Creative Fabrica

Smarter Locks

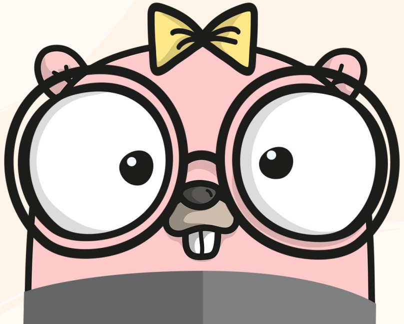
A deep dive on the Runtime Mutex Spin Optimisation

Raiza Claudino

Raizaclaudino

E-Mail
raiza@creativefabrica.com

Creative Fabrica
Westerstraat 187
1015 MA, Amsterdam
the Netherlands



Smarter Locks

A deep dive on the Runtime Mutex Spin Optimisation

Smarter Locks

A deep dive on the Runtime Mutex Spin Optimisation

Overview:

- Motivation
- Behaviour before the optimization
- go1.24 runtime mutex optimization

Smarter Locks

A deep dive on the Runtime Mutex Spin Optimisation

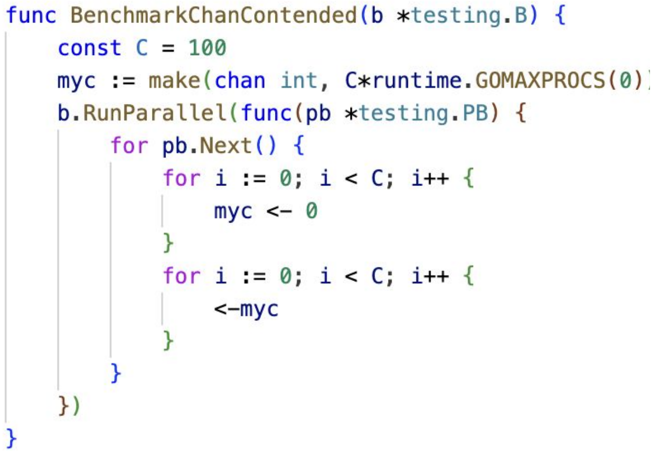
> Several performance improvements to the runtime have decreased CPU overheads by 2–3% on average across a suite of representative benchmarks.

https://tip.golang.org/doc/go1.24

- What was the mutex behaviour before?
- What was proposed and how does it improve CPU overhead?
- What benchmarks?

Smarter Locks

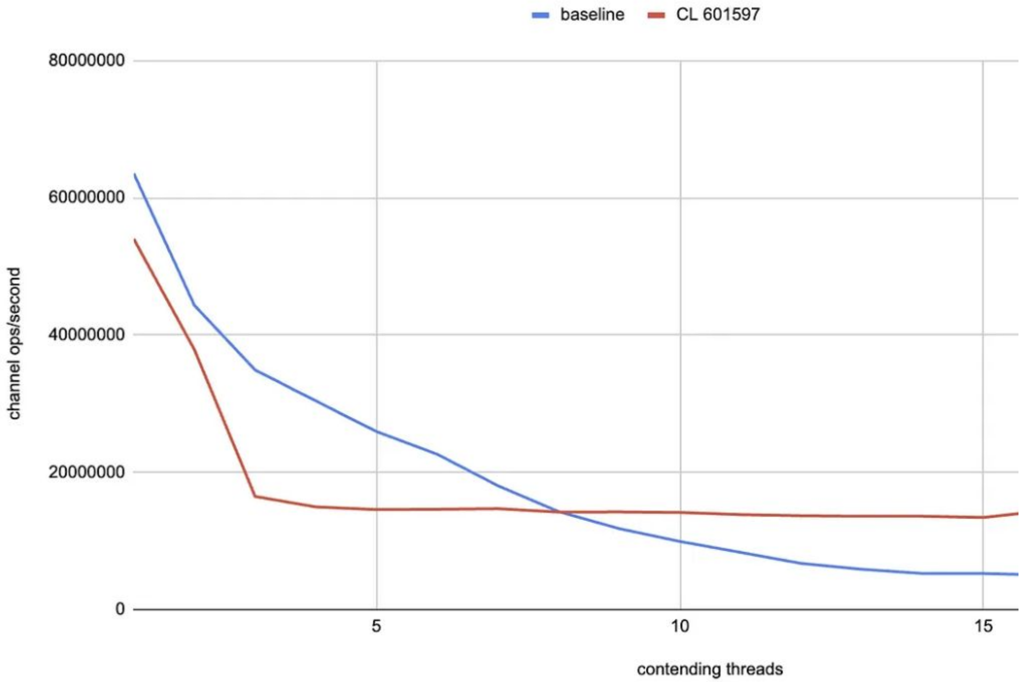
A deep dive on the Runtime Mutex Spin Optimisation



```go
func BenchmarkChanContended(b *testing.B) {
	const C = 100
	myc := make(chan int, C*runtime.GOMAXPROCS(0))
	b.RunParallel(func(pb *testing.PB) {
		for pb.Next() {
			for i := 0; i < C; i++ {
				myc <- 0
			}
			for i := 0; i < C; i++ {
				<-myc
			}
		}
	})
}
```

CL 601597 gives horizontal scaling

Versus baseline, where each additional thread makes the lock even slower



go test runtime -test.run='^$' -test.bench=ChanContended -test.cpu="$(seq 1 20 | tr '\n' ',')" -test.count=10

