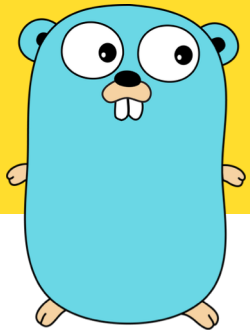


Асинхронная модель в Go

Tinkoff.ru

Оффтоп: 25feb2020 – release go1.14

https://blog.golang.org/go1.14

- Go modules are now production ready
- Embedding interfaces with overlapping method sets
- Improved defer performance
- Goroutines are asynchronously preemptive
- Internal timers are more efficient (time.After, time.Tick, net.Conn.SetDeadline)
- FreeBSD 64/ARM support

План занятия

1. Go Scheduler
2. Goroutines (+Race condition)
3. Locking shared resources (atomic, mutex)
4. Channels
5. Context
6. Pipelines
7. Interactivity (maybe)

Немного фактов о сервисе котировок

22K RPS (Requests Per Second)

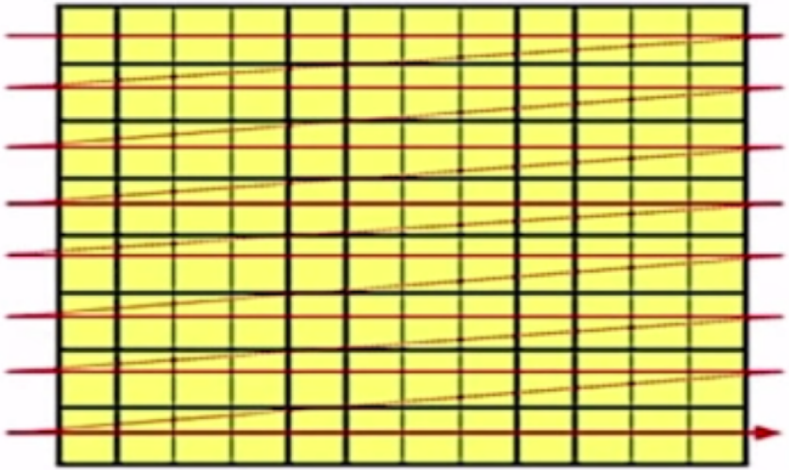
50K MPS (Messages Per Second)

