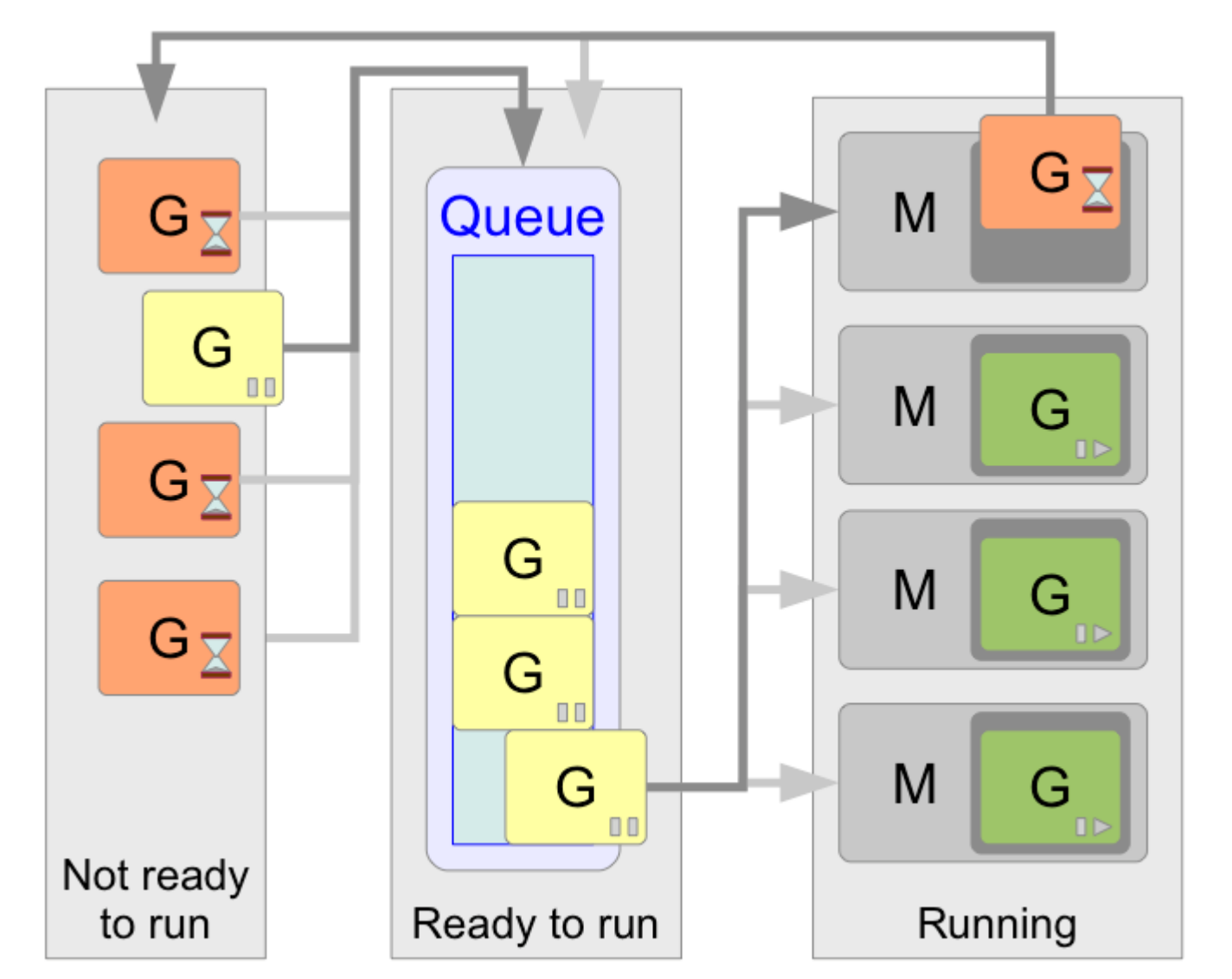
C1 Ядро процессора

КОНКУРЕНТНОСТЬ & ПАРАЛЛЕЛИЗМ В МНОГОЯДЕРНОЙ СИСТЕМЕ



КОНКУРЕНТНОСТЬ В GOLANG

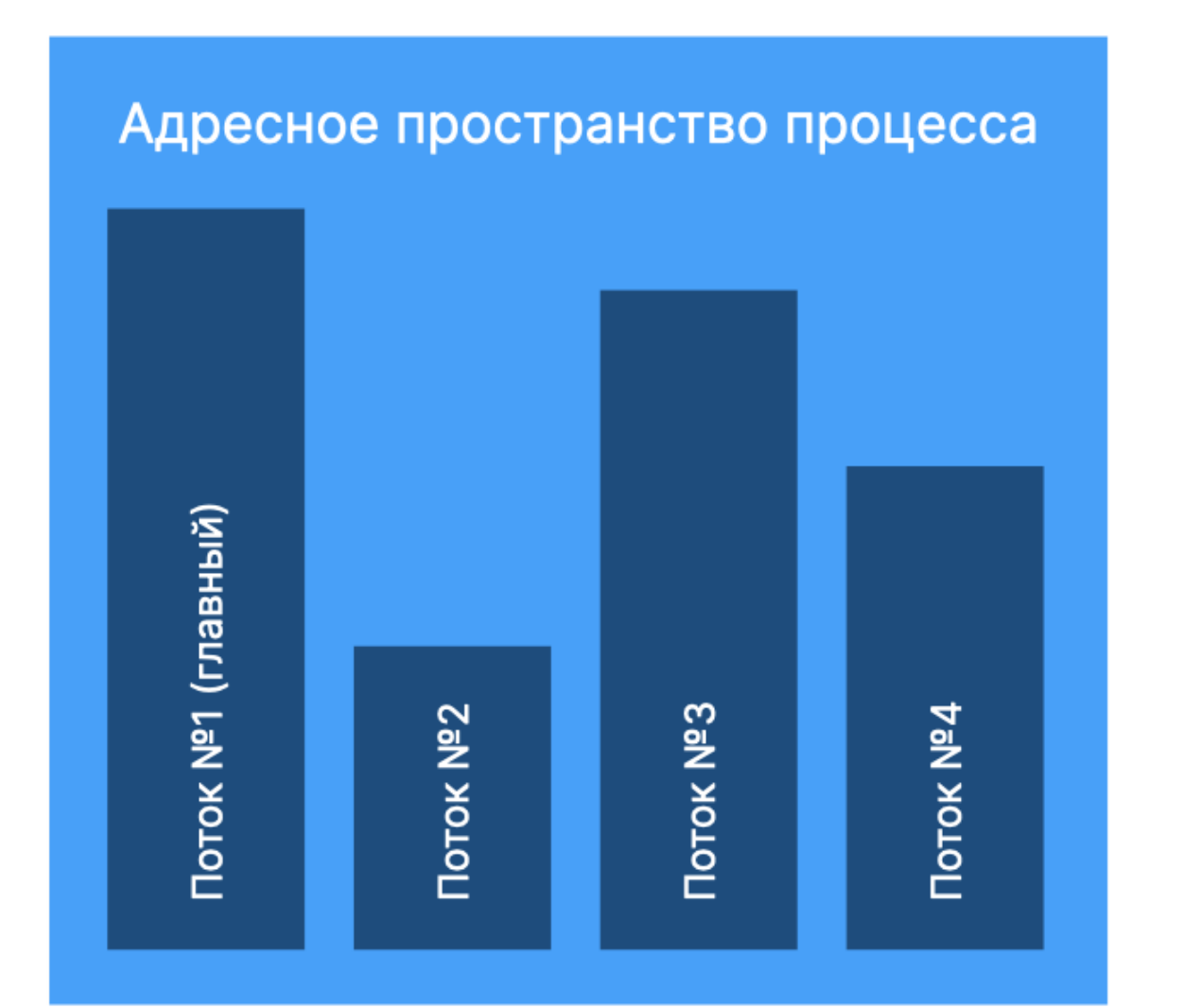
УПРОЩЕННАЯ СХЕМА АСИНХРОННОСТИ В GOLANG



G – горутина

M - поток

ПРОЦЕСС И ПОТОКИ



АСИНХРОННЫЙ GOLANG


```go
func foo() {
    fmt.Println("Hello, world!")
}

// Функцию foo можно вызвать синхронно:
foo() // => Hello, world!

// Или асинхронно:
go foo() // => Hello, world!
```

АСИНХРОННЫЙ GOLANG


```go
func main() {
    fmt.Println("Start main...\n")

    // Объявление канала
    ch := make(chan int)

    go func() {
        // Запись в канал
        ch <- 1
    }()

    // Чтение из канала
    v := <-ch

    fmt.Println(v)

    fmt.Println("\nEnd main...")
}
```

```
Start main...

1

End main...
```

АСИНХРОННЫЙ GOLANG


```go
func main() {
    fmt.Println("Start main...\n")