src/runtime/chan_test.go

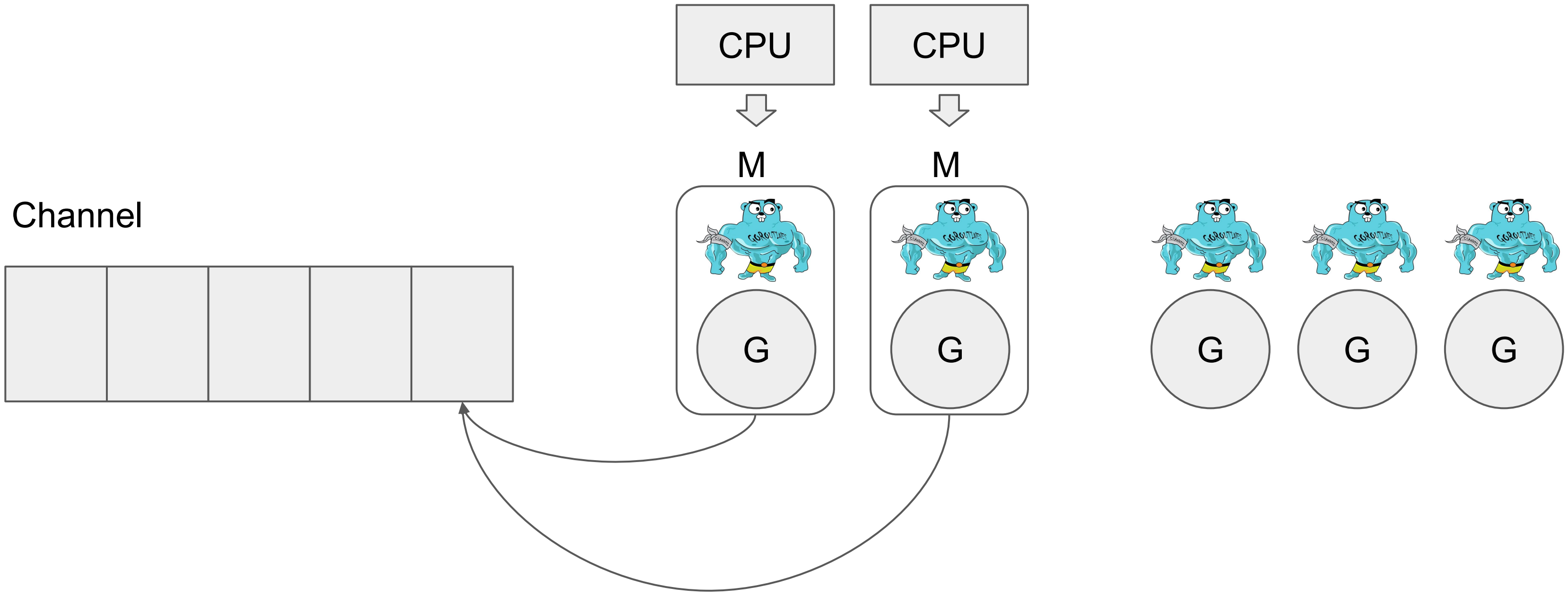
Smarter Locks

A deep dive on the Runtime Mutex Spin Optimisation

Go concurrency primitives

Smarter Locks

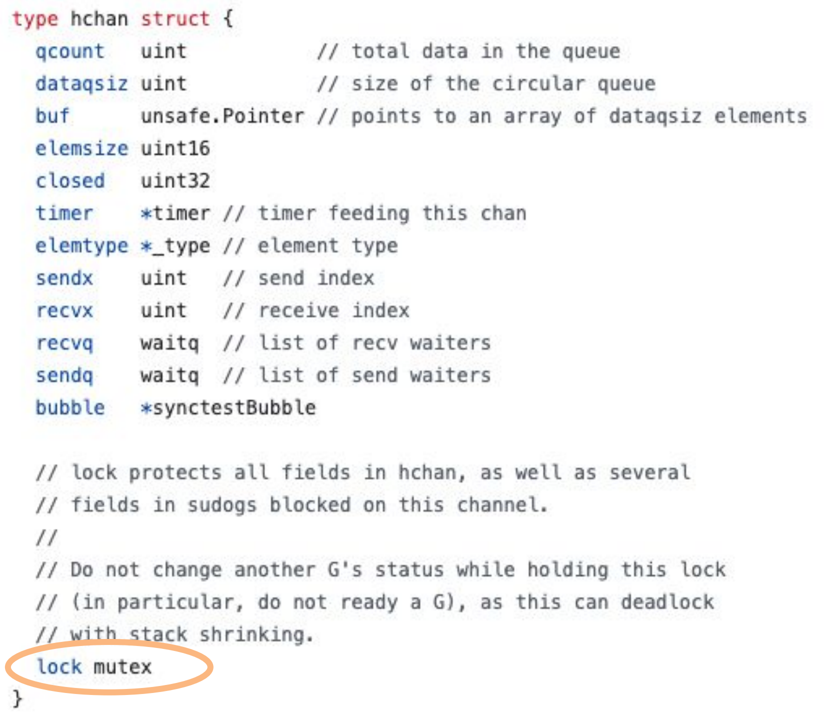
A deep dive on the Runtime Mutex Spin Optimisation



M: OS Threads
G: Goroutines

Smarter Locks

A deep dive on the Runtime Mutex Spin Optimisation



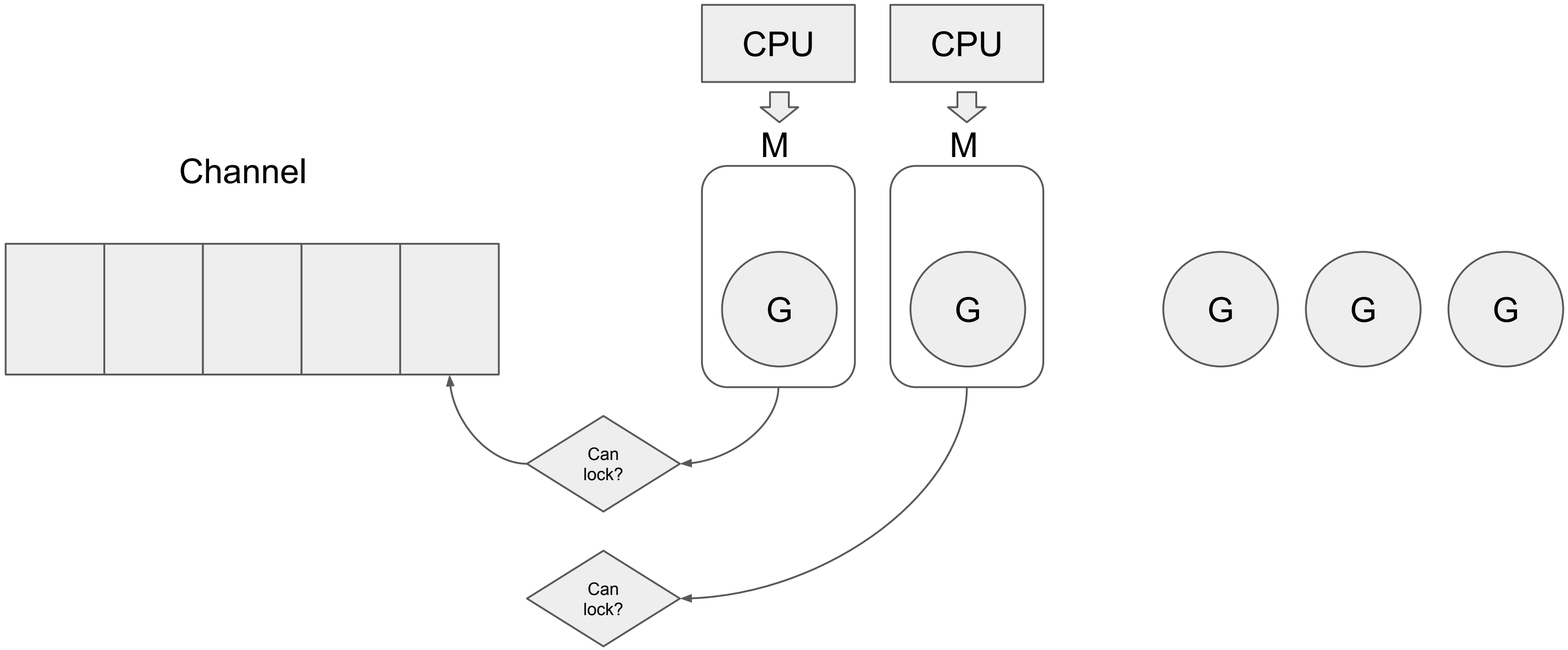
```go
type hchan struct {
  qcount   uint           // total data in the queue
  dataqsiz uint           // size of the circular queue
  buf      unsafe.Pointer // points to an array of dataqsiz elements
  elemsize uint16
  closed   uint32
  timer    *timer // timer feeding this chan
  elemtype *_type // element type
  sendx    uint   // send index
  recvx    uint   // receive index
  recvq    waitq  // list of recv waiters
  sendq    waitq  // list of send waiters
  bubble   *synctestBubble

  // lock protects all fields in hchan, as well as several
  // fields in sudogs blocked on this channel.
  //
  // Do not change another G's status while holding this lock
  // (in particular, do not ready a G), as this can deadlock
  // with stack shrinking.
  lock mutex
}
```

