So You Wanna Go Fast?

Strange Loop 2017

@tyler_treat

BRAVE NEW GEEK

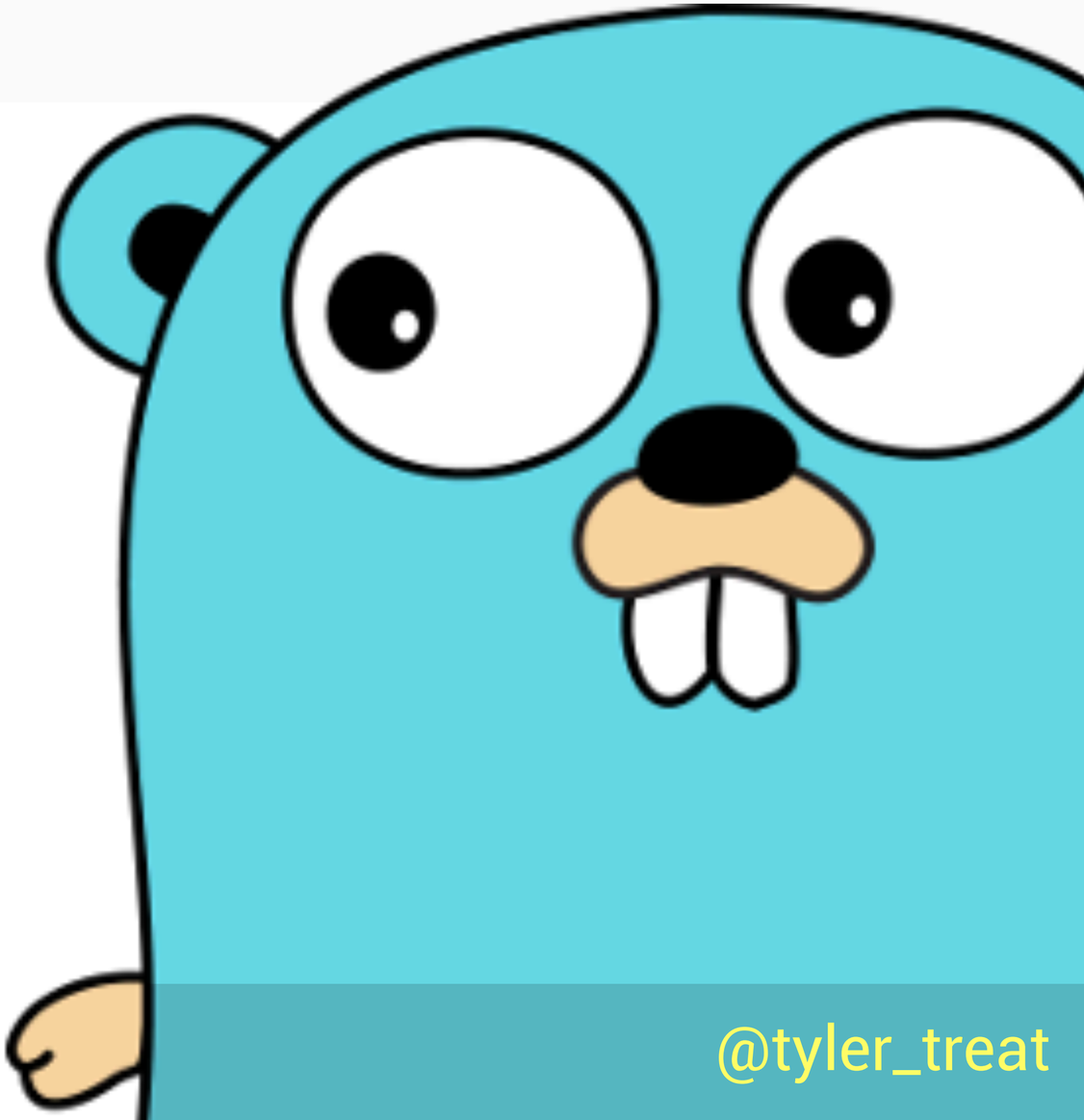
Introspections of a software engineer

MAY 20, 2015 BY TYLER TREAT

Go Is Unapologetically Flawed, Here's Why We Use It

thoughtleading thoughtleading thoughtleading thoughtleading thoughtleading thoughtleading thoughtleading thoughtleading thoughtleading thoughtleading thoughtleading thoughtleading thoughtleading thoughtleading thoughtleading thoughtleading thoughtleading thoughtleading

thoughtleading thoughtleading thoughtleading thoughtleading thoughtleading thoughtleading thoughtleading thoughtleading thoughtleading thoughtleading



@tyler_treat

Make your code faster with

this one weird trick

~~Make your code faster with~~

~~**this one weird trick**~~

So You Wanna Subvert Go?

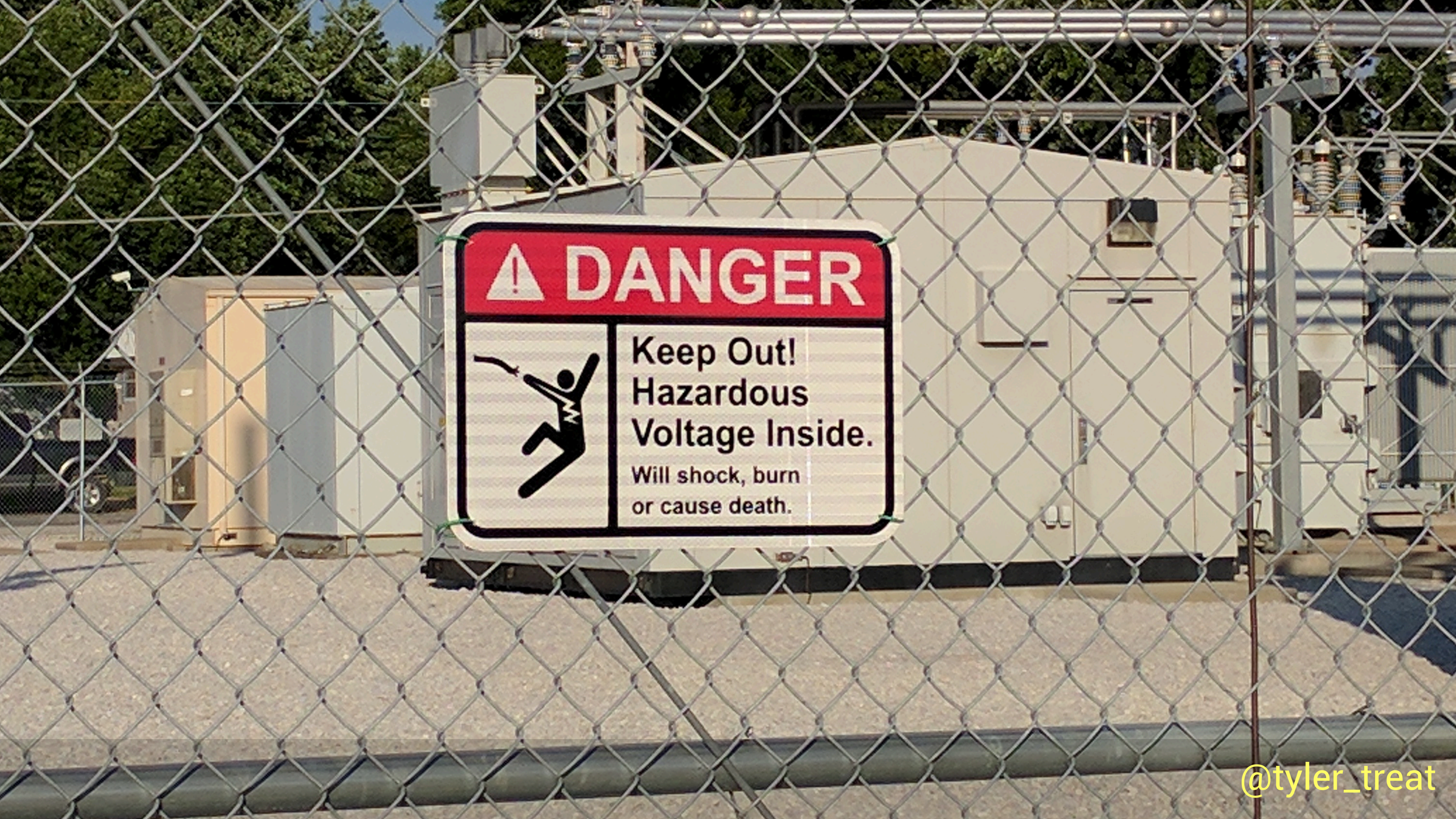
Spoiler Alert:
Go is not a
systems language…

but that doesn’t mean you can’t build internet-scale systems with it.

docker
etcd
Grafana
Cayley
influxdata
NATS
Cockroach
Caddy

@tyler_treat

This is a talk about how to write *terrible* Go code.


DANGER
Keep Out!
Hazardous
Voltage Inside.
Will shock, burn
or cause death.
@tyler_treat

Because this is a talk about **trade-offs.**

Tyler Treat

- Messaging Nerd @ Apcera
- Working on nats.io
- Distributed systems
- bravenewgeek.com

NATS
Open Source. Simple. Secure. Scalable.
Introducing NATS Cloud: NATS, Maintained by our team - so you can focus on innovation, not infrastructure.
Learn More
@tyler_treat

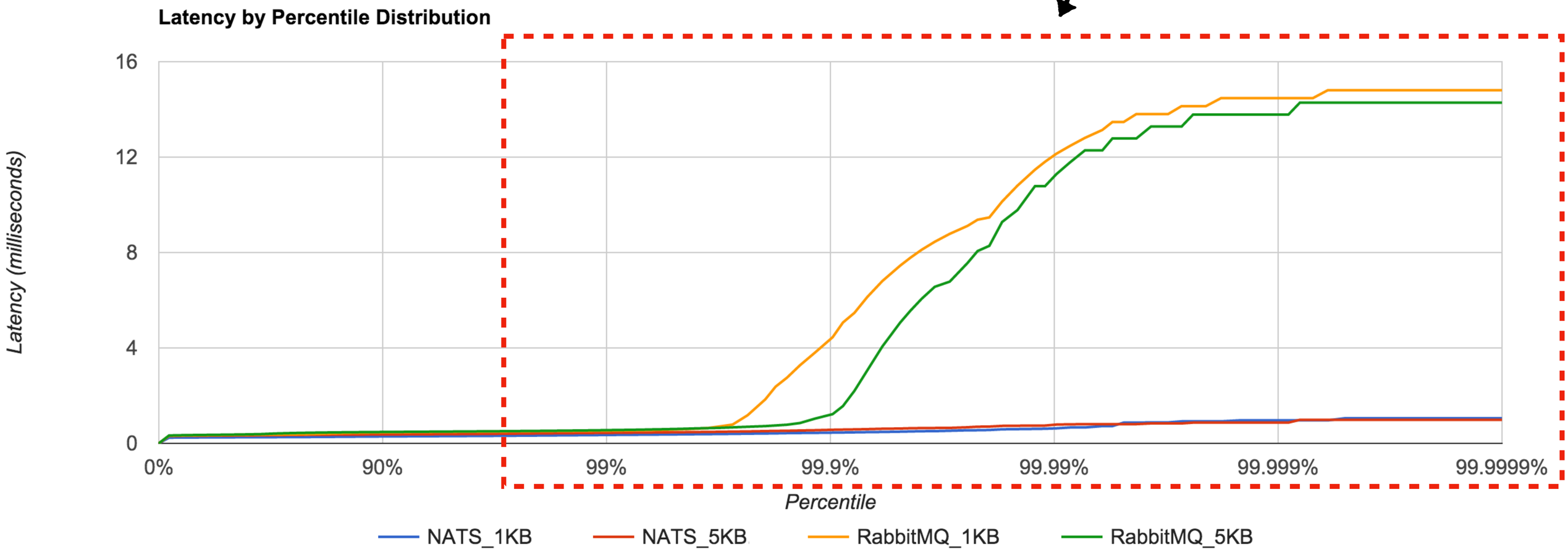
Why does this talk **matter?**

The compiler isn’t magic.

The compiler isn’t magic.

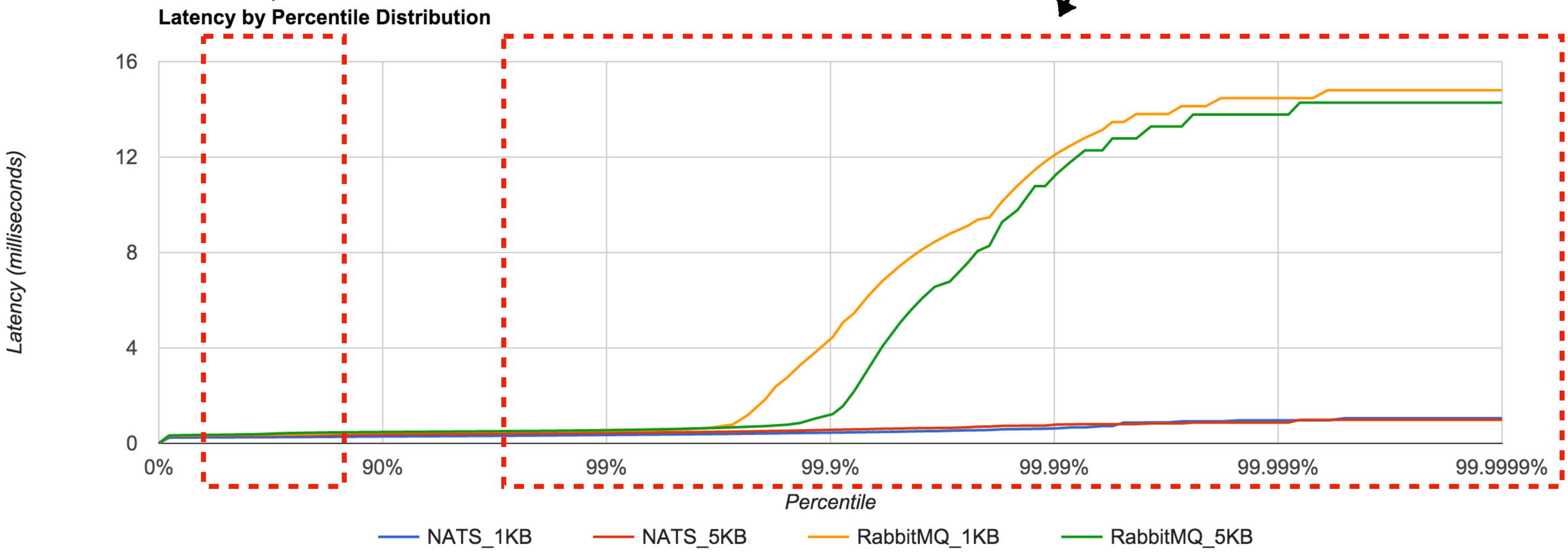
You have to be
mindful of performance
when it matters.

Where bad things hide



Where we're usually looking

Where bad things hide



Tire fires
at scale
@tyler_treat

LN Performance_Reporting.ss

File Edit Data View

15 Publish | Undo | Redo | Save As | Import | Export | Properties | Permissions | Rename

A1 Consolidated

LN Performance_Reporting.ss

- IS-Consolidated
  - IS-Segment 1
  - IS-Segment 2
  - IS-Segment 3
  - IS-Segment 4
  - IS-Segment 5
- Projections
  - Revenue Projections
  - Revenue Budget
  - Operational Metrics
- DRIVERS
  - Dates
  - Company Information
  - Account Mapping
- SOURCES
  - Source_TB
  - Source_Budget
  - Public API

	A	F	G	H	I	K	L	M	N	P	Q	R	S
1	Consolidated												
2	Statement of Earnings			0									
3	3/31/2017	(Shown in millions)				(Shown in millions)				(Shown in millions)			
4		Actuals	Budget			Actuals	Budget			Actuals	Budget		
5		YTD	YTD	Fav/(Unfav)		QTD	QTD	Fav/(Unfav)		MTD	MTD	Fav/(Unfav)	
6		3/31/2017	3/31/2017	Var.	%	3/31/2017	3/31/201	Var.	%	3/31/2017	3/31/2017	Var.	%
7	Installs	5,009	4,920	89	2 %	-7,049	-13,818	6,769	(49)%	1,935	1,803	132	7 %
8	Delivery Service	4,519	4,644	-125	(3)%	-6,852	-13,043	6,191	(47)%	1,626	1,702	-76	(4)%
9	**Service Revenue**	**9,528**	**9,564**	**-36**	**—)%**	**-13,901**	**-26,861**	**12,960**	**(48)%**	**3,561**	**3,505**	**56**	**2 %**
10	Furniture/HomeGoods In-store	2,603	2,636	-33	(1)%	-3,901	-8,499	4,598	(54)%	963	965	-2	(—)%
11	Flooring In-store	2,531	2,571	-40	(2)%	-3,763	-7,220	3,457	(48)%	941	942	-1	(—)%
12	Tools In-store	1,060	1,000	60	6 %	-1,401	-2,813	1,412	(50)%	344	366	-22	(6)%
13	**In-store Revenue**	**6,194**	**6,207**	**-13**	**—)%**	**-9,065**	**-18,532**	**9,467**	**(51)%**	**2,248**	**2,273**	**-25**	**(1)%**
14	Furniture/HomeGoods Online	1,778	1,733	45	3 %	-2,466	-4,720	2,254	(48)%	712	633	79	12 %
15	Flooring Online	1,543	1,607	-64	(4)%	-2,540	-4,513	1,973	(44)%	556	589	-33	(6)%
16	Tools Online	917	953	-36	(4)%	-1,423	-2,780	1,357	(49)%	330	349	-19	(5)%
17	**Online Sales**	**4,238**	**4,293**	**-55**	**(1)%**	**-6,429**	**-12,013**	**5,584**	**(46)%**	**1,598**	**1,571**	**27**	**2 %**
18	**Net Sales**		**20,064**	**-20,064**	**00)%**	**-29,395**	**-57,406**	**28,011**	**(49)%**	**7,407**	**7,349**	**58**	**1 %**
19													
20	Cost of Goods Sold	12,071	12,571	500	4 %	-18,709	-35,253	-16,544	47 %	4,345	4,607	262	6 %
21	Discounts	-288	-300	-12	4 %	445	840	395	47 %	-104	-111	-7	6 %
22	Deferred Interest	256	267	11	4 %	-397	-749	-352	47 %	92	98	6	6 %
23	Gift Cards	63	66	3	5 %	-97	-183	-86	47 %	23	26	3	12 %
24	**Gross margin**	**-12,102**	**7,460**	**-19,562**	**62)%**	**-10,637**	**-22,061**	**11,424**	**(52)%**	**3,051**	**2,729**	**322**	**12 %**
25													
26	Advertising & Promotion	471	491	20	4 %	-729	-1,376	-647	47 %	171	180	9	5 %

@tyler_treat

LN Performance_Reporting.ss

File Edit Data View

15 Publish | Undo | Redo | Save As | Import | Export | Properties | Permissions | Rename

A1 Consolidated

LN Performance_Reporting.ss

- IS-Consolidated
  - IS-Segment 1
  - IS-Segment 2
  - IS-Segment 3
  - IS-Segment 4
  - IS-Segment 5
- Projections
  - Revenue Projections
  - Revenue Budget
  - Operational Metrics
- DRIVERS
  - Dates
  - Company Information
  - Account Mapping
- SOURCES
  - Source_TB
  - Source_Budget
  - Public API

(Shown in millions)

		Actuals YTD 3/31/2017	Budget YTD 3/31/2017	Fav/(Unfav) Var.	%		Fav/(Unfav) Var.	%	Actuals MTD 3/31/2017	Budget MTD 3/31/2017	Fav/(Unfav) Var.	%
7	Installs	5,009	4,920	89	2 %	-13,818	6,769	(49)%	1,935	1,803	132	7 %
8	Delivery Service	4,519	4,644	-125	(3)%	-13,043	6,191	(47)%	1,626	1,702	-76	(4)%
9	**Se ue**	**9,528**	**9,564**	**-36**	**(—)%**	**-26,861**	**12,960**	**(48)%**	**3,561**	**3,505**	**56**	**2 %**
10	Furniture/ ore	2,603	2,636	-33	(1)%	-8,499	4,598	(54)%	963	965	-2	(—)%
11	Floor	2,531	2,571	-40	(2)%	-7,220	3,457	(48)%	941	942	-1	(—)%
12									344	366	-22	(6)%
13									**2,248**	**2,273**	**-25**	**(1)%**
									712	633	79	12 %
									556	589	-33	(6)%
16									330	349	-19	(5)%
17									**1,598**	**1,571**	**27**	**2 %**
18	les		**20,064**	**-20,064**	**00)%**	**-29,395**	**-57,406**	**28,011**	**(49)%**	**7,407**	**7,349**	**58**	**1 %**
19												
20	Cost of Goods	12,071	12,571	500	4 %	-18,709	-35,253	-16,544	47 %	4,345	4,607	262	6 %
21	Discounts	-288	-300	-12	4 %	445	840	395	47 %	-104	-111	-7	6 %
22	Deferred Interest	256	267	11	4 %	-397	-749	-352	47 %	92	98	6	6 %
23	Gift Cards	63	66	3	5 %	-97	-183	-86	47 %	23	26	3	12 %
24	**Gross margin**	**-12,102**	**7,460**	**-19,562**	**62)%**	**-10,637**	**-22,061**	**11,424**	**(52)%**	**3,051**	**2,729**	**322**	**12 %**
25												
26	Advertising & Promotion	471	491	20	4 %	-729	-1,376	-647	47 %	171	180	9	5 %

@tyler_treat

Workiva / go-datastructures

Watch 205 | Unstar 3,238 | Fork 306

Code | Issues 5 | Pull requests 3 | Projects 0 | Wiki | Insights

No description, website, or topics provided.

629 commits | 11 branches | 37 releases | 31 contributors | Apache-2.0

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

tylertreat-wf committed on GitHub Merge pull request #169 from andygrunwald/set-unit-tests ... Latest commit 01ff43b on May 3

augmentedtree	Don't panic on traversing an empty tree	5 months ago
batcher	Ups delta in batcher test case.	2 years ago
bitarray	Merge pull request #149 from jxguan/patch-1	11 months ago
btree	Sort keys before deleting.	a year ago
cache	Use Testify assertions	7 months ago
common	Fixed up imports in the skiplist rangetree.	2 years ago
fibheap	[Test] Revise benchmarks for Fibonacci heap	4 months ago
futures	Merge pull request #157 from seanstrickland-wf/AddHasResultToFutures	8 months ago
hashmap/fastinteger	Fix deletion on collision.	2 years ago
list	Implement list Map	2 years ago
mock	Update mock.	2 years ago
numerics	Fix unit tests	2 years ago
queue	fix queue.Poll() memory leak	a year ago

@tyler_treat

Overview

- Measuring performance
- Language features
- Memory management
- Concurrency and multi-core

Overview

- Measuring performance
- Language features
- Memory management
- Concurrency and multi-core

Disclaimer:

Don’t blindly apply optimizations presented.

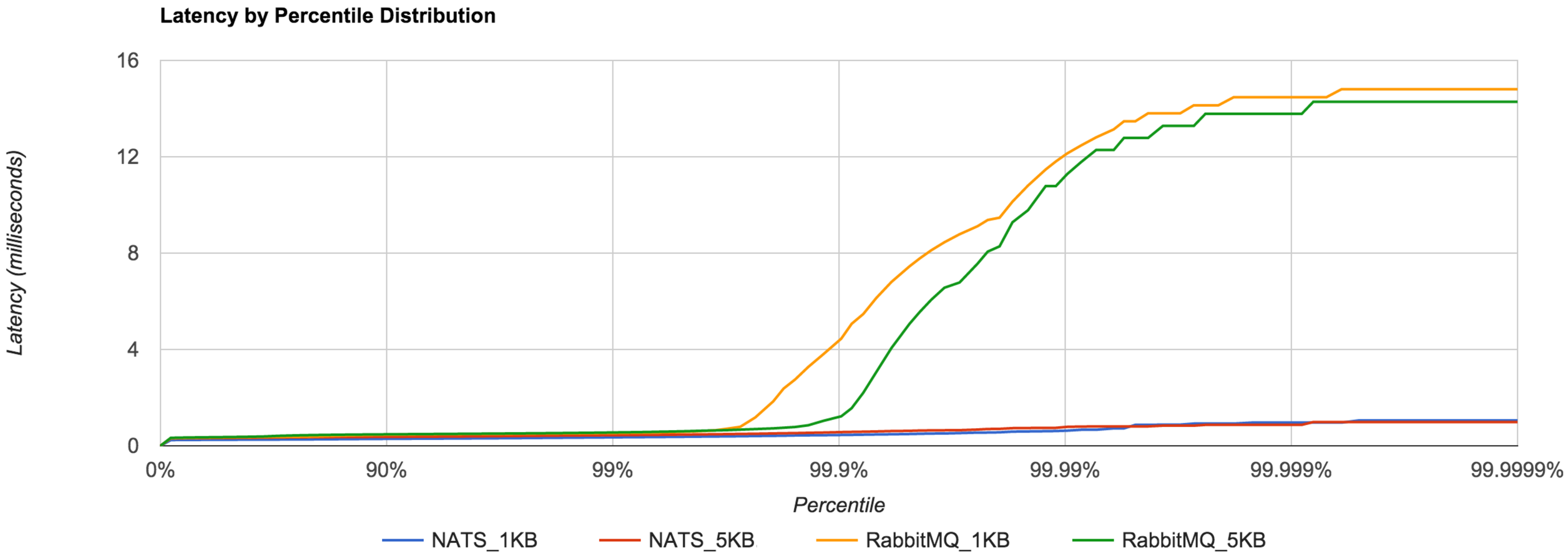
tl;dr of this talk is

“IT DEPENDS!”

Measure
Optimize

Measurement Techniques

- pprof
    - memory
    - cpu
    - blocking
- GODEBUG
    - gctrace
    - schedtrace
    - allocfreetrace
- Benchmarking
    - Code-level: testing.B
    - System-level: HdrHistogram (https://github.com/codahale/hdrhistogram)
      bench (https://github.com/tylertreat/bench)


Latency by Percentile Distribution
16
12
8
4
0
Latency (milliseconds)
0%
90%
99%
99.9%
99.99%
99.999%
99.9999%
Percentile
NATS_1KB
NATS_5KB
RabbitMQ_1KB
RabbitMQ_5KB

The only way to get good at something is to be **really fucking bad at it** for a long time.

Benchmarking…

a great way to rattle the
Hacker News fart chamber.

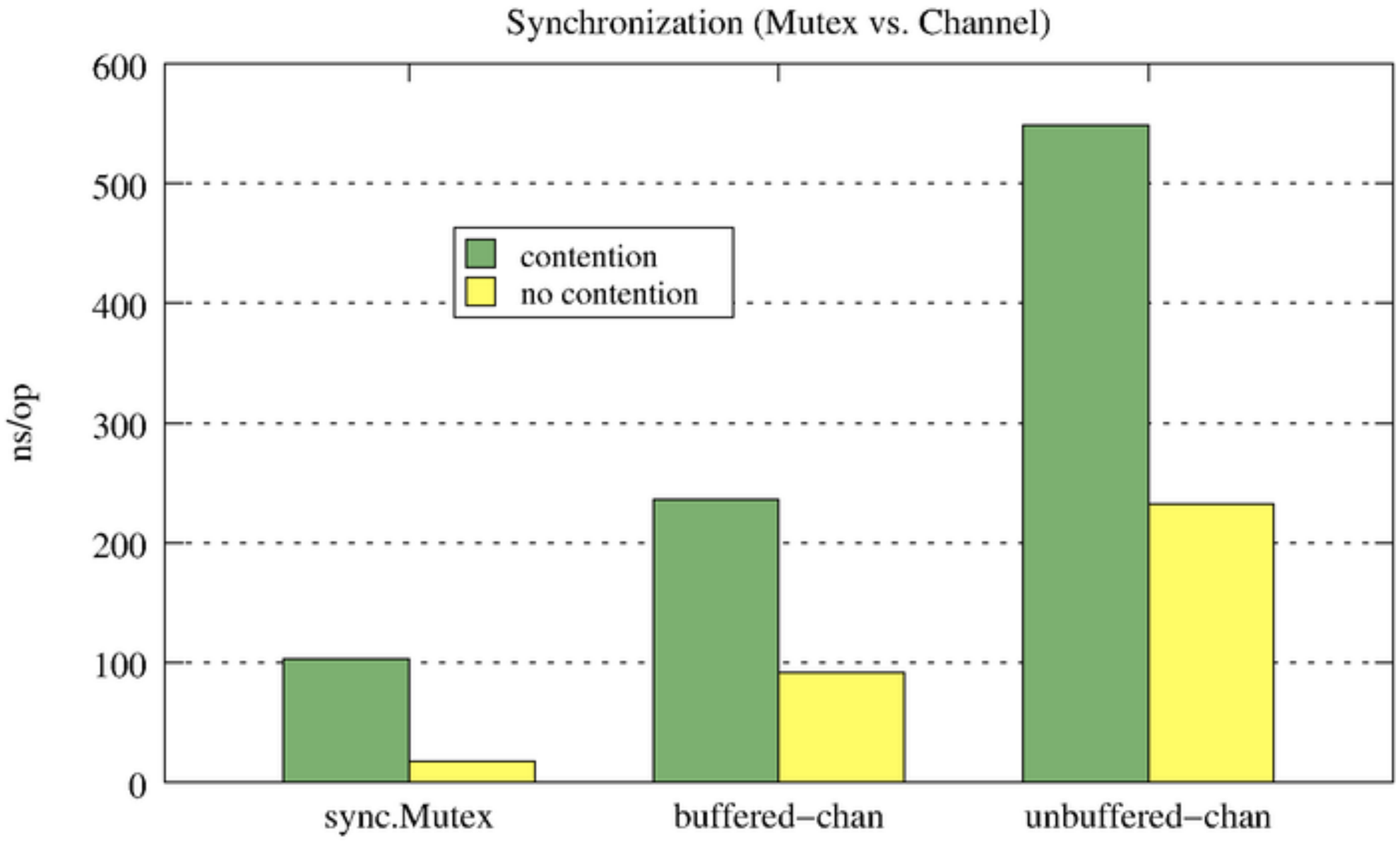
Overview

- Measuring performance
- Language features
- Memory management
- Concurrency and multi-core

channels

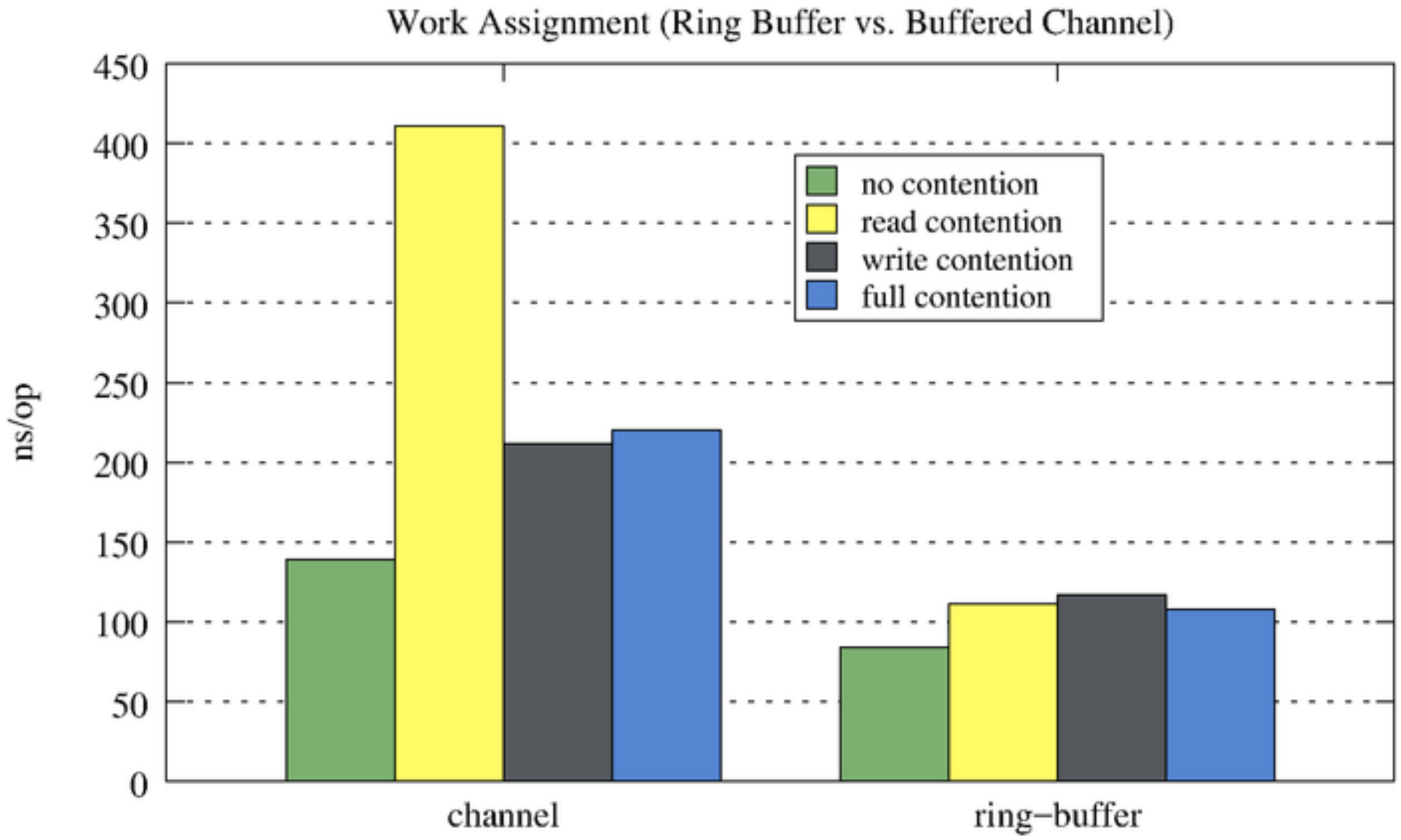
“Instead of explicitly using locks to mediate access to shared data, Go encourages the use of channels to pass references to data between goroutines.”

https://blog.golang.org/share-memory-by-communicating


Synchronization (Mutex vs. Channel)
ns/op
600
500
400
300
200
100
0
contention
no contention
sync.Mutex
buffered–chan
unbuffered–chan

IT CAME TO THEM WITH
A MESSAGE.

