




Golang Developer. Professional

otus.ru

 Проверить, идет ли запись

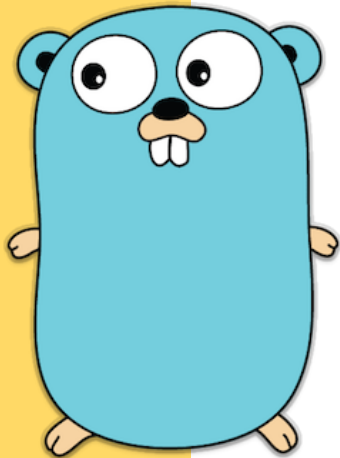
Меня хорошо видно && слышно?

 Ставим “+”, если все хорошо
“-”, если есть проблемы

Тема вебинара

Дополнительные примитивы синхронизации

Рубаха Юрий



Правила вебинара



Активно
участвуем



Off-topic обсуждаем
в учебной группе



Задаем вопрос
в чат или голосом



Вопросы вижу в чате,
могу ответить не сразу

Условные обозначения



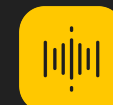
Индивидуально



Время, необходимое
на активность



Пишем в чат



Говорим голосом



Документ



Ответьте себе или
задайте вопрос

О чем будем говорить

- go memory model
- sync.RWMutex
- sync.Cond
- sync.Pool
- atomic

Go memory model

```
func main() {  
    text := ""  
    isInit := false  
  
    go func() {  
        text = "go-go-go"  
        isInit = true  
    }()  
  
    for !isInit {  
        time.Sleep(time.Nanosecond)  
    }  
    fmt.Println(text)  
}
```

Go memory model

<https://golang.org/ref/mem>

If you must read the rest of this document to understand the behavior of your program, you are being too clever.

Don't be clever.

sync.Mutex

```
type Counters struct {  
    mu sync.Mutex  
    m  map[string]int  
}  
  
func (c *Counters) Load(key string) (int, bool) {  
    c.mu.Lock()  
    defer c.mu.Unlock()  
    val, ok := c.m[key]  
    return val, ok  
}  
  
func (c *Counters) Store(key string, value int) {  
    c.mu.Lock()  
    defer c.mu.Unlock()  
    c.m[key] = value  
}
```