6 x 8CPU x 16Gb RAM

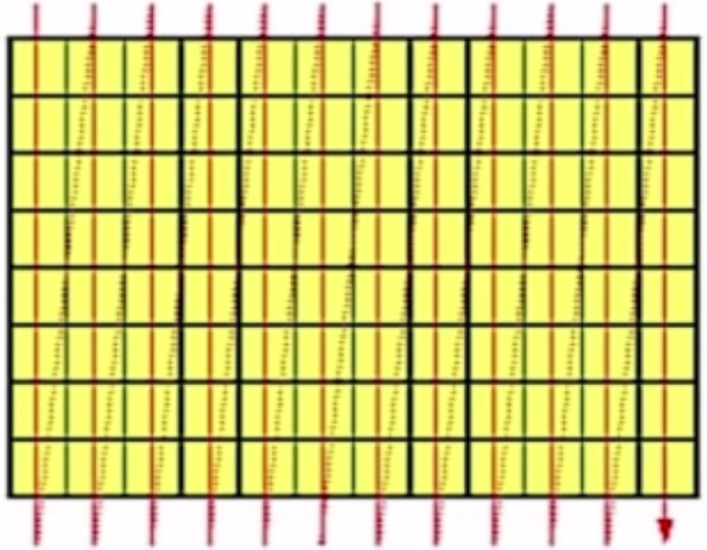
40ms perc999

Go Scheduler

Немного классики

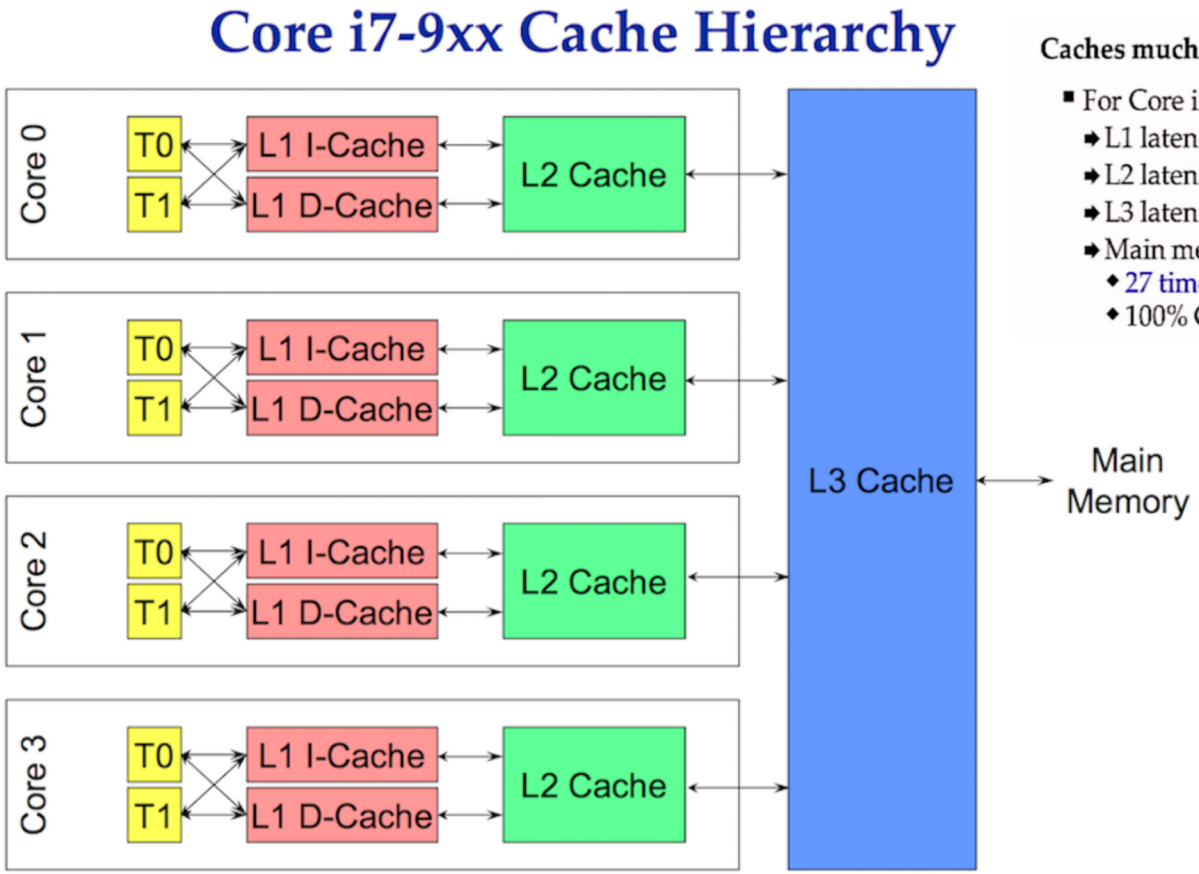


Row Major



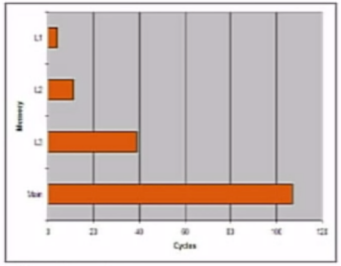
Column Major

Кэш лайны



Caches much faster than main memory.

- For Core i7-9xx:
  - L1 latency is 4 cycles.
  - L2 latency is 11 cycles.
  - L3 latency is 39 cycles.
  - Main memory latency is 107 cycles.
    - 27 times slower than L1!
    - 100% CPU utilization ⇒ >99% CPU idle time!