BUT THEY COULD NOT UNDERSTAND ITS ALIEN LANGUAGE
USE CHANNELS TO COORDINATE, NOT SYNCHRONIZE.
FUCKING WHAT?
poorlydrawnlines.com


Work Assignment (Ring Buffer vs. Buffered Channel)
ns/op
450
400
350
300
250
200
150
100
50
0
no contention
read contention
write contention
full contention
channel
ring−buffer

golang / go

Watch 2,311 | Unstar 30,025 | Fork 4,030

Code | Issues 2,891 | Pull requests 0 | Wiki | Insights

runtime: lock-free channels #8899

New issue

Open dvyukov opened this issue on Oct 7, 2014 · 49 comments

dvyukov commented on Oct 7, 2014 Member

```
Here is design doc:
https://docs.google.com/document/d/1yIAYmbvL3JxOKOjuCyon7JhW4cSv1wy5hC0ApeGMV9s/pub

Here is the implementation:
https://golang.org/cl/12544043/

I've benchmarked the change of a real app where chans were abandoned in favor of other
sync primitives due to perf reasons:
https://groups.google.com/d/msg/golang-dev/0IElw_BbTrk/cGHMdNoHGQEJ
and the change provides 23% end-to-end speedup making chans significantly faster than
the other synchronization.
```

4

dvyukov added accepted release-none repo-main performance labels on Oct 7, 2014

rsc added this to the Unplanned milestone on Apr 9, 2015

Assignees

dvyukov

Labels

Performance

Projects

None yet

Milestone

Unplanned

Notifications

Subscribe

You're not receiving notifications from this thread.

@tyler_treat

defer

golang / go

Watch 2,261 | Unstar 29,165 | Fork 3,918

Code | Issues 2,818 | Pull requests 0 | Wiki | Insights

runtime: defer is slow #14939

New issue

Open minux opened this issue on Mar 23, 2016 · 20 comments

minux commented on Mar 23, 2016 Member

```
package p
import "testing"
//go:noinline
func defers() (r int) {
        defer func() {
                r = 42
        }()
        return 0
}
func BenchmarkDefer(b *testing.B) {
        for i := 0; i < b.N; i++ {
                defers()
        }
}
```

On my system, BenchmarkDefer uses 77.7ns/op. This issue arises from investigation of #9704: if I remove the "defer endcgo(mp)" and place the call at the end of the func cgocall, the benchmark in #9704 will improve from 144ns/op to 63.7ns/op. (Note: we can't

eliminate the defer in func cgocall though, as it will break defer/recover

Assignees

aclements

Labels

early-in-cycle

Performance

Projects

None yet

Milestone

Go1.10

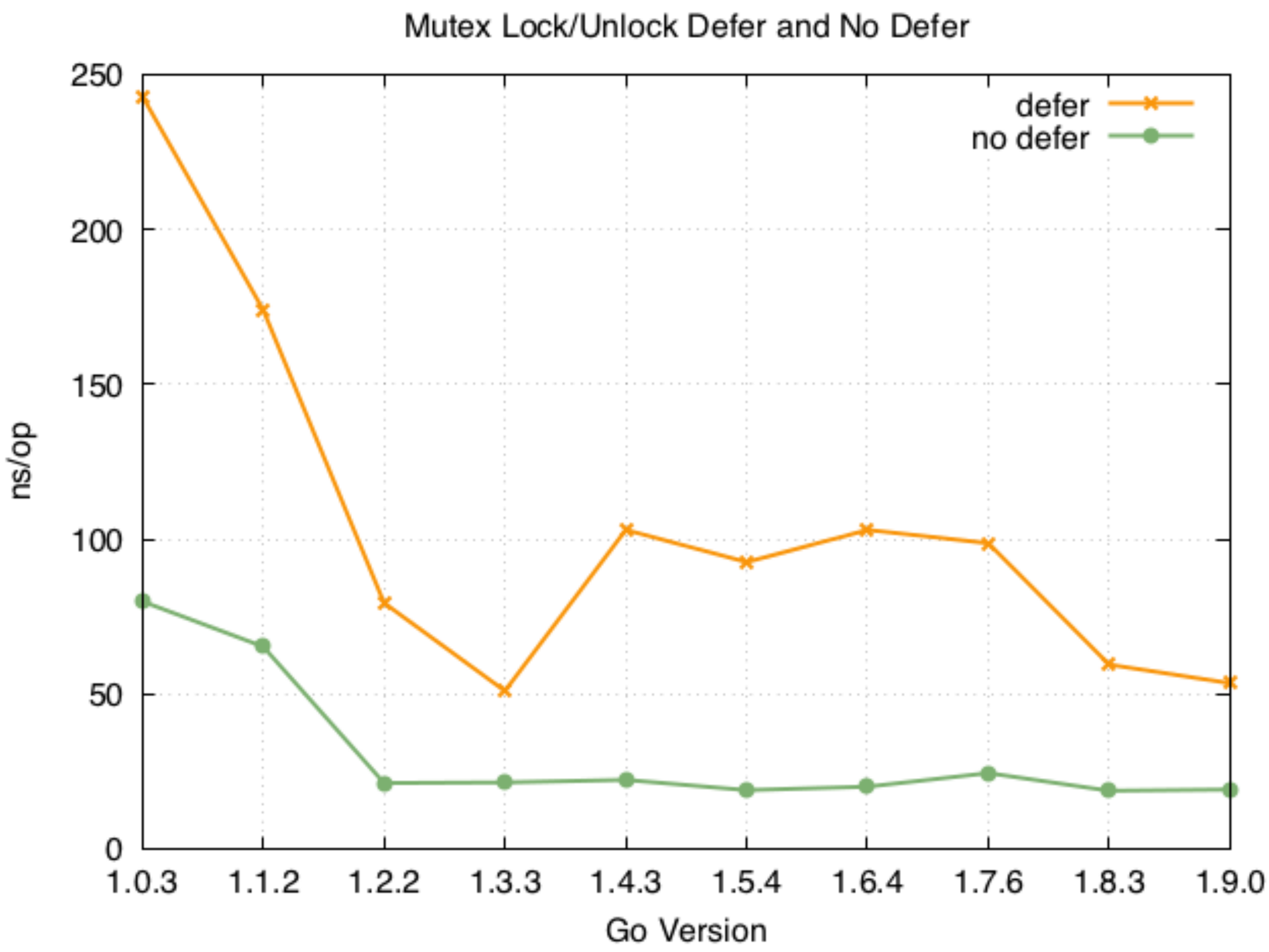
Notifications

Subscribe

You're not receiving notifications from this thread.

@tyler_treat

Is defer still slow?


Mutex Lock/Unlock Defer and No Defer
250
200
150
100
50
0
ns/op
defer
no defer
1.0.3
1.1.2
1.2.2
1.3.3
1.4.3
1.5.4
1.6.4
1.7.6
1.8.3
1.9.0
Go Version

The Secret Life of interface{}

```go
type Stringer interface {
        String() string
}
```

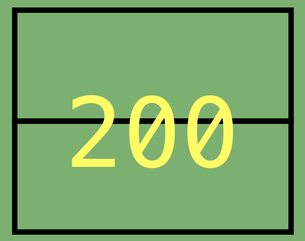
https://research.swtch.com/interfaces

```go
type Stringer interface {
        String() string
}

type Binary uint64
```

https://research.swtch.com/interfaces

```go
type Stringer interface {
        String() string
}

type Binary uint64
```

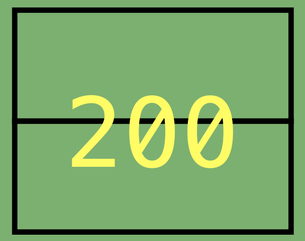
```go
b := Binary(200)
```



https://research.swtch.com/interfaces

```go
type Stringer interface {
        String() string
}

type Binary uint64

func (i Binary) String() string {
    return strconv.FormatUint(uint64(i), 2)
}
```

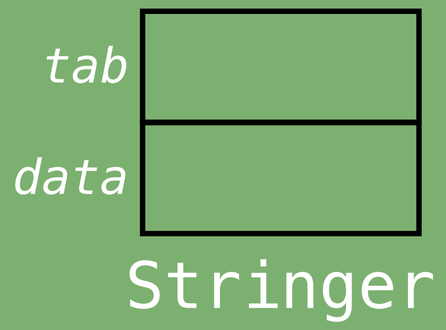
```go
b := Binary(200)
```



https://research.swtch.com/interfaces

```
type Stringer interface {
        String() string
}
```

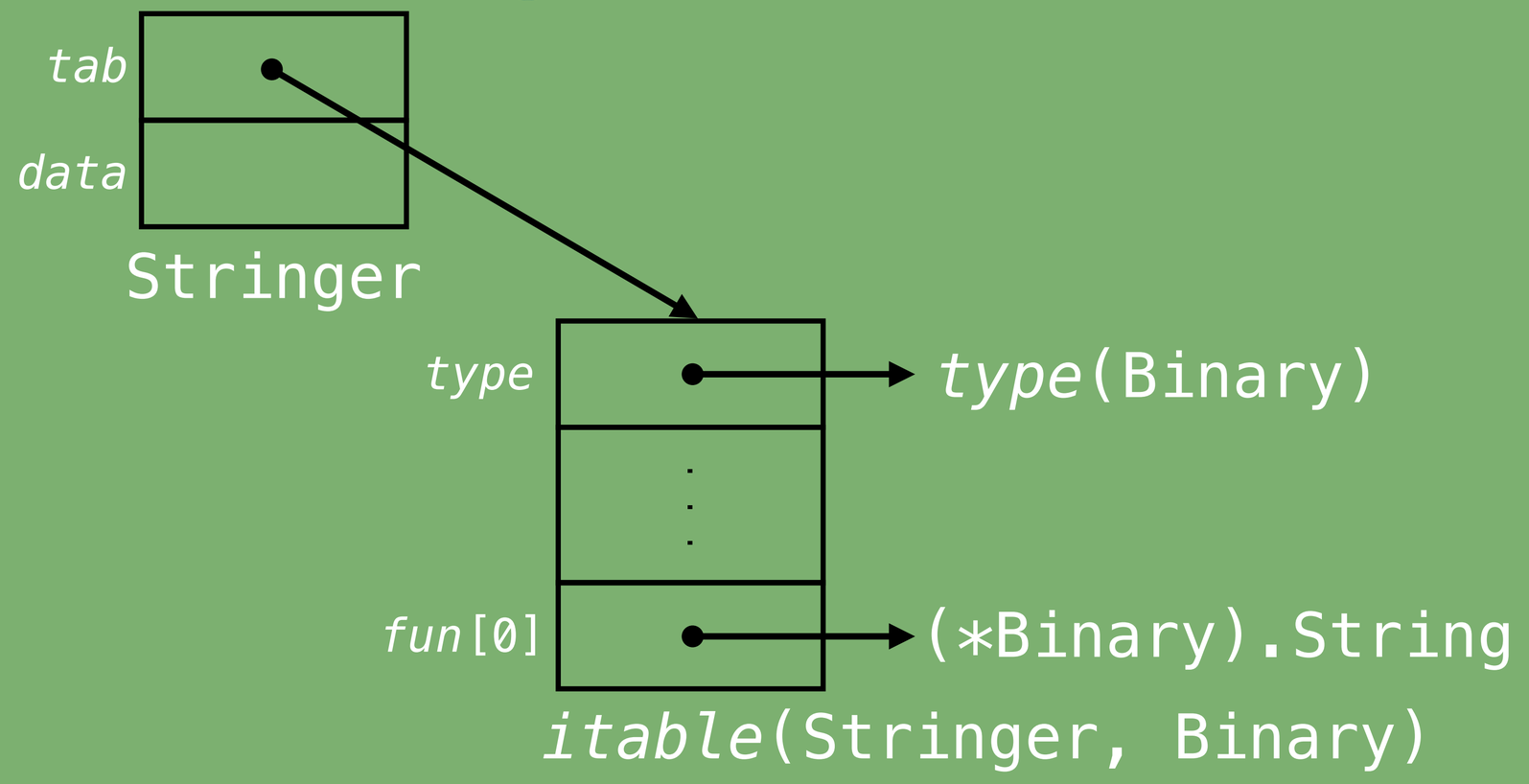
```
s := Stringer(b)
```



https://research.swtch.com/interfaces

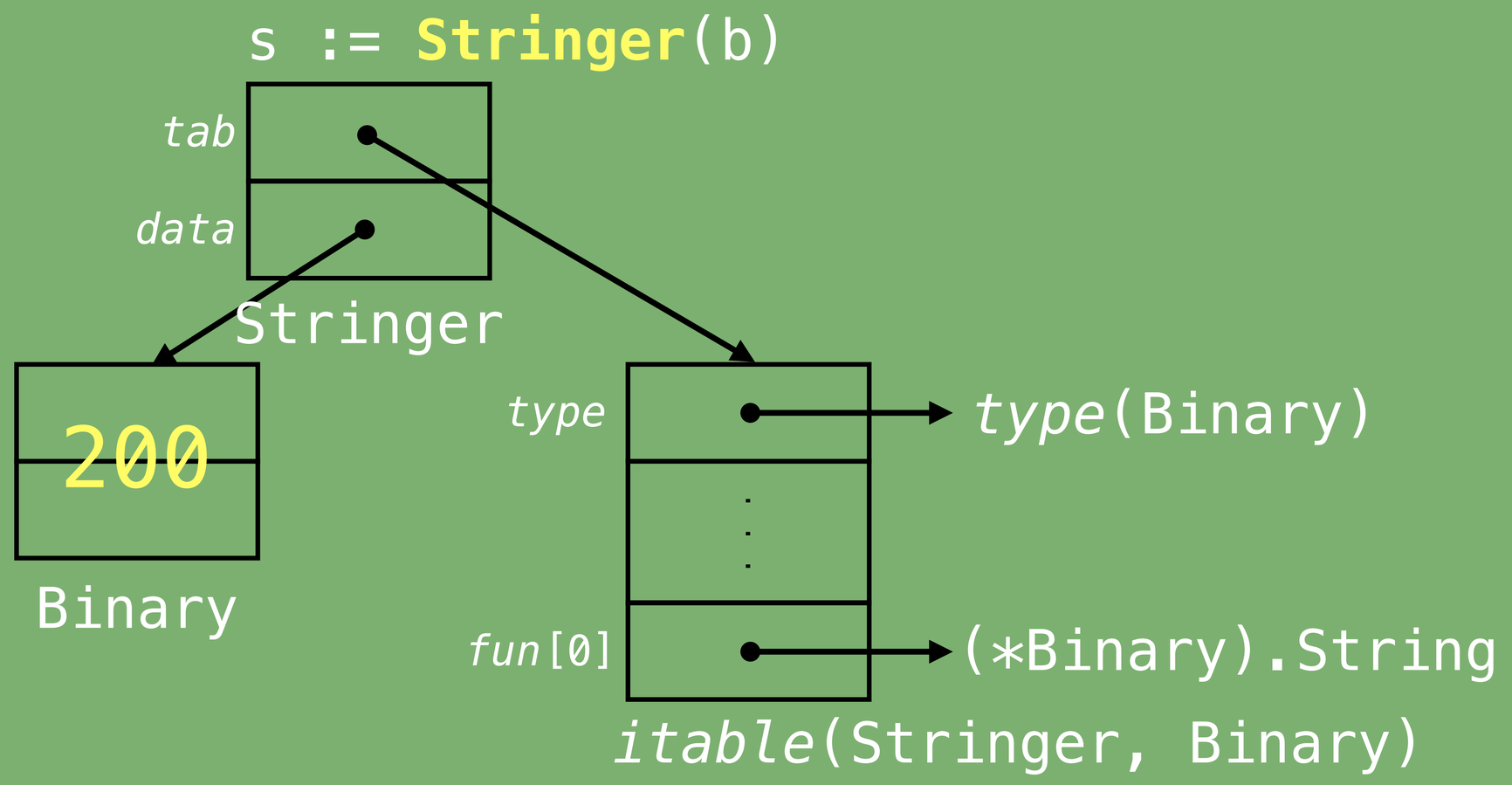
```
type Stringer interface {
        String() string
}
```

```
s := Stringer(b)
```



https://research.swtch.com/interfaces

```
type Stringer interface {
        String() string
}
```



https://research.swtch.com/interfaces

```
BenchmarkMethodCall-8          2000000000          0.29 ns/op
BenchmarkMethodCallIface-8     1000000000          2.30 ns/op
```

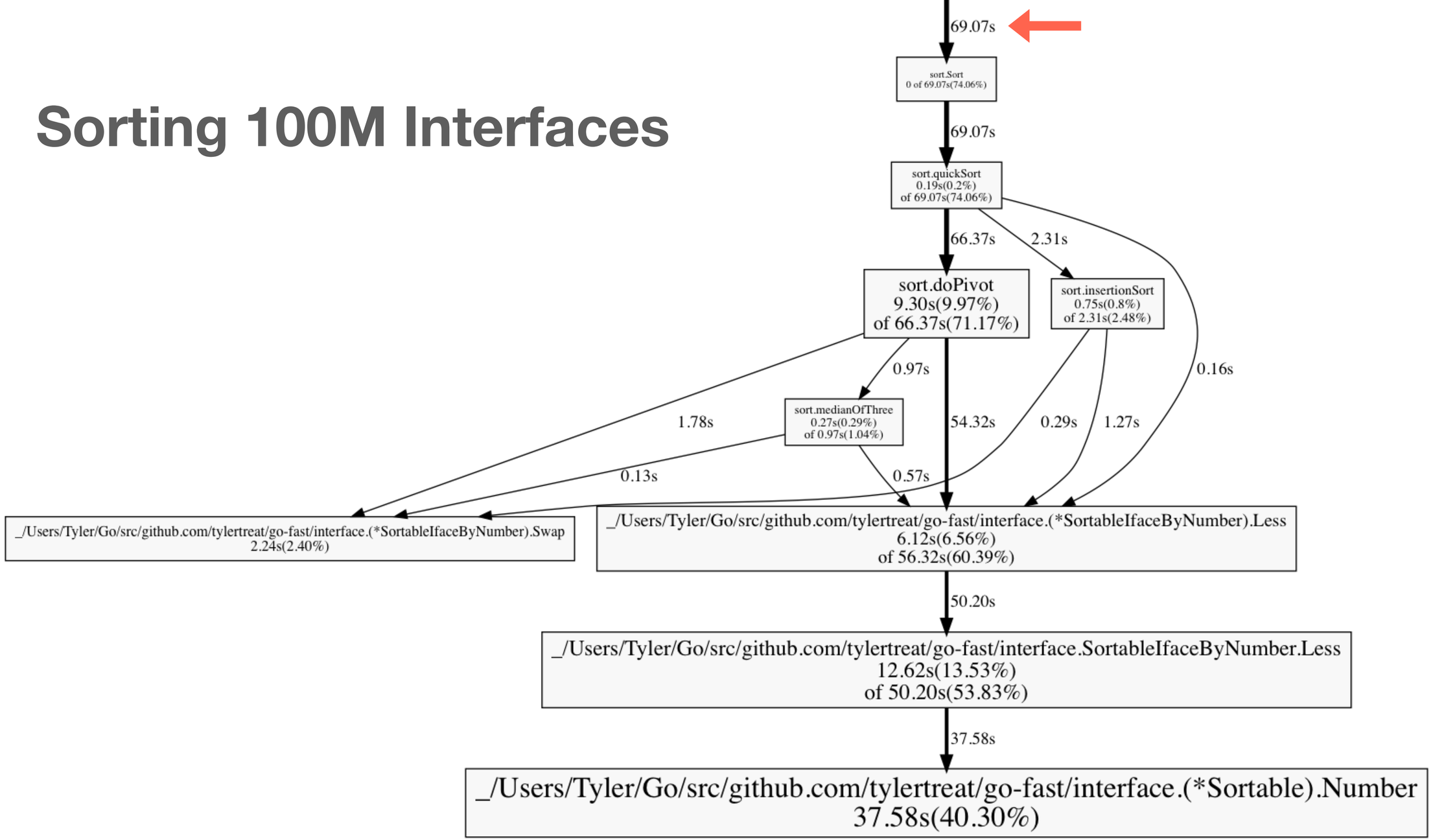
So what?

```go
62 type SortableIface interface {
63     Number() int
64 }
65
66 type Sortable struct {
67     number int
68 }
69
70 func (s Sortable) Number() int {
71     return s.number
72 }
73
74 type SortableIfaceByNumber []SortableIface
75
76 func (a SortableIfaceByNumber) Len() int           { return len(a) }
77 func (a SortableIfaceByNumber) Swap(i, j int)      { a[i], a[j] = a[j], a[i] }
78 func (a SortableIfaceByNumber) Less(i, j int) bool { return a[i].Number() < a[j].Number() }
79
80 type SortableByNumber []Sortable
81
82 func (a SortableByNumber) Len() int           { return len(a) }
83 func (a SortableByNumber) Swap(i, j int)      { a[i], a[j] = a[j], a[i] }
84 func (a SortableByNumber) Less(i, j int) bool { return a[i].Number() < a[j].Number() }
```

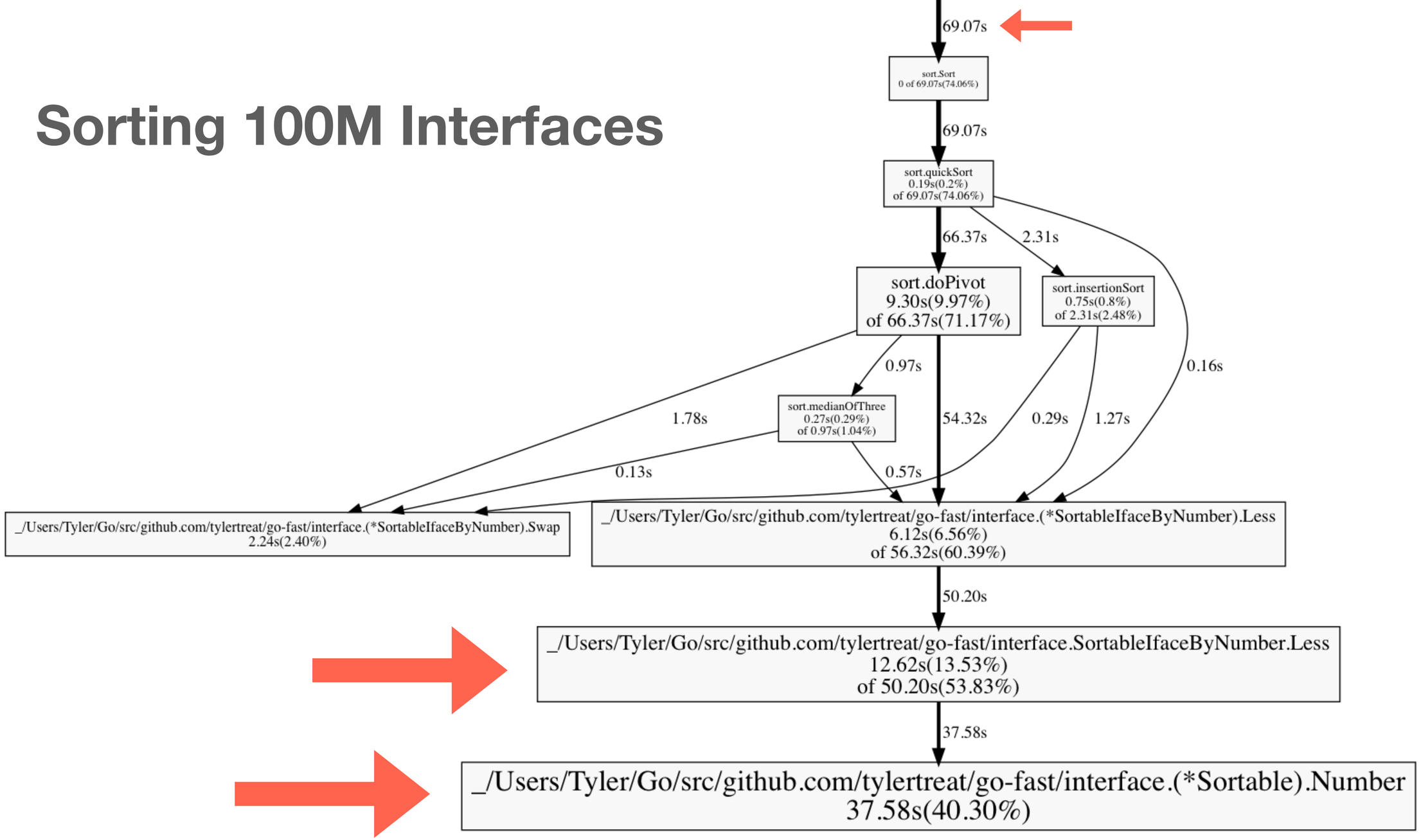
@tyler_treat

Sorting 100M Structs
300
250
200
150
100
50
0
seconds
Go 1.4.3
Go 1.8.3
struct
iface

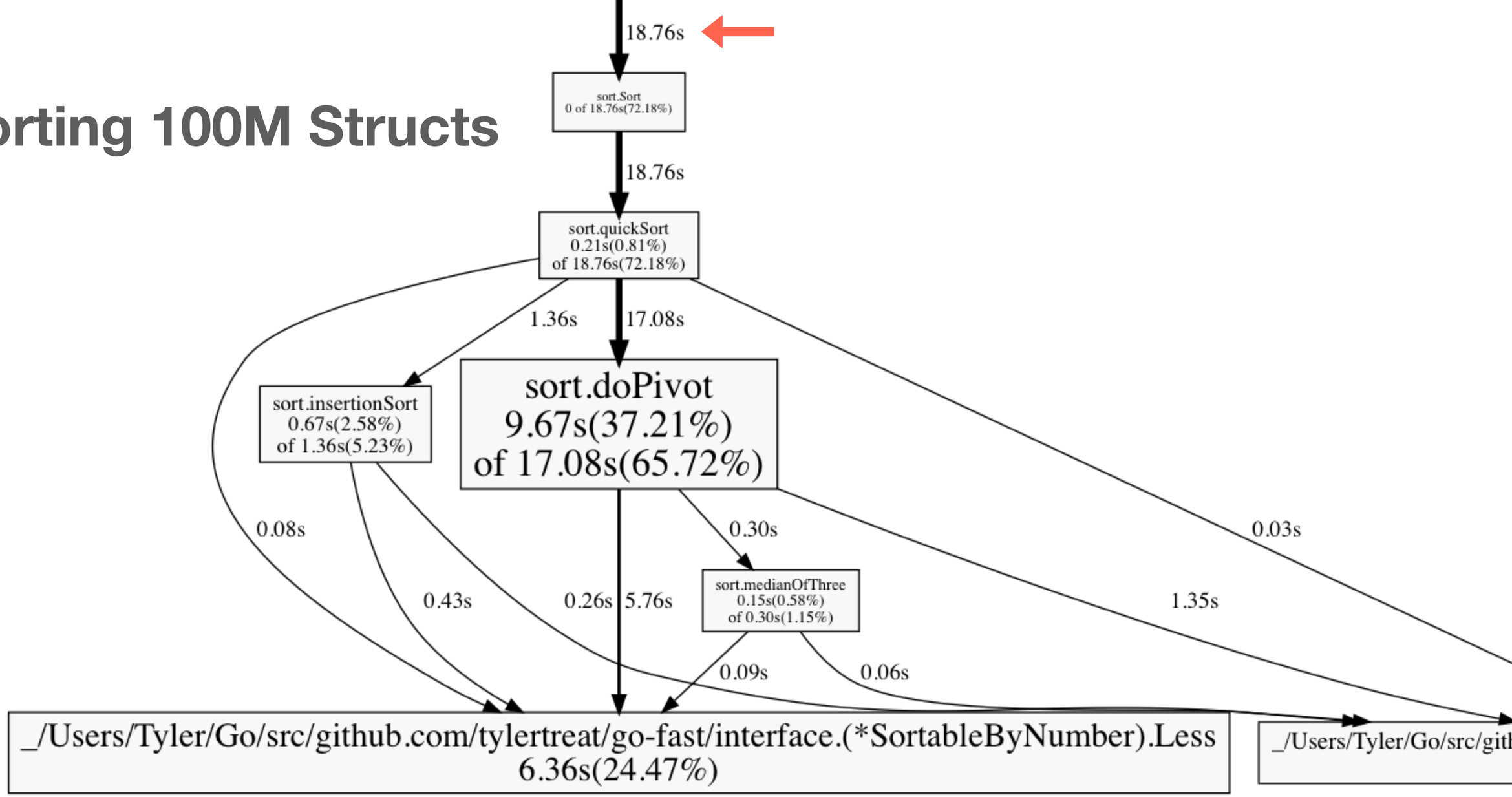
Sorting 100M Interfaces



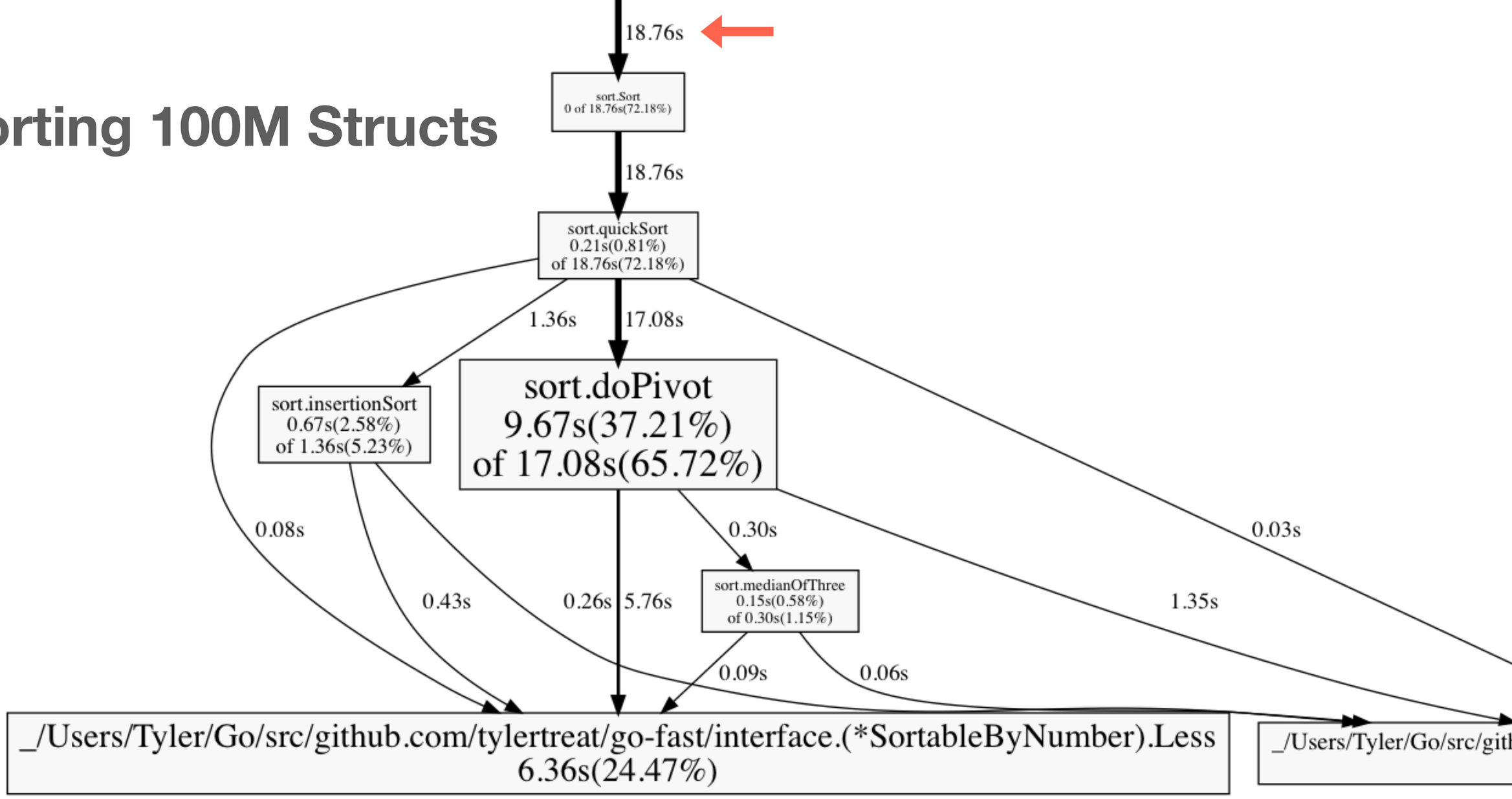
Sorting 100M Interfaces



Sorting 100M Structs



Sorting 100M Structs



$ go test -bench=. -gcflags="-m"

```
./iface_test.go:70: can inline Sortable.Number
./iface_test.go:76: can inline SortableIfaceByNumber.Len
./iface_test.go:77: can inline SortableIfaceByNumber.Swap
./iface_test.go:82: can inline SortableByNumber.Len
./iface_test.go:83: can inline SortableByNumber.Swap
./iface_test.go:84: can inline SortableByNumber.Less
./iface_test.go:84: inlining call to Sortable.Number
./iface_test.go:84: inlining call to Sortable.Number
```

```
$ go test -bench=. -gcflags="-m"
```

```
./iface_test.go:70: can inline Sortable.Number
./iface_test.go:76: can inline SortableIfaceByNumber.Len
./iface_test.go:77: can inline SortableIfaceByNumber.Swap
./iface_test.go:82: can inline SortableByNumber.Len
./iface_test.go:83: can inline SortableByNumber.Swap
./iface_test.go:84: can inline SortableByNumber.Less
./iface_test.go:84: inlining call to Sortable.Number
./iface_test.go:84: inlining call to Sortable.Number
```