https://goplay.tools/snippet/Qp-w_QsOleR

https://go.dev/play/p/Qp-w_QsOleR

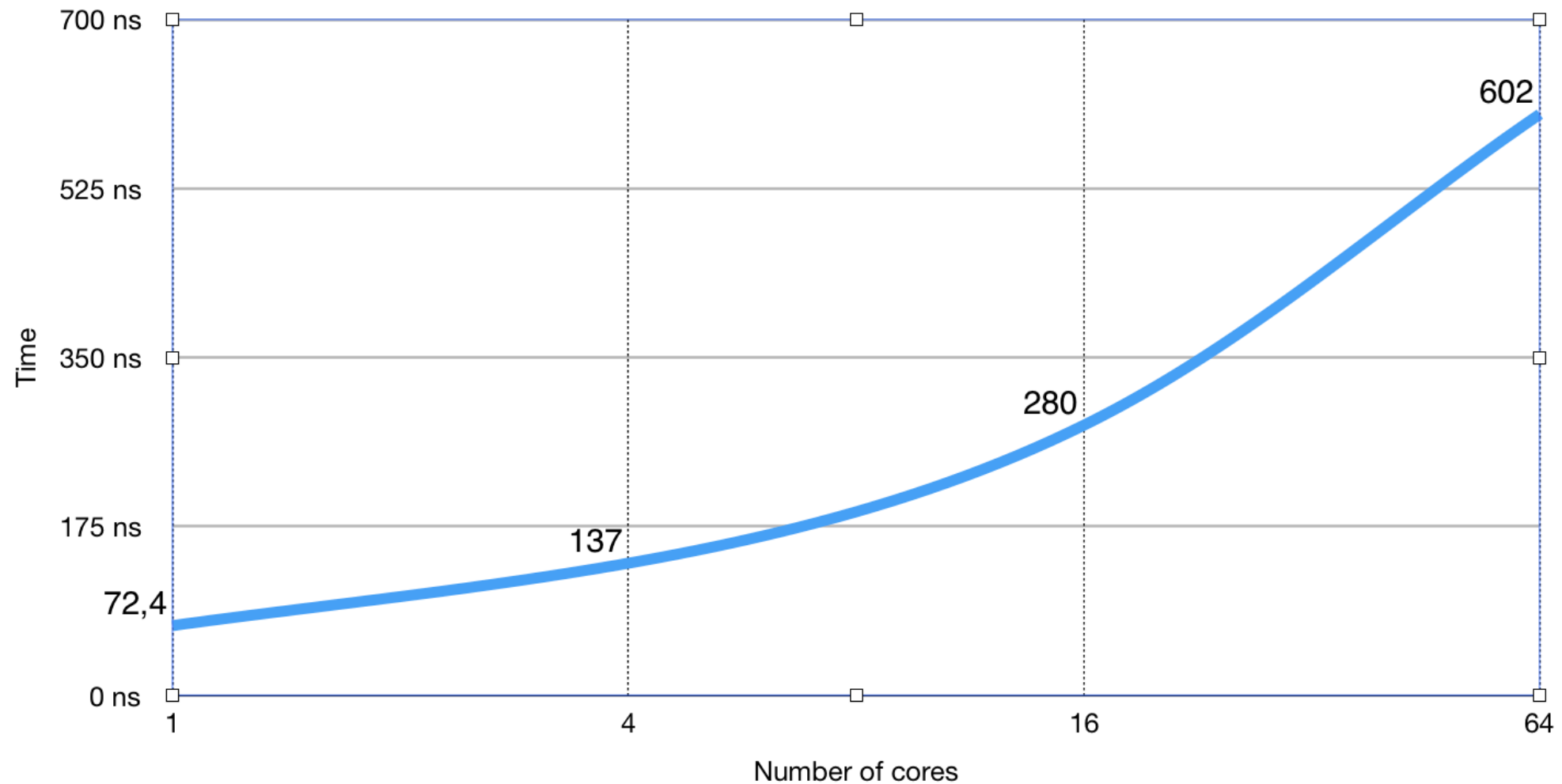
sync.RWMutex

```
type Counters struct {  
    mu sync.RWMutex  
    m  map[string]int  
}  
  
func (c *Counters) Load(key string) (int, bool) {  
    c.mu.RLock()  
    defer c.mu.RUnlock()  
    val, ok := c.m[key]  
    return val, ok  
}  
  
func (c *Counters) Store(key string, value int) {  
    c.mu.Lock()  
    defer c.mu.Unlock()  
    c.m[key] = value  
}
```