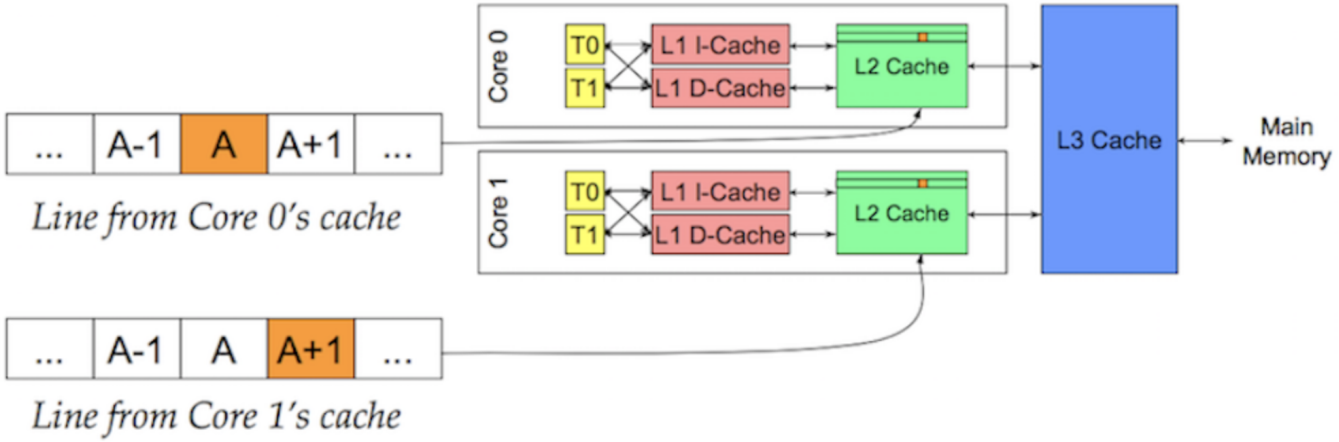


False sharing

False Sharing

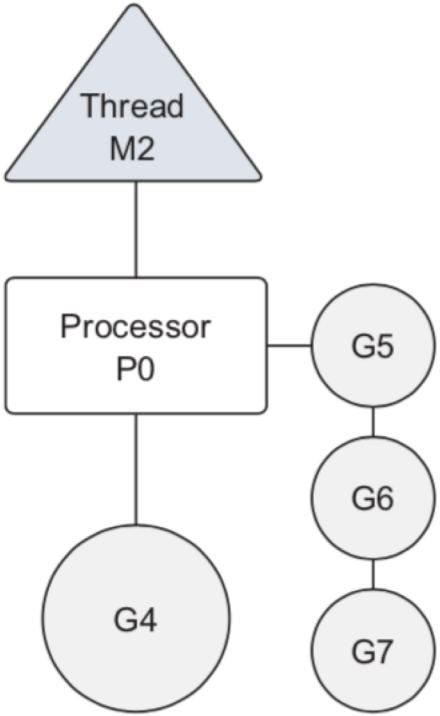
Suppose Core 0 accesses *A* and Core 1 accesses *A+1*.

- *Independent* pieces of memory; concurrent access is safe.
- But *A* and *A+1* probably map to the same cache line.
  - ➡ If so, Core 0's writes to *A* invalidates *A+1*'s cache line in Core 1.
    - And vice versa.
    - This is *false sharing*.