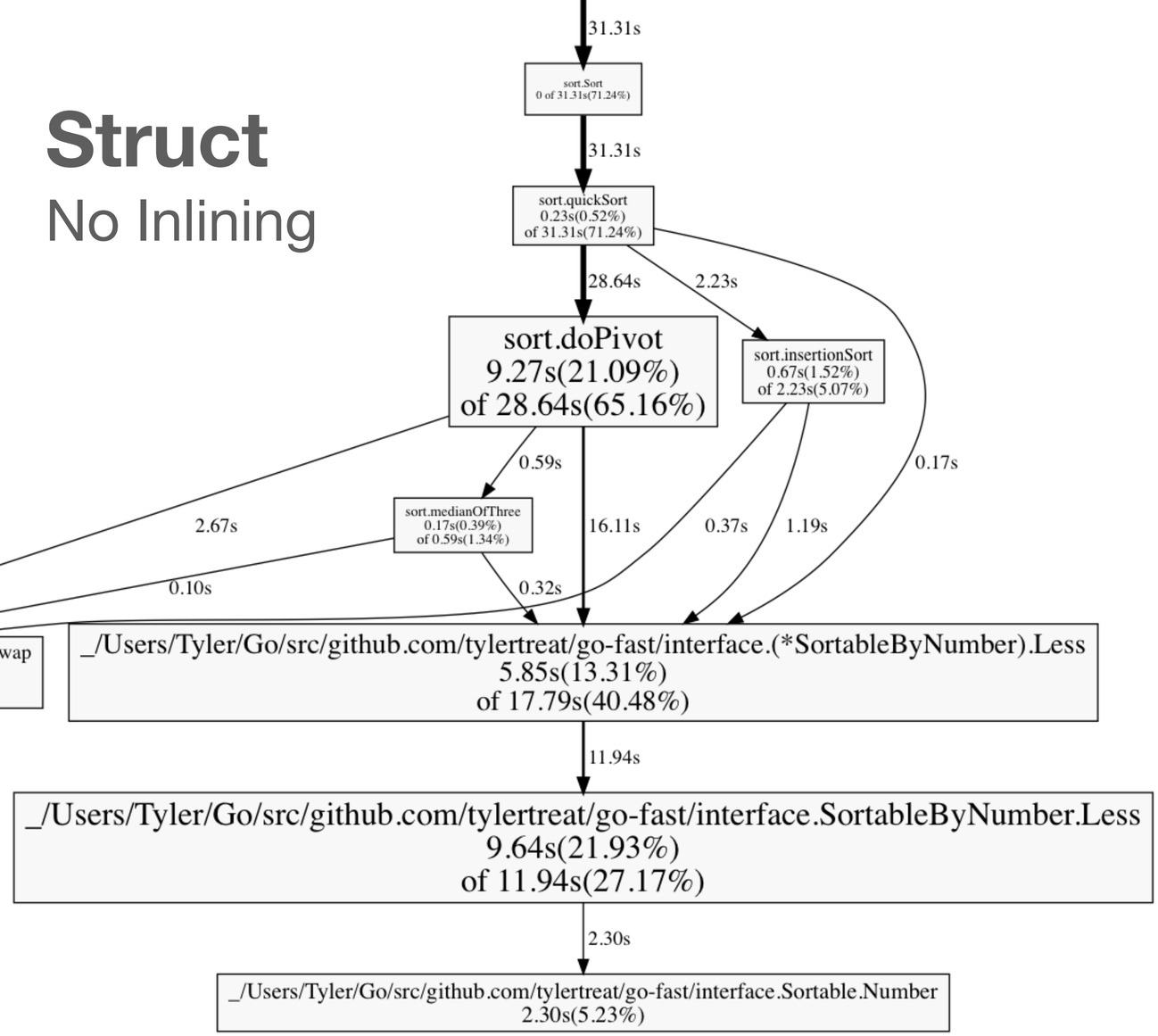
```go
62 type SortableIface interface {
63     Number() int
64 }
65
66 type Sortable struct {
67     number int
68 }
69
70 func (s Sortable) Number() int {
71     return s.number
72 }
73
74 type SortableIfaceByNumber []SortableIface
75
76 func (a SortableIfaceByNumber) Len() int           { return len(a) }
77 func (a SortableIfaceByNumber) Swap(i, j int)      { a[i], a[j] = a[j], a[i] }
78 func (a SortableIfaceByNumber) Less(i, j int) bool { return a[i].Number() < a[j].Number() }
79
80 type SortableByNumber []Sortable
81
82 func (a SortableByNumber) Len() int           { return len(a) }
83 func (a SortableByNumber) Swap(i, j int)      { a[i], a[j] = a[j], a[i] }
84 func (a SortableByNumber) Less(i, j int) bool { return a[i].Number() < a[j].Number() }
```

@tyler_treat

```
$ go test -bench=. -gcflags="-l"
```

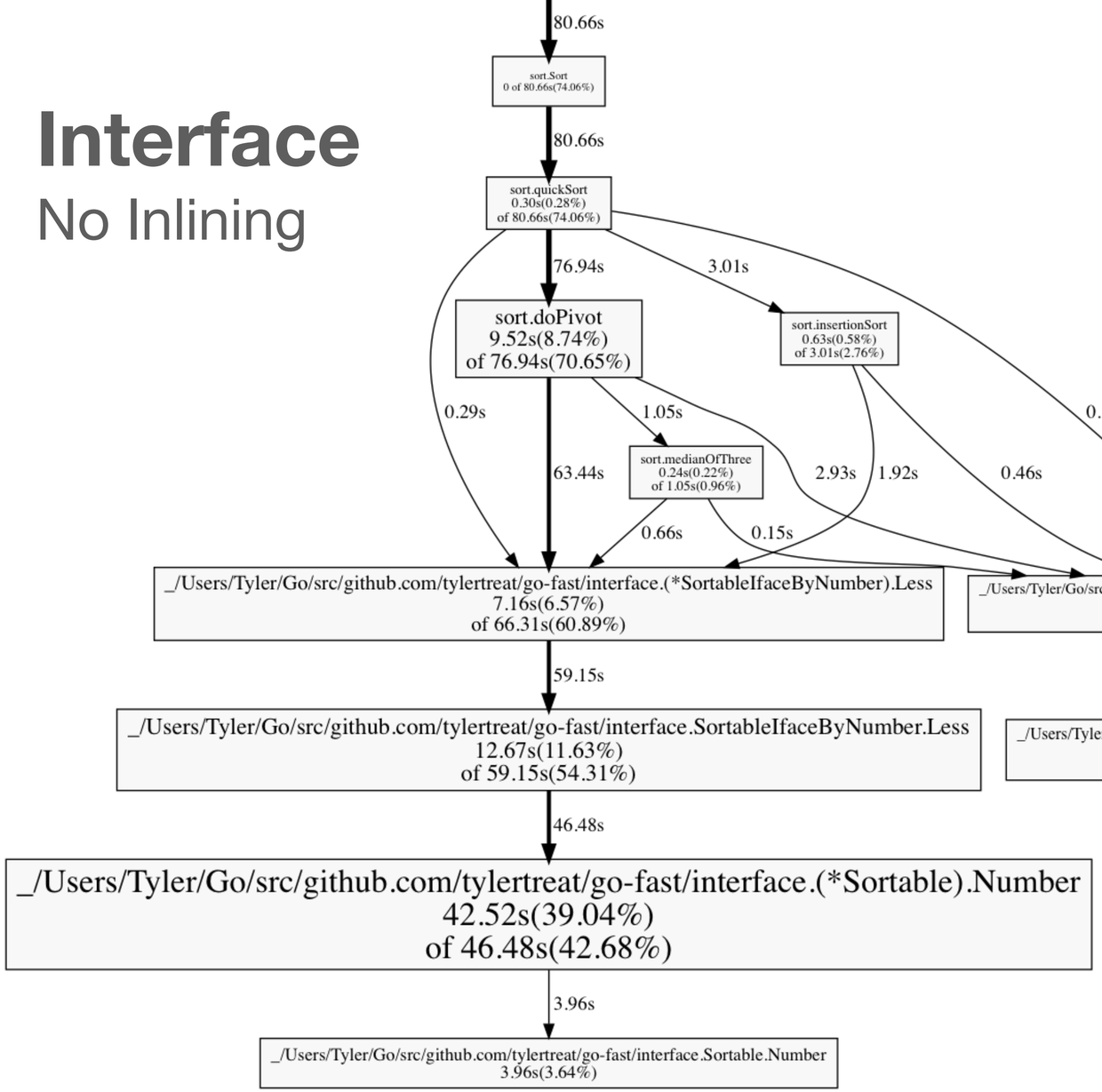
Struct

No Inlining



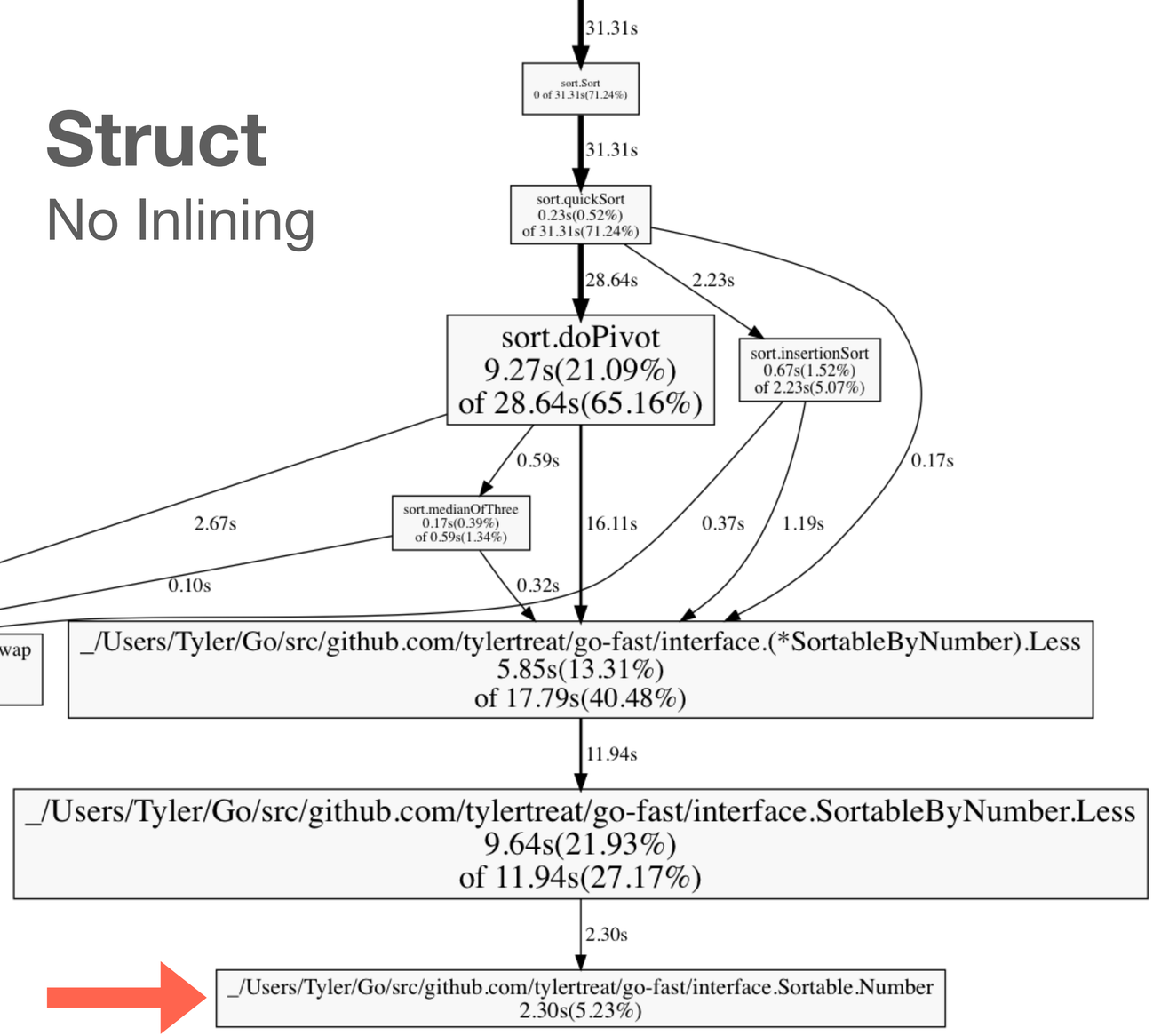
Interface

No Inlining



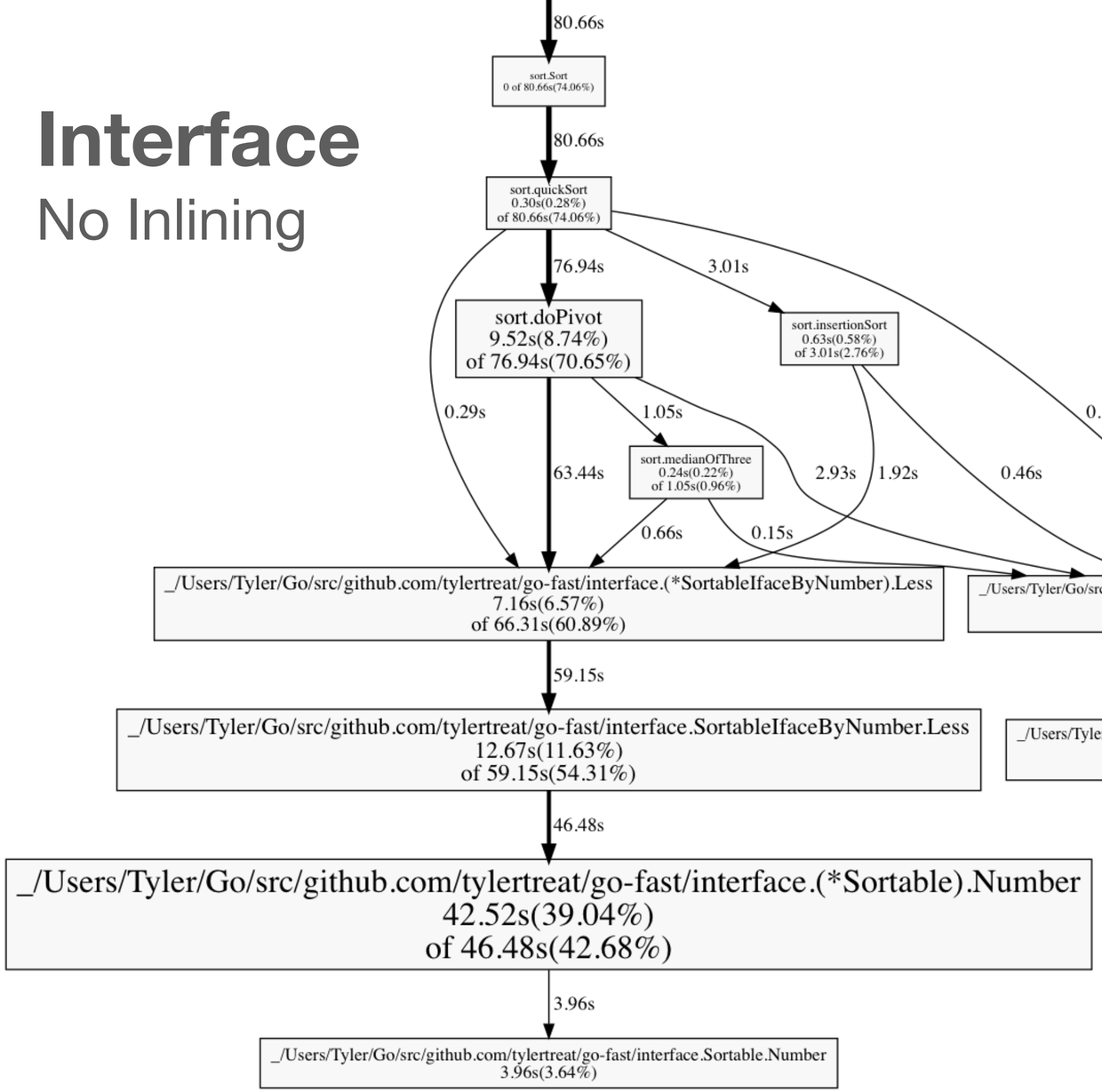
Struct

No Inlining



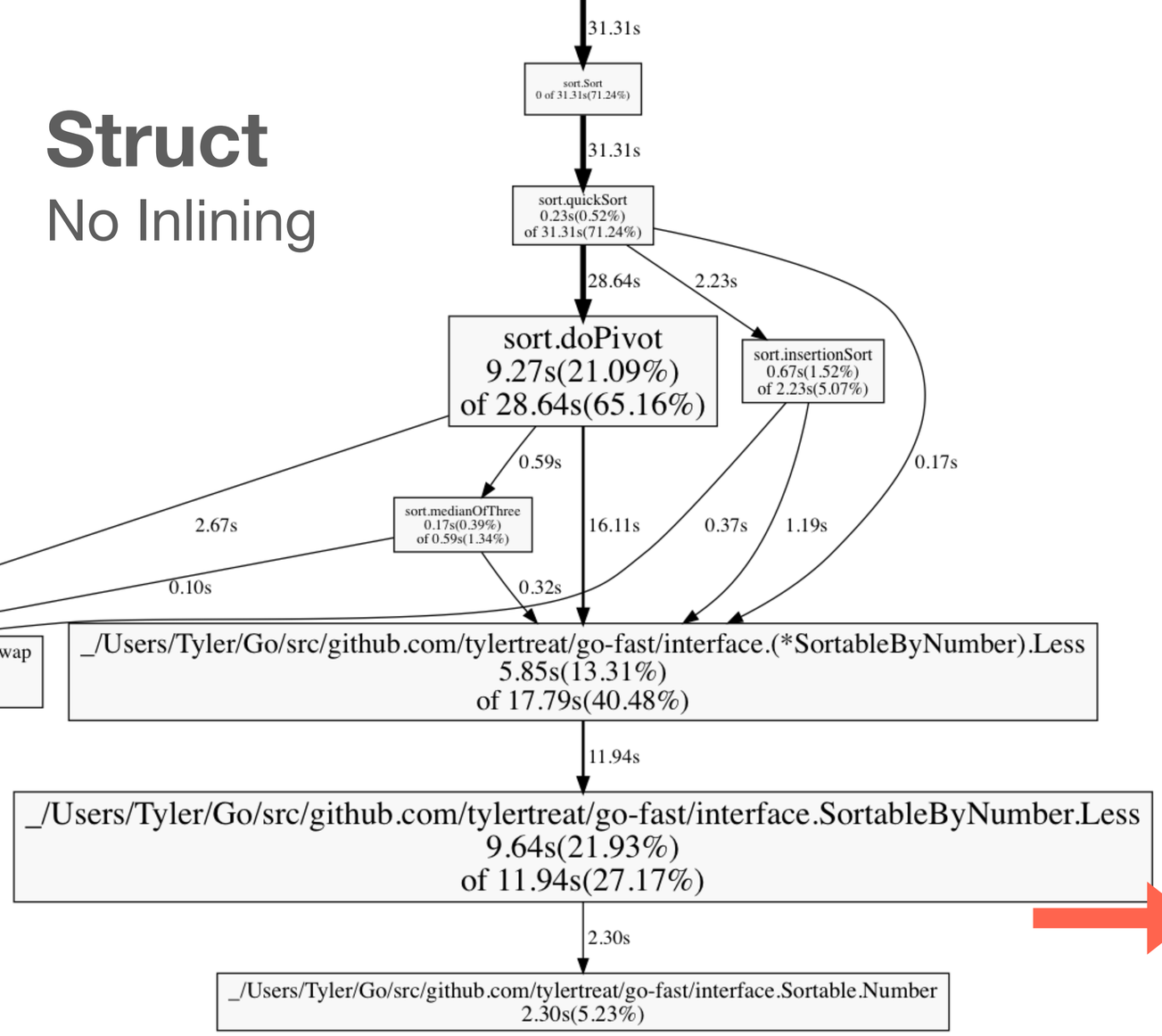
Interface

No Inlining



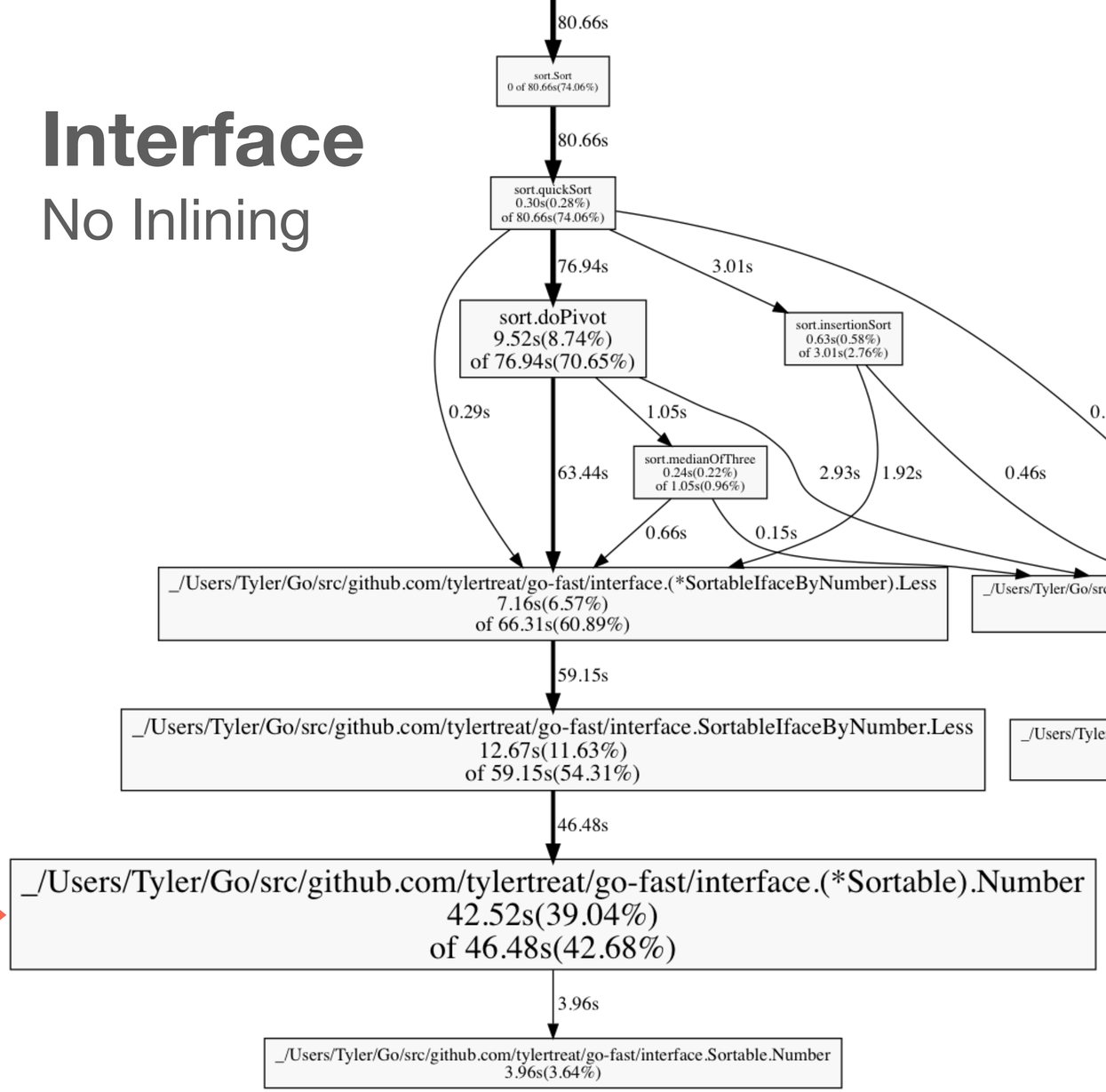
Struct

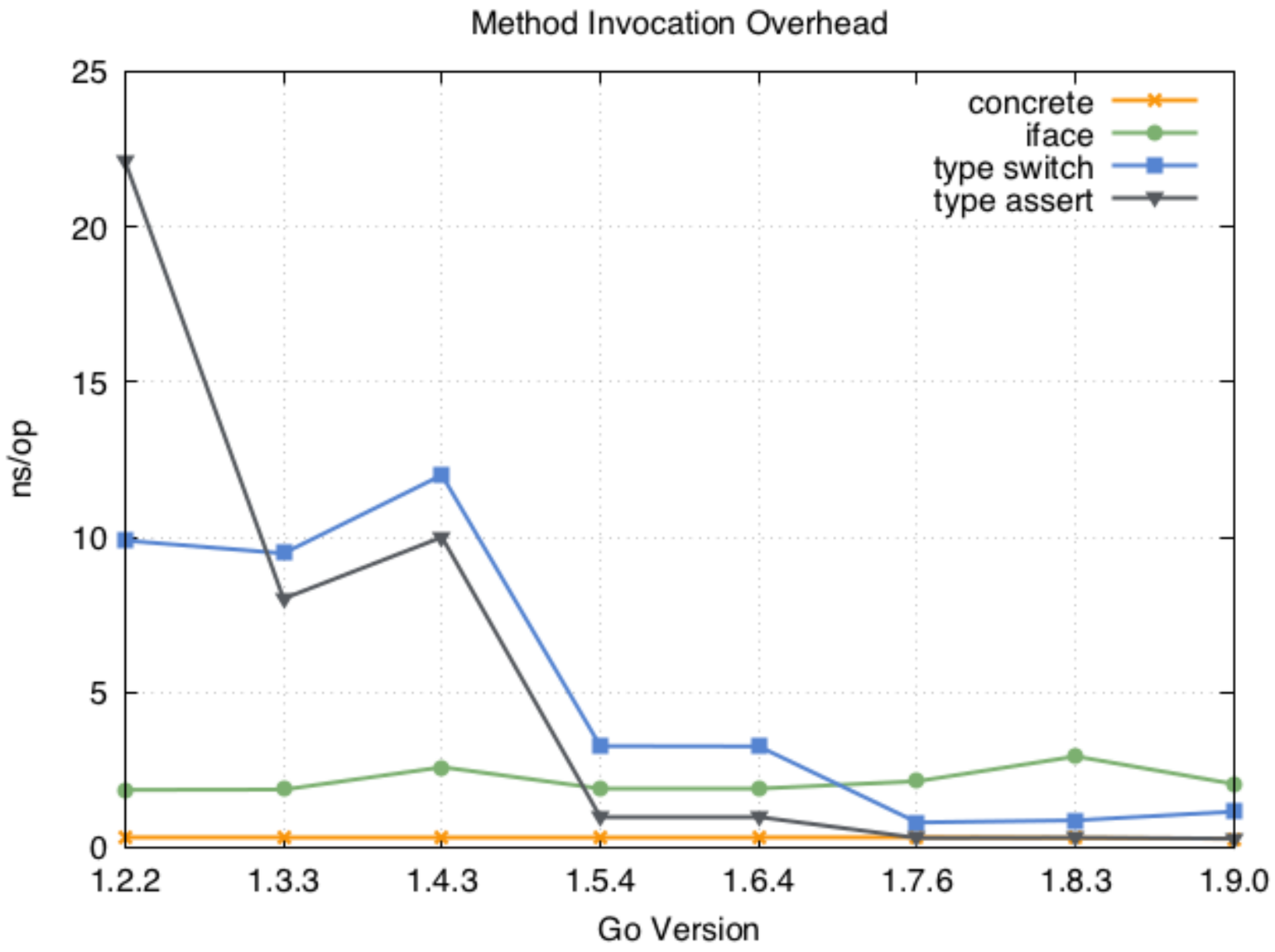
No Inlining

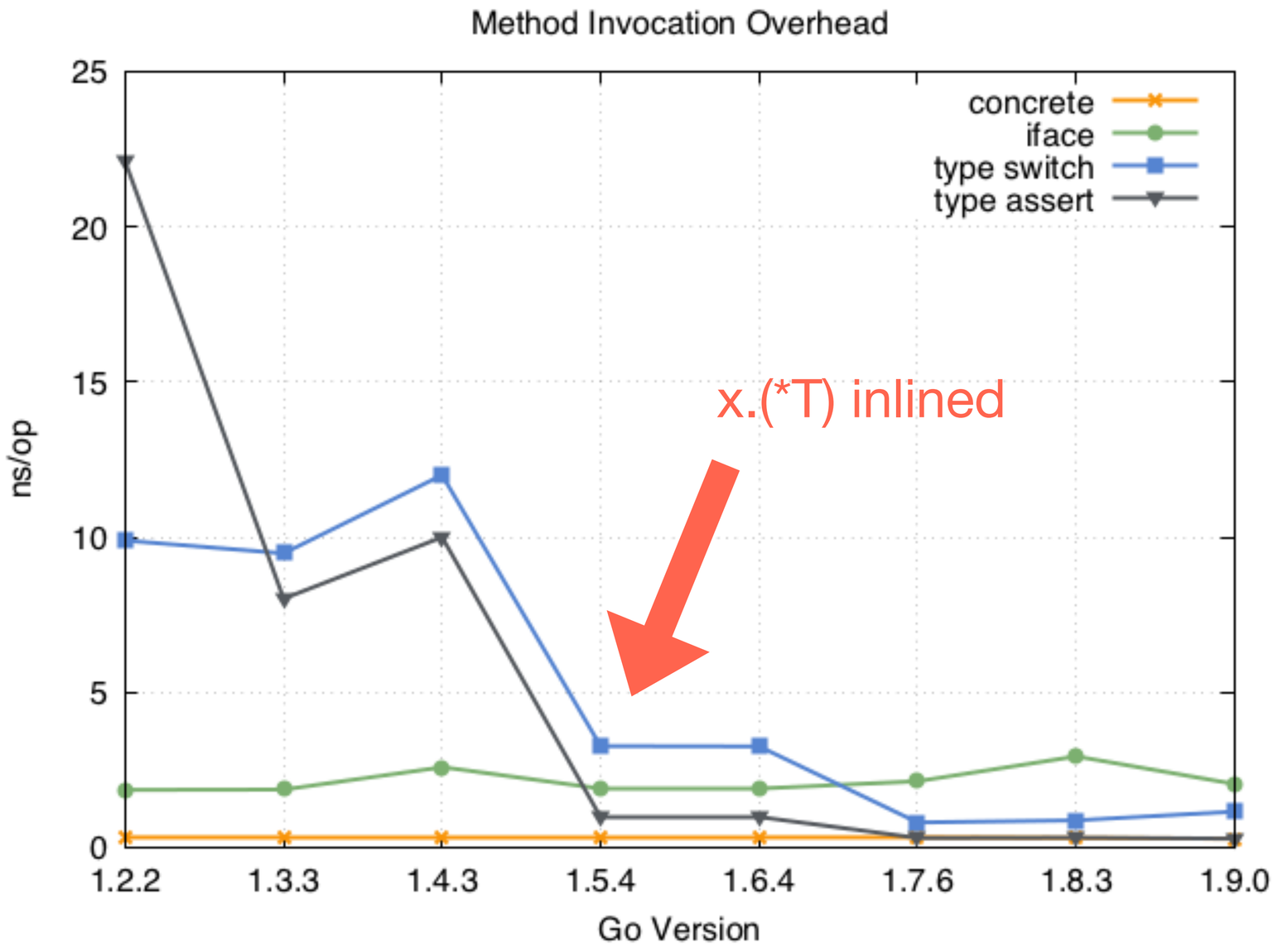


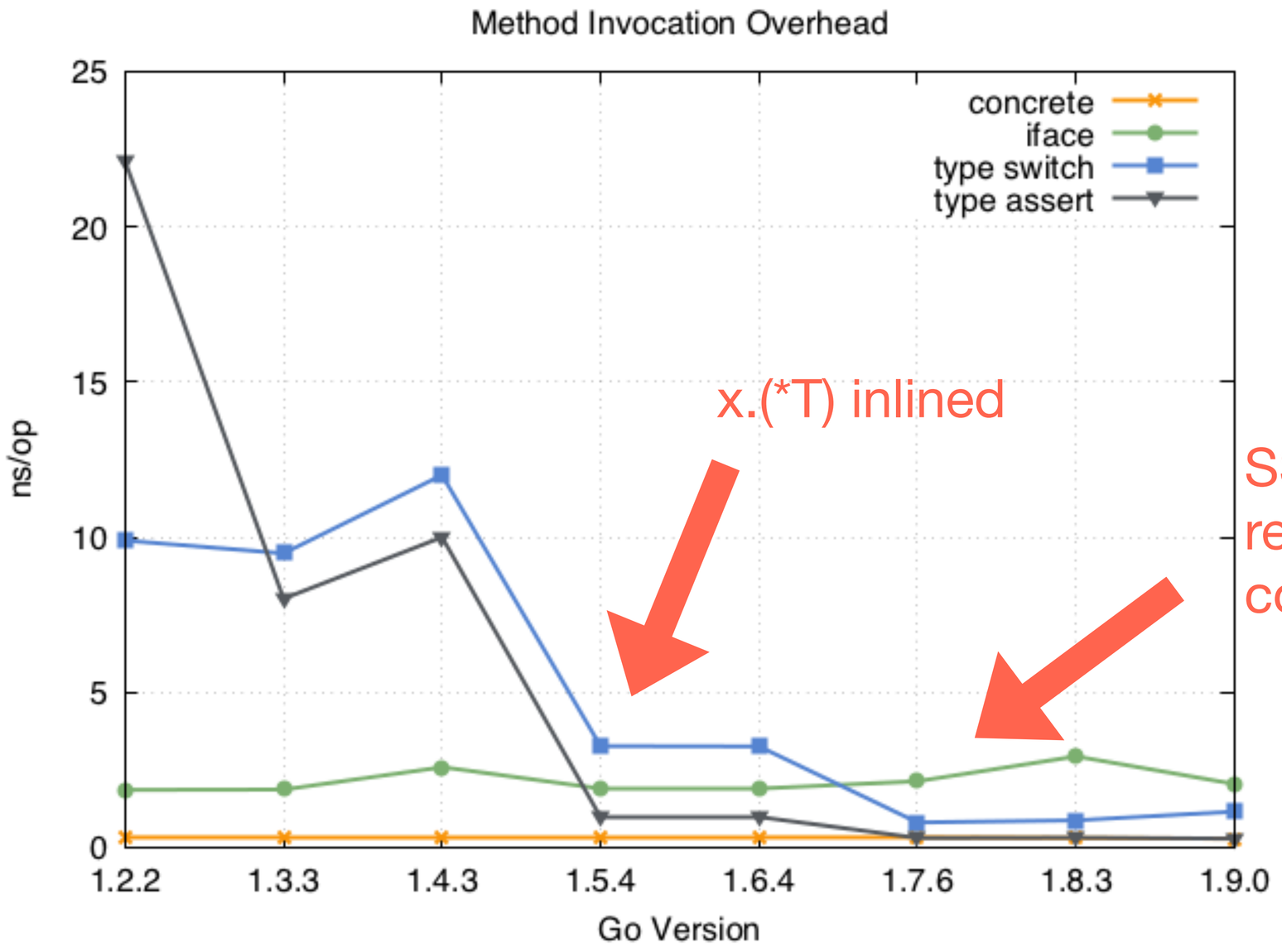
Interface

No Inlining




Method Invocation Overhead
ns/op
25
20
15
10
5
0
1.2.2
1.3.3
1.4.3
1.5.4
1.6.4
1.7.6
1.8.3
1.9.0
Go Version
concrete
iface
type switch
type assert