- This is where the optimization is 🎉
- What was happening before?

src/runtime/chan.go

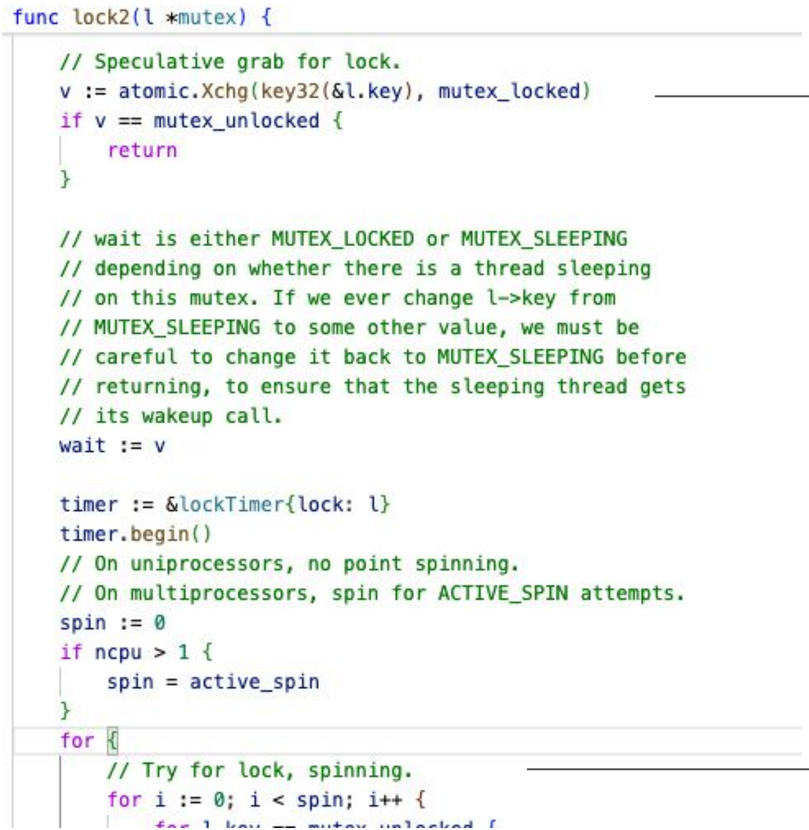
Smarter Locks

A deep dive on the Runtime Mutex ~~Spin Optimisation~~ on go 1.23



Smarter Locks

A deep dive on the Runtime Mutex ~~Spin Optimisation~~ on go 1.23



```go
func lock2(l *mutex) {
    // Speculative grab for lock.
    v := atomic.Xchg(key32(&l.key), mutex_locked)
    if v == mutex_unlocked {
        return
    }

    // wait is either MUTEX_LOCKED or MUTEX_SLEEPING
    // depending on whether there is a thread sleeping
    // on this mutex. If we ever change l->key from
    // MUTEX_SLEEPING to some other value, we must be
    // careful to change it back to MUTEX_SLEEPING before
    // returning, to ensure that the sleeping thread gets
    // its wakeup call.
    wait := v

    timer := &lockTimer{lock: l}
    timer.begin()
    // On uniprocessors, no point spinning.
    // On multiprocessors, spin for ACTIVE_SPIN attempts.
    spin := 0
    if ncpu > 1 {
        spin = active_spin
    }
    for {
        // Try for lock, spinning.
        for i := 0; i < spin; i++ {
            for l.key == mutex_unlocked {
```

OS Atomic call:

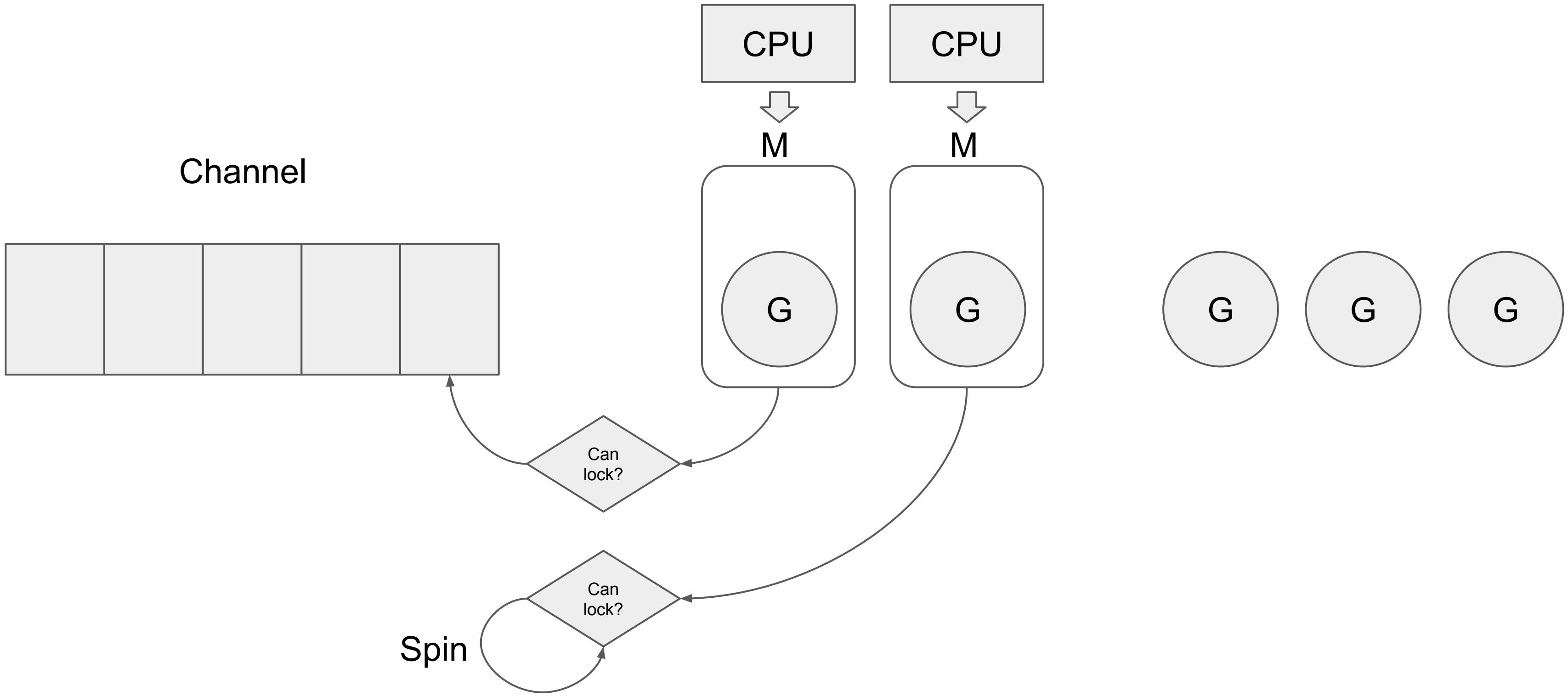
- Check and Set

If lock was not acquired "spin" for a while

src/runtime/lock_futex.go

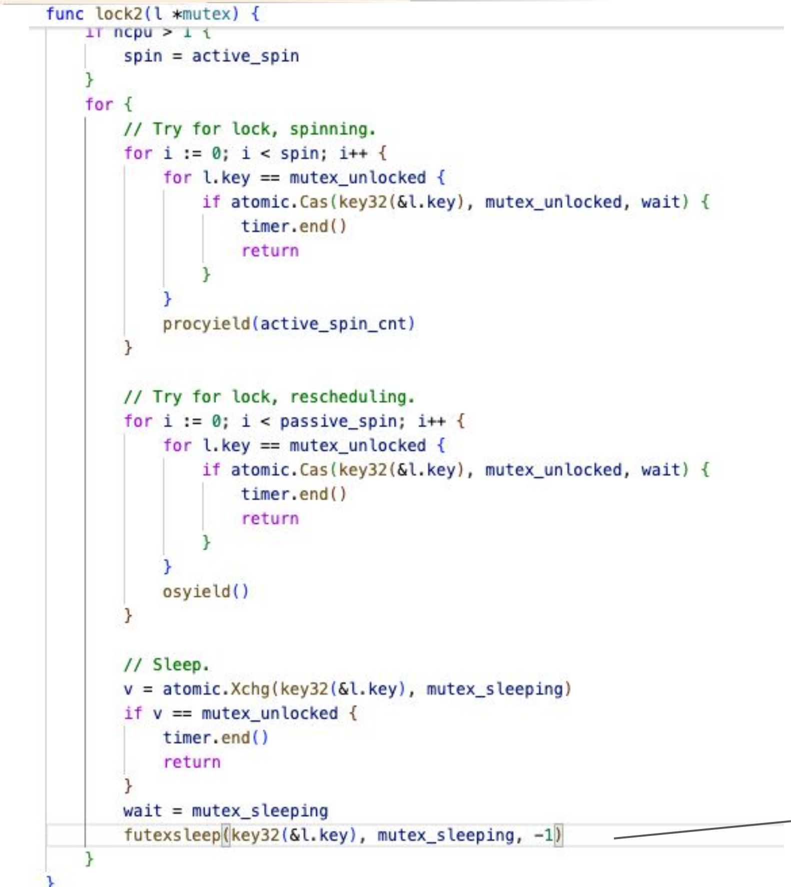
Smarter Locks

A deep dive on the Runtime Mutex ~~Spin Optimisation~~ on go 1.23



Smarter Locks

A deep dive on the Runtime Mutex ~~Spin Optimisation~~ on go 1.23



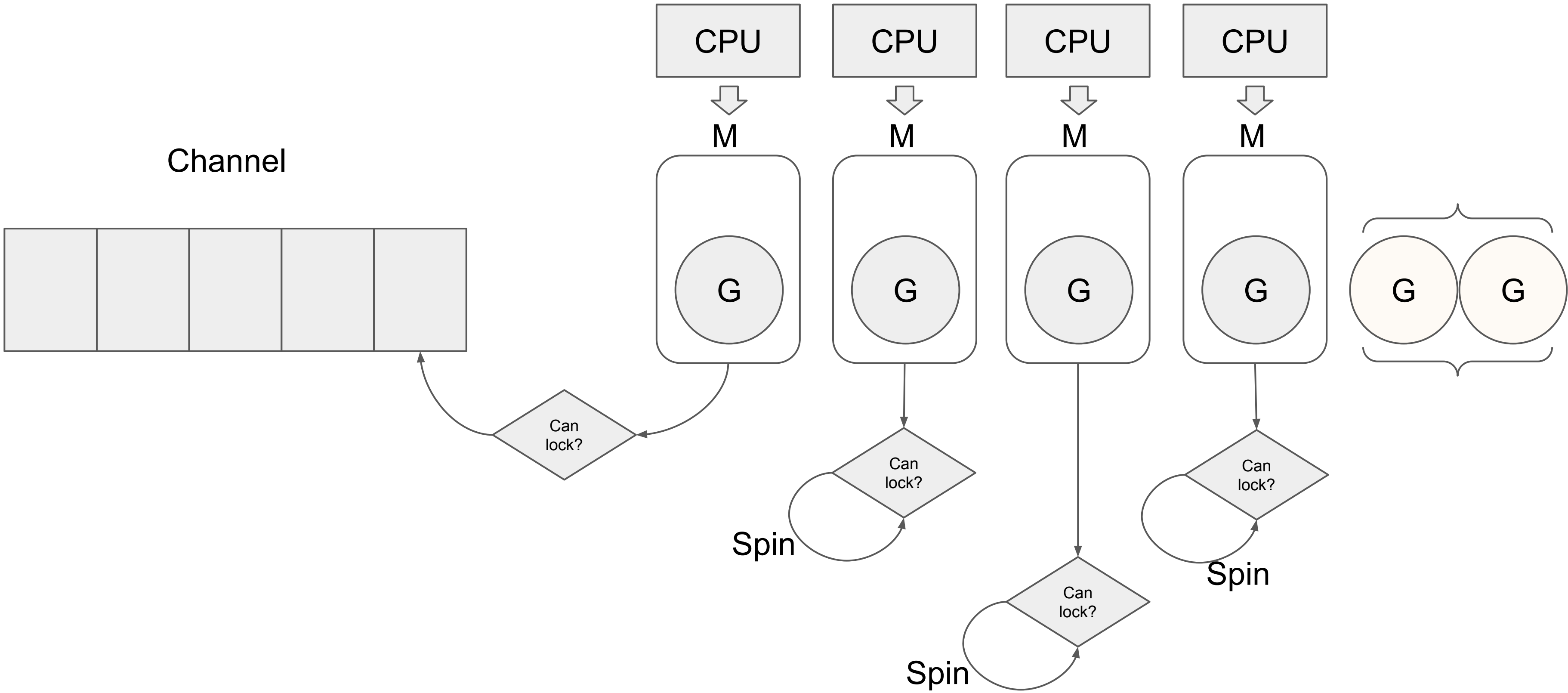
```go
func lock2(l *mutex) {
    if ncpu > 1 {
        spin = active_spin
    }
    for {
        // Try for lock, spinning.
        for i := 0; i < spin; i++ {
            for l.key == mutex_unlocked {
                if atomic.Cas(key32(&l.key), mutex_unlocked, wait) {
                    timer.end()
                    return
                }
            }
            procyield(active_spin_cnt)
        }

        // Try for lock, rescheduling.
        for i := 0; i < passive_spin; i++ {
            for l.key == mutex_unlocked {
                if atomic.Cas(key32(&l.key), mutex_unlocked, wait) {
                    timer.end()
                    return
                }
            }
            osyield()
        }

        // Sleep.
        v = atomic.Xchg(key32(&l.key), mutex_sleeping)
        if v == mutex_unlocked {
            timer.end()
            return
        }
        wait = mutex_sleeping
        futexsleep(key32(&l.key), mutex_sleeping, -1)
    }
}
```