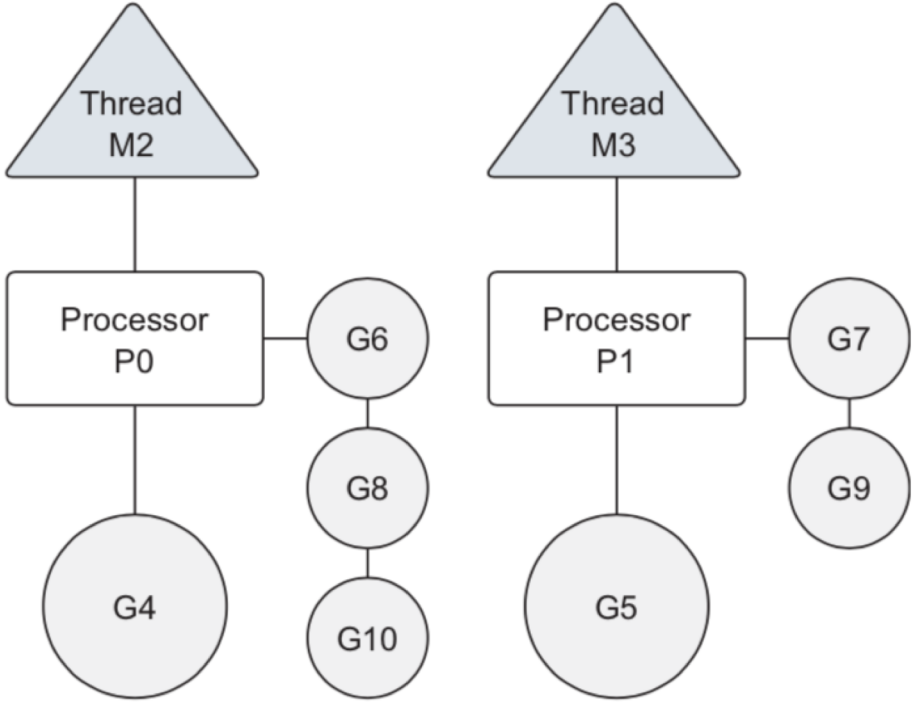


Concurrency vs Parallelism

Concurrency



Parallelism



Goroutines

Введение в go func()

```
1  func main() {
2     go spinner(100 * time.Millisecond)
3     fibN := fib(45)
4     fmt.Printf("\rFibonacci(%d) = %d\n", n, fibN)
5  }
6  func spinner(delay time.Duration) {
7     for {
8        for _, r := range `-\|/` {
9           fmt.Printf("\r%c", r)
10          time.Sleep(delay)
11       }
12    }
13 }
14 func fib(x int) int {
15    if x < 2 {
16       return x
17    }
18    return fib(x-1) + fib(x-2)
19 }
```

WaitGroup для оркестрации

```
1  runtime.GOMAXPROCS(1)
2  var wg sync.WaitGroup
3  wg.Add(2)
4  fmt.Println("Starting...")
5  go func() {
6     defer wg.Done()
7     for char := 'a'; char < 'a'+26; char++ {
8        //runtime.Gosched()
9        fmt.Printf("%c ", char)
10    }
11 }()
12 go func() {
13    defer wg.Done()
14    for char := 'A'; char < 'A'+26; char++ {
15       //runtime.Gosched()
16       fmt.Printf("%c ", char)
17    }
18 }()
19 wg.Wait()
20 fmt.Println("\nFinished")
```

Как WaitGroup передать в функцию

```
1  func main() {
2     runtime.GOMAXPROCS(2)
3     var wg sync.WaitGroup
4     wg.Add(2)
5     fmt.Println("Starting...")
6     go printABC('A', &wg)
7     go printABC('a', &wg)
8     wg.Wait()
9     fmt.Println("\nFinished")
10 }
11 func printABC(firstLetter rune, wg *sync.WaitGroup) {
12    defer wg.Done()
13    for i := 0; i < 10; i++ {
14       for char := firstLetter; char < firstLetter+26; char++ {
15          fmt.Printf("%c ", char)
16       }
17    }
18 }
```

Гонка?

```
1  var counter int
2  func main() {
3     runtime.GOMAXPROCS(1)
4     var wg sync.WaitGroup
5     wg.Add(2)
6     go incCounter(&wg) //routine #1
7     go incCounter(&wg) //routine #2
8     wg.Wait()
9     fmt.Println("Final counter: ", counter)
10 }
11 func incCounter(wg *sync.WaitGroup) {
12    defer wg.Done()
13    for i := 0; i < 2; i++ {
14       value := counter
15       runtime.Gosched()
16       value++
17       counter = value
18    }
19 }
```

Goroutines // Race detector

> go run –race main.go

```
WARNING: DATA RACE
Write at 0x000001213840 by goroutine 7:
  main.incCounter()
      /Users/sergey/fintech/cmd/3-race/main.go:26 +0x8e
Previous read at 0x000001213840 by goroutine 6:
  main.incCounter()
      /Users/sergey/fintech/cmd/3-race/main.go:23 +0x6d
Goroutine 7 (running) created at:
  main.main()
      /Users/sergey/fintech/cmd/3-race/main.go:16 +0xd1
Goroutine 6 (running) created at:
  main.main()
      /Users/sergey/fintech/cmd/3-race/main.go:15 +0xaf
==================
Final counter:  2
Found 1 data race(s)
exit status 66
```