Method Invocation Overhead
ns/op
25
20
15
10
5
0
1.2.2
1.3.3
1.4.3
1.5.4
1.6.4
1.7.6
1.8.3
1.9.0
Go Version
concrete
iface
type switch
type assert
x.(*T) inlined



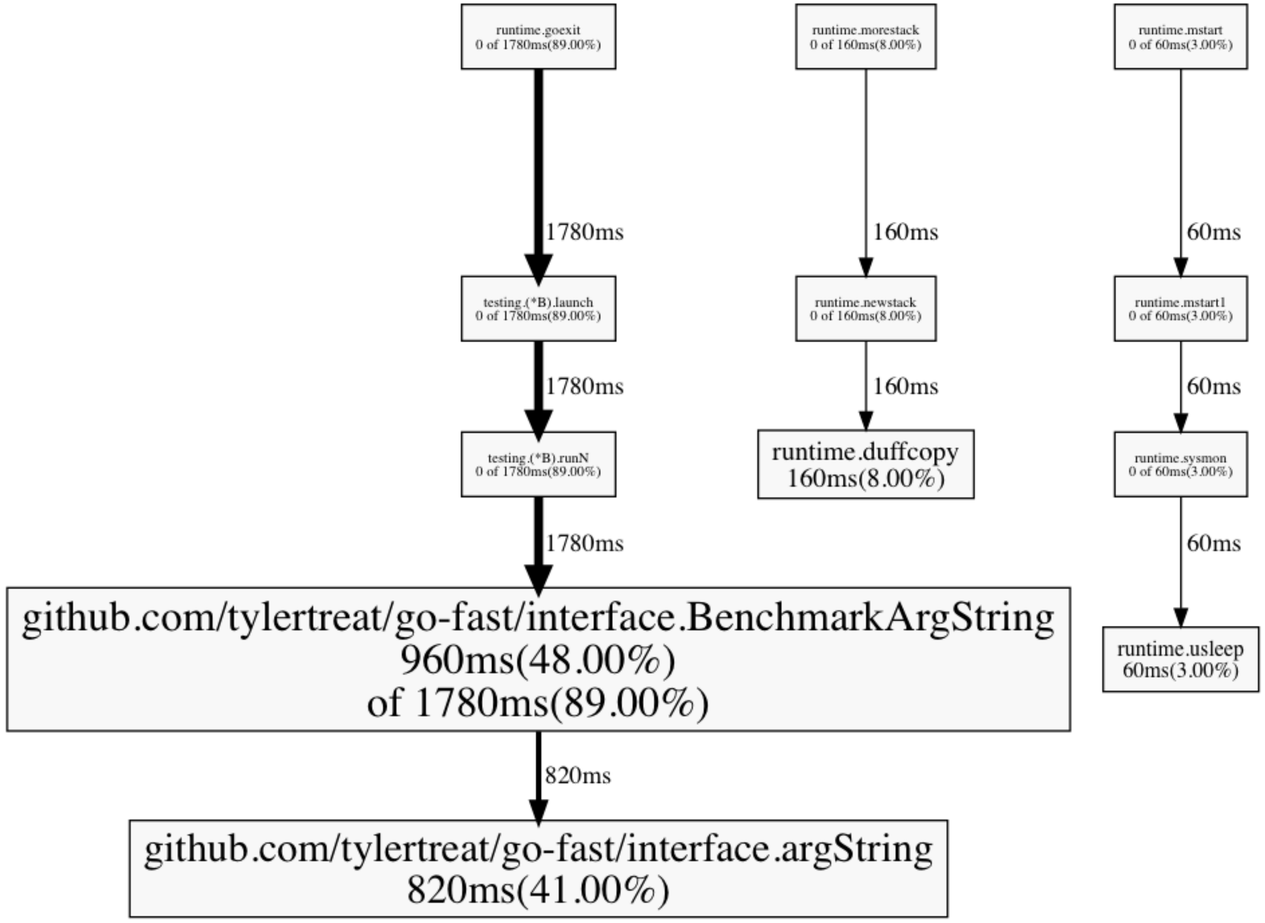
SSA backend & remaining type conversions inlined

```go
8  func BenchmarkArgIface(b *testing.B) {
9      print := false
10     for i := 0; i < b.N; i++ {
11         argIface("foo", print)
12     }
13 }
14
15 func BenchmarkArgString(b *testing.B) {
16     print := false
17     for i := 0; i < b.N; i++ {
18         argString("foo", print)
19     }
20 }
21
22 func argIface(arg interface{}, print bool) {
23     if print {
24         fmt.Println(arg)
25     }
26 }
27
28 func argString(arg string, print bool) {
29     if print {
30         fmt.Println(arg)
31     }
32 }
```

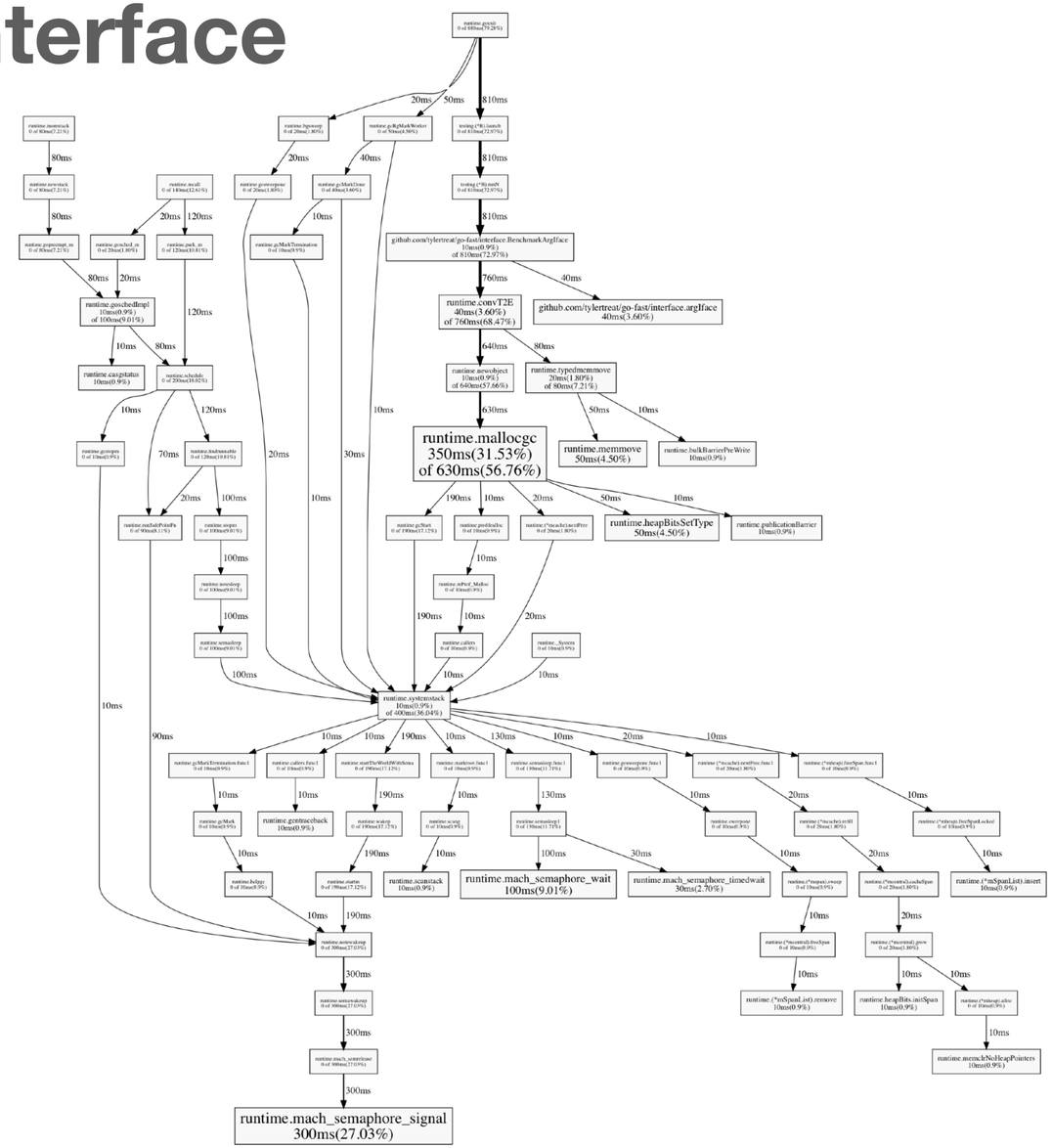
@tyler_treat

```
BenchmarkArgIface-4      50000000          40.4 ns/op
BenchmarkArgString-4     1000000000        2.25 ns/op
```

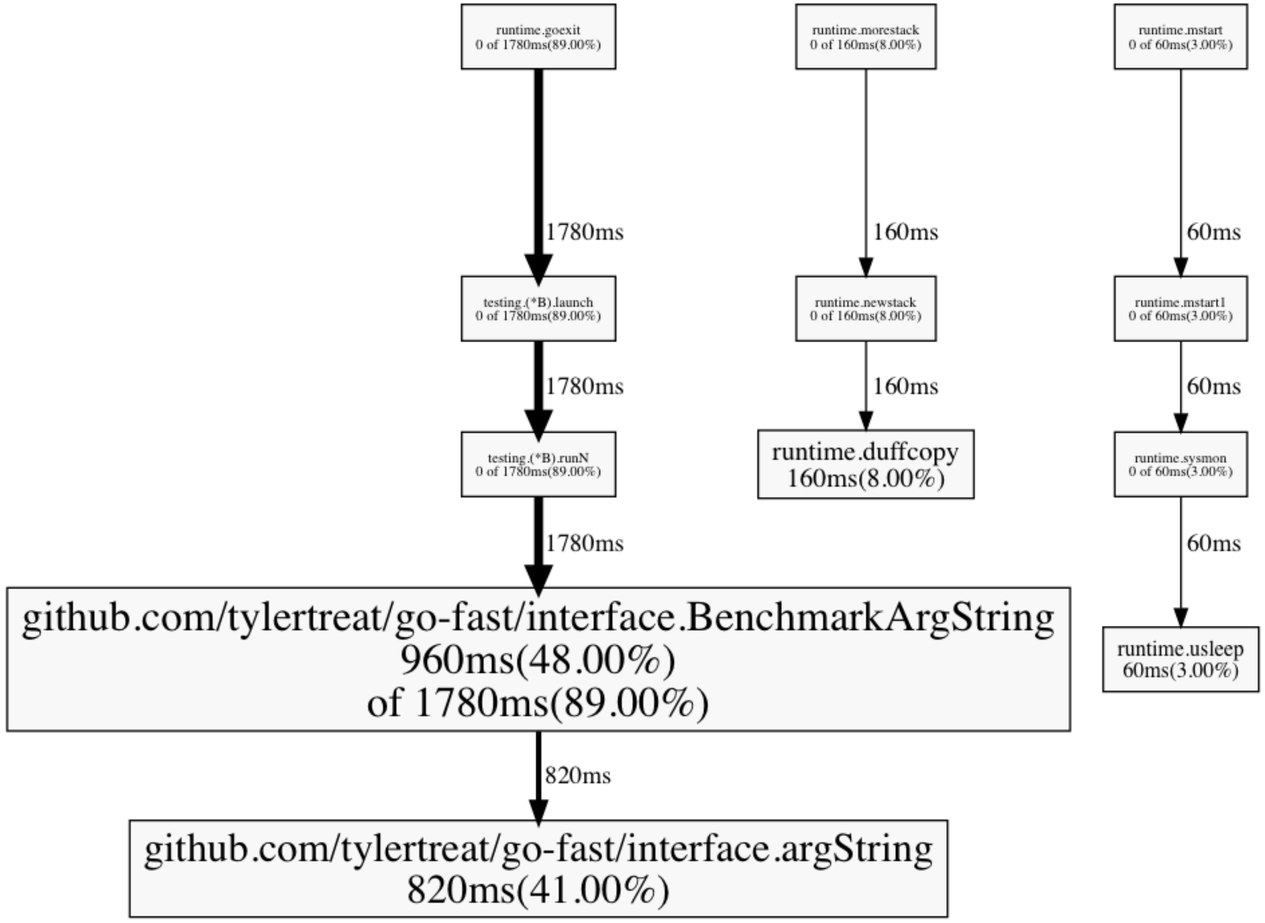
Struct



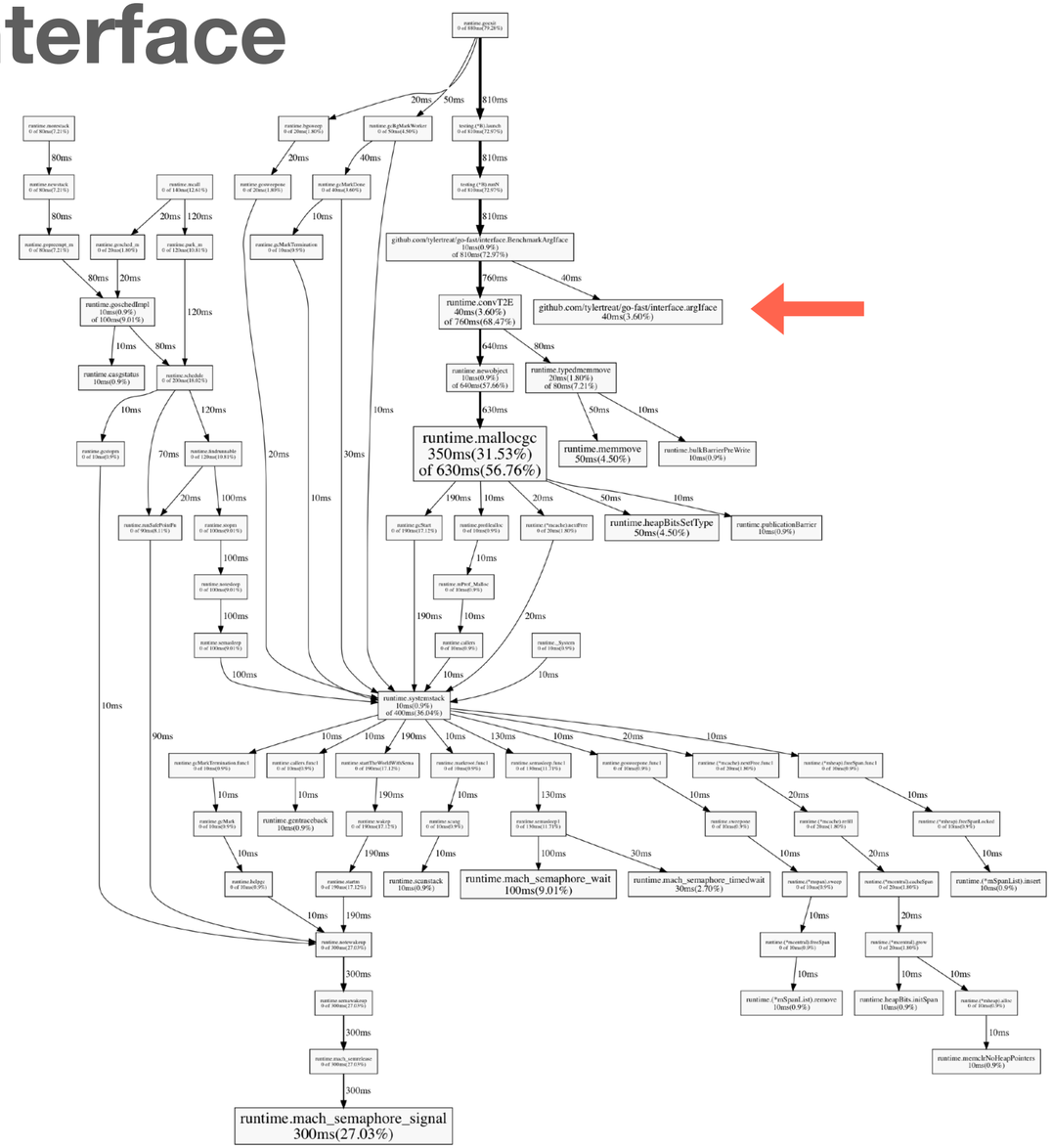
Interface

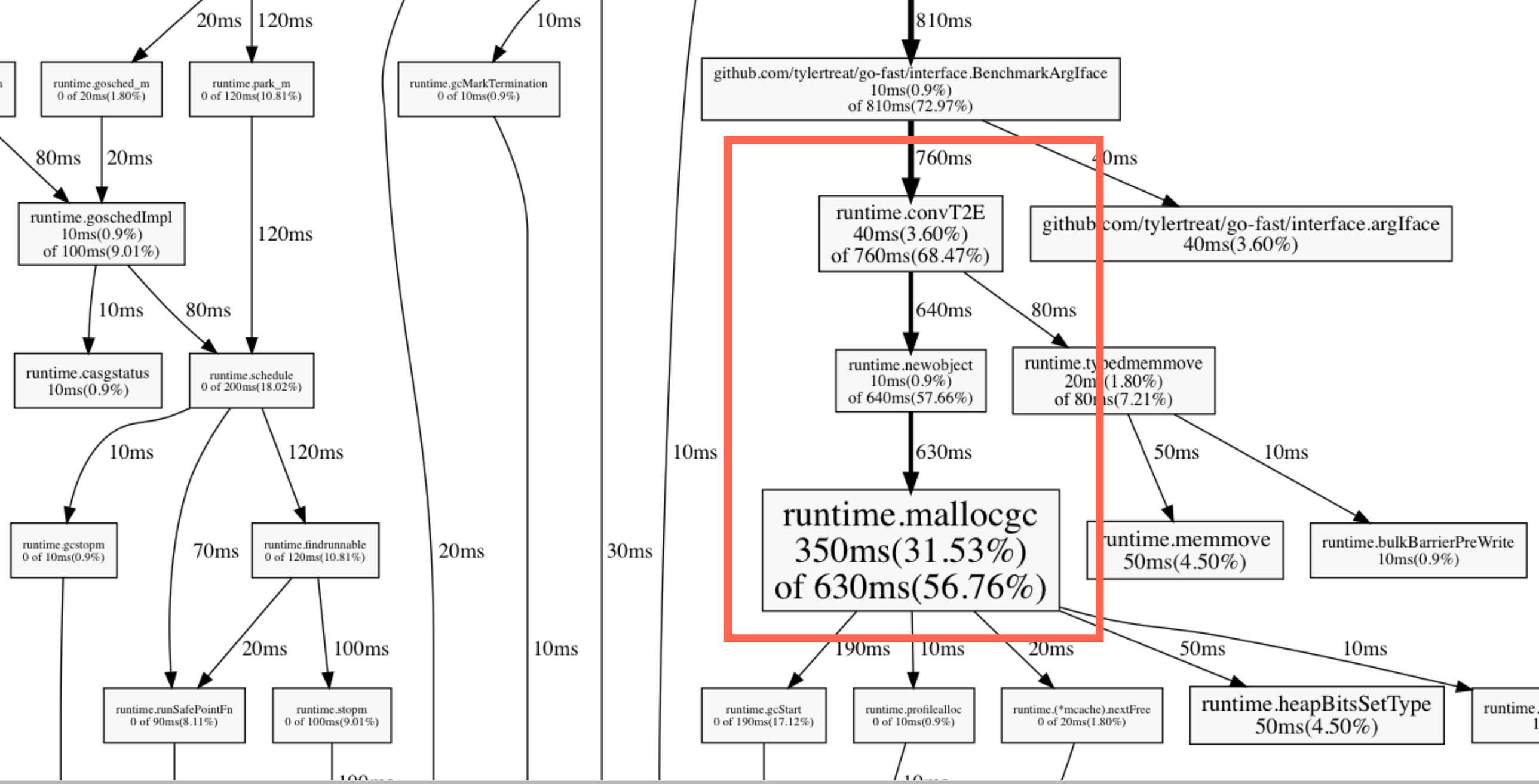


Struct



Interface




20ms
120ms
10ms
runtime.gosched_m
0 of 20ms(1.80%)
runtime.park_m
0 of 120ms(10.81%)
runtime.gcMarkTermination
0 of 10ms(0.9%)
80ms
20ms
runtime.goschedImpl
10ms(0.9%)
of 100ms(9.01%)
120ms
10ms
80ms
runtime.casgstatus
10ms(0.9%)
runtime.schedule
0 of 200ms(18.02%)
10ms
120ms
runtime.gcstopm
0 of 10ms(0.9%)
70ms
runtime.findrunnable
0 of 120ms(10.81%)
20ms
20ms
100ms
10ms
runtime.runSafePointFn
0 of 90ms(8.11%)
runtime.stopm
0 of 100ms(9.01%)
810ms
github.com/tylertreat/go-fast/interface.BenchmarkArgIface
10ms(0.9%)
of 810ms(72.97%)
760ms
runtime.convT2E
40ms(3.60%)
of 760ms(68.47%)
40ms
640ms
80ms
runtime.newobject
10ms(0.9%)
of 640ms(57.66%)
20ms(1.80%)
of 80ms(7.21%)
10ms
630ms
50ms
10ms
runtime.mallocgc
350ms(31.53%)
of 630ms(56.76%)
runtime.memmove
50ms(4.50%)
runtime.bulkBarrierPreWrite
10ms(0.9%)
30ms
190ms
10ms
20ms
50ms
10ms
runtime.gcStart
0 of 190ms(17.12%)
runtime.profilealloc
0 of 10ms(0.9%)
runtime.(*mcache).nextFree
0 of 20ms(1.80%)
runtime.heapBitsSetType
50ms(4.50%)

@tyler_treat

$ go test -bench=. -gcflags="-S"

```
LEAQ    go.string."foo"(SB), DX
MOVQ    DX, ""..autotmp_0+40(SP)
MOVQ    $3, ""..autotmp_0+48(SP)
LEAQ    type.string(SB), BX
MOVQ    BX, (SP)
LEAQ    ""..autotmp_0+40(SP), SI
MOVQ    SI, 8(SP)
PCDATA  $0, $1
CALL    runtime.convT2E(SB)
MOVQ    24(SP), AX
MOVQ    16(SP), CX
MOVQ    CX, (SP)
MOVQ    AX, 8(SP)
MOVB    $0, 16(SP)
PCDATA  $0, $1
CALL    "".argIface(SB)
```

$ go test -bench=. -gcflags="-S"

```
LEAQ    go.string."foo"(SB), DX
MOVQ    DX, ""..autotmp_0+40(SP)
MOVQ    $3, ""..autotmp_0+48(SP)
LEAQ    type.string(SB), BX
MOVQ    BX, (SP)
LEAQ    ""..autotmp_0+40(SP), SI
MOVQ    SI, 8(SP)
PCDATA  $0, $1
CALL    runtime.convT2E(SB)
MOVQ    24(SP), AX
MOVQ    16(SP), CX
MOVQ    CX, (SP)
MOVQ    AX, 8(SP)
MOVB    $0, 16(SP)
PCDATA  $0, $1
CALL    "".argIface(SB)
```

$ go test -bench=. -gcflags="-S"

```
LEAQ    go.string."foo"(SB), DX
MOVQ    DX, ""..autotmp_0+40(SP)
MOVQ    $3, ""..autotmp_0+48(SP)
LEAQ    type.string(SB), BX
MOVQ    BX, (SP)
LEAQ    ""..autotmp_0+40(SP), SI
MOVQ    SI, 8(SP)
PCDATA  $0, $1
CALL    runtime.convT2E(SB)
MOVQ    24(SP), AX
MOVQ    16(SP), CX
MOVQ    CX, (SP)
MOVQ    AX, 8(SP)
MOVB    $0, 16(SP)
PCDATA  $0, $1
CALL    "".argIface(SB)
```

```
LEAQ    go.string."foo"(SB), DX
MOVQ    DX, (SP)
MOVQ    $3, 8(SP)
MOVB    $0, 16(SP)
PCDATA  $0, $0
CALL    "".argString(SB)
```

Key Insight:

If performance matters, write type-specific code.

Overview

- Measuring performance
- Language features
- Memory management
- Concurrency and multi-core

byte to string conversions

Derek Collison
@derekcollison

Following

A @nats_io community contribution from @nussjustin that results in ~2M msgs/sec bump. Now at > 11M msgs/sec!

```
Benchmark____PubNo_Payload-8    20000000                88.6 ns/op
Benchmark____Pub8b_Payload-8    20000000                89.8 ns/op
Benchmark___Pub32b_Payload-8    20000000                97.3 ns/op
Benchmark__Pub256B_Payload-8    10000000               129 ns/op
Benchmark____Pub1K_Payload-8     5000000               216 ns/op
Benchmark____Pub4K_Payload-8     1000000              1123 ns/op
Benchmark____Pub8K_Payload-8      500000              2309 ns/op
Benchmark___________PubSub-8    10000000               210 ns/op
Benchmark___PubSubTwoConns-8    10000000               205 ns/op
Benchmark___PubTwoQueueSub-8    10000000               231 ns/op
Benchmark__PubFourQueueSub-8    10000000               233 ns/op
Benchmark_PubEightQueueSub-8     5000000               231 ns/op
```

11:40 AM - 5 Apr 2016 from San Francisco, CA

11 Retweets **24** Likes

3 11 24

@tyler_treat

nats-io / gnatsd

Unwatch 239 | Unstar 3,145 | Fork 348

Code | Issues 46 | Pull requests 8 | Projects 0 | Settings | Insights

Remove an unnecessary string allocation #237

Edit

Merged **derekcollison** merged 1 commit into `nats-io:master` from `unknown repository` on Apr 5, 2016

Conversation 2 | Commits 1 | Files changed 1 | +2 −3

nussjustin commented on Apr 5, 2016 Contributor

This removes an potentially unneeded allocation when processing messages. Previous to this change, the processMsg function always converted the subject from a []byte to a string, thus allocating, and used it to lookup or set a cache entry.

By moving the conversion for the cache lookup into the map access, we can use an optimization introduced in Go 1.3 that avoids allocating the string for the map lookup.

In case where we insert into the cache we still need to allocate, because we need a string for the Match function and also because setting the map key still needs to allocate, if the key isn't already in the map (if I remember correctly).

These are the results from the publish benchmarks from github.com/nats-io/nats/test:

```
benchmark                          old ns/op     new ns/op     delta
BenchmarkPublishSpeed-4            394           339           -13.96%
```

Reviewers

No reviews—request one

Assignees

No one—assign yourself

Labels

None yet

Projects

None yet

Milestone

No milestone

@tyler_treat

5 server/client.go

View

```
@@ -787,20 +787,19 @@ func (c *client) processMsg(msg []byte) {
787 787         var r *SublistResult
788 788         var ok bool
789 789
790     -       subject := string(c.pa.subject)
791     -
792 790         if srv != nil {
793 791                 genid = atomic.LoadUint64(&srv.sl.genid)
794 792         }
795 793
796 794         if genid == c.cache.genid && c.cache.results != nil {
797     -               r, ok = c.cache.results[subject]
    795 +               r, ok = c.cache.results[string(c.pa.subject)]
798 796         } else {
799 797                 // reset
800 798                 c.cache.results = make(map[string]*SublistResult)
801 799                 c.cache.genid = genid
802 800         }
803 801         if !ok {
    802 +               subject := string(c.pa.subject)
804 803                 r = srv.sl.Match(subject)
805 804                 c.cache.results[subject] = r
806 805         }
```

What’s going on here?

golang / go

Watch 2,294 | Unstar 29,646 | Fork 3,989

Code | Issues 2,858 | Pull requests 0 | Wiki | Insights

cmd/gc: optimized map[string] lookup from []byte key #3512

Closed **bradfitz** opened this issue on Apr 11, 2012 · 22 comments

bradfitz commented on Apr 11, 2012 — Owner

```
key := []byte("some key")  // from network, etc

var m map[string]T
....
v, ok := m[string(key)]  // It would be nice if this were optimized away and didn't
actually make a copy & garbage
....
```

rsc commented on Apr 11, 2012 — Contributor

Comment 1:

```
In general this might introduce races.  If you pass in a []byte to the key lookup,
there is no guarantee that something is not editing the bytes during the lookup.
But it could probably be done so that it would just fail the lookup instead of
```

Assignees

No one assigned

Labels

FrozenDueToAge

GarbageCollector

Performance

Projects

None yet

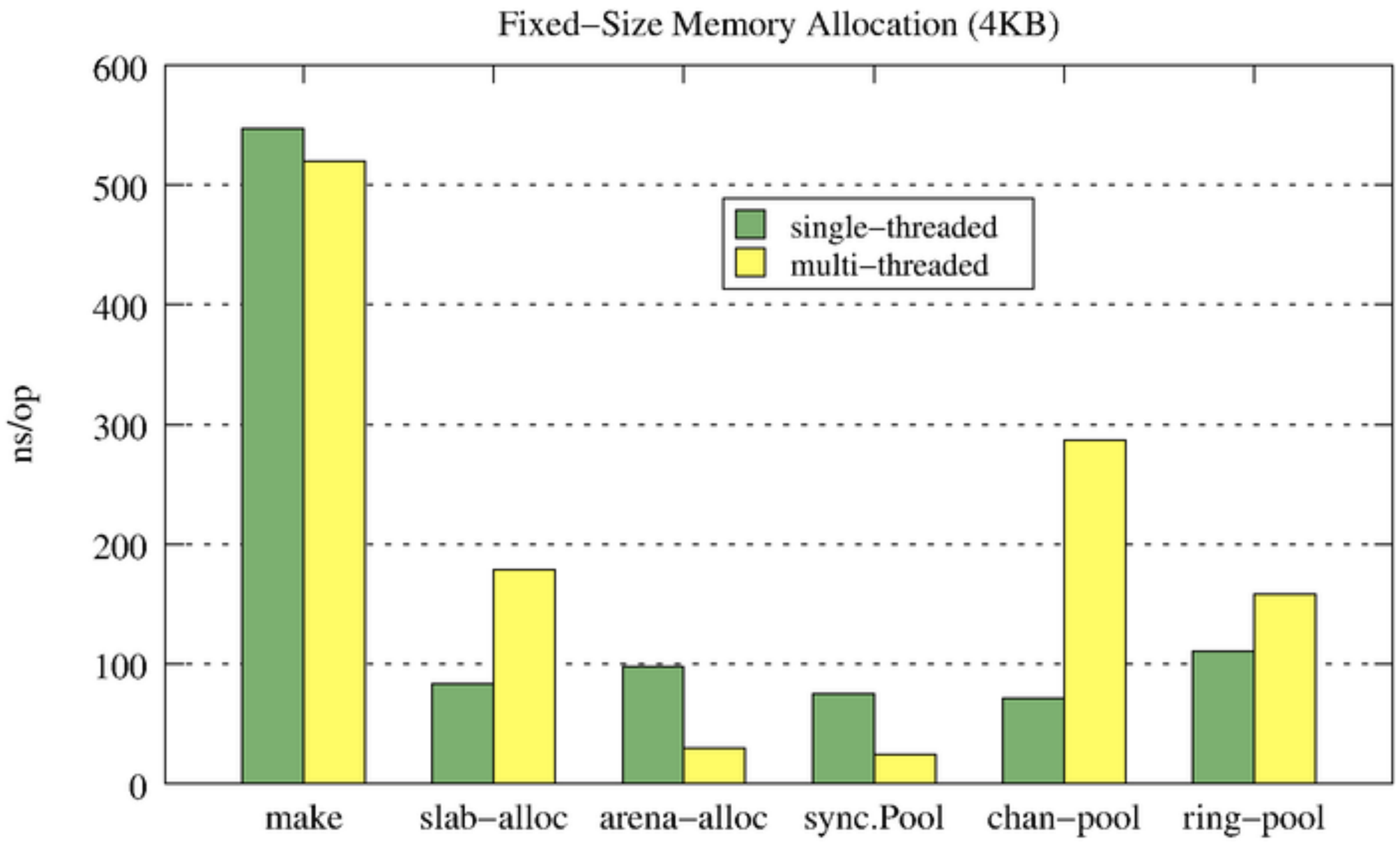
Milestone

No milestone

Notifications

@tyler_treat

memory allocation


Fixed–Size Memory Allocation (4KB)
ns/op
600
500
400
300
200
100
0
single–threaded
multi–threaded
make
slab–alloc
arena–alloc
sync.Pool
chan–pool
ring–pool

How is sync.Pool so fast?