    // Объявление канала
    ch := make(chan int)

    // Запись в канал
    ch <- 1

    // Чтение из канала
    v := <-ch

    fmt.Println(v)

    fmt.Println("\nEnd main...")
}
```

```
Start main...

fatal error: all goroutines are asleep - deadlock!

goroutine 1 [chan send]:
main.main()
        /opt/src/916554/main.go:15 +0x7a
```

АСИНХРОННЫЙ GOLANG


```go
func main() {
    fmt.Println("Start main...\n")

    // Объявление канала
    ch := make(chan int, 1)

    // Запись в канал
    ch <- 1

    // Чтение из канала
    v := <-ch

    fmt.Println(v)

    fmt.Println("\nEnd main...")
}
```

```
Start main...

1

End main...
```

АСИНХРОННЫЙ GOLANG


```go
func main() {
	s := []int{7, 2, 8, -9, 4, 0}

	// Инициализируем канал
	c := make(chan int)

	// Делим слайс пополам:
	go sum(s[:len(s)/2], c)
	go sum(s[len(s)/2:], c)

	// Читаем из c
	x, y := <-c, <-c

	fmt.Println(x, y, x+y)
}
```

```go
func sum(s []int, c chan int) {
	sum := 0
	for _, v := range s {
		sum += v
	}
	// Записываем сумму в c
	c <- sum
}
```

```
-5 17 12
```

АСИНХРОННЫЙ GOLANG


```
1    ch := make(chan int, 2)
```

АСИНХРОННЫЙ GOLANG


```go
package main

import "fmt"

func main() {
    ch := make(chan string, 4)

    ch <- "hello"
    ch <- "darkness"
    ch <- "my"
    ch <- "old"

    fmt.Println(<-ch)
    fmt.Println(<-ch)
    fmt.Println(<-ch)
    fmt.Println(<-ch)
}
```

```
hello
darkness
my
old
```

АСИНХРОННЫЙ GOLANG


```go
func main() {
	ch := make(chan string, 4)

	ch <- "hello"
	ch <- "darkness"
	ch <- "my"
	ch <- "old"
	ch <- "kek" // extra message

	fmt.Println(<-ch)
	fmt.Println(<-ch)
	fmt.Println(<-ch)
	fmt.Println(<-ch)
}
```

```
fatal error: all goroutines are asleep - deadlock!

goroutine 1 [chan send]:
main.main()
	/opt/src/916562/main.go:12 +0x8c
```