<https://goplay.tools/snippet/1ltioa-0cXF>

<https://go.dev/play/p/1ltioa-0cXF>

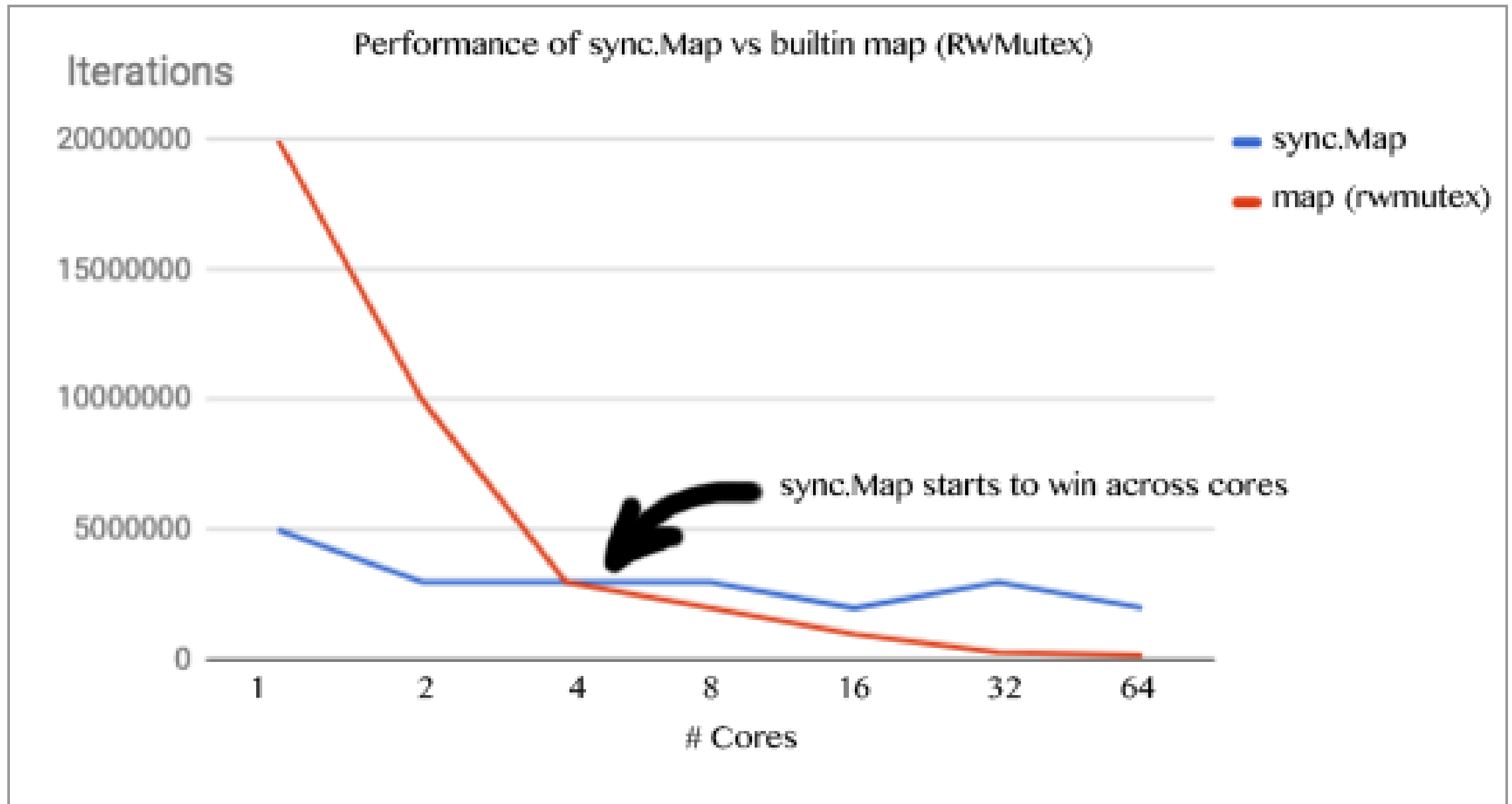
Cache contention



При блокировке на чтение каждое ядро обновляет счетчик. Что приводит к инвалидации счетчика в кэше других ядер.
<https://habr.com/ru/post/338718/>

sync.Map

Решает проблему частых чтений для map + rwmutex.



sync.Map

```
type Map struct {  
}  
  
func (m *Map) Delete(key interface{})  
func (m *Map) Load(key interface{}) (value interface{}, ok bool)  
func (m *Map) LoadOrStore(key, value interface{}) (actual interface{}, loaded bool)  
func (m *Map) Range(f func(key, value interface{}) bool)  
func (m *Map) Store(key, value interface{})
```

<https://www.youtube.com/watch?v=C1EtfDnsdDs>

sync.Map

```
type Counters struct {  
    m sync.Map  
}  
  
func (c *Counters) Load(key string) (int, bool) {  
    val, ok := c.m.Load(key)  
    if !ok {  
        return 0, false  
    }  
    return val.(int), true  
}  
  
func (c *Counters) Store(key string, value int) {  
    c.m.Store(key, value)  
}  
  
func (c *Counters) Range(f func(k string, v int) bool) {  
    c.m.Range(func(k, v interface{}) bool {  
        return f(k.(string), v.(int))  
    })  
}
```

<https://goplay.tools/snippet/SYciXadco3q>

<https://go.dev/play/p/SYciXadco3q>

Практика

```
// Сделайте тип Worker потокобезопасным  
// с помощью mutex или rwmutex.  
// Объясните свой выбор.
```

```
type Worker struct {  
    ready bool  
}  
  
func (w *Worker) setReady() {  
    w.ready = true  
}  
  
func (w *Worker) CheckReady() bool {  
    return w.ready  
}
```