Per-CPU storage!

```go
// pin pins the current goroutine to P, disables preemption and returns poolLocal pool for the P.
// Caller must call runtime_procUnpin() when done with the pool.
func (p *Pool) pin() *poolLocal {
    pid := runtime_procPin()
    // In pinSlow we store to localSize and then to local, here we load in opposite order.
    // Since we've disabled preemption, GC cannot happen in between.
    // Thus here we must observe local at least as large localSize.
    // We can observe a newer/larger local, it is fine (we must observe its zero-initialized-ness).
    s := atomic.LoadUintptr(&p.localSize) // load-acquire
    l := p.local                          // load-consume
    if uintptr(pid) < s {
        return indexLocal(l, pid)
    }
    return p.pinSlow()
}
```

https://golang.org/src/sync/pool.go

```go
// pin pins the current goroutine to P, disables preemption and returns poolLocal pool for the P.
// Caller must call runtime_procUnpin() when done with the pool.
func (p *Pool) pin() *poolLocal {
    pid := runtime_procPin()
    // In pinSlow we store to localSize and then to local, here we load in opposite order.
    // Since we've disabled preemption, GC cannot happen in between.
    // Thus here we must observe local at least as large localSize.
    // We can observe a newer/larger local, it is fine (we must observe its zero-initialized-ness).
    s := atomic.LoadUintptr(&p.localSize) // load-acquire
    l := p.local                          // load-consume
    if uintptr(pid) < s {
        return indexLocal(l, pid)
    }
    return p.pinSlow()
}
```

https://golang.org/src/sync/pool.go

golang / go

Watch 2,311 | Unstar 29,988 | Fork 4,028

Code | Issues 2,884 | Pull requests 0 | Wiki | Insights

sync: per-cpu storage #8281

New issue

Open hyangah opened this issue on Jun 24, 2014 · 23 comments

hyangah commented on Jun 24, 2014 Contributor

```
Provide a way to create/access cpu local objects. This may make it easy to  batch data
per-CPU to reduce global locking. This is a feature request.

Potential use case:

scalable stats across many goroutines - stat updates are batched in the per-CPU stat
object. Periodically or on-demand, the stat batches are aggregated to a global stat
object.

This is similar to sync.Pool in that it will help reducing lock contention, but
different in that the object lifetime should be explicit and the per-CPU objects need to
be accessible from other CPUs if necessary.
```

1

ianlancetaylor commented on Jun 24, 2014 Contributor

Assignees

No one assigned

Labels

Thinking

Projects

None yet

Milestone

Unplanned

Notifications

Subscribe

You're not receiving notifications

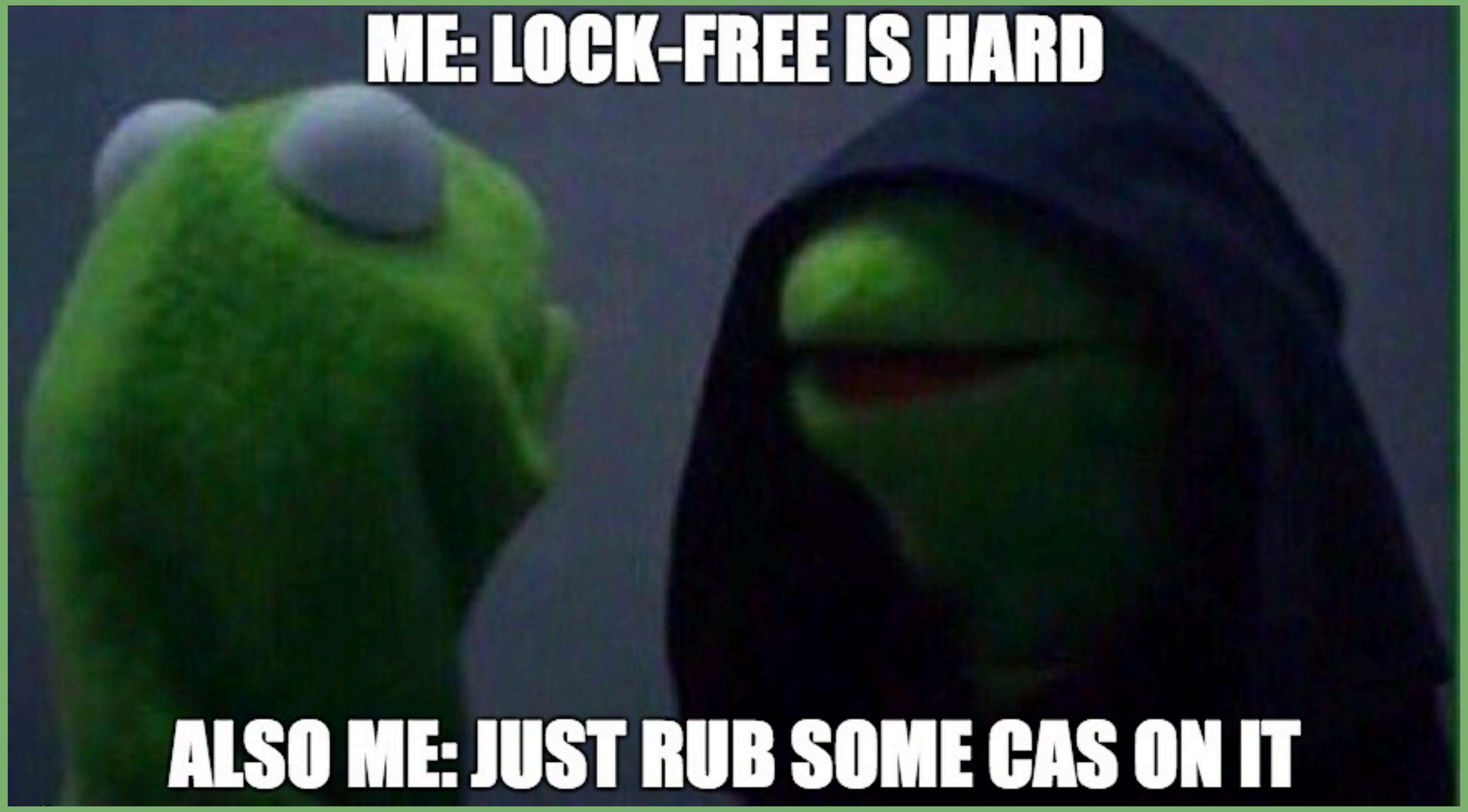
@tyler_treat

Overview

- Measuring performance
- Language features
- Memory management
- Concurrency and multi-core

“We generally don’t want sync/atomic to be used at all…Experience has shown us again and again that **very very few people are capable of writing correct code that uses atomic operations**…”

—Ian Lance Taylor



@tyler_treat

Fast Topic Matching

Subscribers

Subscriber	Match Request
1	forex.usd
2	forex.*
3	stock.nasdaq.msft

Messages

Message	Topic
1	forex.gbp
2	stock.nyse.ibm
3	stock.nyse.ge
4	forex.eur
5	forex.usd
...	...
N	stock.nasdaq.msft

http://bravenewgeek.com/fast-topic-matching/

Fast Topic Matching

Subscribers

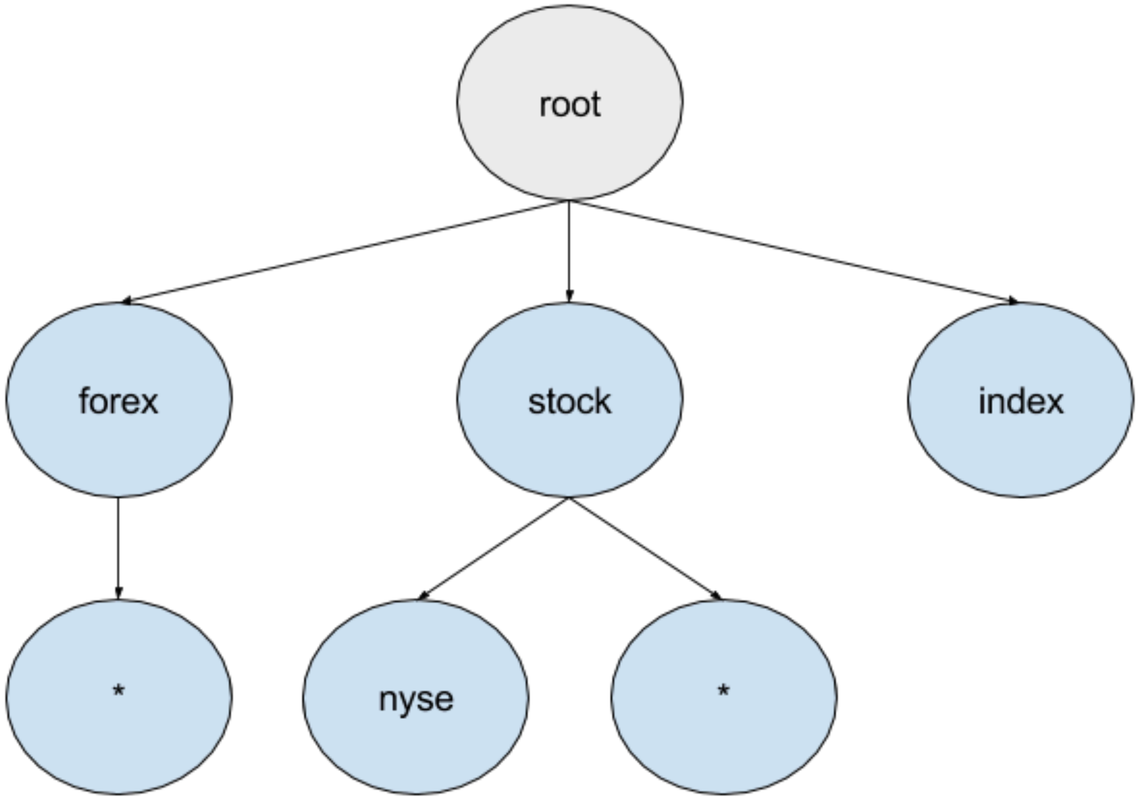
Subscriber	Match Request
1	forex.usd
2	forex.*
3	stock.nasdaq.msft

Messages

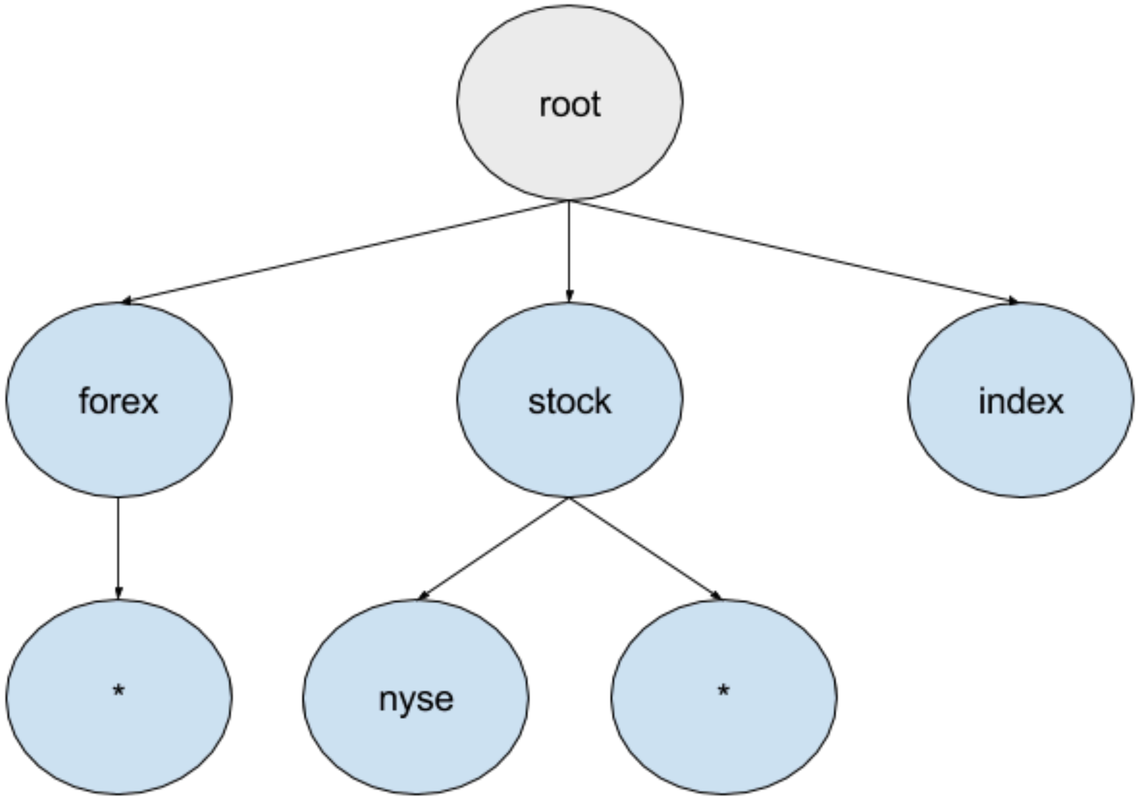
Message	Topic
1	forex.gbp
2	stock.nyse.ibm
3	stock.nyse.ge
4	forex.eur
5	forex.usd
...	...
N	stock.nasdaq.msft

http://bravenewgeek.com/fast-topic-matching/

Fast Topic Matching



Fast Topic Matching



Concurrent Tries with Efficient Non-Blocking Snapshots

Aleksandar Prokopec — EPFL — aleksandar.prokopec@epfl.ch

Nathan G. Bronson — Stanford — ngbronson@gmail.com

Phil Bagwell — Typesafe — phil.bagwell@typesafe.com

Martin Odersky — EPFL — martin.odersky@epfl.ch

Abstract

We describe a non-blocking concurrent hash trie based on shared-memory single-word compare-and-swap instructions. The hash trie supports standard mutable lock-free operations such as insertion, removal, lookup and their conditional variants. To ensure space-efficiency, removal operations compress the trie when necessary.

We show how to implement an efficient lock-free snapshot operation for concurrent hash tries. The snapshot operation uses a single-word compare-and-swap and avoids copying the data structure eagerly. Snapshots are used to implement consistent iterators and a linearizable size retrieval. We compare concurrent hash trie performance with other concurrent data structures and evaluate the performance of the snapshot operation.

Categories and Subject Descriptors E.1 [*Data structures*]: Trees

General Terms Algorithms

Keywords hash trie, concurrent data structure, snapshot, non-blocking

1. Introduction

When designing concurrent data structures, lock-freedom is an important concern. Lock-free data structures generally guarantee better robustness than their lock-based variants [11], as they are unaffected by thread delays. A fundamental difference is that lock-free data structures can continue to work correctly in the presence of failures, whereas a failure may prevent progress indefinitely with a lock-based data structure.

While obtaining a consistent view of the contents of the data structure at a single point in time is a trivial matter for data structures accessed sequentially, it is not clear how to do this in a non-blocking concurrent setting. A consistent view of the data structure at a single point in time is called a *snapshot*. Snapshots can be

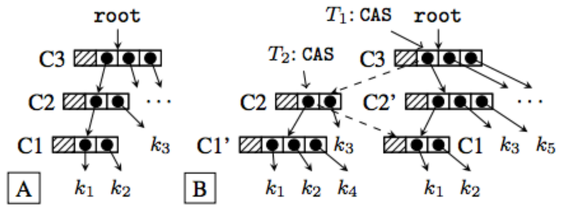


Figure 1. Hash tries

2. We introduce a non-blocking, atomic constant-time snapshot operation. We show how to use them to implement atomic size retrieval, consistent iterators and an atomic clear operation.
3. We present benchmarks that compare performance of concurrent tries against other concurrent data structures across different architectures.

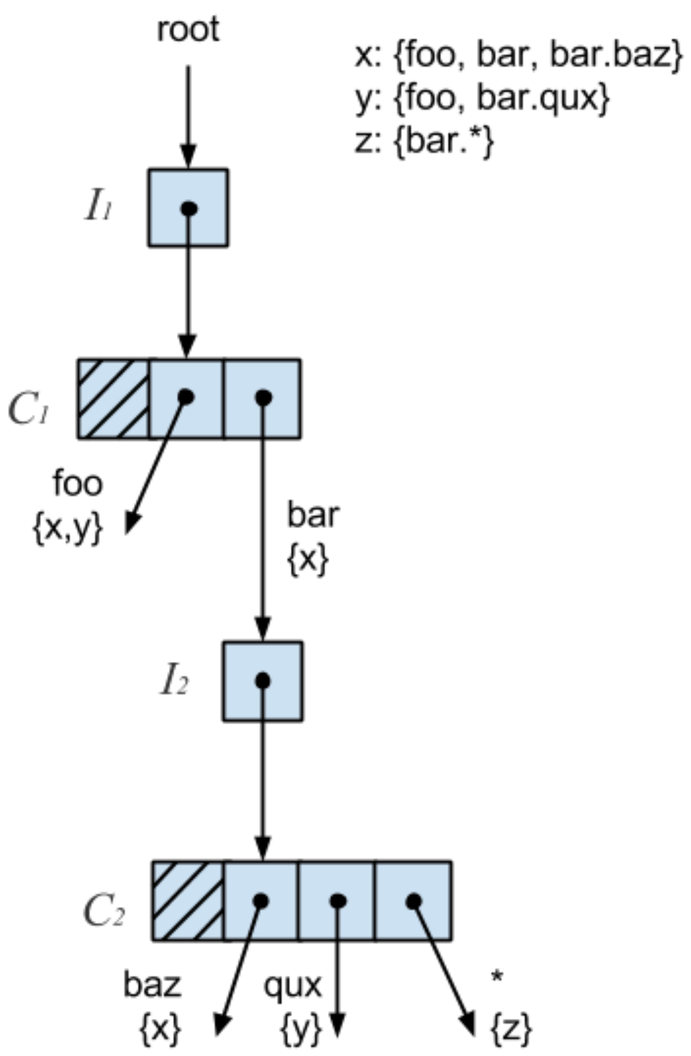
Section 2 illustrates usefulness of snapshots. Section 3 describes basic operations on concurrent tries. Section 4 describes snapshots. Section 5 presents various benchmarks. Section 6 contains related work and Section 7 concludes.

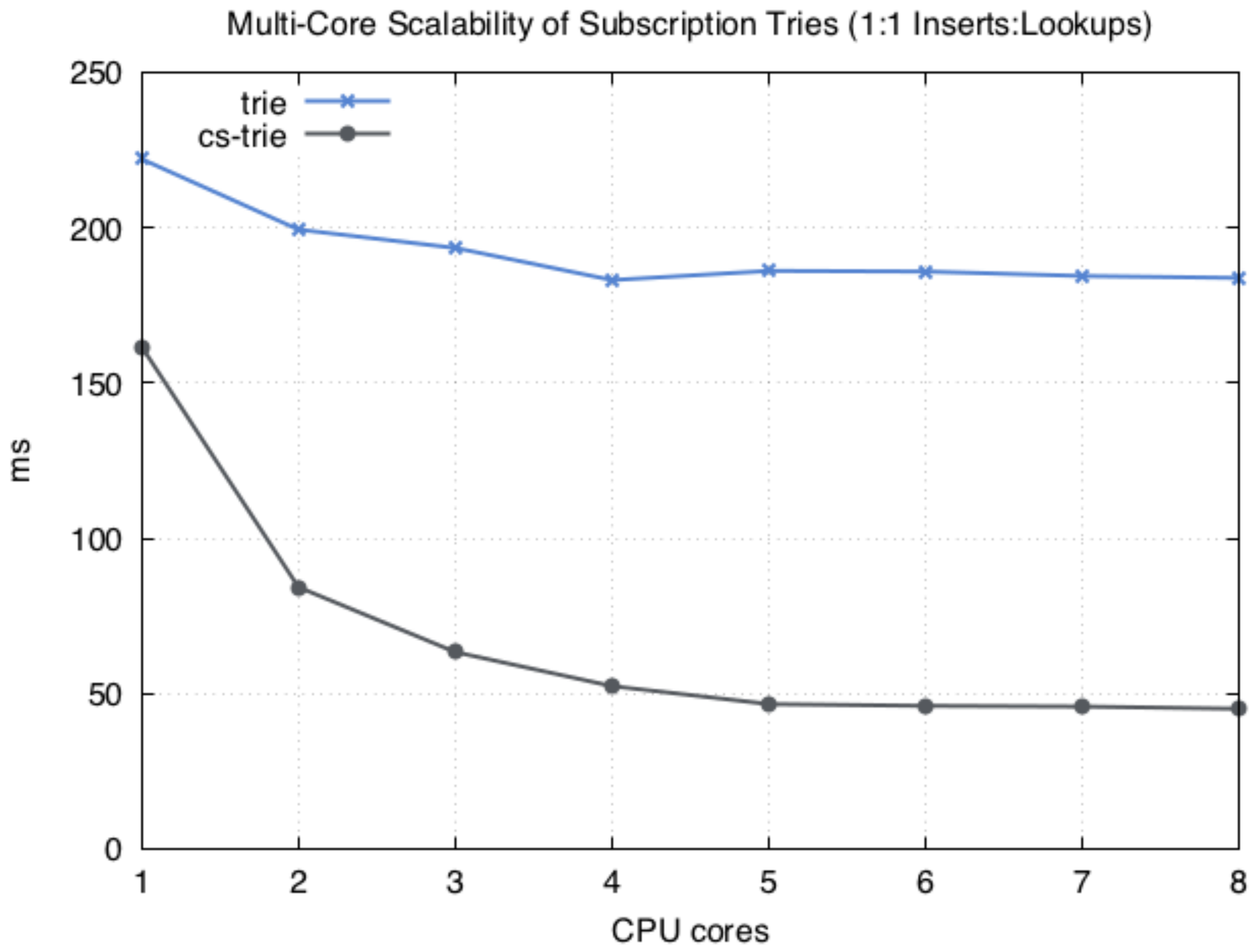
2. Motivation

Most stock concurrent collection implementations include operations such as the atomic lookup, insert and remove operations. Operations that require global data structure information or induce a global change in the data structure, such as size retrieval, iterator creation or deleting all the elements are typically implemented with no atomicity guarantees (e.g. the Java `ConcurrentHashMap` and the `ConcurrentSkipListMap` [14]) or require a global lock. Ide-

@tyler_treat

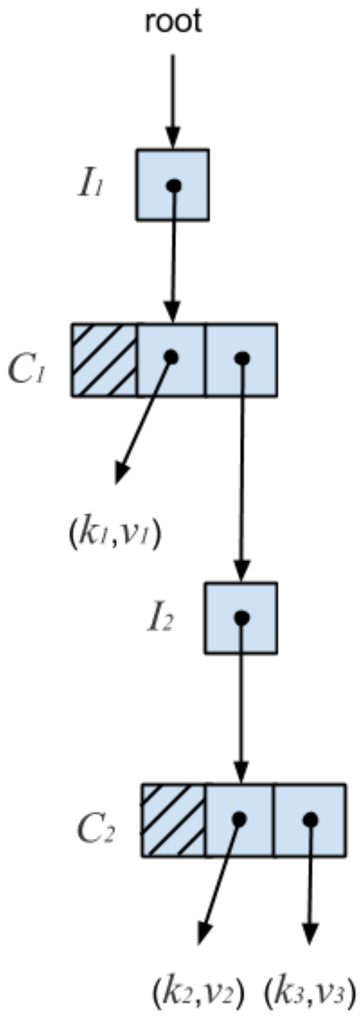
Fast Topic Matching





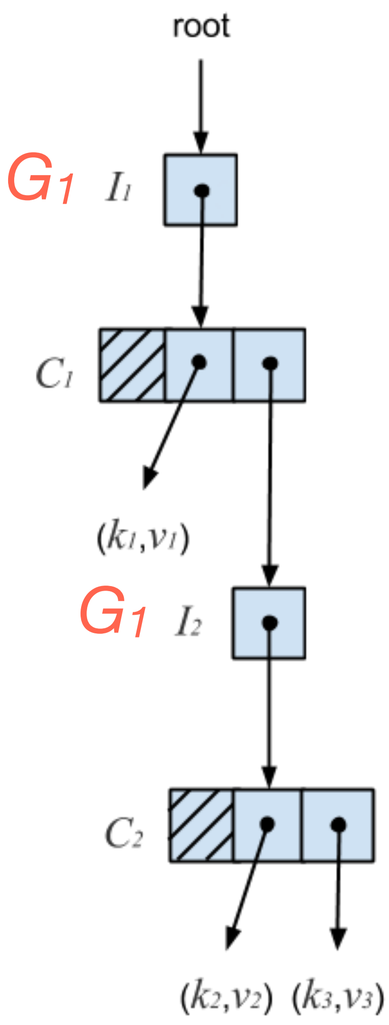
Concurrent
80,000 inserts
80,000 lookups

Ctrie



Ctrie

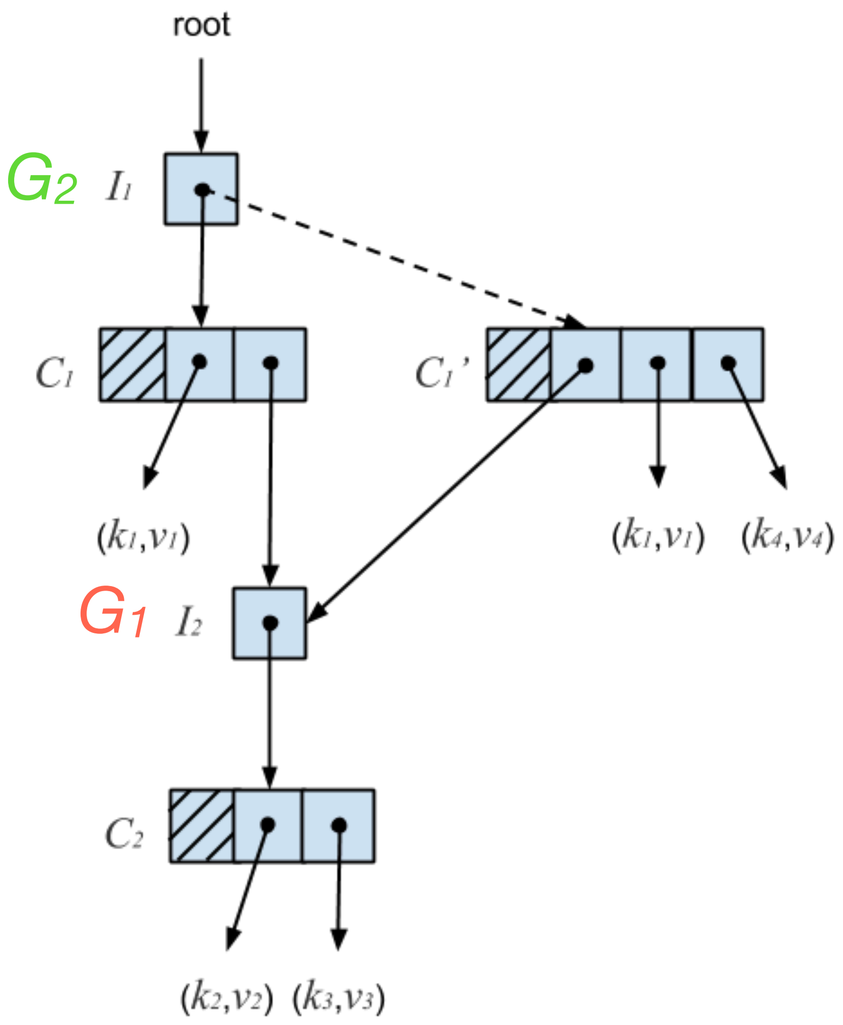
1. Assign a generation, G_1, to each I-node (empty struct).



Ctrie

1. Assign a generation, G_1, to each I-node (empty struct).

2. Add new node by copying I-node with updated branch and generation then GCAS, i.e. **atomically**:
   - compare I-nodes to detect tree mutations.
   - compare root generations to detect snapshots.



A Practical Multi-Word Compare-and-Swap Operation

Timothy L. Harris, Keir Fraser and Ian A. Pratt

University of Cambridge Computer Laboratory, Cambridge, UK
{tim.harris,keir.fraser,ian.pratt}@cl.cam.ac.uk

Abstract. Work on non-blocking data structures has proposed extending processor designs with a compare-and-swap primitive, CAS2, which acts on two arbitrary memory locations. Experience suggested that current operations, typically single-word compare-and-swap (CAS1), are not expressive enough to be used alone in an efficient manner. In this paper we build CAS2 from CAS1 and, in fact, build an arbitrary multi-word compare-and-swap (CASN). Our design requires only the primitives available on contemporary systems, reserves a small and constant amount of space in each word updated (either 0 or 2 bits) and permits non-overlapping updates to occur concurrently. This provides compelling evidence that current primitives are not only universal in the theoretical sense introduced by Herlihy, but are also universal in their use as foundations for practical algorithms. This provides a straightforward mechanism for deploying many of the interesting non-blocking data structures presented in the literature that have previously required CAS2.

1 Introduction

CASN is an operation for shared-memory systems that reads the contents of a series of locations, compares these against specified values and, if they all match, updates the locations with a further set of values. All this is performed atomically with respect to other CASN operations and specialized reads. The implementation of a non-blocking multi-word compare-and-swap operation has been the focus of many research papers [7, 10, 2, 3, 15, 5]. As we will show none of these provides a solution that is practicable in terms of the operations it requires from the processor, its storage costs and the features it supplies.

@tyler_treat

Workiva / go-datastructures

Unwatch 209 | Unstar 3,294 | Fork 316

Code | Issues 6 | Pull requests 4 | Projects 0 | Wiki | Insights

ctrie data race #122

New issue

Closed newhook opened this issue on Dec 24, 2015 · 9 comments

newhook commented on Dec 24, 2015

I just got a report of data race in the ctrie impl on committed.

```
func (c *Ctrie) rdcssRoot(old *iNode, expected *mainNode, nv *iNode) bool {
    desc := &iNode{
        rdcss: &rdcssDescriptor{
            old:      old,
            expected: expected,
            nv:       nv,
        },
    }
    if c.casRoot(old, desc) {
        c.rdcssComplete(false)
        return desc.rdcss.committed
    }
    return false
}
```

```
func (c *Ctrie) rdcssComplete(abort bool) *iNode {
```

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Notifications

Unsubscribe

You're receiving notifications because you commented.

@tyler_treat

The Go race detector doesn’t protect you from *doing dumb stuff*.

```
            trie.Snapshot()
        }
        wg.Done()
    }()
    wg.Wait()
    fmt.Println("size", trie.Size())
}
```

Expected size is 7, but if you run it enough times you should see values less than 7 occasionally.

tylertreat commented on Dec 26, 2015 Contributor

Issue appears to be with GCAS. I made an incorrect assumption from Java (where `new Object() != new Object()`). In Go, this is not quite the case: https://play.golang.org/p/BBde4cX1GC. As a result, GCAS always succeeds, despite the root generation changing during a snapshot.

tylertreat commented on Dec 26, 2015 Contributor

I believe I have it fixed with this: https://github.com/Workiva/go-datastructures/compare/master...tylertreat:fixes?expand=1. I haven't been able to reproduce after applying those fixes.

@newhook can you confirm this branch fixes your problem?

newhook commented on Dec 27, 2015

@tyler_treat

```
uintptr(unsafe.Pointer(&s)) + unsafe.Offsetof(s.f) == uintptr(unsafe.Pointer(&s.f))
```

Computer architectures may require memory addresses to be *aligned*; that is, for addresses of a variable to be a multiple of a factor, the variable's type's *alignment*. The function `Alignof` takes an expression denoting a variable of any type and returns the alignment of the (type of the) variable in bytes. For a variable `x`:

```
uintptr(unsafe.Pointer(&x)) % unsafe.Alignof(x) == 0
```

Calls to `Alignof`, `Offsetof`, and `Sizeof` are compile-time constant expressions of type `uintptr`.

Size and alignment guarantees

For the numeric types, the following sizes are guaranteed:

```
type                                size in bytes

byte, uint8, int8                    1
uint16, int16                        2
uint32, int32, float32               4
uint64, int64, float64, complex64    8
complex128                          16
```

The following minimal alignment properties are guaranteed:

1. For a variable `x` of any type: `unsafe.Alignof(x)` is at least 1.
2. For a variable `x` of struct type: `unsafe.Alignof(x)` is the largest of all the values `unsafe.Alignof(x.f)` for each field `f` of `x`, but at least 1.
3. For a variable `x` of array type: `unsafe.Alignof(x)` is the same as `unsafe.Alignof(x[0])`, but at least 1.

A struct or array type has size zero if it contains no fields (or elements, respectively) that have a size greater than zero. Two distinct zero-size variables may have the same address in memory.

Except as noted, the content of this page is licensed under the Creative Commons Attribution 3.0 License, and code is licensed under a BSD license.
Terms of Service | Privacy Policy

@tyler_treat

Code | Issues 6 | Pull requests 4 | Projects 0 | Wiki | Insights

Fix Ctrie snapshotting #123

Edit

Merged tylertreat-wf merged 7 commits into `Workiva:master` from `tylertreat:fixes` on Jan 4, 2016

Conversation 4 | Commits 7 | Files changed 2

Changes from all commits | 2 files | +61 −17

Unified | Split | Review changes

37 trie/ctrie/ctrie.go

View

```
@@ -63,8 +63,10 @@ type Ctrie struct {
 }

 // generation demarcates Ctrie snapshots. We use a heap-allocated reference
-// instead of an integer to avoid integer overflows.
-type generation struct{}
+// instead of an integer to avoid integer overflows. Struct must have a field
+// on it since two distinct zero-size variables may have the same address in
+// memory.
+type generation struct{ _ int }

 // iNode is an indirection node. I-nodes remain present in the Ctrie even as
 // nodes above and below change. Thread-safety is achieved in part by
@@ -320,7 +322,7 @@ func (c *Ctrie) Snapshot() *Ctrie {
                root := c.readRoot()
                main := gcasRead(root, c)
                if c.rdcssRoot(root, main, root.copyToGen(&generation{}, c)) {
```

@tyler_treat

Side note:

unsafe is, in fact, unsafe.