АСИНХРОННЫЙ GOLANG


```go
ch := make(chan string)
// some work...
close(ch)
```

АСИНХРОННЫЙ GOLANG


```go
func main() {
    ch := make(chan string)

    go func(c chan string) {
        c <- "hello"
        c <- "darkness"
        c <- "my"
        c <- "old"
        c <- "kek"
        close(c)
    }(ch)

    for {
        value, isOpen := <- ch
        if !isOpen {
            break
        } else {
            fmt.Println(value)
        }
    }
}
```

```
hello
darkness
my
old
kek
```

АСИНХРОННЫЙ GOLANG


```go
func main() {
    ch := make(chan string)

    go func(c chan string) {
        c <- "hello"
        c <- "darkness"
        c <- "my"
        c <- "old"
        c <- "kek"
        close(c)
    }(ch)

    for v := range ch {
        fmt.Println(v)
    }
}
```

```
hello
darkness
my
old
kek
```

АСИНХРОННЫЙ GOLANG


```go
func main() {
    ch1 := make(chan int)
    ch2 := make(chan int)
    cancelCh := make(chan struct{})

    go func() {
        ch1 <- 1
        ch1 <- 11
        ch1 <- 111
        ch1 <- 1111
        ch1 <- 11111
        cancelCh <- struct{}{}
    }()

    go func() {
        ch2 <- 2
        ch2 <- 22
        ch2 <- 222
        ch2 <- 2222
        ch2 <- 22222
        cancelCh <- struct{}{}
    }()

MYLABEL:
    for {
        select {
        case v1 := <- ch1:
            fmt.Println(v1)
        case v2 := <- ch2:
            fmt.Println(v2)
        case _ = <- cancelCh:
            break MYLABEL
        }
    }
}
```

```
2
22
1
11
111
1111
11111
```

АСИНХРОННЫЙ GOLANG


```go
func main() {
    tick := time.Tick(100 * time.Millisecond)
    boom := time.After(500 * time.Millisecond)
    for {
        select {
        case <-tick:
            fmt.Println("tick.")
        case <-boom:
            fmt.Println("BOOM!")
            return
        default:
            fmt.Println("    .")
            time.Sleep(50 * time.Millisecond)
        }
    }
}
```

```
.
    .
tick.
    .
    .
tick.
    .
    .
tick.
    .
    .
tick.
    .
    .
tick.
BOOM!
```

АСИНХРОННЫЙ GOLANG


```go
func main() {
    var wg sync.WaitGroup

    wg.Add(1)
    go func() {
        defer wg.Done()
        time.Sleep(time.Millisecond*300)
        fmt.Println("First done")
    }()

    wg.Add(1)
    go func() {
        defer wg.Done()
        time.Sleep(time.Millisecond*800)
        fmt.Println("Second done")
    }()

    fmt.Println("Waiting...\n")
    wg.Wait()
    fmt.Println("\nMain done")
}
```

```
Waiting...

First done
Second done

Main done
```

АСИНХРОННЫЙ GOLANG


```go
func main() {
    var wg sync.WaitGroup
    for i := 0; i < 5; i++ {
        wg.Add(1)
        go func() {
            defer wg.Done()
            time.Sleep(time.Millisecond*100)
            fmt.Println("work")
        }()
    }

    fmt.Println("Waiting...\n")
    wg.Wait()
    fmt.Println("\nMain done")
}
```

```
Waiting...

work
work
work
work
work

Main done
```