<https://goplay.tools/snippet/djgdo2jollq>

<https://go.dev/play/p/djgdo2jollq>

sync.Cond

Cond(ition variable) - механизм для ожидания горутинами сигнала о событии

```
type Cond struct {  
    L Locker  
}  
  
func NewCond(l Locker) *Cond  
  
func (c *Cond) Wait() // разблокирует c.L, ждет сигнала и снова блокирует c.L  
  
func (c *Cond) Broadcast() // будит все горутин, которые ждут c  
  
func (c *Cond) Signal() // будит одну горутину, которая ждет c, если такая есть
```

sync.Cond

```
func worker() {  
    for task := range tasks {  
        task()  
    }  
}  
  
func produce(task func()) {  
    tasks <- task  
}
```

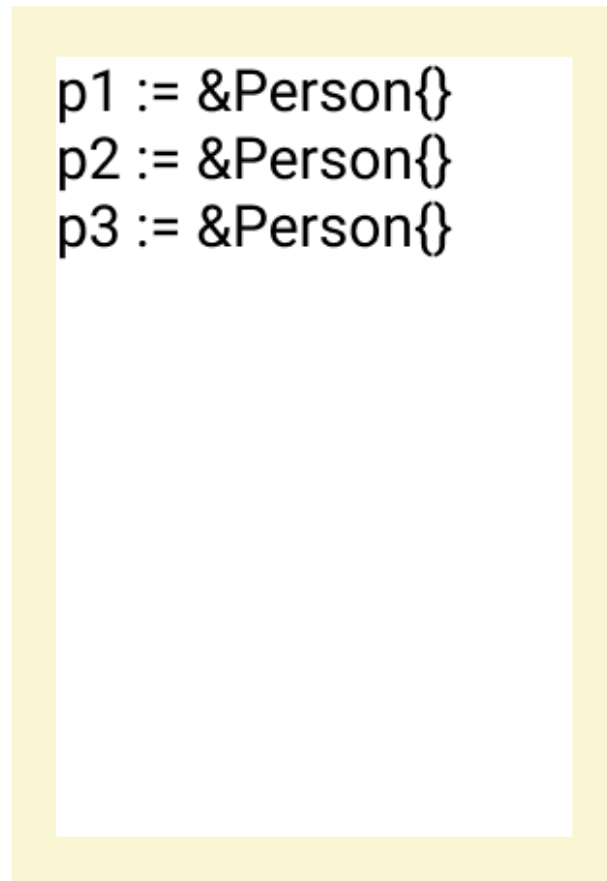

sync.Cond

```
func worker() {  
    var task func()  
  
    for !stopped {  
        mu.Lock()  
        for len(tasks) == 0 { // <== for, а не if! Почему?  
            cond.Wait()  
        }  
        task, tasks = tasks[0], tasks[1:]  
        mu.Unlock()  
  
        task()  
    }  
}  
  
func produce(task func()) {  
    mu.Lock()  
    tasks = append(tasks, task)  
    mu.Unlock()  
  
    cond.Broadcast()  
}
```

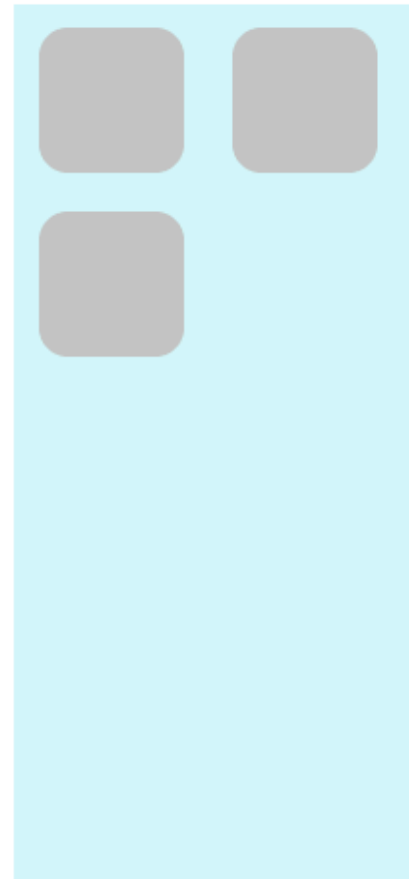
<https://goplay.tools/snippet/FPuOkRPJzDh>