“Packages that import unsafe may depend on internal properties of the Go implementation. We reserve the right to make changes to the implementation that may break such programs.”

https://golang.org/doc/go1compat

Tyler Treat
@tyler_treat

Golang protip:

import yolo "unsafe"

9:36 PM - 17 Aug 2017

2 Retweets **16** Likes

1 2 16

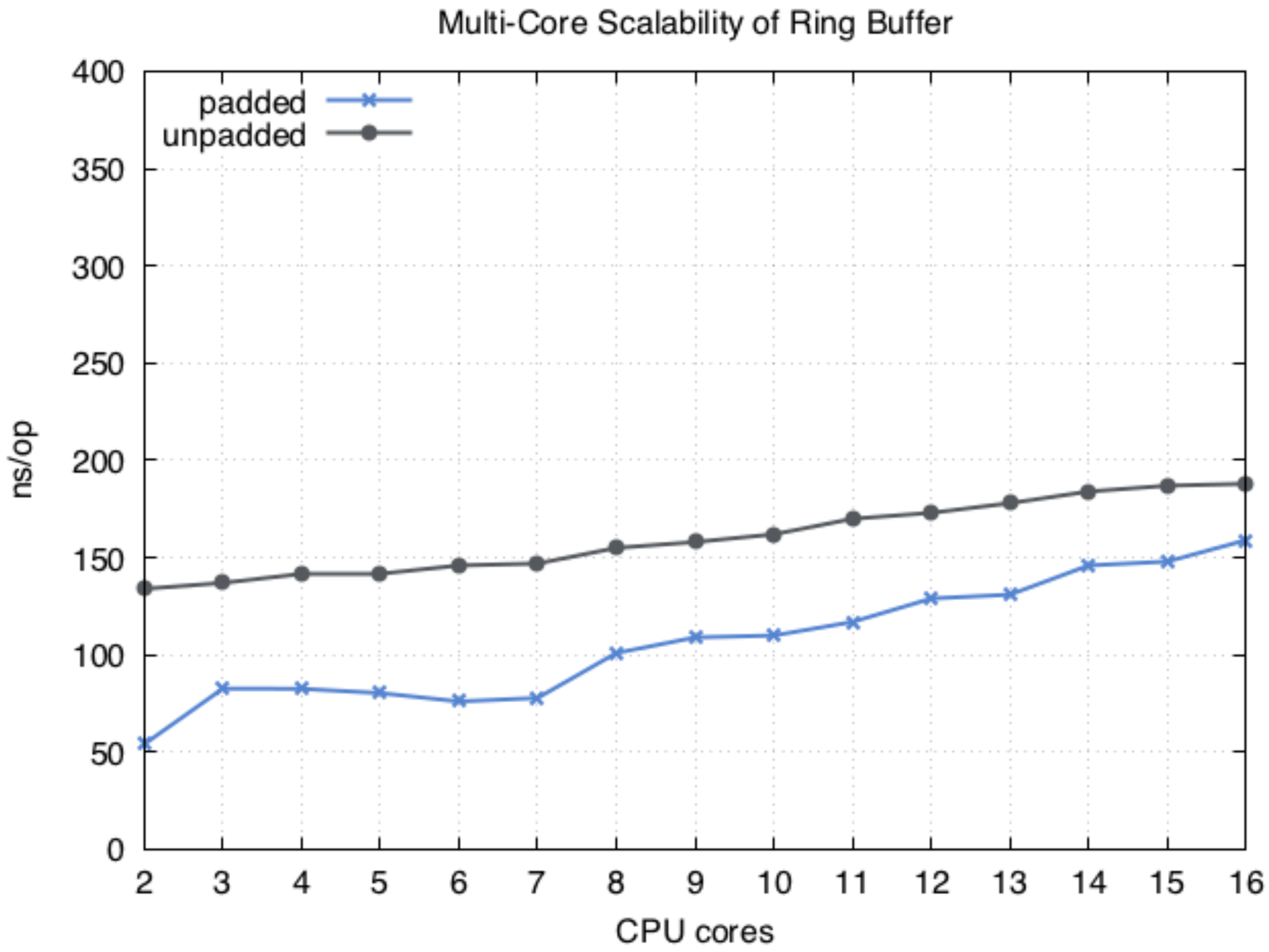
Key Insight:
Struct layout can make a big difference.

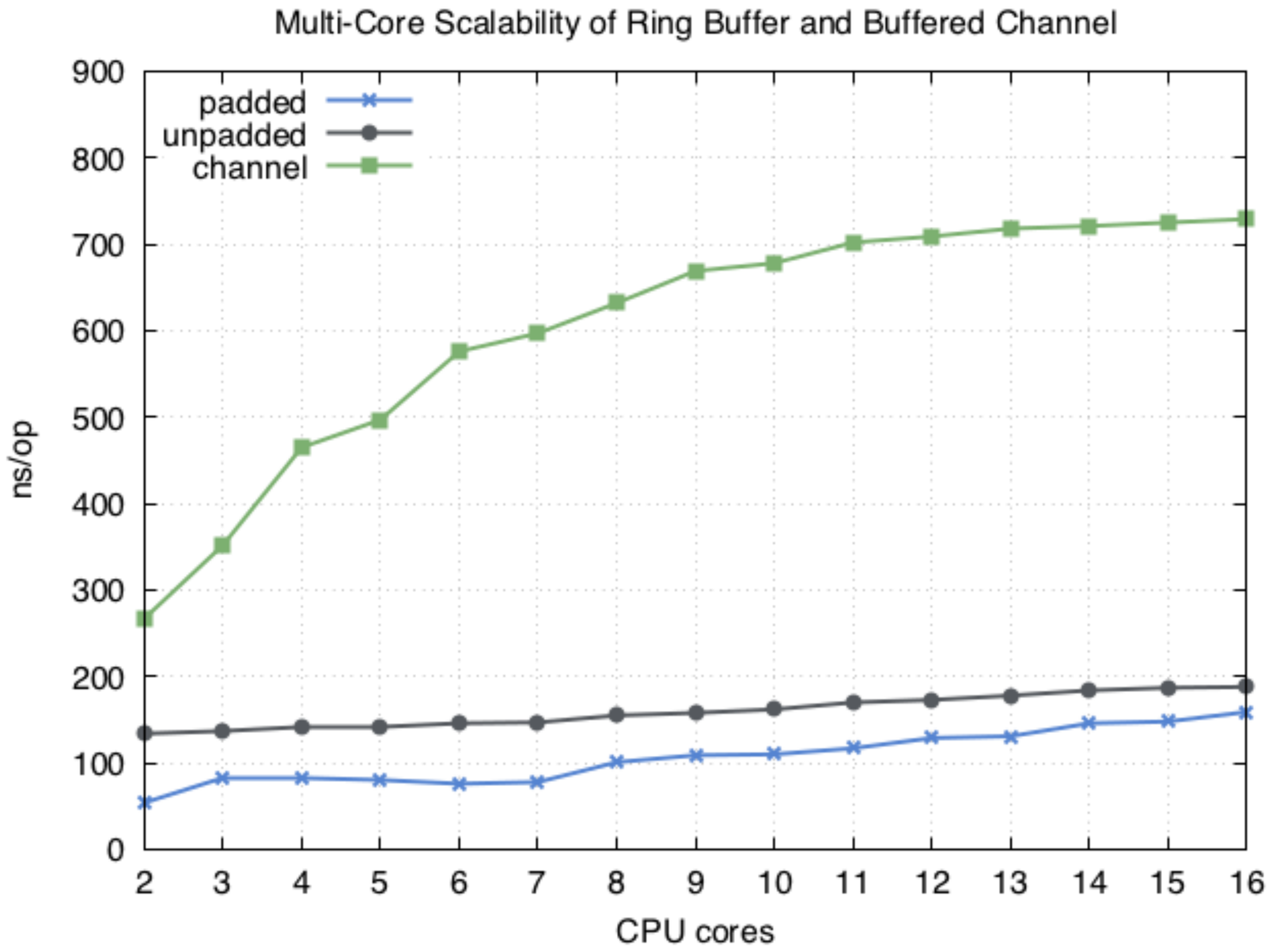
Mechanical Sympathy

@tyler_treat

```go
// RingBuffer is a MPMC buffer that achieves threadsafety with CAS operations
// only.  A put on full or get on empty call will block until an item
// is put or retrieved.  Calling Dispose on the RingBuffer will unblock
// any blocked threads with an error.  This buffer is similar to the buffer
// described here: http://www.1024cores.net/home/lock-free-algorithms/queues/bounded-mpmc-queue
// with some minor additions.
type RingBuffer struct {
    _padding0      [8]uint64
    queue          uint64
    _padding1      [8]uint64
    dequeue        uint64
    _padding2      [8]uint64
    mask, disposed uint64
    _padding3      [8]uint64
    nodes          nodes
}
```

https://github.com/Workiva/go-datastructures/blob/master/queue/ring.go


Multi-Core Scalability of Ring Buffer
padded
unpadded
ns/op
400
350
300
250
200
150
100
50
0
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
CPU cores


Multi-Core Scalability of Ring Buffer and Buffered Channel
padded
unpadded
channel
ns/op
900
800
700
600
500
400
300
200
100
0
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
CPU cores

golang / go

Watch 2,311 | Unstar 29,989 | Fork 4,028

Code | Issues 2,884 | Pull requests 0 | Wiki | Insights

sync: RWMutex scales poorly with CPU count #17973

New issue

Open bcmills opened this issue on Nov 17, 2016 · 39 comments

bcmills commented on Nov 17, 2016 • edited

Member

On a machine with many cores, the performance of `sync.RWMutex.R{Lock,Unlock}` degrades dramatically as `GOMAXPROCS` increases.

This test program:

```
package benchmarks_test

import (
	"fmt"
	"sync"
	"testing"
)

func BenchmarkRWMutex(b *testing.B) {
	for ng := 1; ng <= 256; ng <<= 2 {
		b.Run(fmt.Sprint(ng), func(b *testing.B) {
			var mu sync.RWMutex
			mu.Lock()

			var wg sync.WaitGroup
```

Assignees

No one assigned

Labels

NeedsFix

Performance

Projects

None yet

Milestone

Go1.10

Notifications

Subscribe

You're not receiving notifications

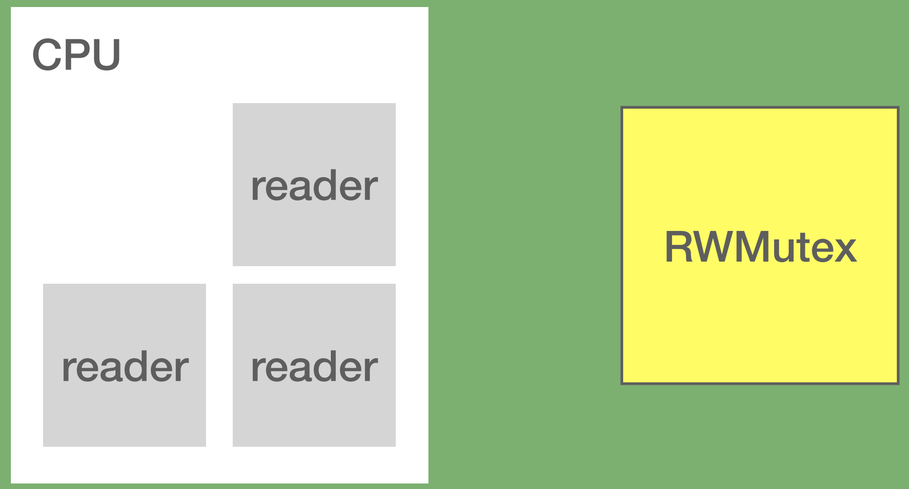
@tyler_treat

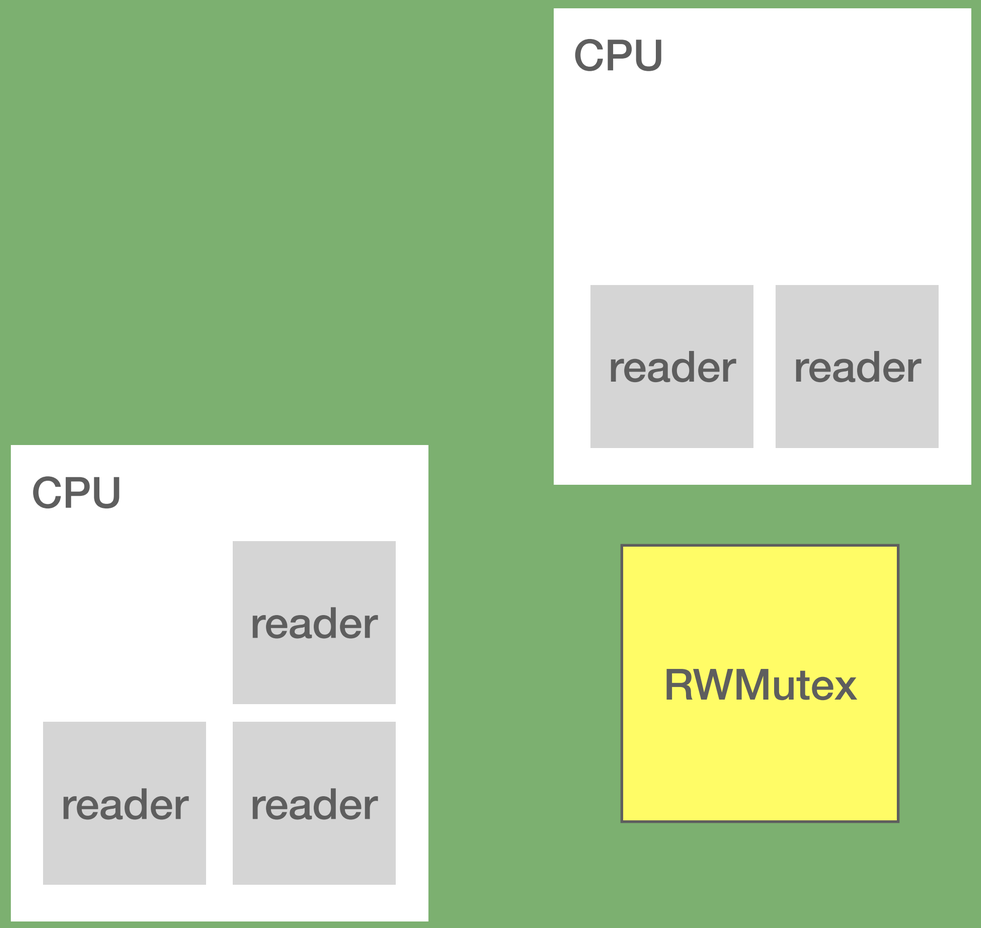
```go
// An RWMutex is a reader/writer mutual exclusion lock.
// The lock can be held by an arbitrary number of readers or a single writer.
// RWMutexes can be created as part of other structures;
// the zero value for a RWMutex is an unlocked mutex.
//
// An RWMutex must not be copied after first use.
//
// If a goroutine holds a RWMutex for reading, it must not expect this or any
// other goroutine to be able to also take the read lock until the first read
// lock is released. In particular, this prohibits recursive read locking.
// This is to ensure that the lock eventually becomes available;
// a blocked Lock call excludes new readers from acquiring the lock.
type RWMutex struct {
    w           Mutex  // held if there are pending writers
    writerSem   uint32 // semaphore for writers to wait for completing readers
    readerSem   uint32 // semaphore for readers to wait for completing writers
    readerCount int32  // number of pending readers
    readerWait  int32  // number of departing readers
}
```

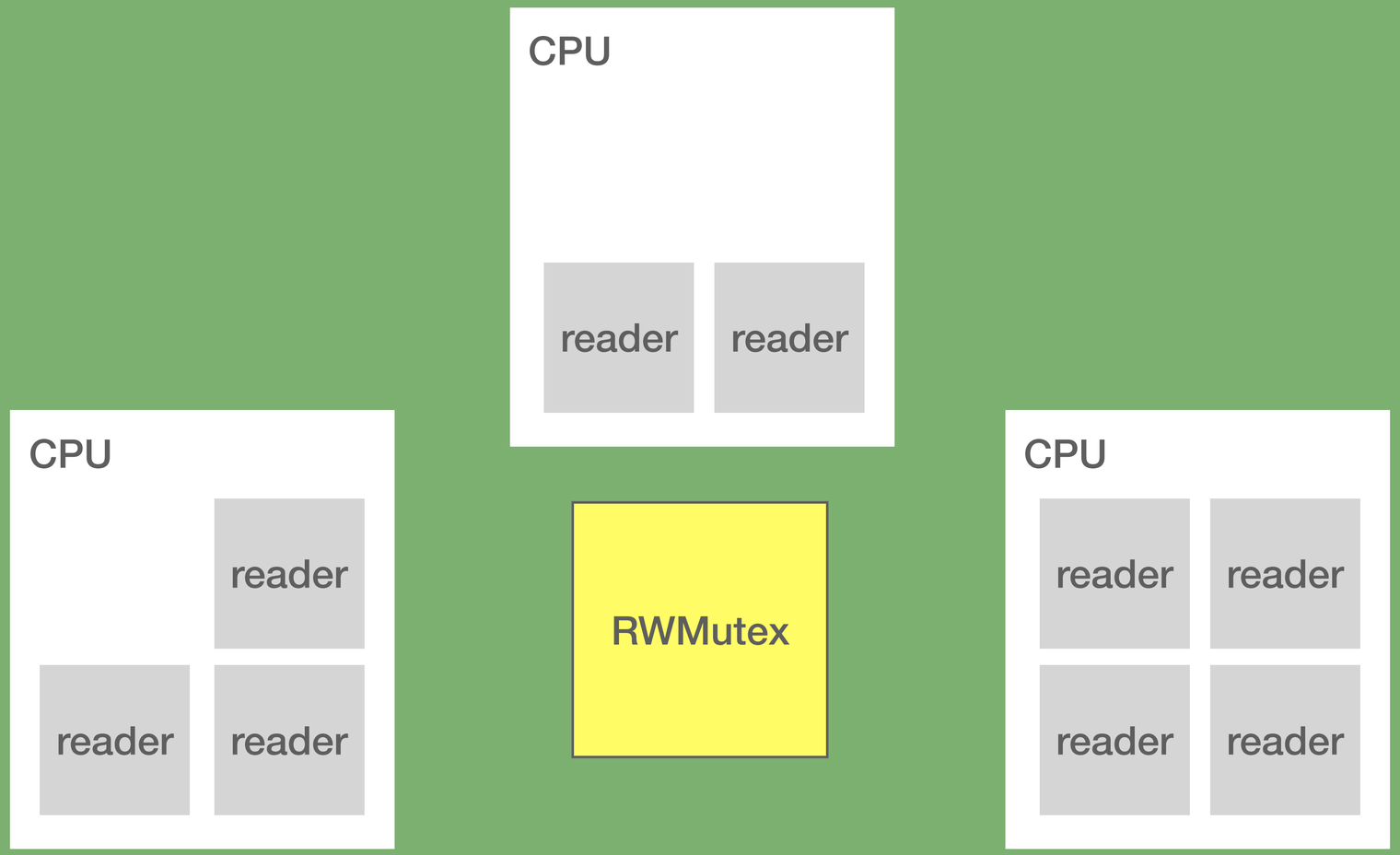
https://golang.org/src/sync/rwmutex.go

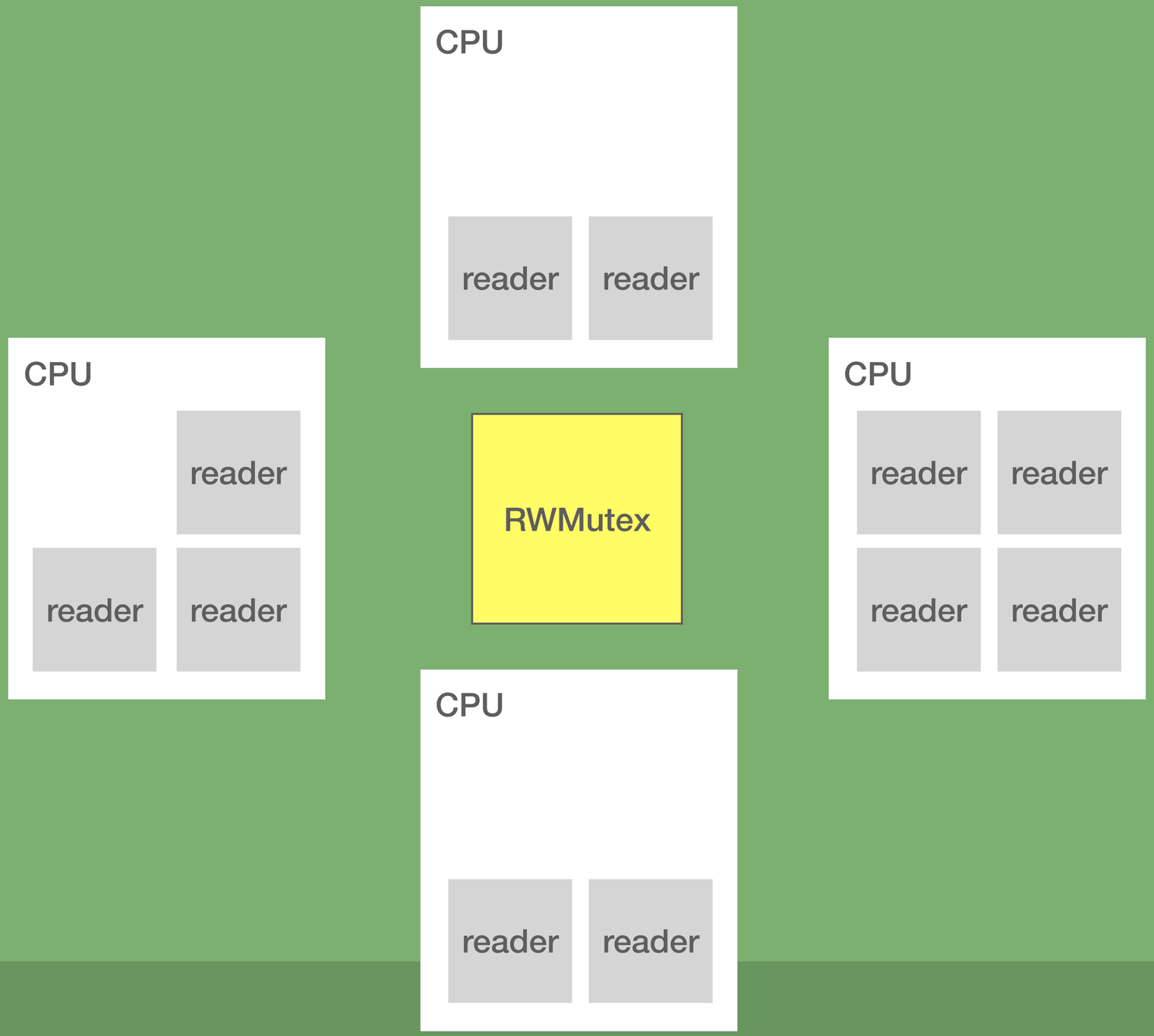
```go
// An RWMutex is a reader/writer mutual exclusion lock.
// The lock can be held by an arbitrary number of readers or a single writer.
// RWMutexes can be created as part of other structures;
// the zero value for a RWMutex is an unlocked mutex.
//
// An RWMutex must not be copied after first use.
//
// If a goroutine holds a RWMutex for reading, it must not expect this or any
// other goroutine to be able to also take the read lock until the first read
// lock is released. In particular, this prohibits recursive read locking.
// This is to ensure that the lock eventually becomes available;
// a blocked Lock call excludes new readers from acquiring the lock.
type RWMutex struct {
    w           Mutex  // held if there are pending writers
    writerSem   uint32 // semaphore for writers to wait for completing readers
    readerSem   uint32 // semaphore for readers to wait for completing writers
    readerCount int32  // number of pending readers
    readerWait  int32  // number of departing readers
}
```

https://golang.org/src/sync/rwmutex.go

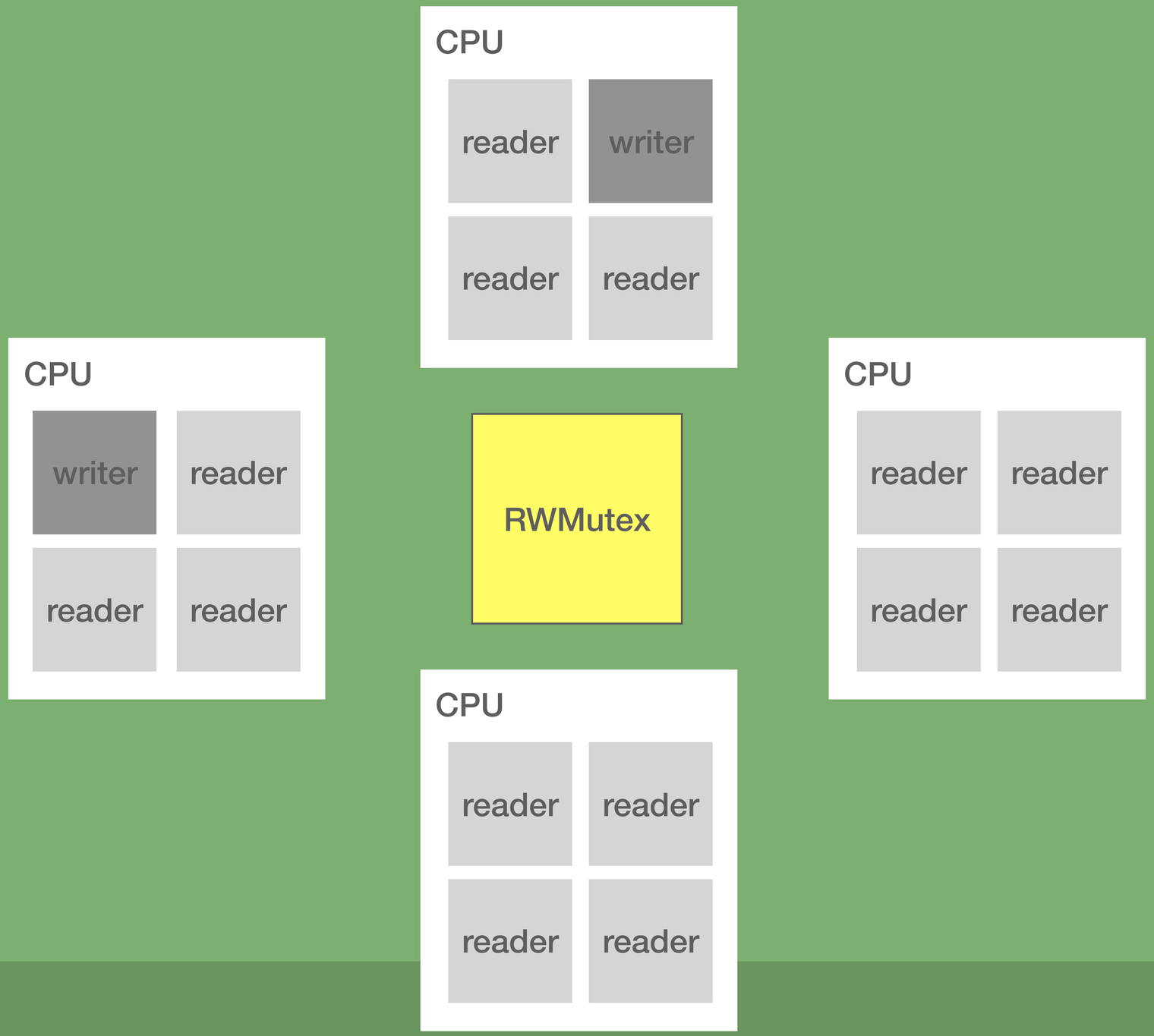

CPU
reader
reader
reader
RWMutex


CPU
reader
reader
CPU
reader
reader
reader
RWMutex

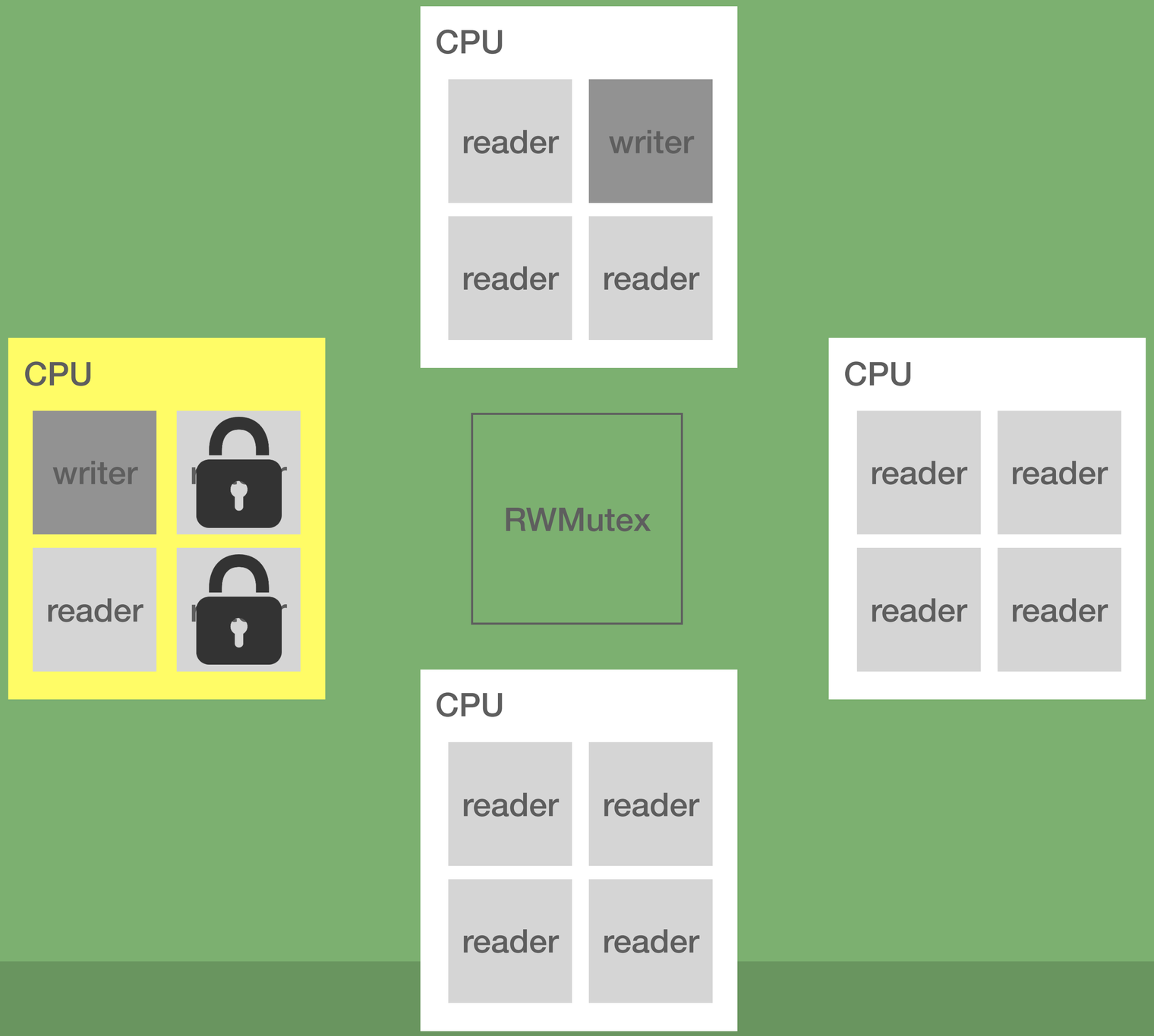

CPU
reader
reader
CPU
reader
reader
reader
RWMutex
CPU
reader
reader
reader
reader


CPU
reader
reader
CPU
reader
reader
reader
RWMutex
CPU
reader
reader
reader
reader
CPU
reader
reader

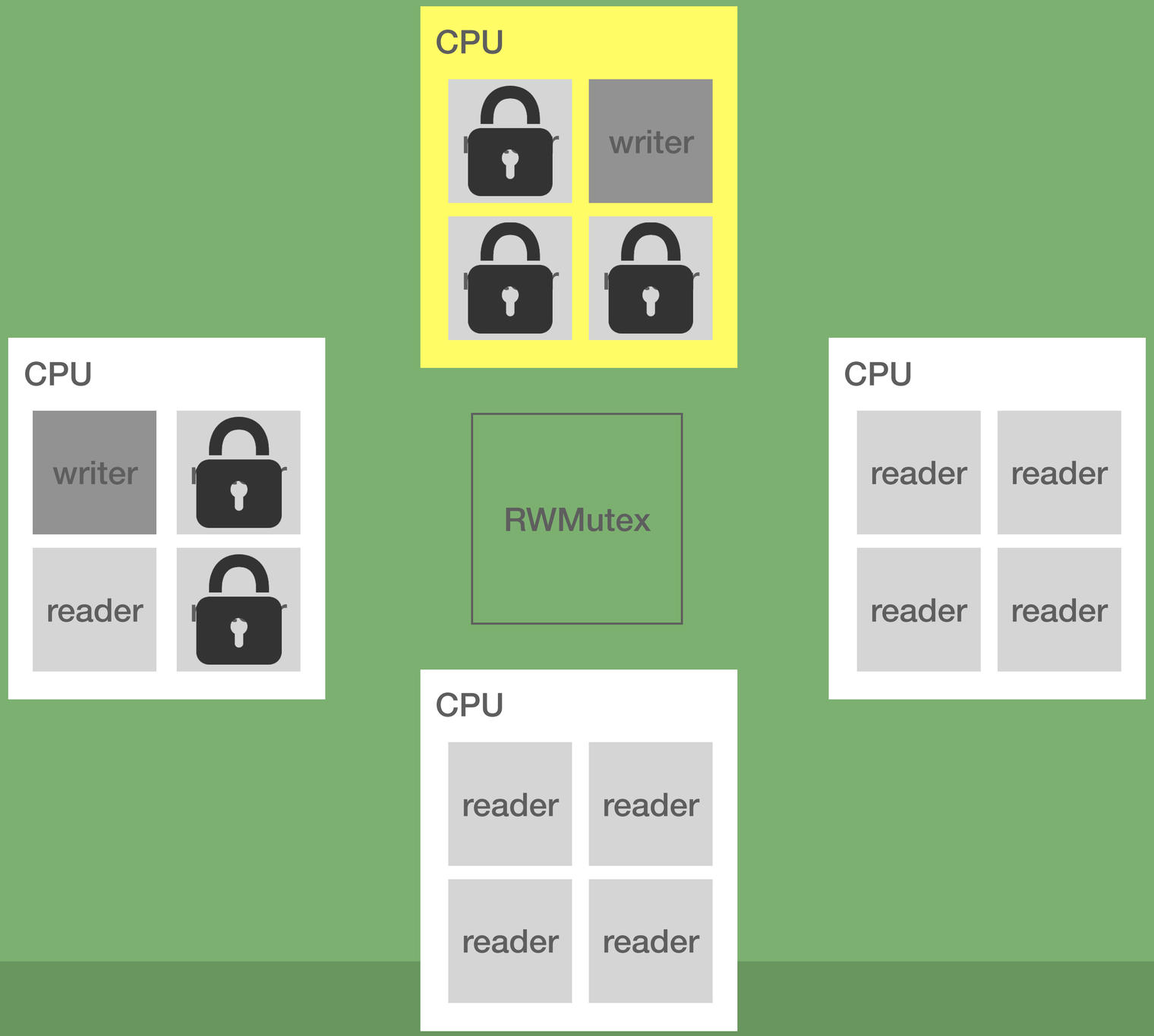
@tyler_treat


CPU
reader
writer
reader
reader
CPU
writer
reader
reader
reader
RWMutex
CPU
reader
reader
reader
reader
CPU
reader
reader
reader
reader