АСИНХРОННЫЙ GOLANG

Go 1.20 ▾ Скачать код Запустить

main.go

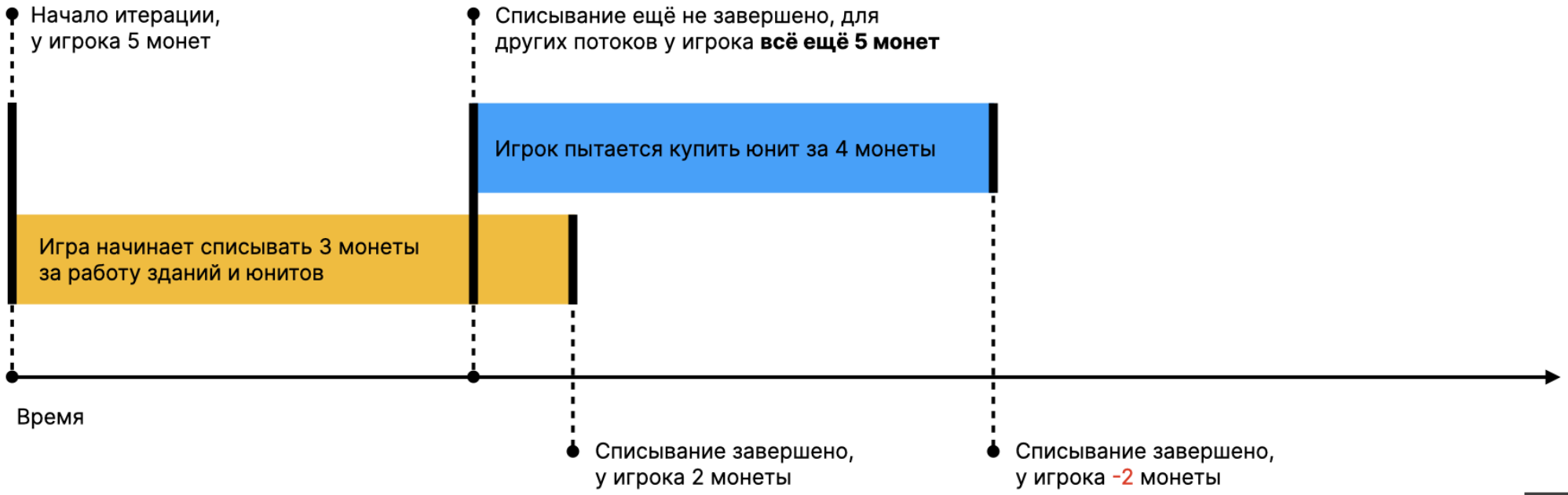
```go
func main() {
    var wg sync.WaitGroup
    for i := 0; i < 5; i++ {
        wg.Add(1)
        go func() {
            defer wg.Done()
            time.Sleep(time.Millisecond*100)
            fmt.Println(i, "work")
        }()
    }

    fmt.Println("Waiting...\n")
    wg.Wait()
    fmt.Println("\nMain done")
}
```

```
Waiting...

5 work
5 work
5 work
5 work
5 work

Main done
```

СОСТОЯНИЕ ГОНКИ

Активные расходы
Непосредственные покупки игрока
(покупка предметов в игровом магазине)

Пассивные расходы
Покупки без явного участия игрока
(расходы на содержание юнитов и зданий)



АСИНХРОННОСТЬ В GOLANG


```go
func main() {
    var (
        wg sync.WaitGroup
        counter int
    )

    for i := 0; i < 99999; i++ {
        wg.Add(1)
        go func() {
            defer wg.Done()
            counter++
        }()
    }

    fmt.Println("Waiting...\n")
    wg.Wait()
    fmt.Println(counter)
}
```

```
Waiting...

99076
```

АСИНХРОННОСТЬ В GOLANG


```go
func main() {
    var (
        mu sync.Mutex
        wg sync.WaitGroup
        counter int
    )

    for i := 0; i < 99999; i++ {
        wg.Add(1)
        go func() {
            defer wg.Done()
            mu.Lock()
            counter++
            mu.Unlock()
        }()
    }

    fmt.Println("Waiting...\n")
    wg.Wait()
    fmt.Println(counter)
}
```

```
Waiting...

99999
```

ИСТОЧНИКИ

- Конкурентность и параллелизм — разные вещи
  https://tproger.ru/explain/concurrency-vs-parallelism
- Конкурентность в Go простым языком
  https://blog.ildarkarymov.ru/posts/go-concurrency/

СПАСИБО ЗА ВНИМАНИЕ :3

08.10.2024