GoBench: A Benchmark Suite of Real-World Go Concurrency Bugs

Ting Yuan, Guangwei Li, Jie Lu, Chen Liu, Lian Li, and Jingling Xue

Institute of Computing Technology
of the Chinese Academy of Sciences

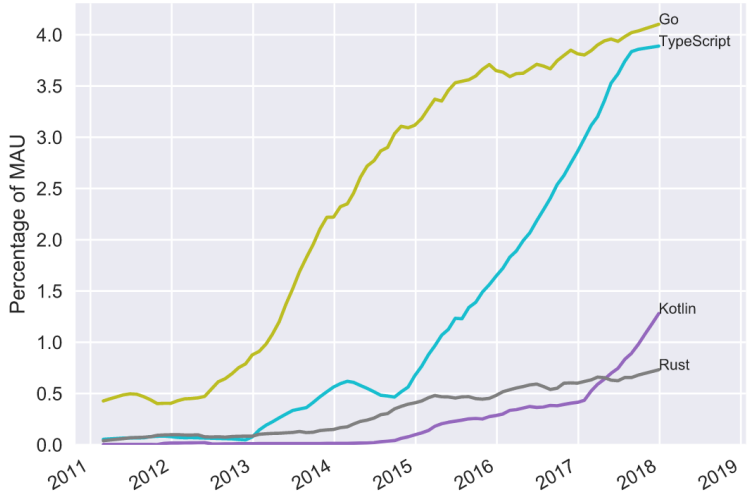
University of Chinese
Academy of Sciences

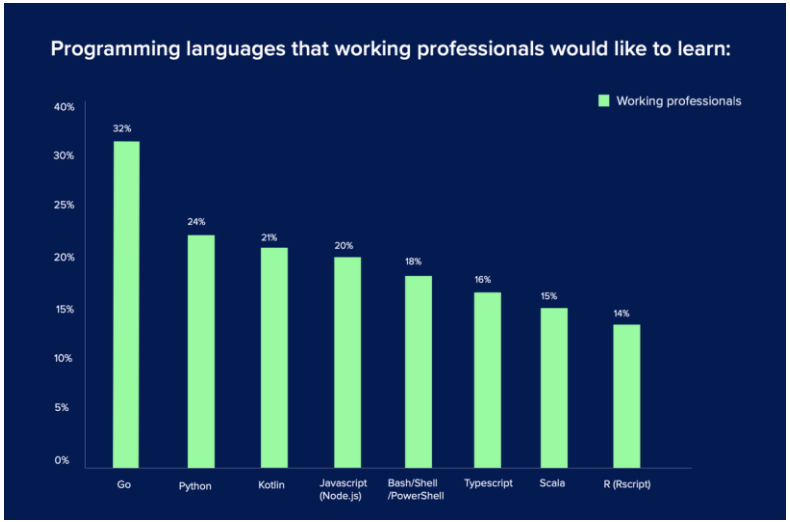
UNSW SYDNEY

University of New South Wales

Go is popular

- Go is a language with the fastest-growing user base since 2011.