golang.org/x/sync/errgroup

```go
var g errgroup.Group
var urls = []string{
   "http://www.golang.org/",
   "http://www.google.com/",
   "http://www.nnsssuuu.com/",
}
for _, url := range urls {
   // Launch a goroutine to fetch the URL.
   url := url // https://golang.org/doc/faq#closures_and_goroutines
   g.Go(func() error {
      // Fetch the URL.
      resp, err := http.Get(url)
      if err == nil {
         resp.Body.Close()
      }
      return err
   })
}
// Wait for all HTTP fetches to complete.
if err := g.Wait(); err == nil {
   fmt.Println("Successfully fetched all URLs.")
}
```

Errgroup with Context

```
Google := func(ctx context.Context, query string) ([]Result, error) {
   g, ctx := errgroup.WithContext(ctx)
   searches := []Search{Web, Image, Video}
   results := make([]Result, len(searches))
   for i, search := range searches {
      i, search := i, search // https://golang.org/doc/faq#closures_and_goroutines
      g.Go(func() error {
         result, err := search(ctx, query)
         if err == nil {
            results[i] = result
         }
         return err
      })
   }
   if err := g.Wait(); err != nil {
      return nil, err
   }
   return results, nil
}
results, err := Google(context.Background(), "golang")
if err != nil {
   fmt.Fprintln(os.Stderr, err)
   return
}
for _, result := range results {
   fmt.Println(result)
}
```

Locking shared resources

sync/atomic

```
1  // SwapInt64 atomically stores new into *addr and returns the previous
2  *addr value.
3  func SwapInt64(addr *int64, new int64) (old int64)
4  // SwapPointer atomically stores new into *addr and returns the previous
5  *addr value.
6  func SwapPointer(addr *unsafe.Pointer, new unsafe.Pointer) (old
7  unsafe.Pointer)
8  // CompareAndSwapInt64 executes the compare-and-swap operation for an int64
9  value.
10
11 func CompareAndSwapInt64(addr *int64, old, new int64) (swapped bool)
12 // CompareAndSwapPointer executes the compare-and-swap operation for a
13 unsafe.Pointer value.
14 func CompareAndSwapPointer(addr *unsafe.Pointer, old, new unsafe.Pointer)
15 (swapped bool)
16
17 // AddInt64 atomically adds delta to *addr and returns the new value.
18 func AddInt64(addr *int64, delta int64) (new int64)
```

sync/atomic

```
1  var counter int64
2
3  func main() {
4     runtime.GOMAXPROCS(1)
5     var wg sync.WaitGroup
6     wg.Add(2)
7     go incCounter(&wg) //routine #1
8     go incCounter(&wg) //routine #2
9     wg.Wait()
10    fmt.Println("Final counter: ", counter)
11 }
12 func incCounter(wg *sync.WaitGroup) {
13    defer wg.Done()
14    for i := 0; i < 2; i++ {
15       atomic.AddInt64(&counter, 1)
16       runtime.Gosched()
17    }
18 }
```

sync/mutex

Мьютекс — это аналог бинарного семафор.

Метод Lock дает go-программе монопольный доступ к участку кода (критическая секция).

Метод Unlock освобождает критический участок кода для доступа из другой go-рутины

sync/mutex

```
1  var counter int
2  var mutex sync.Mutex
3  func main() {
4     var wg sync.WaitGroup
5     wg.Add(2)
6     go incCounter(&wg) //routine #1
7     go incCounter(&wg) //routine #2
8     wg.Wait()
9     fmt.Println("Final counter: ", counter)
10 }
11 func incCounter(wg *sync.WaitGroup) {
12    defer wg.Done()
13    for i := 0; i < 2; i++ {
14       mutex.Lock()
15       value := counter
16       value++
17       counter = value
18       mutex.Unlock()
19    }
20 }
```

Channels

channels

Канал — связующее звено между go-рутинами.