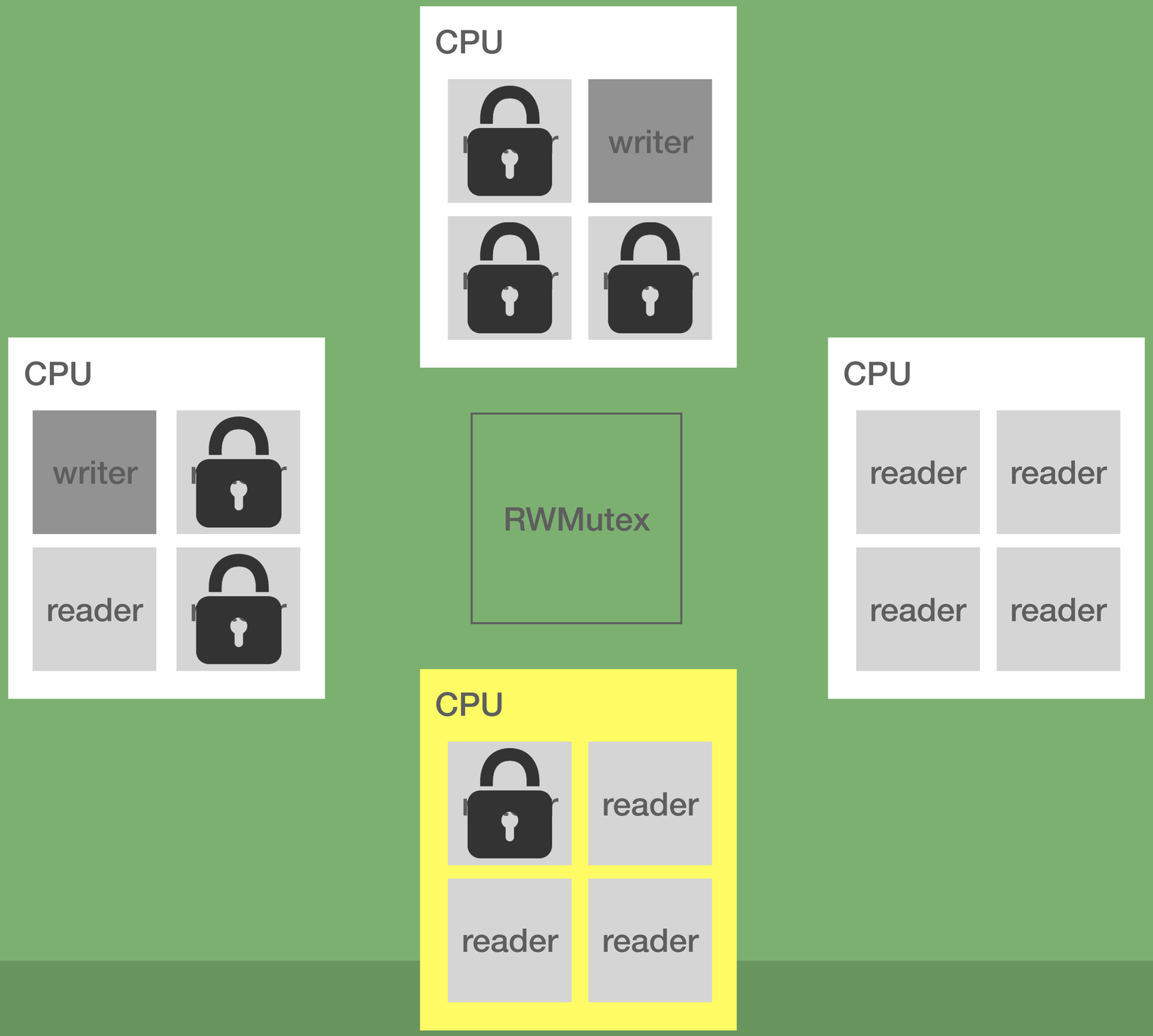
@tyler_treat


CPU
reader
writer
reader
reader
CPU
writer
reader
RWMutex
CPU
reader
reader
reader
reader
CPU
reader
reader
reader
reader

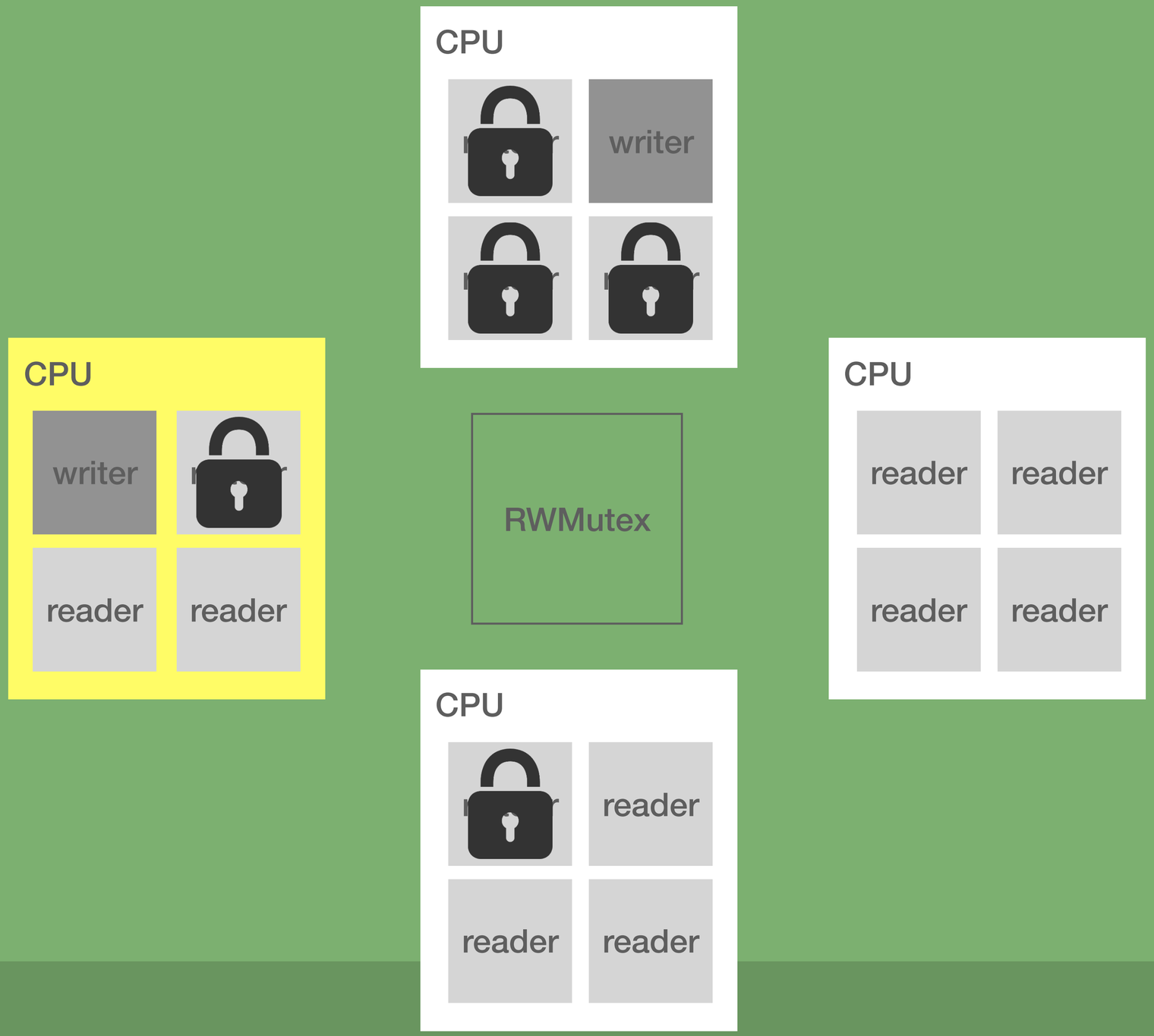
@tyler_treat


CPU
writer
CPU
writer
reader
RWMutex
CPU
reader
reader
reader
reader
CPU
reader
reader
reader
reader

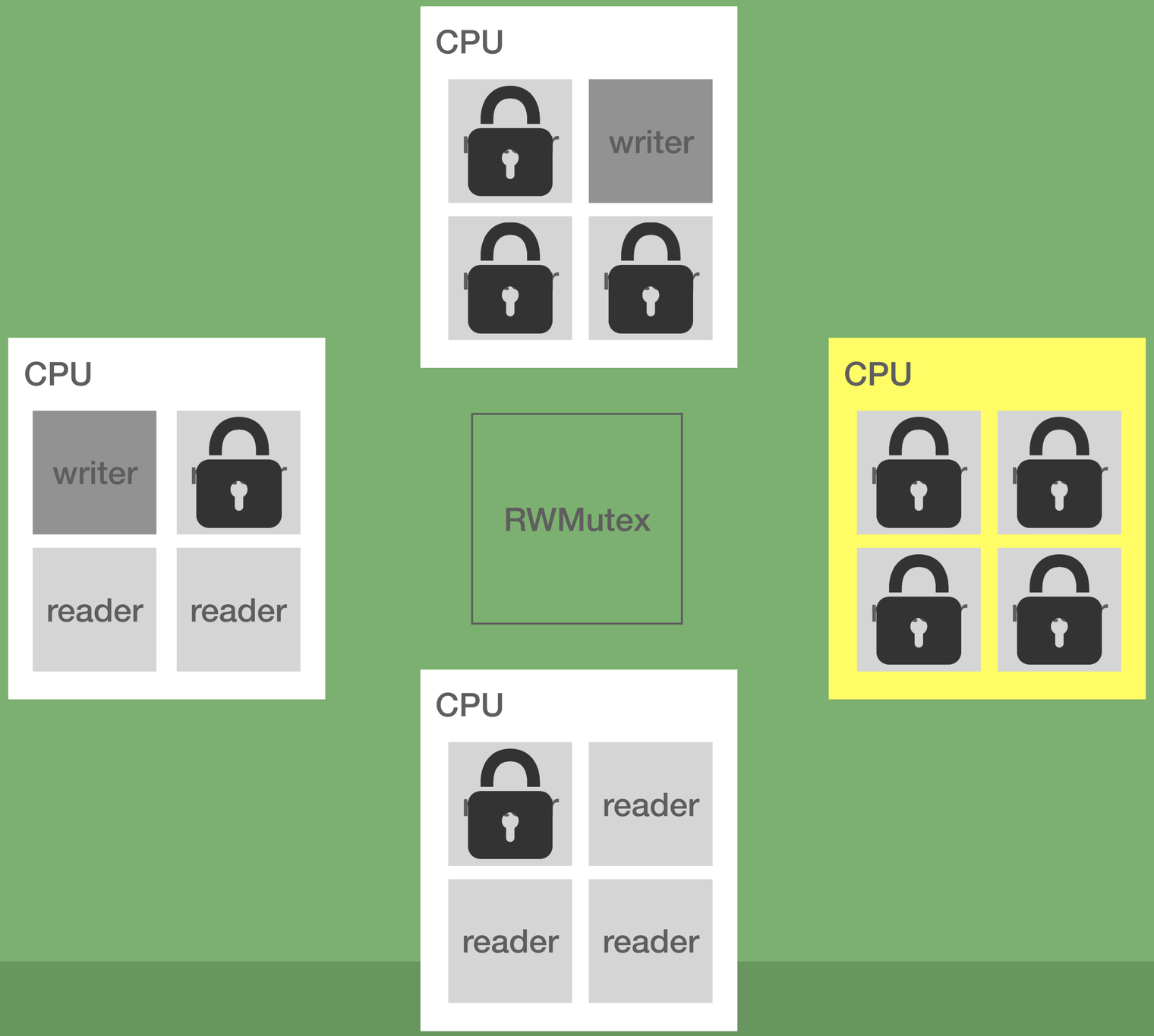
@tyler_treat


CPU
writer
CPU
writer
reader
RWMutex
CPU
reader
reader
reader
reader
CPU
reader
reader
reader

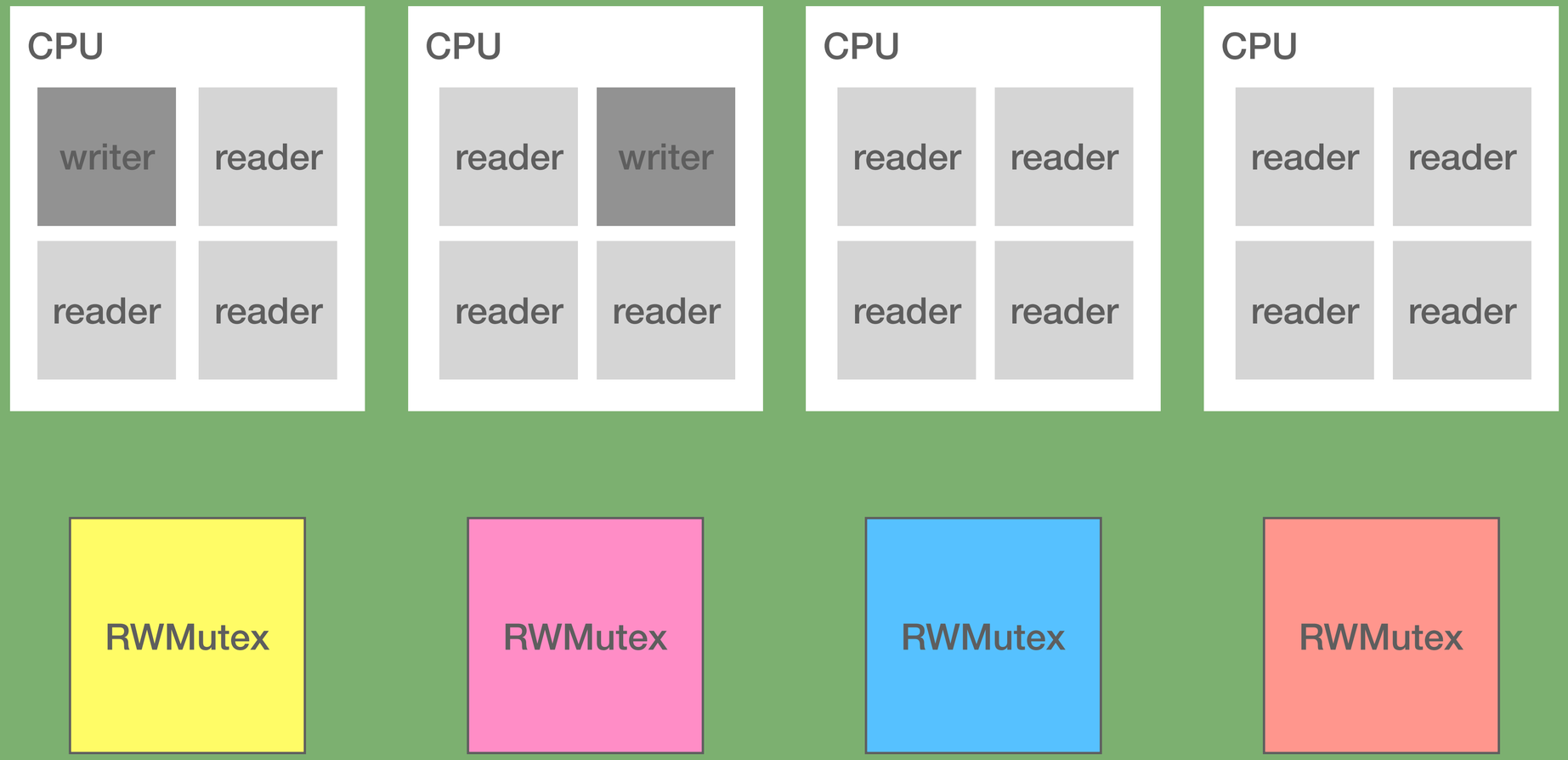
@tyler_treat

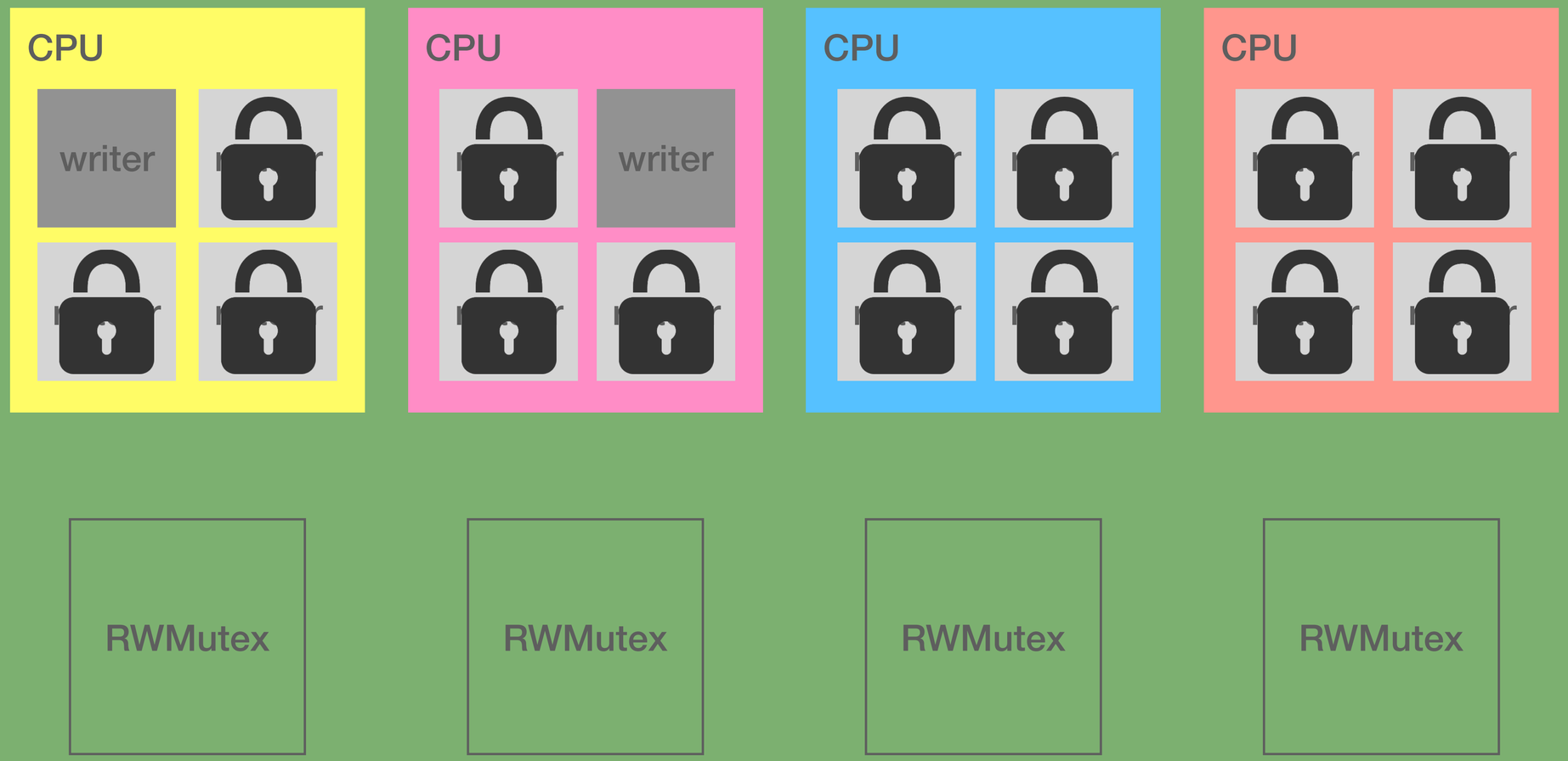

CPU
writer
CPU
writer
reader
reader
RWMutex
CPU
reader
reader
reader
reader
CPU
reader
reader
reader

@tyler_treat

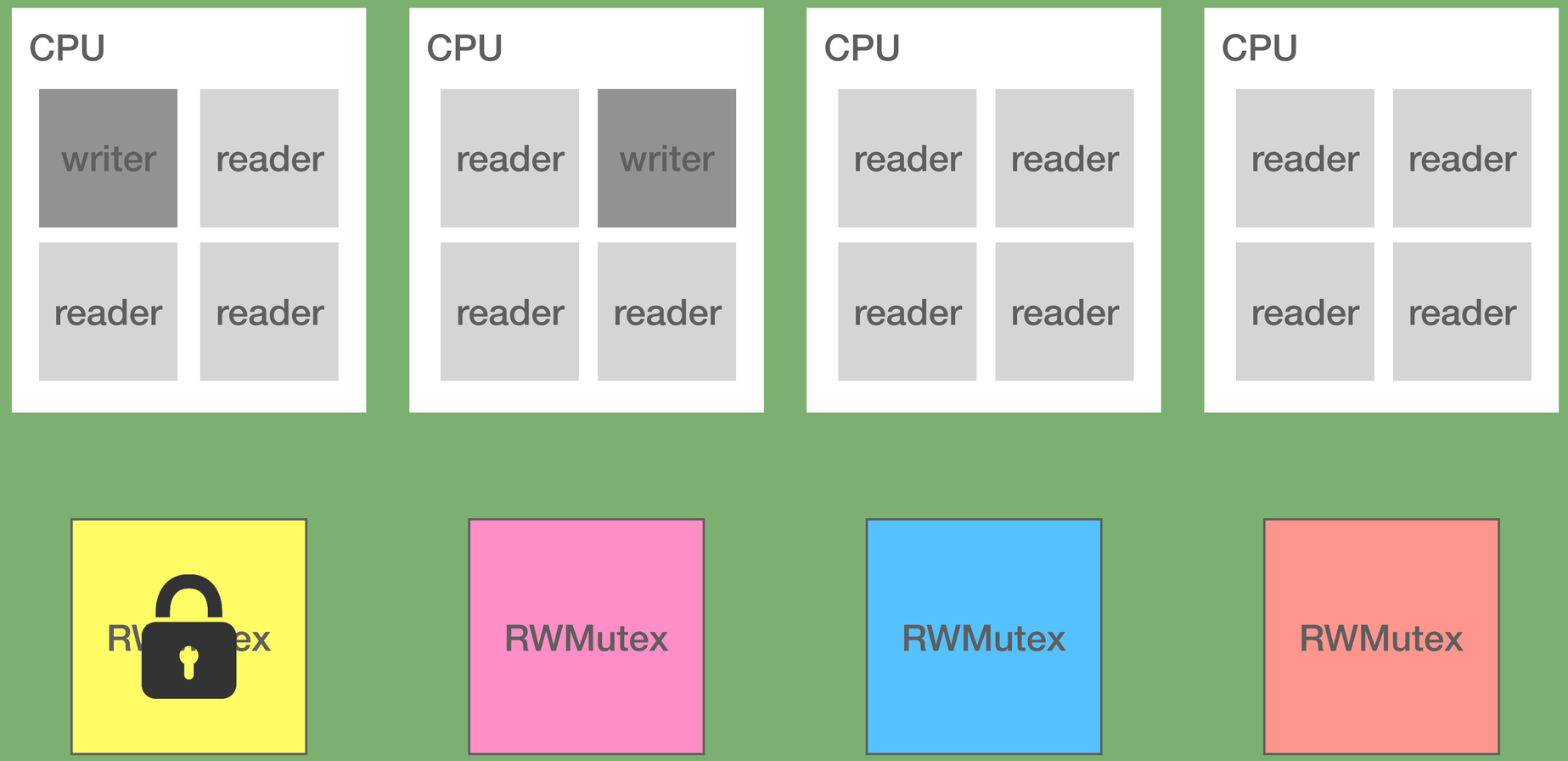

CPU
writer
CPU
writer
reader
reader
RWMutex
CPU
CPU
reader
reader
reader

@tyler_treat

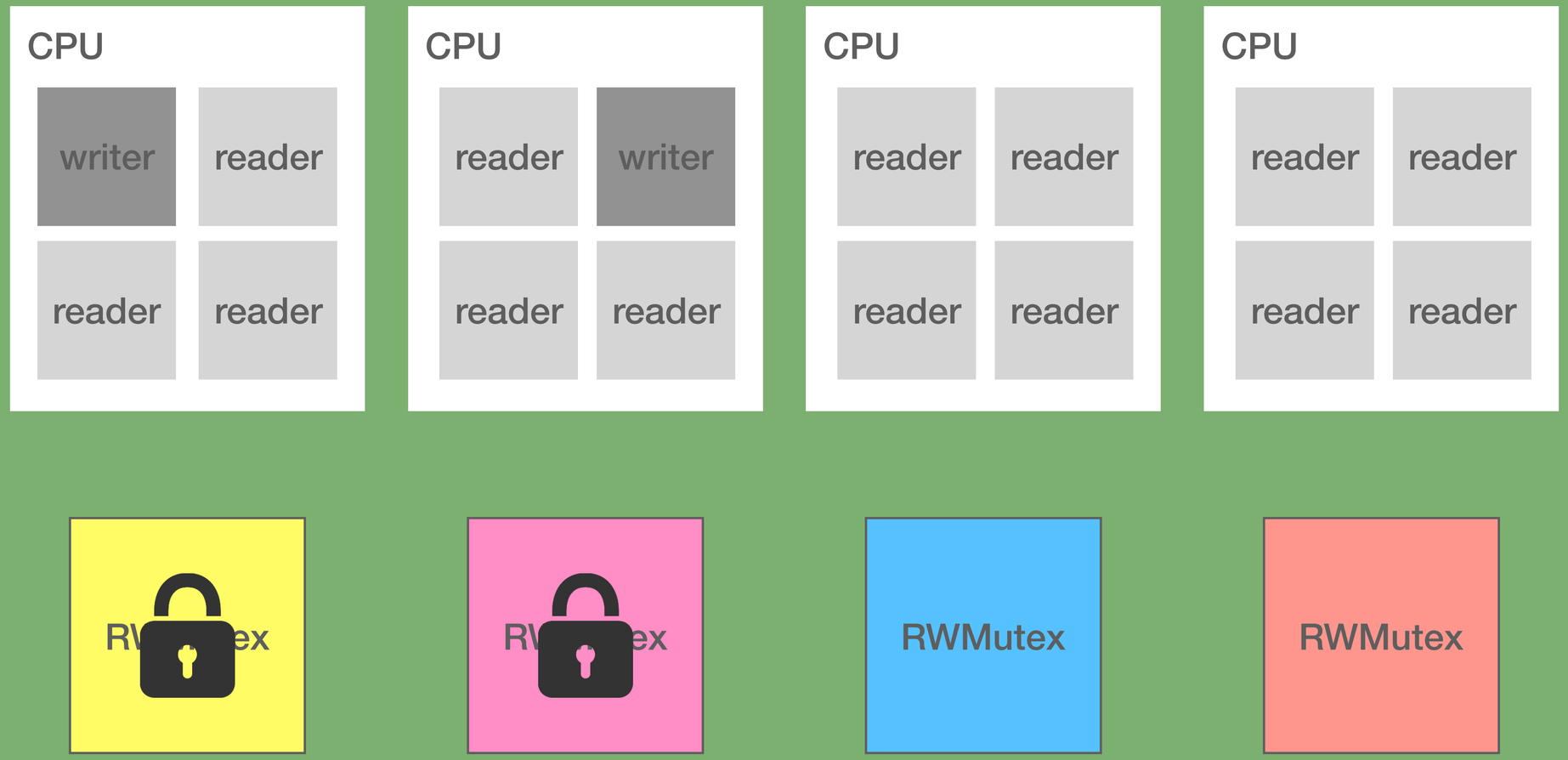

CPU
writer
reader
reader
reader
CPU
reader
writer
reader
reader
CPU
reader
reader
reader
reader
CPU
reader
reader
reader
reader
RWMutex
RWMutex
RWMutex
RWMutex


CPU
writer
CPU
writer
CPU
CPU
RWMutex
RWMutex
RWMutex
RWMutex

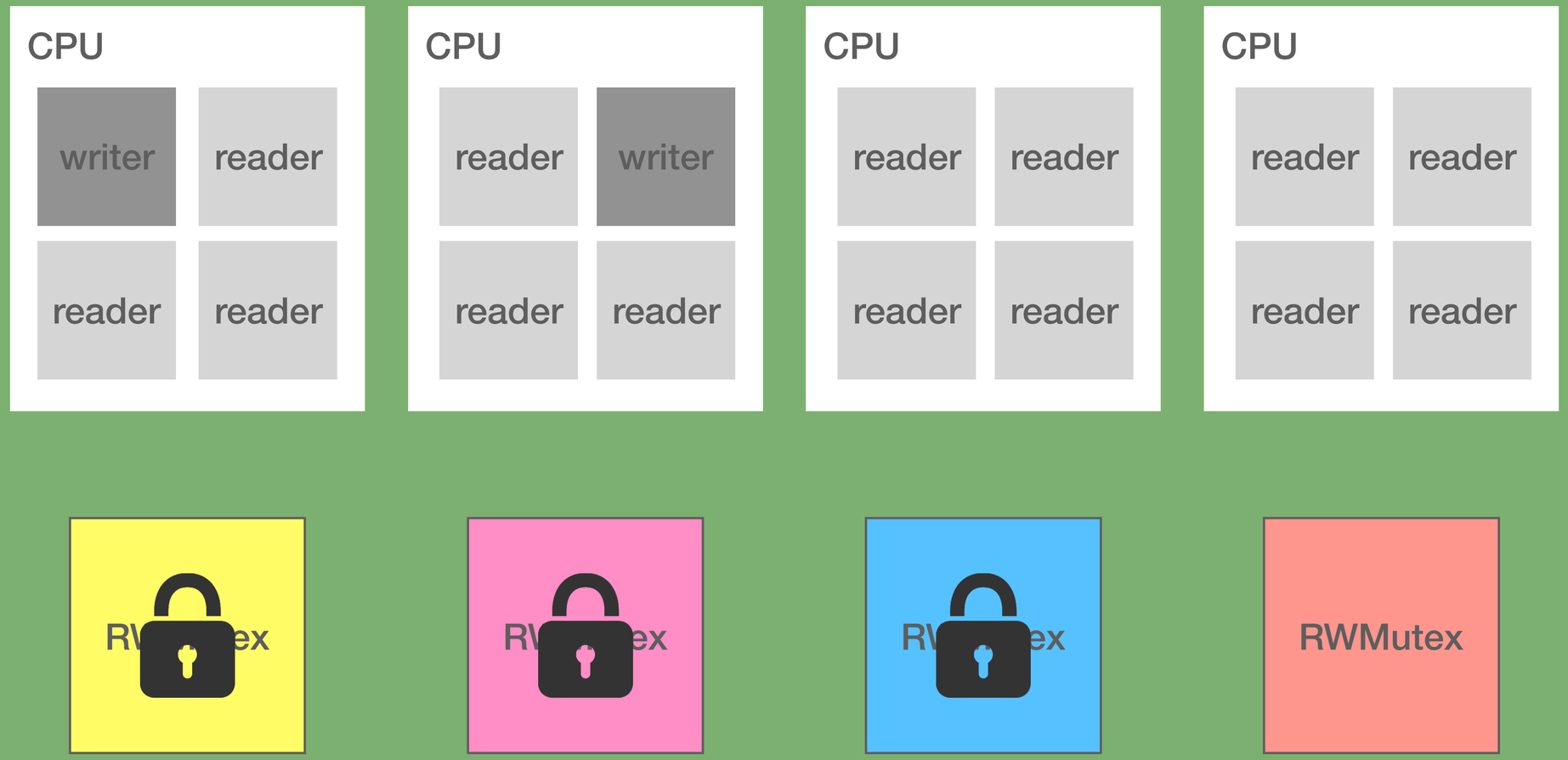
@tyler_treat


CPU
writer
reader
reader
reader
CPU
reader
writer
reader
reader
CPU
reader
reader
reader
reader
CPU
reader
reader
reader
reader
RWMutex
RWMutex
RWMutex

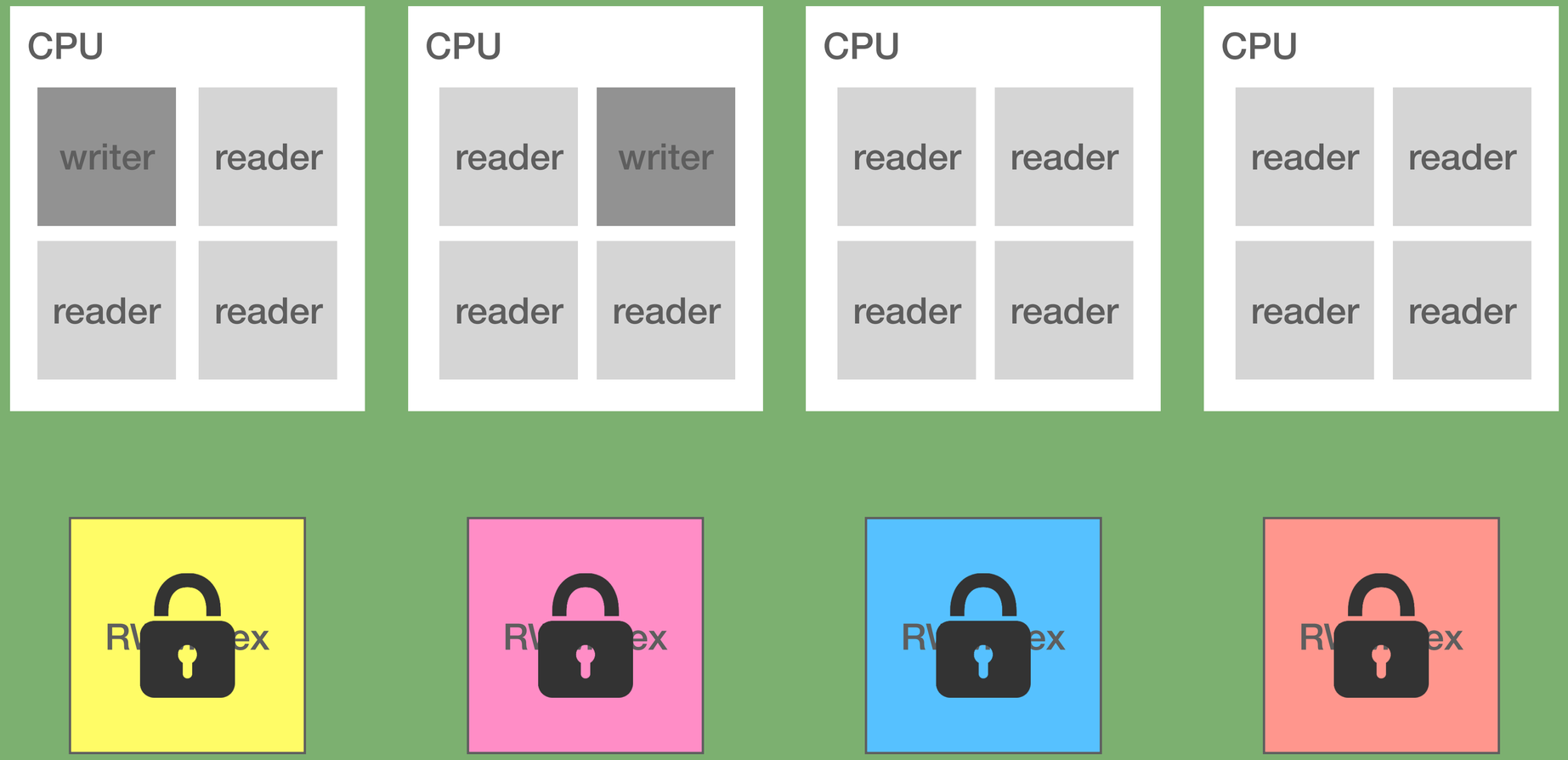
@tyler_treat


CPU
writer
reader
reader
reader
CPU
reader
writer
reader
reader
CPU
reader
reader
reader
reader
CPU
reader
reader
reader
reader
RWMutex
RWMutex