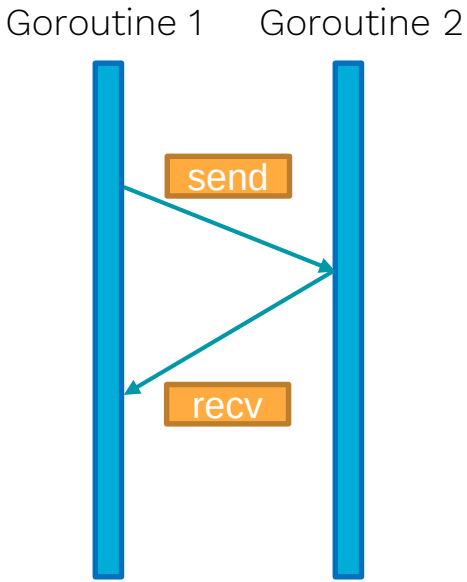


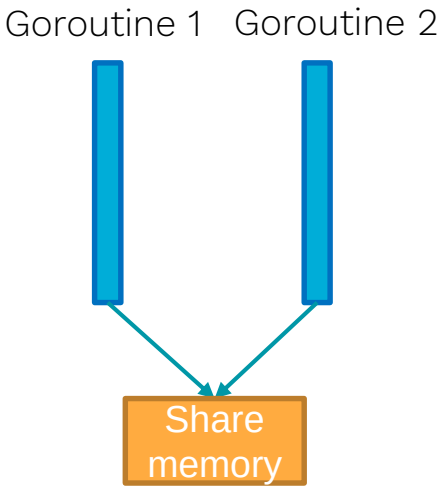
[1] Ranking Programming Languages by GitHub Users

[2] Developers say Google's Go is 'most sought after' programming language of 2020

Concurrency in Go

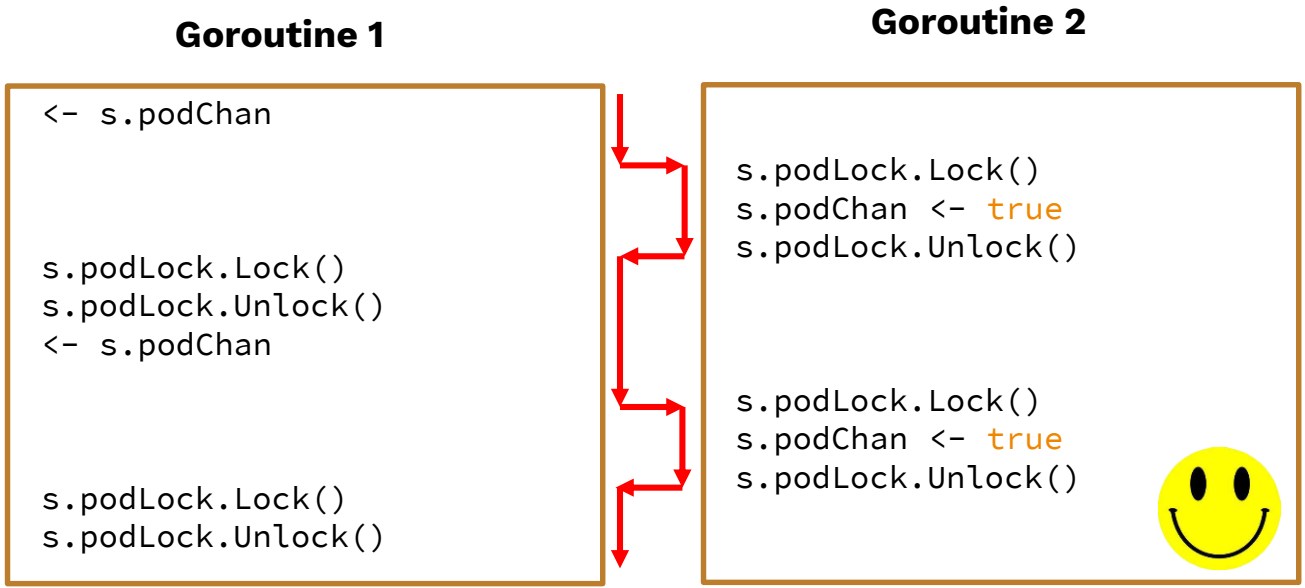
- Message passing and shared memory are widely used in real world Go applications.