Канал используется для передачи данных между go-рутинами.

```
1  //создание небуферизованный канал
2  unbuffered := make(chan int)
3
4  //создание буферизованный канал
5  buffered := make(chan int, 10)
```

```
1  //запись в канал
2  buffered <- 1000
3
4  // чтение из канала
5  value := <-unbuffered
```

Типизированные каналы

Каналы типизированы, но можно использовать пустой интерфейс.

```
1  // канал, из которого доступно только чтение
2  readDataCh := make(<-chan interface{})
3
4  // канал, в который доступна только запись
5  writeDataCh := make(chan<- interface{})
6
7  // канал, в который можно писать и из которого разрешено читать
8  rwDataCh := make(chan interface{})
```

Запись/чтение

Запись и чтение из канала блокирующие

```
1  dataCh := make(chan string)
2  go func() {
3     if true {
4        return
5     }
6     dataCh <- "Hello channel!"
7  }()
8  fmt.Println(<-dataCh)
```

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

goroutine 1 [chan receive]:
main.main()
```

Закрытие канала

Закрывая канал, мы явно сообщаем читающему, что запись закончена

```
1  dataCh := make(chan int)
2  close(dataCh)
3  cnt, ok := <-dataCh
4  fmt.Printf("(%v): %v", ok, cnt)
```

```
(false): 0
```

for-range для чтения

При чтении из канала можно использовать цикл for-range