<https://go.dev/play/p/FPuOkRPJzDh>

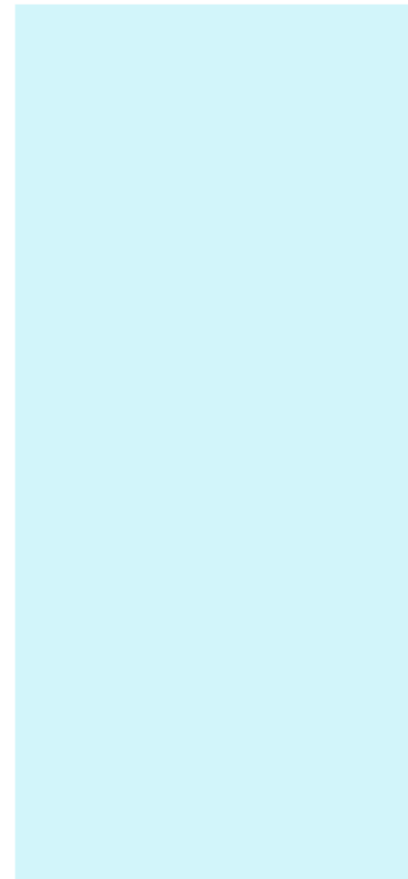
sync.Pool



Memory in use



Garbage



sync.Pool

```
p1 := &Person{}  
p2 := &Person{}  
p3 := &Person{}
```

```
// p1 and p2 → 🗑️
```

Memory in use



Garbage



sync.Pool

```
p1 := &Person{}  
p2 := &Person{}  
p3 := &Person{}
```

```
// p1 and p2 → 🗑️
```

```
p4 := &Person{}
```