Concurrency in Go

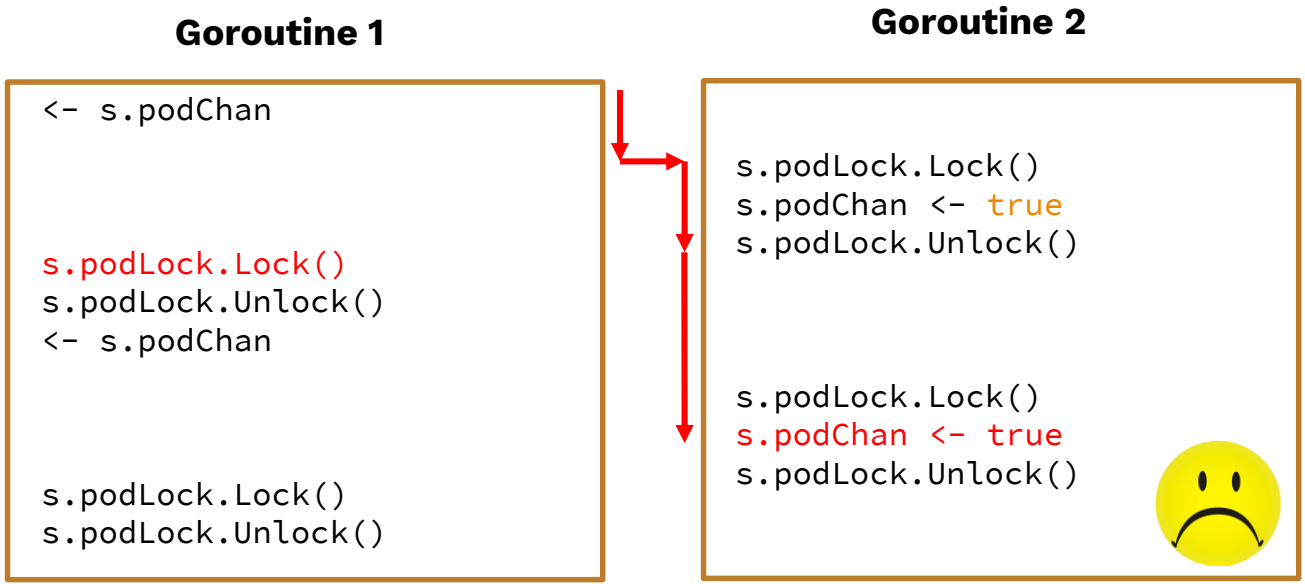
- However, using the two mechanisms together may easily lead to mistakes.



An example from Kubernetes

Concurrency in Go

- However, using the two mechanisms together may easily lead to mistakes.



An example from Kubernetes

Concurrency in Go

- And there are also many Go specific non-blocking bugs

Goroutine 1

```
select {
    case <- s.donec:
        return
    default:
}
```

Goroutine 2

```
close(s.donec)
s.donec = nil
```

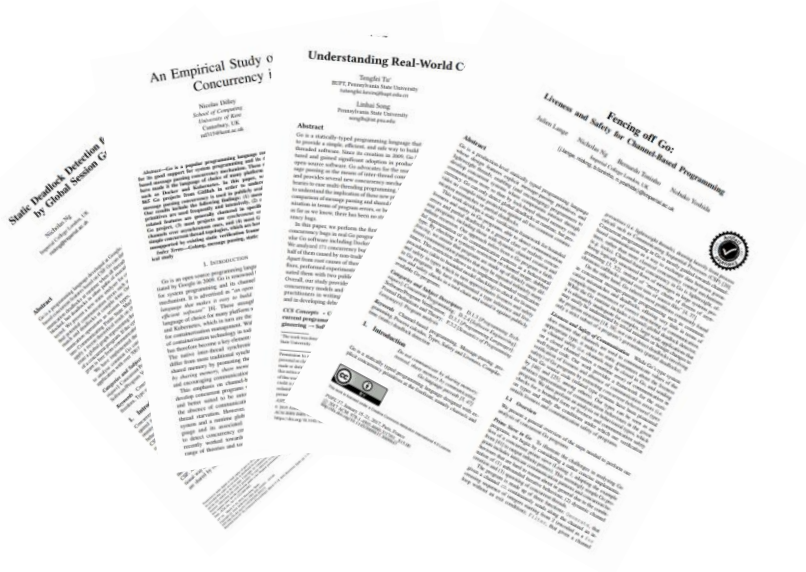
Channel misuse in Istio

```
for _, c := range checks {
    go func() {
        CheckInTxn(&c.Name)
    }
}
```

Anonymous function misuse in CockroachDB

Motivation

- Researches on *Go concurrency bugs* gradually rise



- Open source detectors:
  - goleak
  - go-deadlock
  - dingo-hunter
  - ... ...
- But so far there is no measurement to evaluate concurrency bug detectors on Go!