```
1  dataCh := make(chan int)
2  go func() {
3     defer close(dataCh)
4     for i := 0; i < 5; i++ {
5        dataCh <- i
6     }
7  }()
8  for data := range dataCh {
9     fmt.Printf("%v ", data)
10 }
```

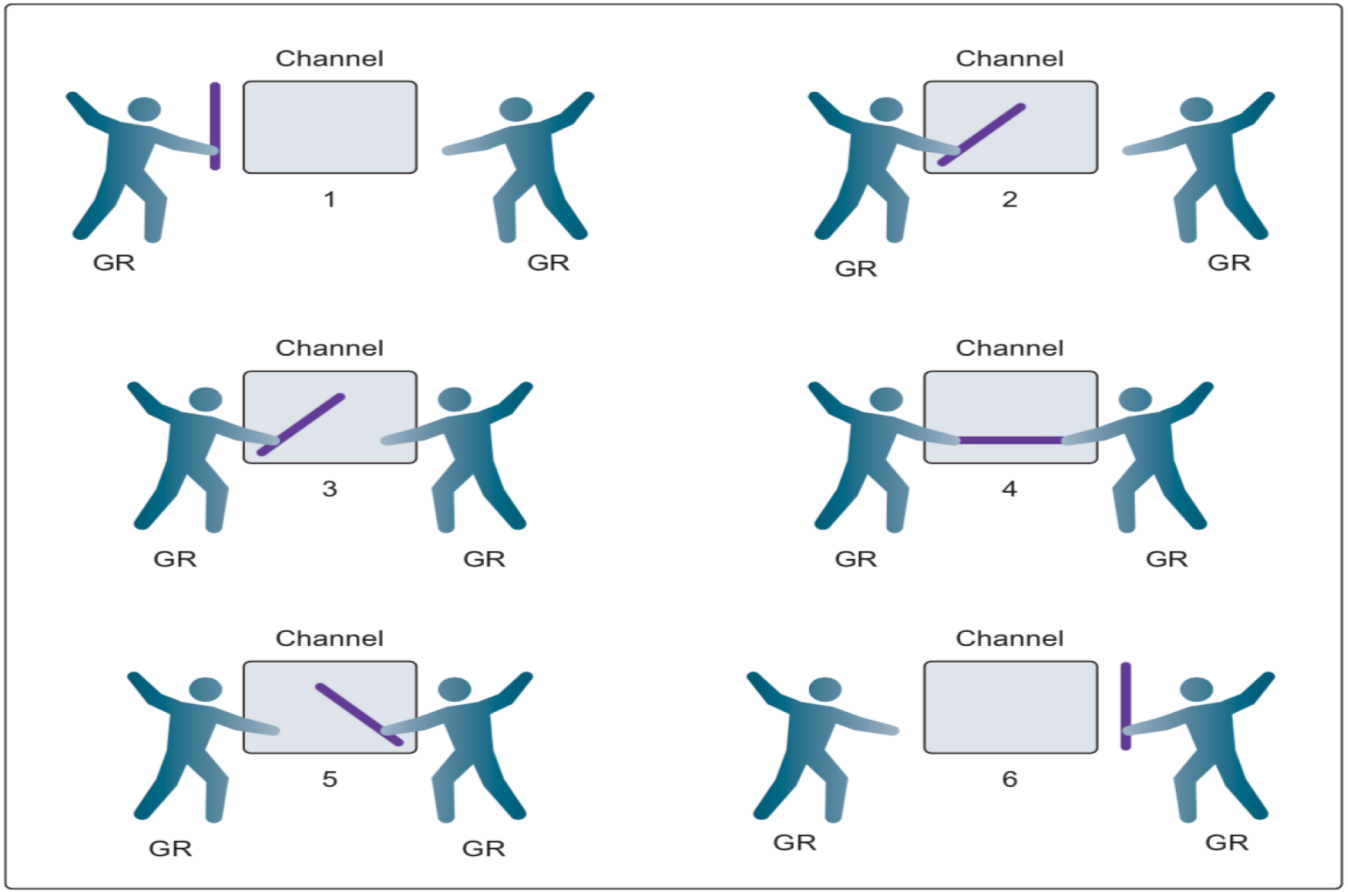
```
0 1 2 3 4
```

channels

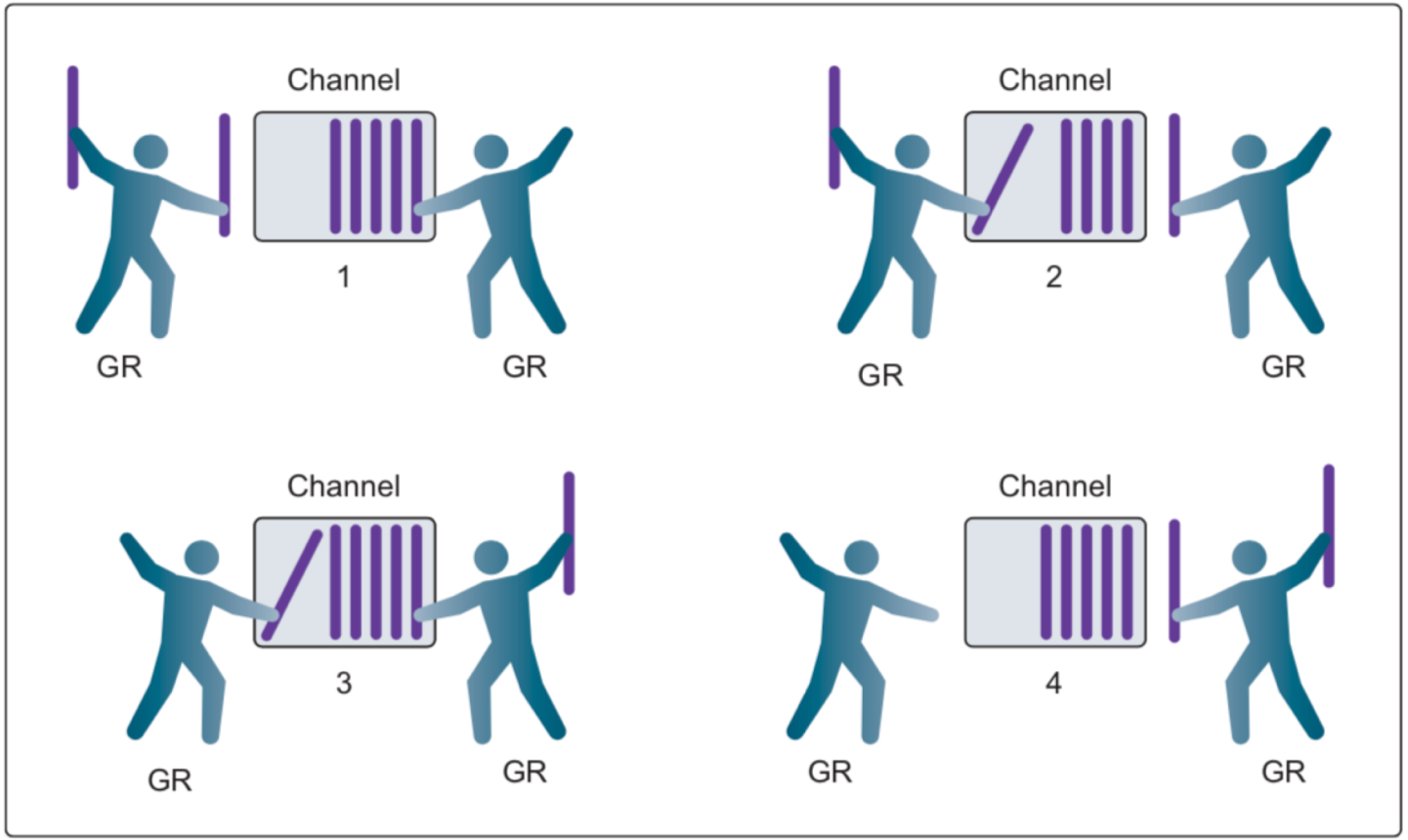
```
1  start := make(chan interface{})
2  var wg sync.WaitGroup
3  for i := 0; i < 4; i++ {
4     wg.Add(1)
5     go func(i int) {
6        defer wg.Done()
7        <-start
8        fmt.Printf("%v begun\n", i)
9     }(i)
10 }
11 fmt.Println("Unblocking...")
12 close(start)
13 wg.Wait()
```