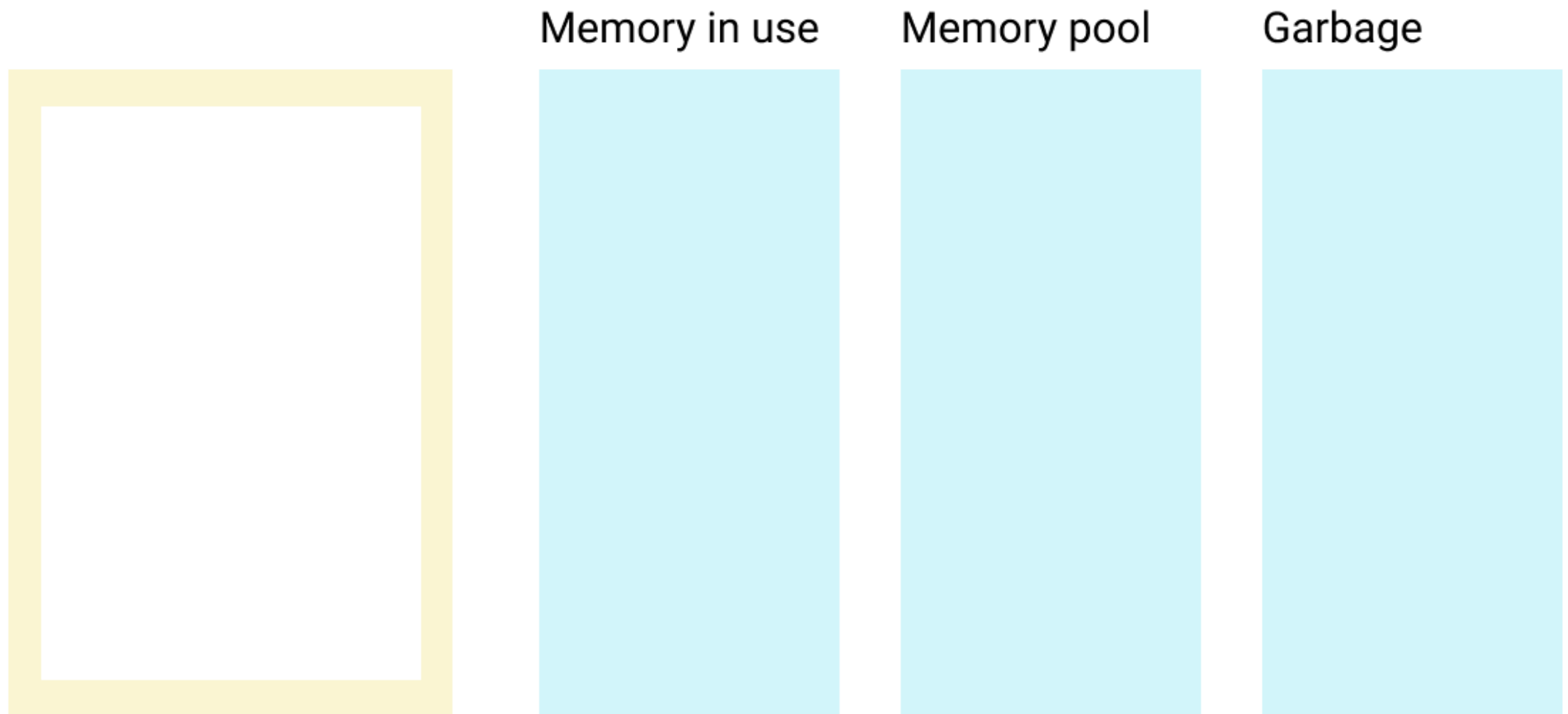
Memory in use



Garbage



sync.Pool



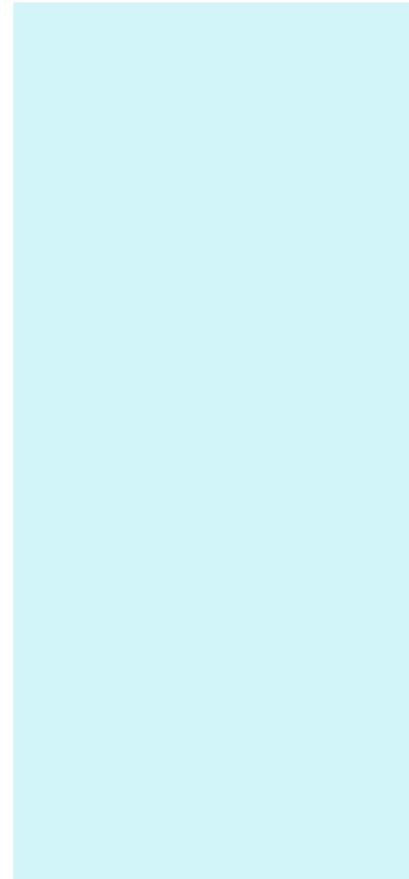
sync.Pool

```
p1 := pool.Get()  
p2 := pool.Get()  
p3 := pool.Get()
```

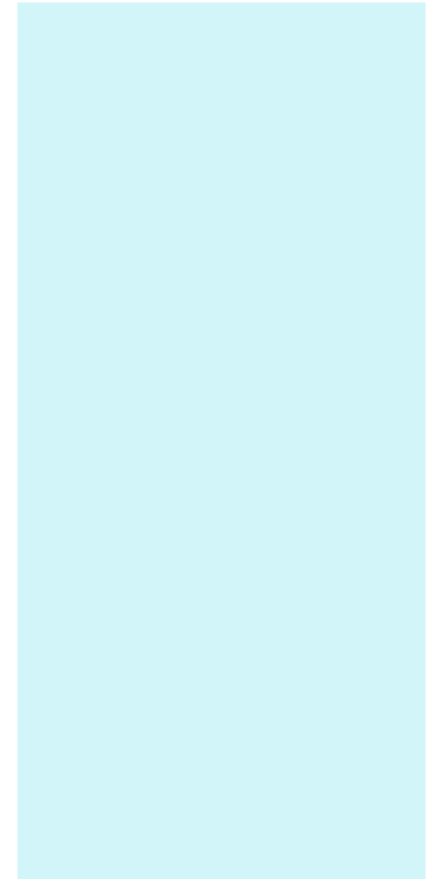
Memory in use



Memory pool



Garbage



sync.Pool

```
p1 := pool.Get()  
p2 := pool.Get()  
p3 := pool.Get()
```

```
pool.Put(p1)  
pool.Put(p2)
```