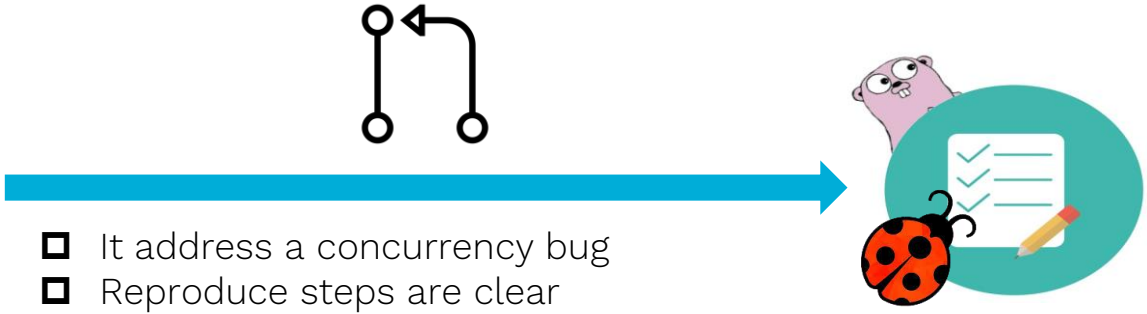
Overview of $GoBench$

- A benchmark suite of real-world $Go\ concurrency\ bugs$
- $GoBench$ is composed of $GoReal$ and $GoKer$
- $GoReal$: 82 representative bugs found in 9 popular open source applications
- $GoKer$: 103 bug kernels extracted from $GoReal$ and a recent study [1]

[1] Tu, Tengfei, et al. “Understanding real-world concurrency bugs in Go”. ASPLOS 2019.

GoReal: Real world bugs

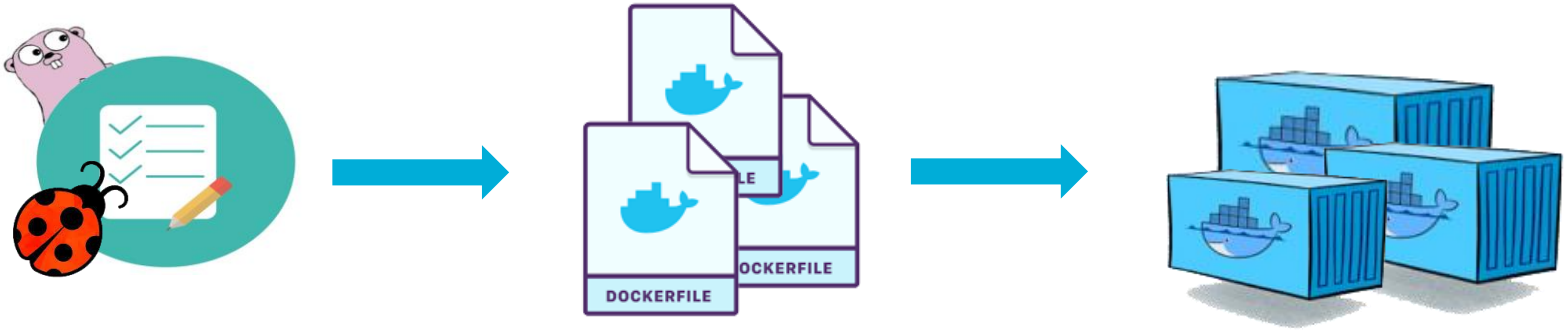
- Collect concurrency bugs in pull requests



- It address a concurrency bug
- Reproduce steps are clear
- There is a test function as the entry point

GoReal: Real world bugs

- Package those bugs into Dockerfiles



GoReal: Real world bugs

- Bug classification

Bug Type (#Bugs)	
Blocking Bugs (40)	Resource Deadlock (9)
	Communication Deadlock (21)
	Mixed Deadlock (10)
Non-blocking Bugs (42)	Traditional Bugs (24)
	Go-specific Bugs (18)