```
Unblocking...
2 begun
3 begun
0 begun
1 begun
```

channels/unbuffered



channels/buffered



select statement

Select statement— связующее звено между каналами.

```
1  var c1, c2 <-chan interface{}
2  select {
3  case <- c1:
4     // do something
5  case <- c2:
6     // do something
7  default:
8     // do something
9  }
```

channels // select statement

```
1  c1 := make(chan interface{});
2  close(c1)
3  c2 := make(chan interface{})
4  close(c2)
5  var c1Count, c2Count int
6  for i := 0; i < 1000; i++ {
7     select {
8     case <-c1:
9        c1Count++
10    case <-c2:
11       c2Count++
12    }
13 }
14 fmt.Printf("c1Count: %d\nc2Count: %d", c1Count, c2Count)
```

```
c1Count: 498
c2Count: 502
```

channels // select statement

```
1  var c<-chan int
2  select {
3  case <-c:
4  case <- time.After(5 * time.Second):
5     fmt.Print("timed out")
6  }
```

Context

Context interface

```
1  // A Context carries a deadline, cancelation signal, and request-scoped values
2  // across API boundaries. Its methods are safe for simultaneous use by multiple
3  // goroutines.
4  type Context interface {
5     // Done returns a channel that is closed when this Context is canceled
6     // or times out.
7     Done() <-chan struct{}
8
9     // Err indicates why this context was canceled, after the Done channel
10    // is closed.
11    Err() error
12
13    // Deadline returns the time when this Context will be canceled, if any.
14    Deadline() (deadline time.Time, ok bool)
15
16    // Value returns the value associated with key or nil if none.
17    Value(key interface{}) interface{}
   }
```

WithValue

```
1  func main() {
2     ctx := context.WithValue(context.Background(), "tracking_id", "some track value")
3     fmt.Println(ctx.Value("tracking_id"))
4  }
```

Что стоит передавать через контекст:
Специфические параметры запроса (например трэк ид)

Что не стоит передавать через контекст:
Всякие глобальные параметры типа логгеров, подключений к базам данных

WithTimeout

```
1  duration := 150 * time.Millisecond
2  ctx, cancel := context.WithTimeout(context.Background(), duration)
3  defer cancel()
4  ch := make(chan int, 1)
5  go func() {
6     //time.Sleep(50 * time.Millisecond)
7     time.Sleep(500 * time.Millisecond)
8     ch <- 123
9  }()
10 select {
11 case d := <-ch:
12    fmt.Printf("completed, result: %d\n", d)
13
14 case <-ctx.Done():
15    fmt.Println("cancelled")
16 }
```

```
cancelled
```

Pipelines

pipelines

https://blog.golang.org/pipelines

Обратная связь

Спасибо за внимание

Tinkoff.ru