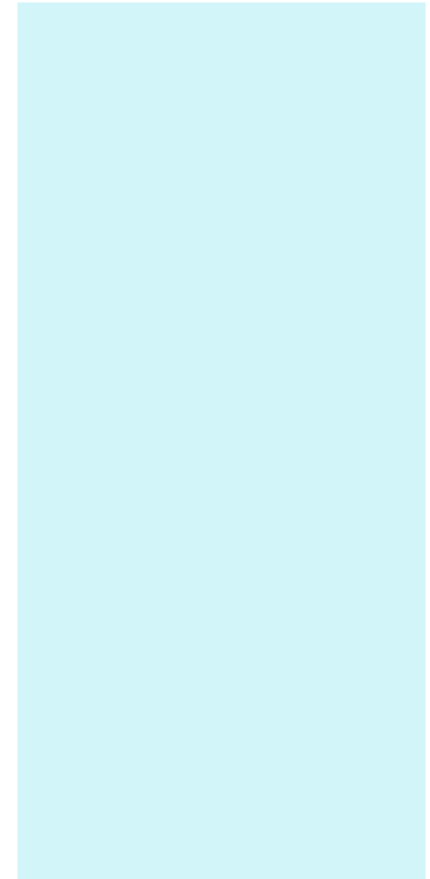
Memory in use



Memory pool



Garbage



sync.Pool

```
p1 := pool.Get()  
p2 := pool.Get()  
p3 := pool.Get()
```

```
pool.Put(p1)  
pool.Put(p2)
```

```
p4 := pool.Get()
```

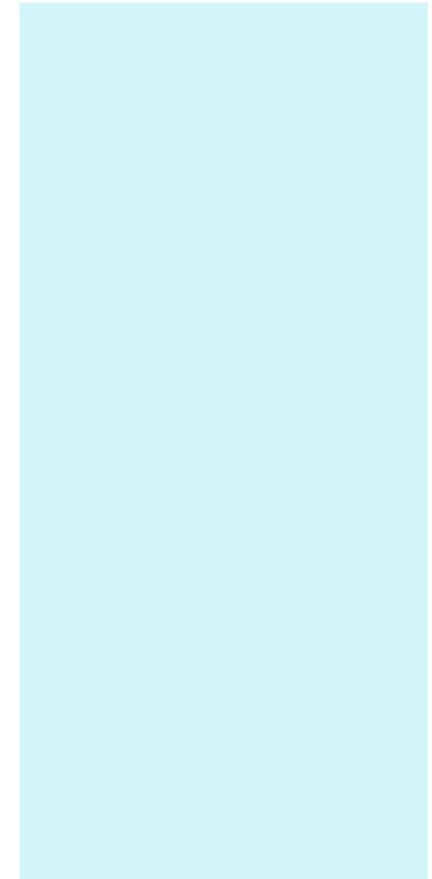
Memory in use



Memory pool



Garbage



sync.Pool

```
p1 := pool.Get()  
p2 := pool.Get()  
p3 := pool.Get()
```

```
pool.Put(p1)  
pool.Put(p2)
```

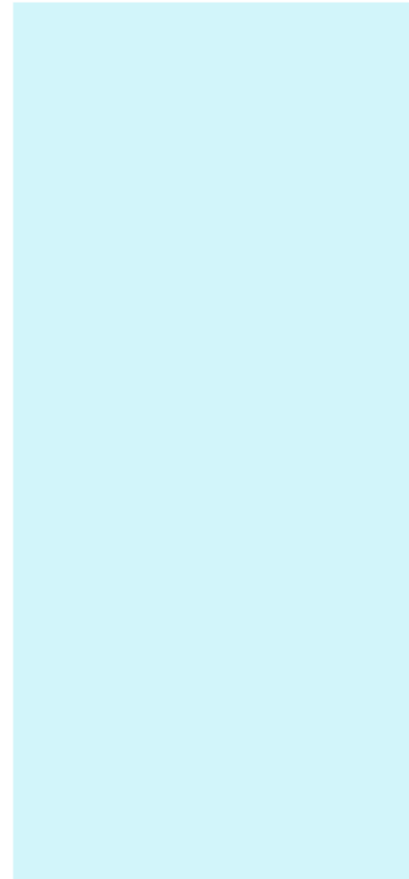
```
p4 := pool.Get()
```

```
// a little later
```