GoKer: Kernels extracted from real world bugs

- 36 bug kernels are reconstructed from a recent study.[1]
- 67 bug kernels are extracted from *GoReal*

There's a deadlock in `assignSimpleTokenToUser`. The function acquires lock `as.simpleTokensMu` and posts to `addSimpleTokenCh` (suppose that the channel is full so it blocks). If the goroutine `simpleTokenTTL-Keeper.run` happens to hit `<-tokenTicker.C`, it will try to acquire `simpleTokensMu` while calling `delete-TokenFunc`. Since only the goroutine `simpleToken-TTLKeeper.run` can drain `addSimpleTokenCh`, the lock is never released.

```
G1 [semacquire]:
.../auth.newDeleter.func1(...)
.../auth.(*simpleTokenTTLKeeper).run(...)
created by .../etcd/auth.NewSimpleTokenTTLKeeper

G4 [chan send]:
.../auth.(*simpleTokenTTLKeeper).addSimpleToken(...)
.../auth.(*tokenSimple).assignSimpleTokenToUser(...)
.../auth.(*tokenSimple).assign(...)
.../auth.(*authStore).Authenticate(...)
created by .../etcd/auth.TestHammerSimpleAuthenticate
```

[1] Tu, Tengfei, et al. "Understanding real-world concurrency bugs in Go". ASPLOS 2019.

GoKer: Kernels extracted from real world bugs

- We manually extract the code snippets into a separate test function.

```
func newDeleterFunc(t *tokenSimple) func(string) {
    return func(tk string) {
        t.simpleTokensMu.Lock()
        defer t.simpleTokensMu.Unlock()
-       if username, ok := t.simpleTokens[tk]; ok {
-           plog.Infof("deleting token %s for user %s", tk, username)
-           delete(t.simpleTokens, tk)
-       }
    }
}
type simpleTokenTTLKeeper struct {
    tokens              map[string]time.Time
-   addSimpleTokenCh    chan string
+   addSimpleTokenCh    chan struct{}
-   resetSimpleTokenCh  chan string
-   deleteSimpleTokenCh chan string
-   stopCh              chan chan struct{}
    deleteTokenFunc     func(string)
}
```

```
func TestEtcd7492(t *testing.T) {
    ... ...
    as := setupAuthStore()    // Fork G1
    var wg sync.WaitGroup
-   wg.Add(len(users))
+   wg.Add(3)
-   for u := range users {
+   for i := 0;i < 3; i ++ {
-       go func(user string) {
+       go func() {    // Fork G2, G3, and G4
            defer wg.Done()
-           ... ...
-           _, err := as.AuthInfoFromCtx(ctx)
-           if err != nil {
-               t.Fatal(err)
-           }
-       }(u)
+           as.Authenticate()
+       }()
    }
-   time.Sleep(time.Millisecond)
    wg.Wait()
    ... ...
}
```

GoKer: Kernels extracted from real world bugs

- Bug classification

Bug Type (#Bugs)	
Blocking Bugs (68)	Resource Deadlock (23)
	Communication Deadlock (29)
	Mixed Deadlock (16)
Non-blocking Bugs (35)	Traditional Bugs (21)
	Go-specific Bugs (14)