OS Call to sleep M

src/runtime/lock_futex.go

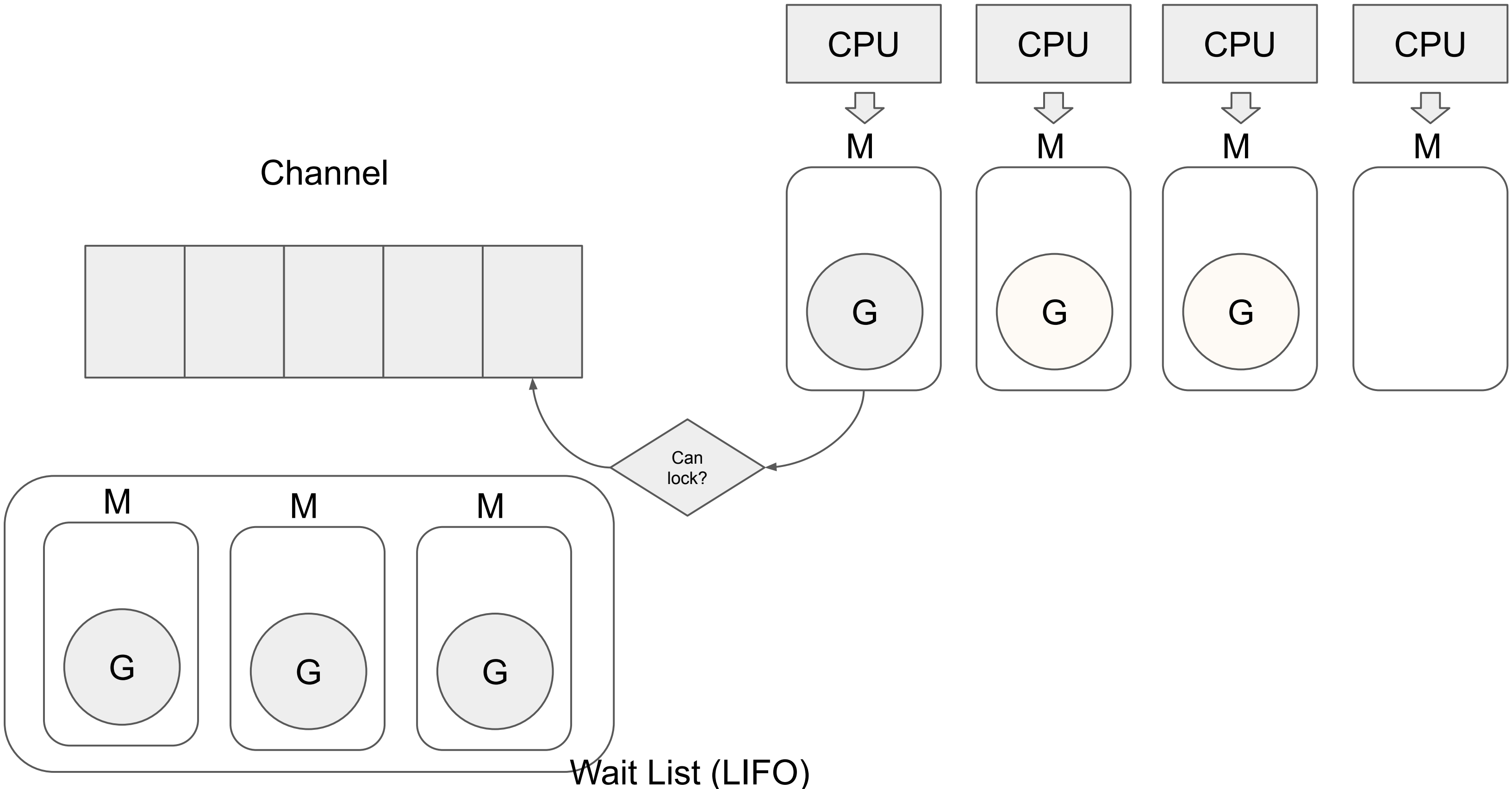
Smarter Locks

A deep dive on the Runtime Mutex ~~Spin Optimisation~~ on go 1.23



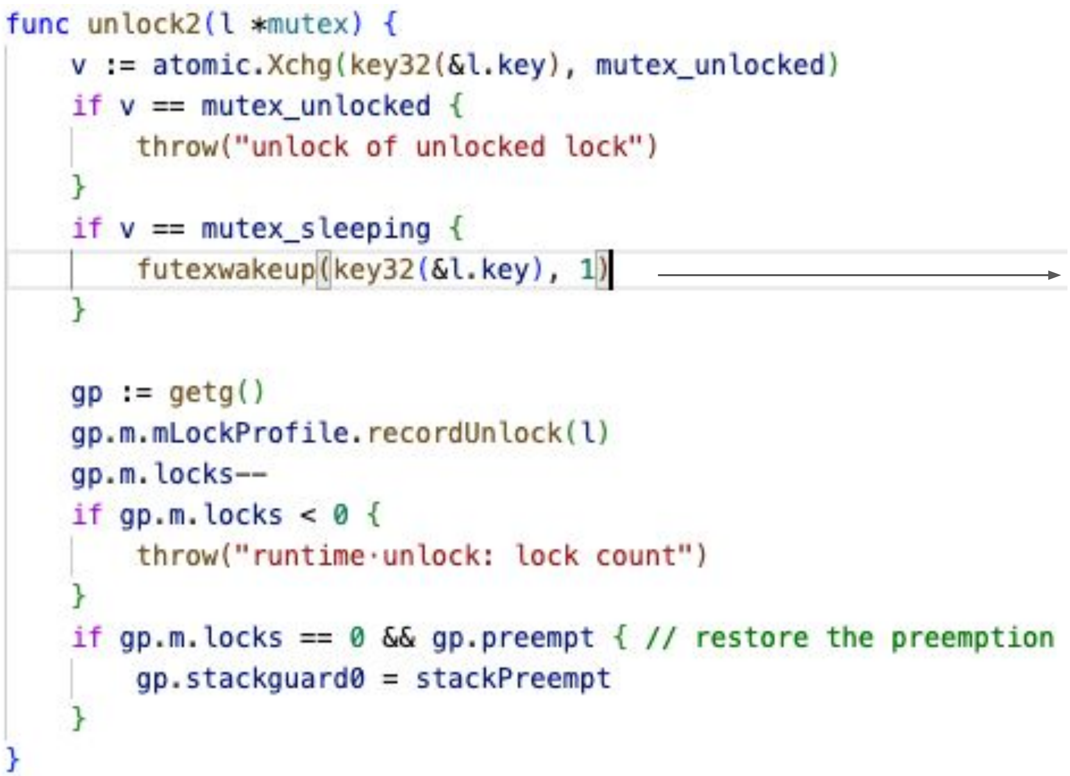
Smarter Locks

A deep dive on the Runtime Mutex ~~Spin Optimisation~~ on go 1.23



Smarter Locks

A deep dive on the Runtime Mutex ~~Spin Optimisation~~ on go 1.23



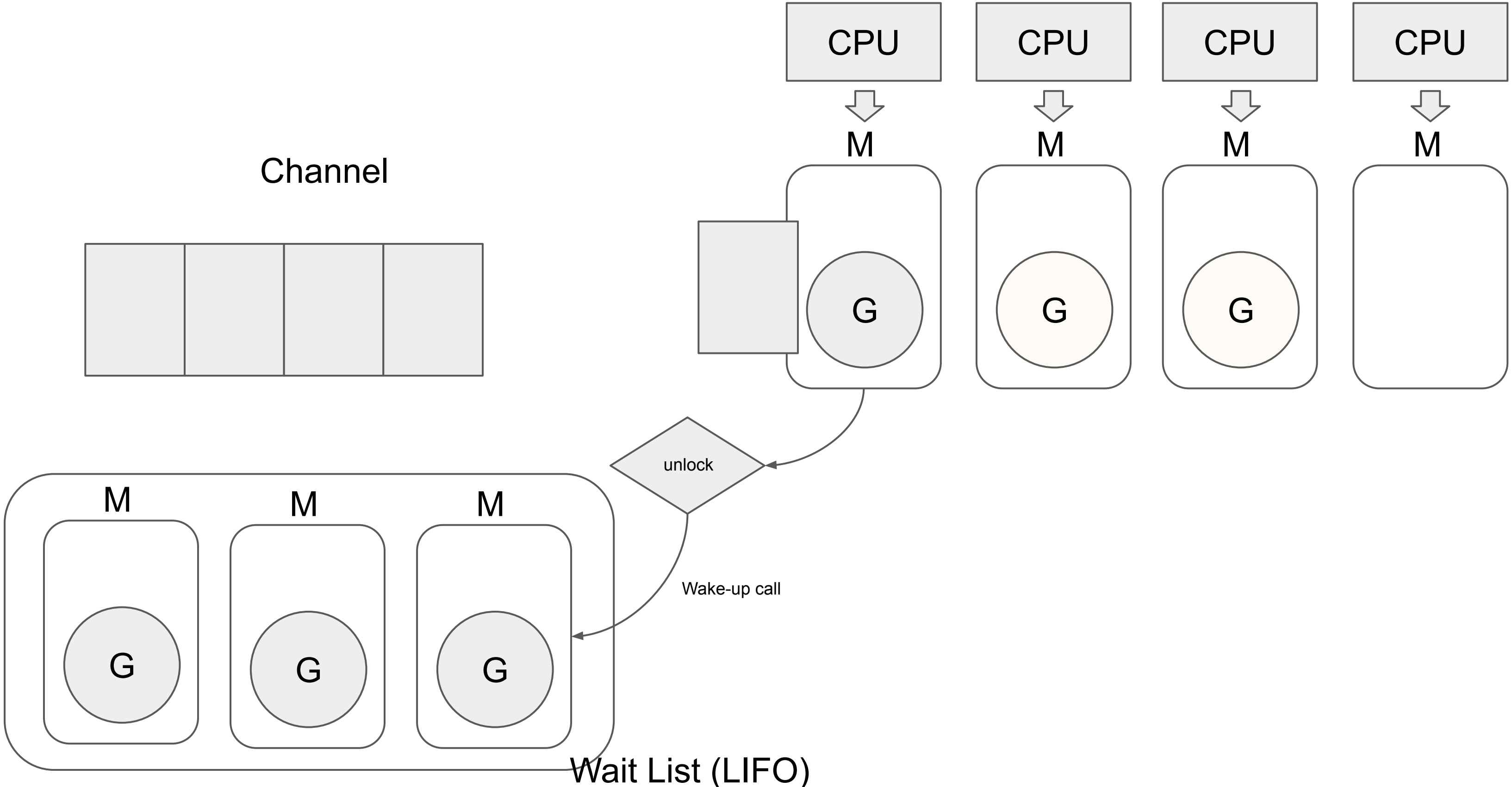
```go
func unlock2(l *mutex) {
	v := atomic.Xchg(key32(&l.key), mutex_unlocked)
	if v == mutex_unlocked {
		throw("unlock of unlocked lock")
	}
	if v == mutex_sleeping {
		futexwakeup(key32(&l.key), 1)
	}

	gp := getg()
	gp.m.mLockProfile.recordUnlock(l)
	gp.m.locks--
	if gp.m.locks < 0 {
		throw("runtime·unlock: lock count")
	}
	if gp.m.locks == 0 && gp.preempt { // restore the preemption
		gp.stackguard0 = stackPreempt
	}
}
```