Memory in use



Memory pool



Garbage



sync.Pool

```
type Pool struct {  
    New func() interface{}  
}  
  
func (p *Pool) Get() interface{}  
  
func (p *Pool) Put(x interface{})
```

sync.Pool

```
type Person struct {  
    name string  
}  
  
type PersonPool struct {  
    pool sync.Pool  
}  
  
func NewPersonPool() *PersonPool {  
    return &PersonPool{  
        pool: sync.Pool{  
            New: func() interface{} { return new(Person) },  
        },  
    }  
}  
  
func (p *PersonPool) Get() *Person {  
    return p.pool.Get().(*Person)  
}  
  
func (p *PersonPool) Put(person *Person) {  
    p.pool.Put(person)  
}
```

sync.Pool

```
func BenchmarkWithPool(b *testing.B) {
    pool := NewPersonPool()

    for i := 0; i < b.N; i++ {
        person := pool.Get()
        person.name = "Ivan"
        pool.Put(person)
    }
}

func BenchmarkWithoutPool(b *testing.B) {
    for i := 0; i < b.N; i++ {
        person := &Person{name: "Ivan"}
        gPerson = person
    }
}
```

sync.Pool

```
func BenchmarkWithPoolGC(b *testing.B) {
    pool := NewPersonPool()

    for i := 0; i < b.N; i++ {
        person := pool.Get().(*Person)
        person.name = "Ivan"
        pool.Put(person)
        if (i % gcFreq) == 0 {
            runtime.GC()
        }
    }
}

func BenchmarkWithoutPoolGC(b *testing.B) {
    for i := 0; i < b.N; i++ {
        person := &Person{name: "Ivan"}
        gPerson = person
        if (i % gcFreq) == 0 {
            runtime.GC()
        }
    }
}
```

atomic: Add

```
func main() {  
    wg := sync.WaitGroup{}  
  
    var v int64  
    for i := 0; i < 1000; i++ {  
        wg.Add(1)  
        go func() {  
            atomic.AddInt64(&v, 1) // атомарный v++  
            wg.Done()  
        }()  
    }  
  
    wg.Wait()  
    fmt.Println(v)  
}
```

<https://goplay.tools/snippet/SHwBoHdsUsg>

<https://goplay.tools/snippet/SHwBoHdsUsg>

atomic: Store/Load

```
func main() {  
    wg := sync.WaitGroup{}  
    var luckyNumber int32 = 7  
  
    wg.Add(2)  
  
    go func() {  
        atomic.StoreInt32(&luckyNumber, 42)  
        wg.Done()  
    }()  
  
    go func() {  
        fmt.Println(atomic.LoadInt32(&luckyNumber))  
        wg.Done()  
    }()  
  
    wg.Wait()  
}
```

<https://goplay.tools/snippet/1CKmqzRzXfb>

<https://go.dev/play/p/1CKmqzRzXfb>

atomic: CompareAndSwap

Вариант реализации sync.Once

```
const (
    onceInit int32 = iota
    onceInit int32 = iota
    onceDoing
    onceDone
)

type Once struct {
    state int32
}

func (o *Once) Do(fn func()) {
    if atomic.CompareAndSwapInt32(&o.state, onceInit, onceDoing) {
        fn()
        atomic.StoreInt32(&o.state, onceDone)
        return
    }

    for atomic.LoadInt32(&o.state) == onceDoing {
        time.Sleep(10 * time.Nanosecond)
    }
}
```

<https://goplay.tools/snippet/QayPezK-9xl>

<https://go.dev/play/p/QayPezK-9xl>

Вопросы?



Ставим “+”,
если вопросы есть



Ставим “-”,
если вопросов нет



**Заполните, пожалуйста,
опрос о занятии
по ссылке в чате**

Следующий вебинар

Concurrency patterns



Ссылка на вебинар будет в ЛК за 15 минут



Материалы к занятию в ЛК — можно изучать



Обязательный материал обозначен красной лентой

Спасибо за внимание!

Приходите на следующие вебинары