Evaluation

- Blocking bugs

Static tools: *dingo-hunter*
Dynamic tools: *go-leak, go-deadlock*

- Non-blocking bugs

Dynamic tools: *built-in race detector (Go-rd)*

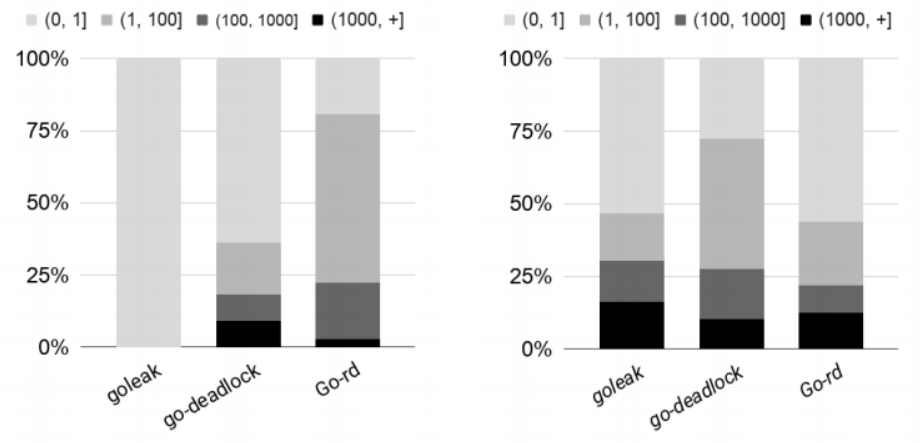
Blocking bugs

Suite	Bug Type	*goleak*	*go-deadlock*	*dingo-hunter*
		# TP/FN/FP	# TP/FN/FP	# TP/FN/FP
GoReal	Resource Deadlock	1/8/1	7/2/0	-/-/-
	Communication Deadlock	8/13/0	1/20/4	-/-/-
	Mixed Deadlock	3/7/1	4/6/3	-/-/-
	Total	12/28/2	12/28/7	-/-/-
GoKer	Resource Deadlock	14/9/0	23/0/0	0/23/0
	Communication Deadlock	20/9/0	0/29/0	1/28/0
	Mixed Deadlock	9/7/0	6/10/0	0/16/0
	Total	43/25/0	29/39/0	1/67/0

Non-blocking bugs

Suite	Bug Type	*Go-rd*		
		#TP	#FN	#FP
GoReal	Traditional	23	1	0
	Go-specific	13	5	0
	Total	36	6	0
GoKer	Traditional	21	0	0
	Go-specific	11	3	0
	Total	32	3	0

Efficiency of dynamic tools

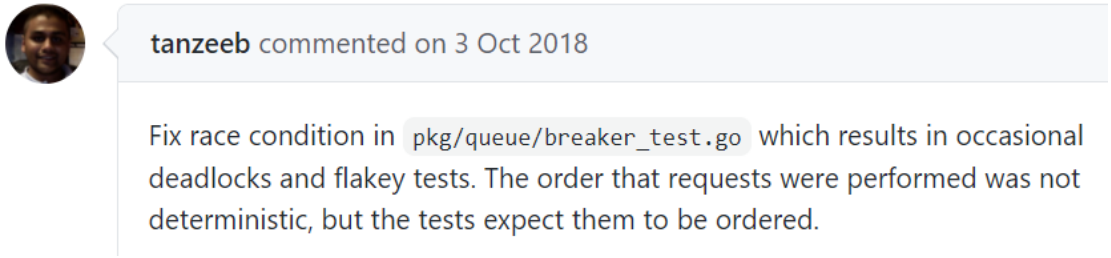


(a) GoReal (b) GoKer

Fig. 10. Percentage distribution for the (average) number of runs falling into each of the four given intervals that is needed by a tool in finding a bug.

A case study (serving#2137)

- A mixed deadlock bug reported in Knative/serving



https://github.com/knative/serving/pull/2137

A case study (serving#2137)

- Goroutines in this case are spawn within a for loop. Multiple buffered channels are involved in the mixed deadlock, and their buffer sizes are different.

- Currently, there is no static tool that can detect it. Dynamic tools require tens of thousands of times to trigger the bug.

- You can try its bug kernel: https://github.com/timmyyuan/gobench/blob/master/gobench/goker/blocking/serving/2137/serving2137_test.go

Conclusion

- $GoBench$ is the first benchmark suite of real-world $Go\ concurrency\ bugs$.
- Static tools need to improve the effectiveness of finding concurrency bugs in Go.
- Dynamic tools need to improve the efficiency of finding concurrency bugs in Go.

Conclusion

- We publish $GoBench$ at https://github.com/timmyyuan/gobench
- We believe $GoBench$ will be instrumental in helping researchers understand concurrency bugs in Go and develop effective tools for their detection.

THANK YOU

Q&A

Contact

- Ting Yuan
  - yuanting@ict.ac.cn