OS Call to wake the thread up

src/runtime/lock_futex.go

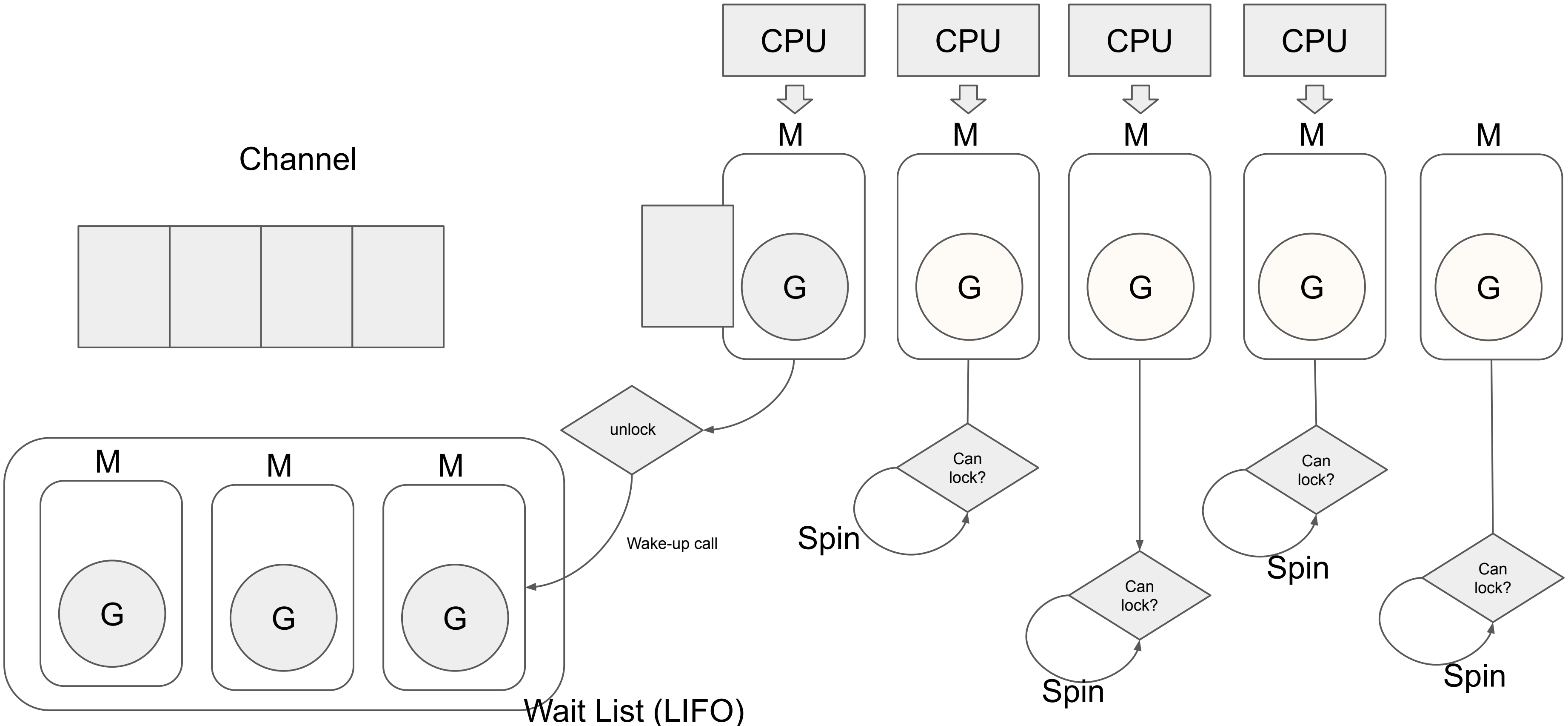
Smarter Locks

A deep dive on the Runtime Mutex ~~Spin Optimisation~~ on go 1.23



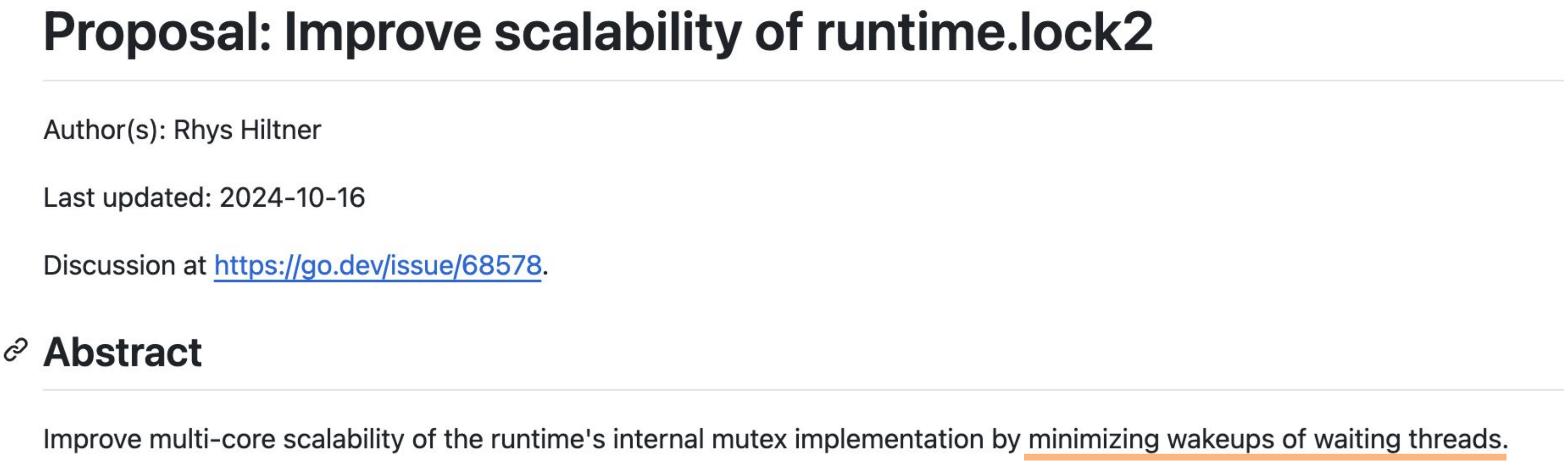
Smarter Locks

A deep dive on the Runtime Mutex ~~Spin Optimisation~~ on go 1.23



Smarter Locks

A deep dive on the Runtime Mutex Spin Optimisation



Proposal: Improve scalability of runtime.lock2

Author(s): Rhys Hiltner

Last updated: 2024-10-16

Discussion at https://go.dev/issue/68578.