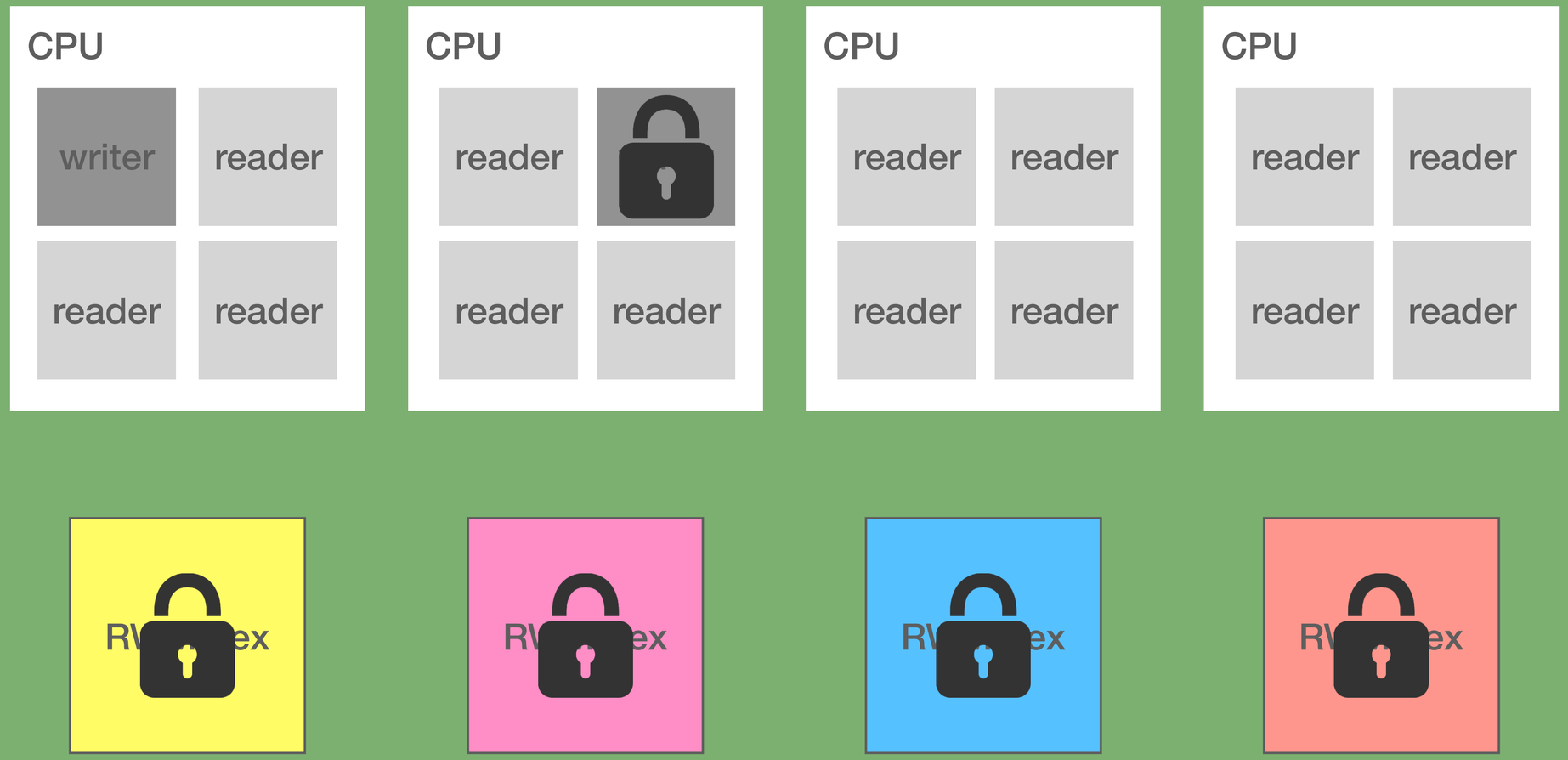
@tyler_treat


CPU
writer
reader
reader
reader
CPU
reader
writer
reader
reader
CPU
reader
reader
reader
reader
CPU
reader
reader
reader
reader
RWMutex

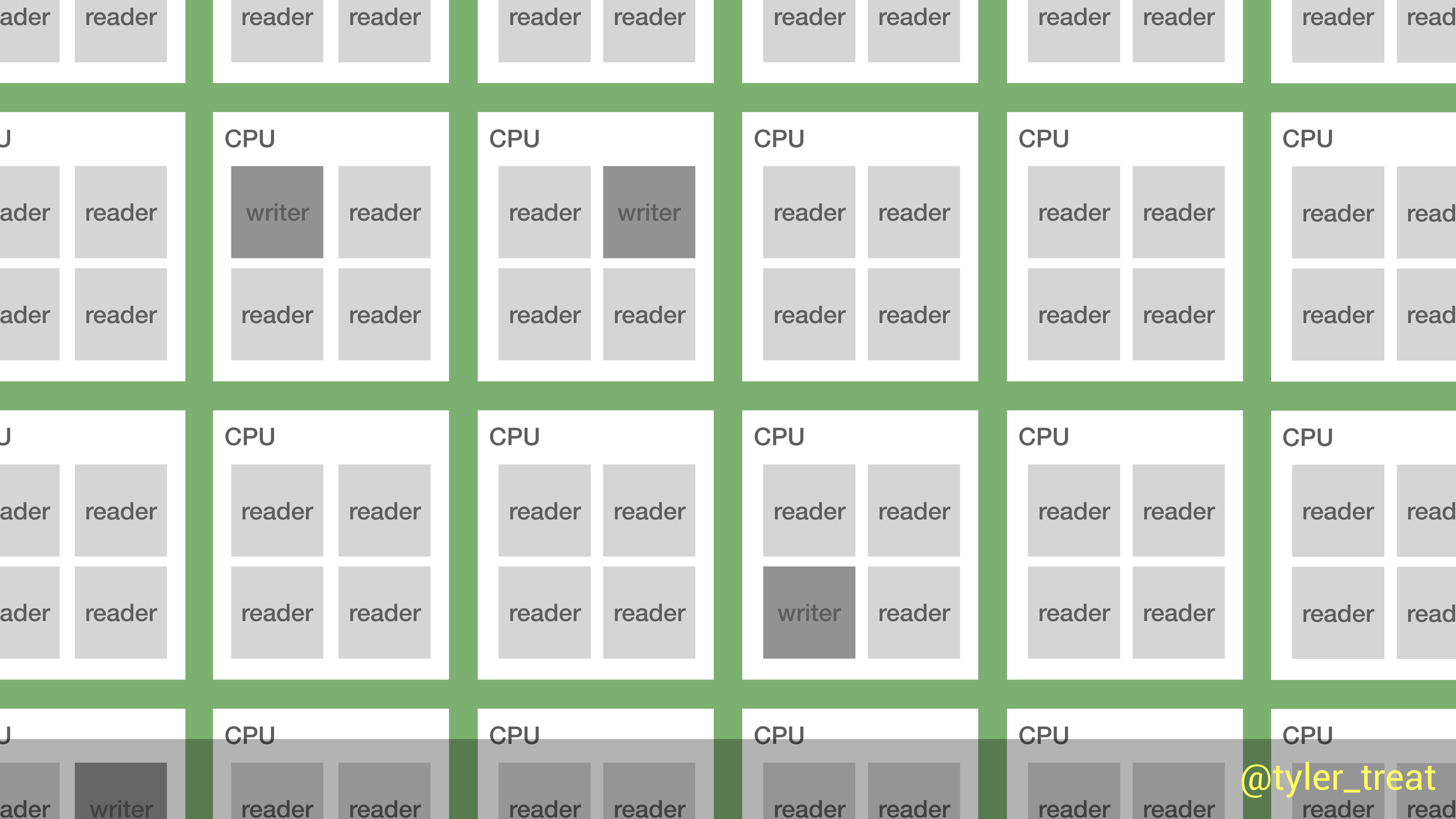
@tyler_treat


CPU
writer
reader
reader
reader
CPU
reader
writer
reader
reader
CPU
reader
reader
reader
reader
CPU
reader
reader
reader
reader

@tyler_treat


CPU
writer
reader
reader
reader
CPU
reader
reader
reader
CPU
reader
reader
reader
reader
CPU
reader
reader
reader
reader

@tyler_treat


CPU
writer
reader
@tyler_treat

golang / go

Watch 2,311 | Unstar 30,018 | Fork 4,030

Code | Issues 2,890 | Pull requests 0 | Wiki | Insights

proposal: sync: support for sharded values #18802

New issue

Open aclements opened this issue on Jan 26 · 33 comments

aclements commented on Jan 26 Member

Per-CPU sharded values are a useful and common way to reduce contention on shared write-mostly values. However, this technique is currently difficult or impossible to use in Go (though there have been attempts, such as **@jonhoo**'s https://github.com/jonhoo/drwmutex and **@bcmills**' https://go-review.googlesource.com/#/c/35676/).

We propose providing an API for creating and working with sharded values. Sharding would be encapsulated in a type, say `sync.Sharded`, that would have `Get() interface{}`, `Put(interface{})`, and `Do(func(interface{}))` methods. `Get` and `Put` would always have to be paired to make `Do` possible. (This is actually the same API that was proposed in #8281 (comment) and rejected, but perhaps we have a better understanding of the issues now.) This idea came out of off-and-on discussions between at least **@rsc**, **@hyangah**, **@RLH**, **@bcmills**, **@Sajmani**, and myself.

This is a counter-proposal to various proposals to expose the current thread/P ID as a way to implement sharded values (#8281, #18590). These have been turned down as exposing low-level implementation details, tying Go to an API that may be inappropriate or difficult to support in the future, being difficult to use correctly (since the ID may change at any time), being difficult to specify, and as being broadly susceptible to abuse.

Assignees

aclements

Labels

Proposal

Proposal-Hold

Projects

None yet

Milestone

Proposal

Notifications

Subscribe

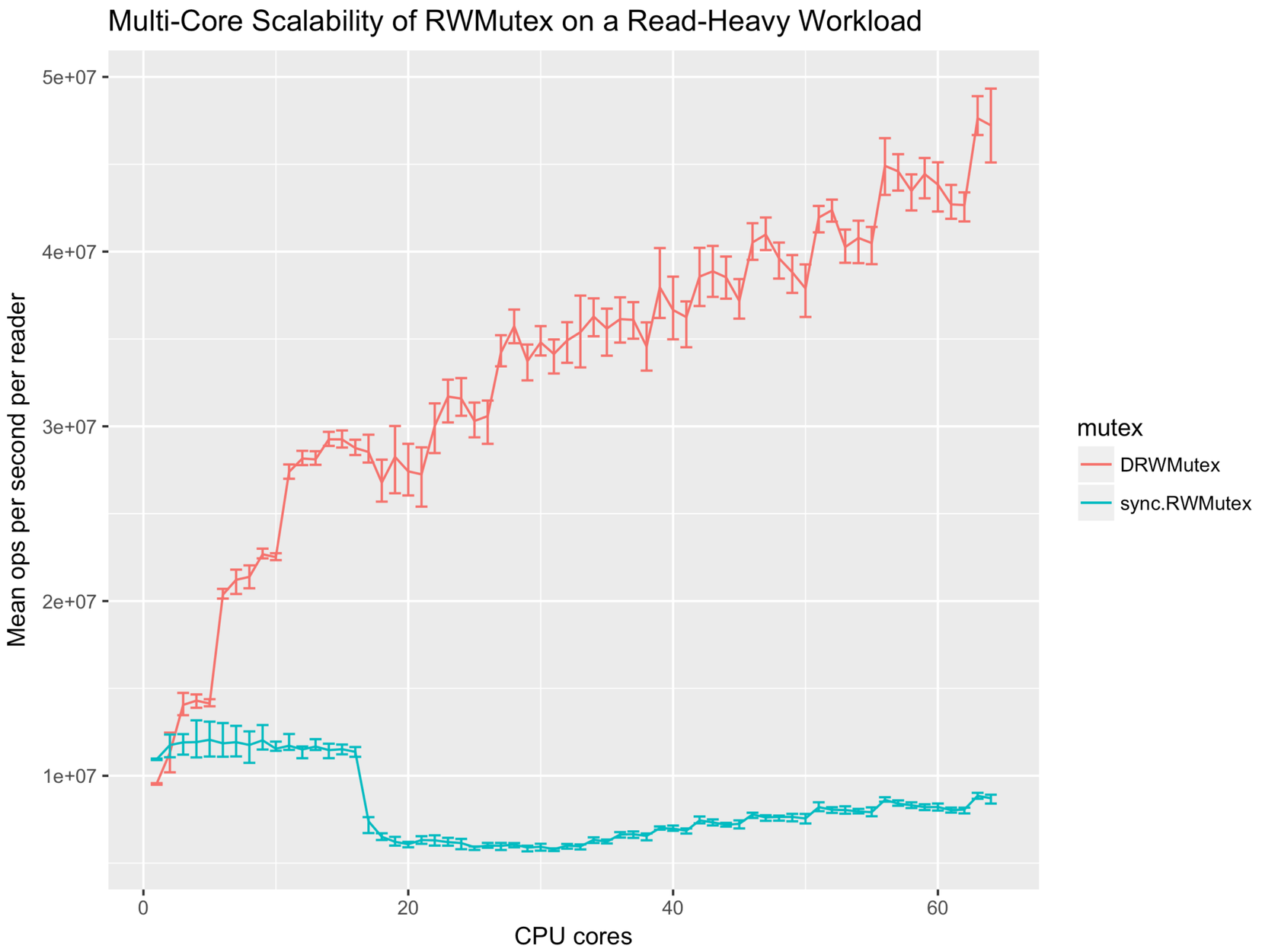
@tyler_treat

How to create CPU->RWMutex mapping?

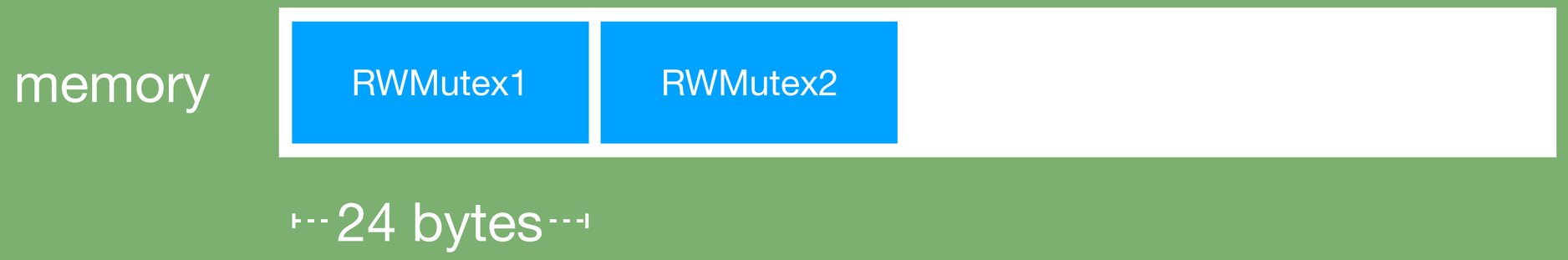
```
#include "textflag.h"

// func cpu() uint64
TEXT ·cpu(SB),NOSPLIT,$0-8
    MOVL    $0x01, AX // version information
    MOVL    $0x00, BX // any leaf will do
    MOVL    $0x00, CX // any subleaf will do

    // call CPUID
    BYTE $0x0f
    BYTE $0xa2

    SHRQ    $24, BX // logical cpu id is put in EBX[31-24]
    MOVQ    BX, ret+0(FP)
    RET

```

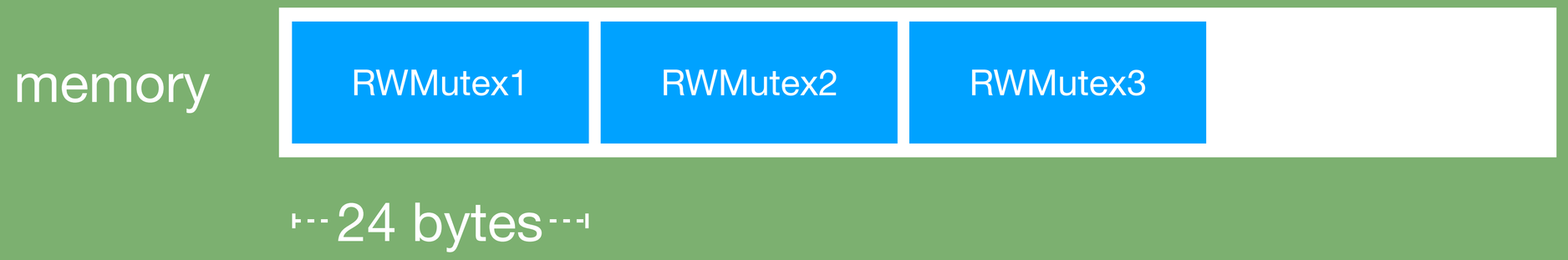
https://github.com/jonhoo/drwmutex/blob/master/cpu_amd64.s

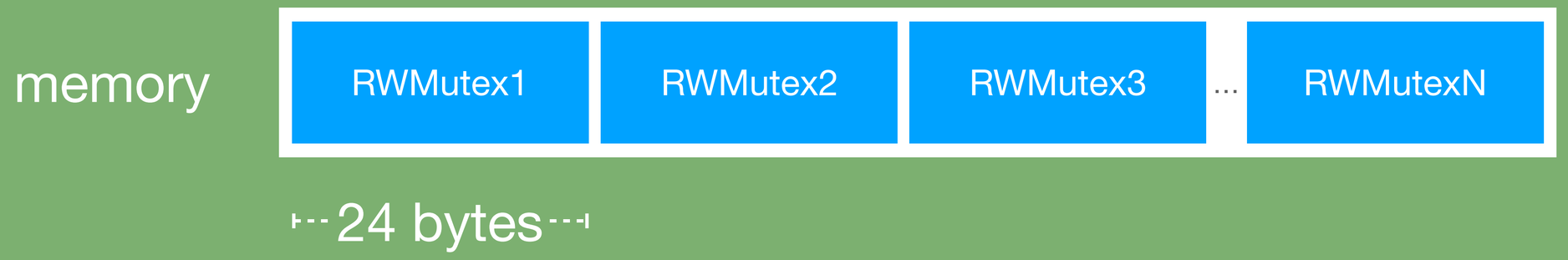
/proc/cpuinfo

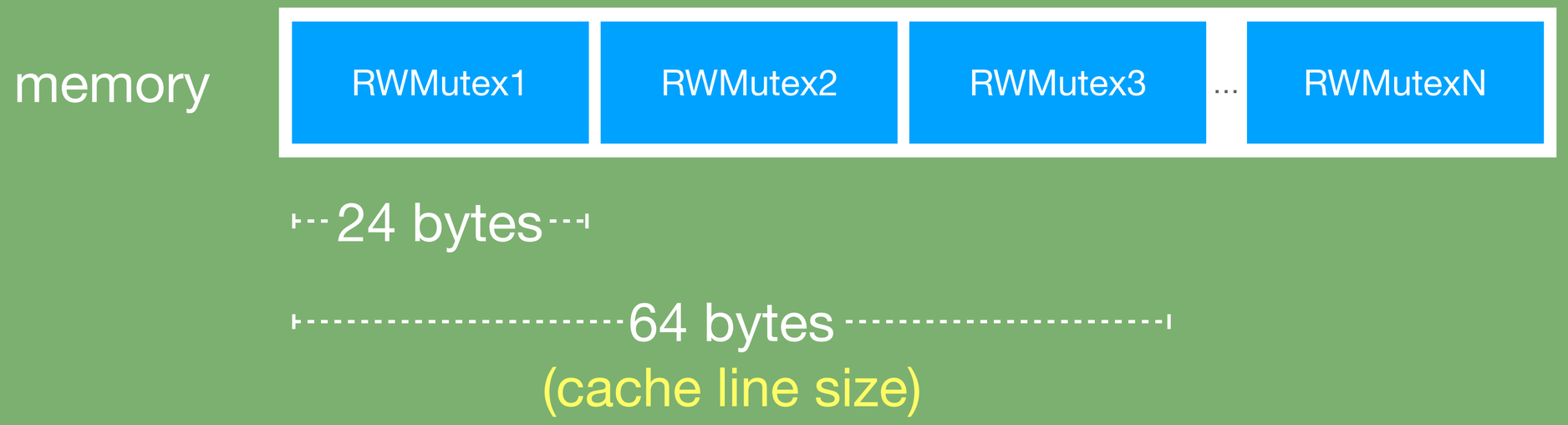

Multi-Core Scalability of RWMutex on a Read-Heavy Workload
Mean ops per second per reader
5e+07
4e+07
3e+07
2e+07
1e+07
0
20
40
60
CPU cores
mutex
DRWMutex
sync.RWMutex

memory
RWMutex1
24 bytes

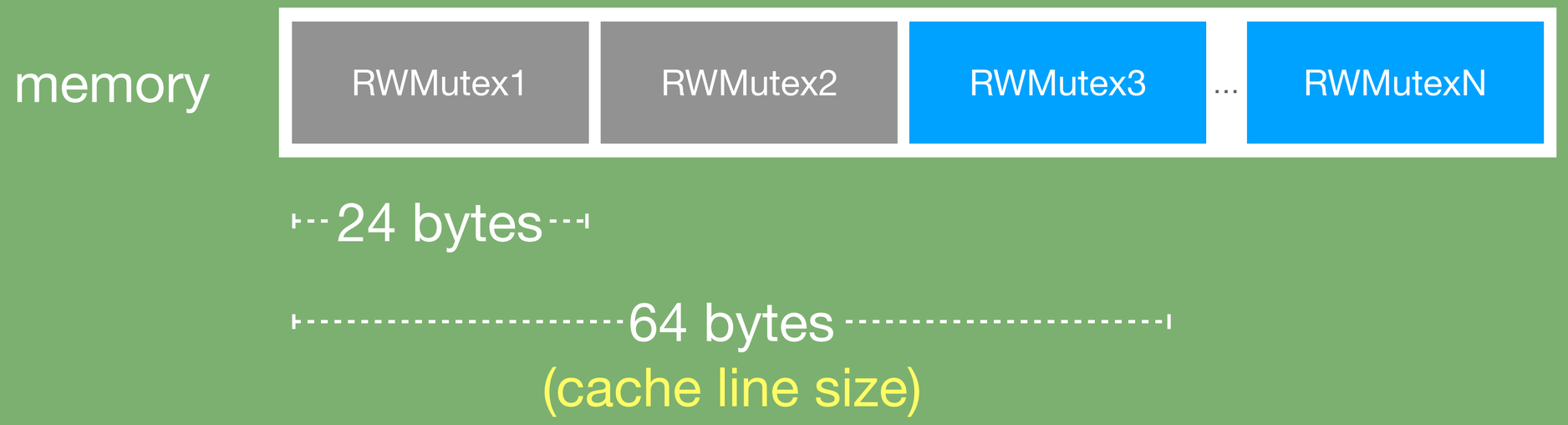

memory
RWMutex1
RWMutex2
24 bytes


memory
RWMutex1
RWMutex2
RWMutex3
24 bytes


memory
RWMutex1
RWMutex2
RWMutex3
...
RWMutexN
24 bytes


memory
RWMutex1
RWMutex2
RWMutex3
...
RWMutexN
24 bytes
64 bytes
(cache line size)

Cache rules everything around me



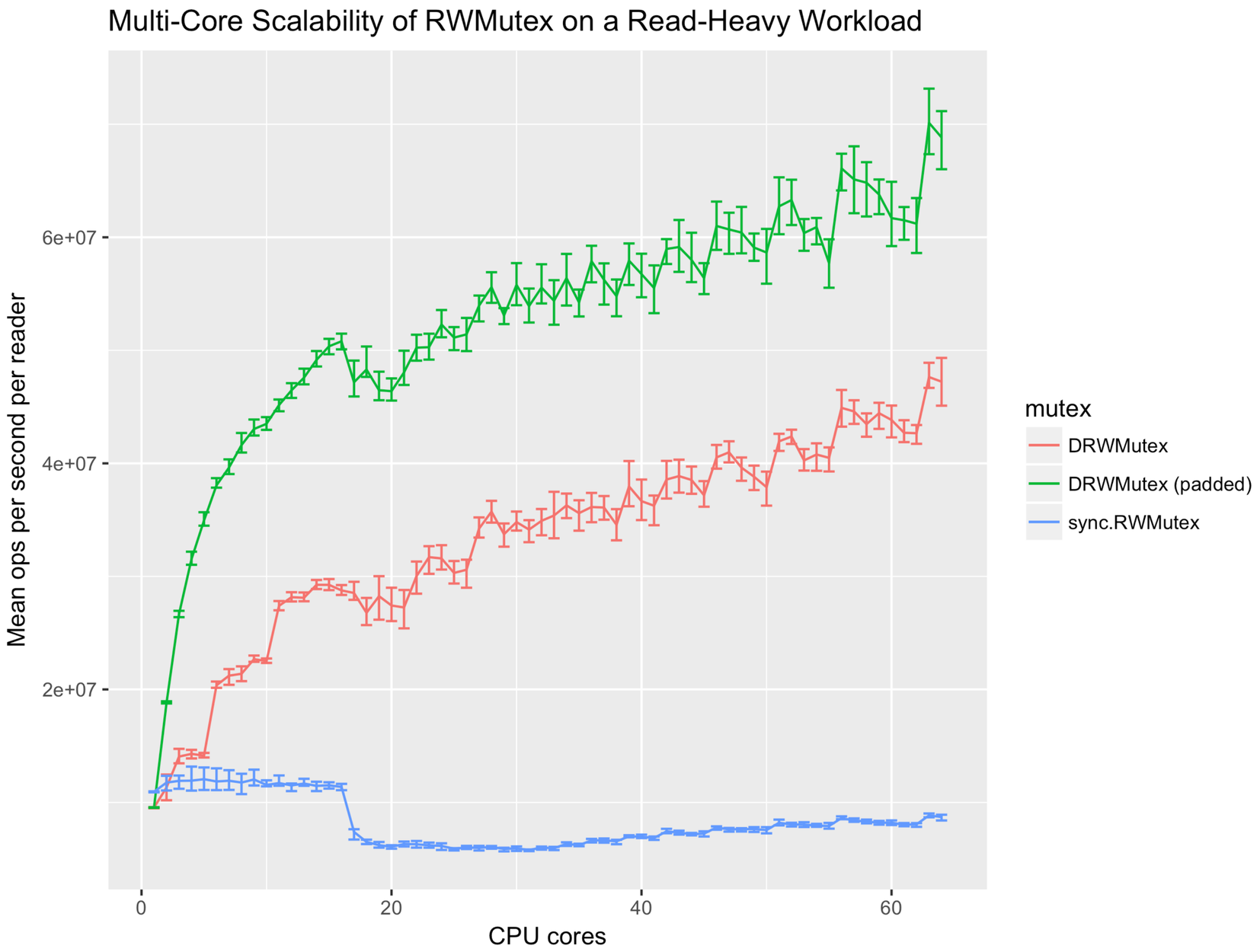
```go
type paddedRWMutex struct {
    _  [8]uint64 // Pad by cache-line size to prevent false sharing.
    mu sync.RWMutex
}

type DRWMutex []paddedRWMutex

// New returns a new, unlocked, distributed RWMutex.
func New() DRWMutex {
    return make(DRWMutex, len(cpus))
}
```

https://github.com/jonhoo/drwmutex/blob/master/drwmutex.go

```go
25 type paddedRWMutex struct {
26     _  [8]uint64 // Pad by cache-line size to prevent false sharing.
27     mu sync.RWMutex
28 }
29
30 type DRWMutex []paddedRWMutex
31
32 // New returns a new, unlocked, distributed RWMutex.
33 func New() DRWMutex {
34     return make(DRWMutex, len(cpus))
35 }
```

https://github.com/jonhoo/drwmutex/blob/master/drwmutex.go

Cache rules everything around me


Multi-Core Scalability of RWMutex on a Read-Heavy Workload
Mean ops per second per reader
6e+07
4e+07
2e+07
0
20
40
60
CPU cores
mutex
DRWMutex
DRWMutex (padded)
sync.RWMutex

type Map

Map is a concurrent map with amortized-constant-time loads, stores, and deletes. It is safe for multiple goroutines to call a Map's methods concurrently.

The zero Map is valid and empty.

A Map must not be copied after first use.

```
type Map struct {
    // contains filtered or unexported fields
}
```

func (*Map) Delete

```
func (m *Map) Delete(key interface{})
```

Delete deletes the value for a key.

func (*Map) Load

```
func (m *Map) Load(key interface{}) (value interface{}, ok bool)
```

Load returns the value stored in the map for a key, or nil if no value is present. The ok result indicates whether value was found in the map.

func (*Map) LoadOrStore

```
func (m *Map) LoadOrStore(key, value interface{}) (actual interface{}, loaded bool)
```

@tyler_treat

type Map

Map is a concurrent map with amortized-constant-time loads, stores, and deletes. It is safe for multiple goroutines to call a Map's methods concurrently.

The zero Map is valid and empty.

A Map must not be copied after first use.

```
type Map struct {
    // contains filtered or unexported fields
}
```

func (*Map) Delete

```
func (m *Map) Delete(key interface{})
```

Delete deletes the value for a key.

func (*Map) Load

```
func (m *Map) Load(key interface{}) (value interface{}, ok bool)
```

Load returns the value stored in the map for a key, or nil if no value is present. The ok result indicates whether value was found in the map.

func (*Map) LoadOrStore

```
func (m *Map) LoadOrStore(key, value interface{}) (actual interface{}, loaded bool)
```

@tyler_treat

Go makes concurrency easy enough to be dangerous.

Conclusions

1

The standard library provides general solutions (and they're generally what you should use).

2

Seemingly small, idiomatic decisions can have profound performance implications.

3

The Go toolchain has lots of tools for analyzing your code—learn them.

4

Go’s compiler and runtime continue to improve.

5

Performance profile can change dramatically between releases.

6

Relying on assumptions can be fatal.

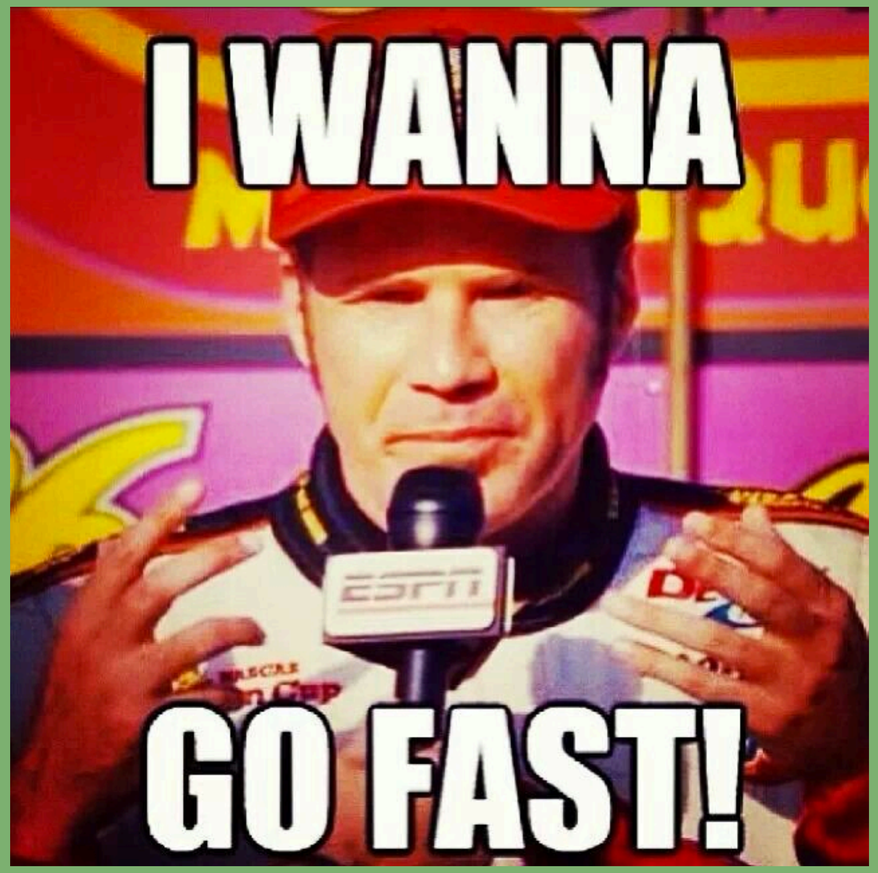
7

Code is marginal,
architecture is material.

8

Peeking behind the curtains can pay dividends.

9

Above all, optimize for the right trade-off.

Thanks!