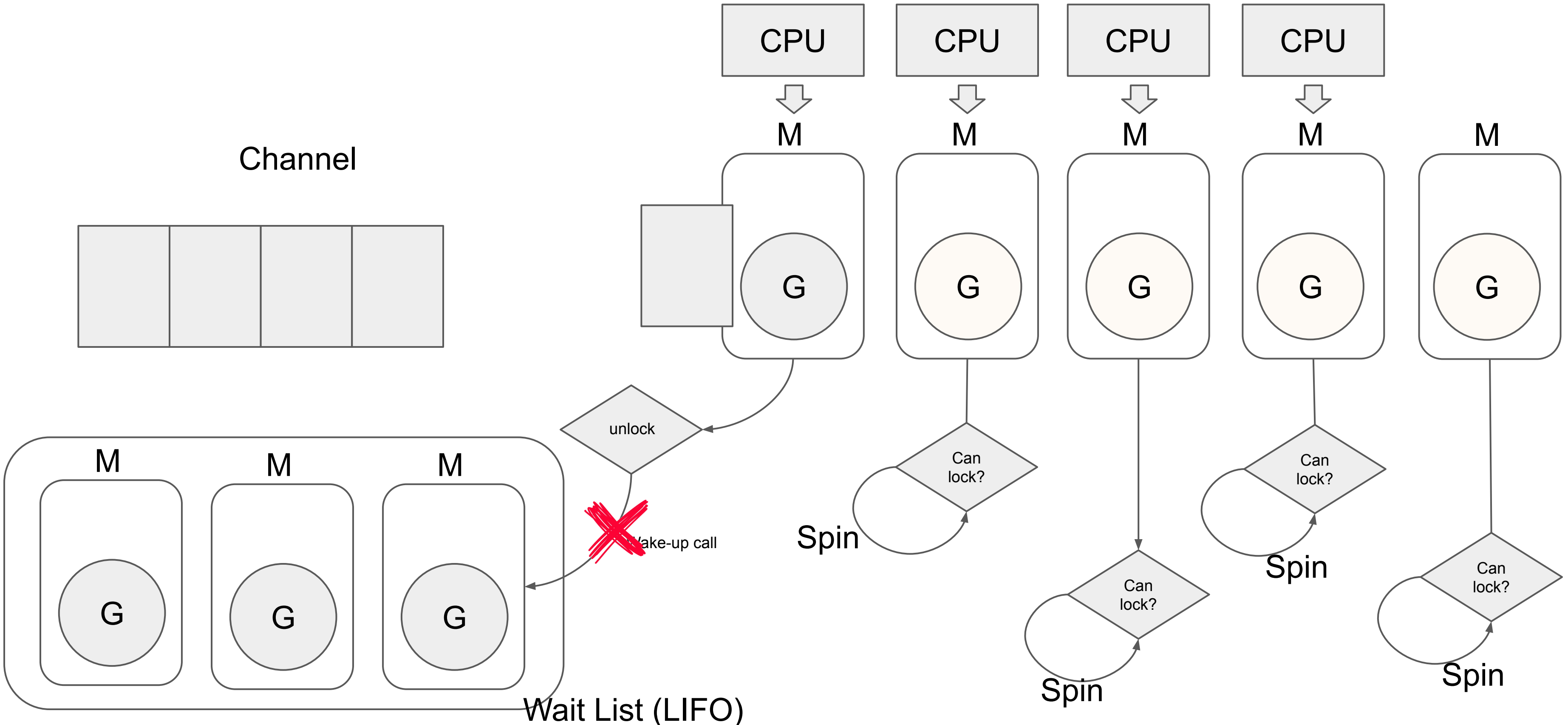
Abstract

Improve multi-core scalability of the runtime's internal mutex implementation by minimizing wakeups of waiting threads.

https://github.com/golang/proposal/blob/master/design/68578-mutex-spinbit.md

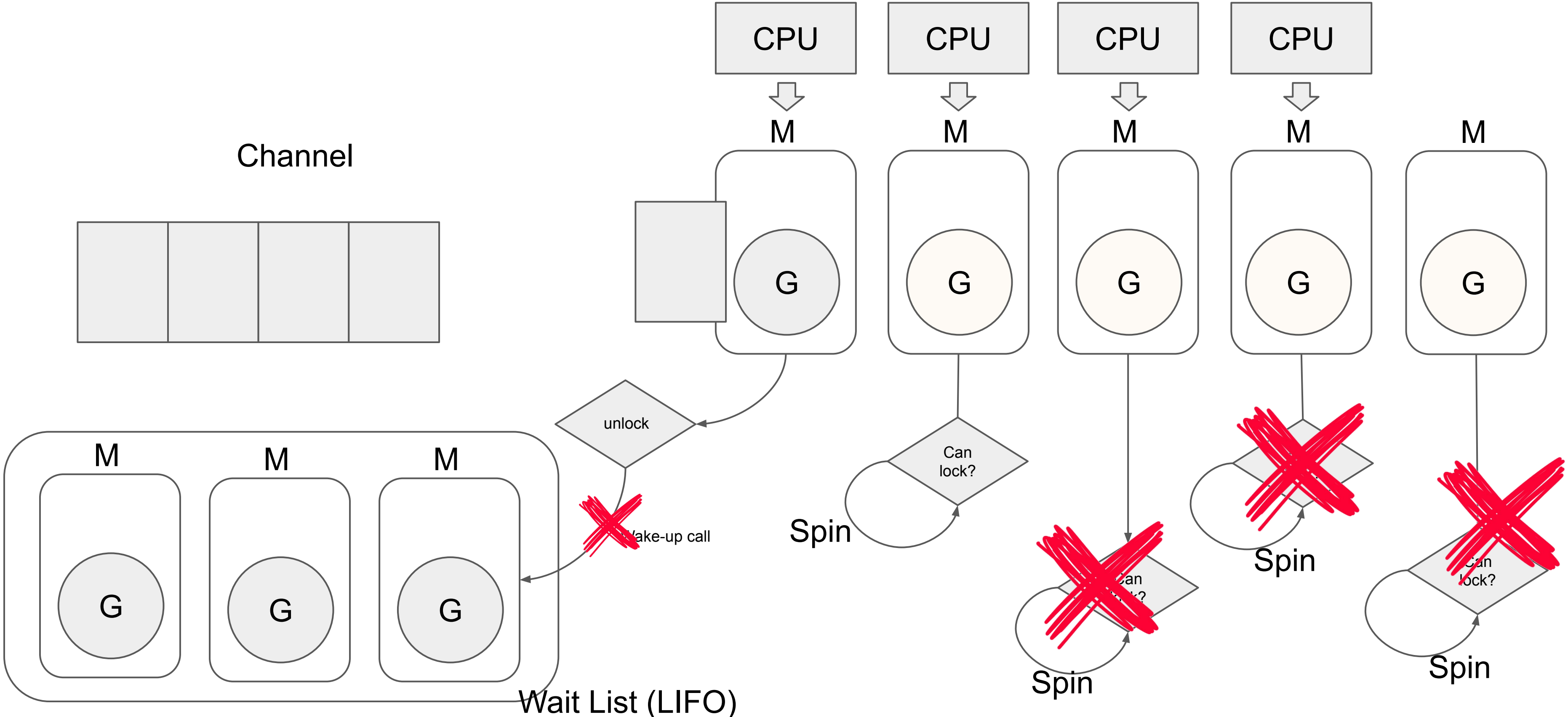
Smarter Locks

A deep dive on the Runtime Mutex Spin Optimisation



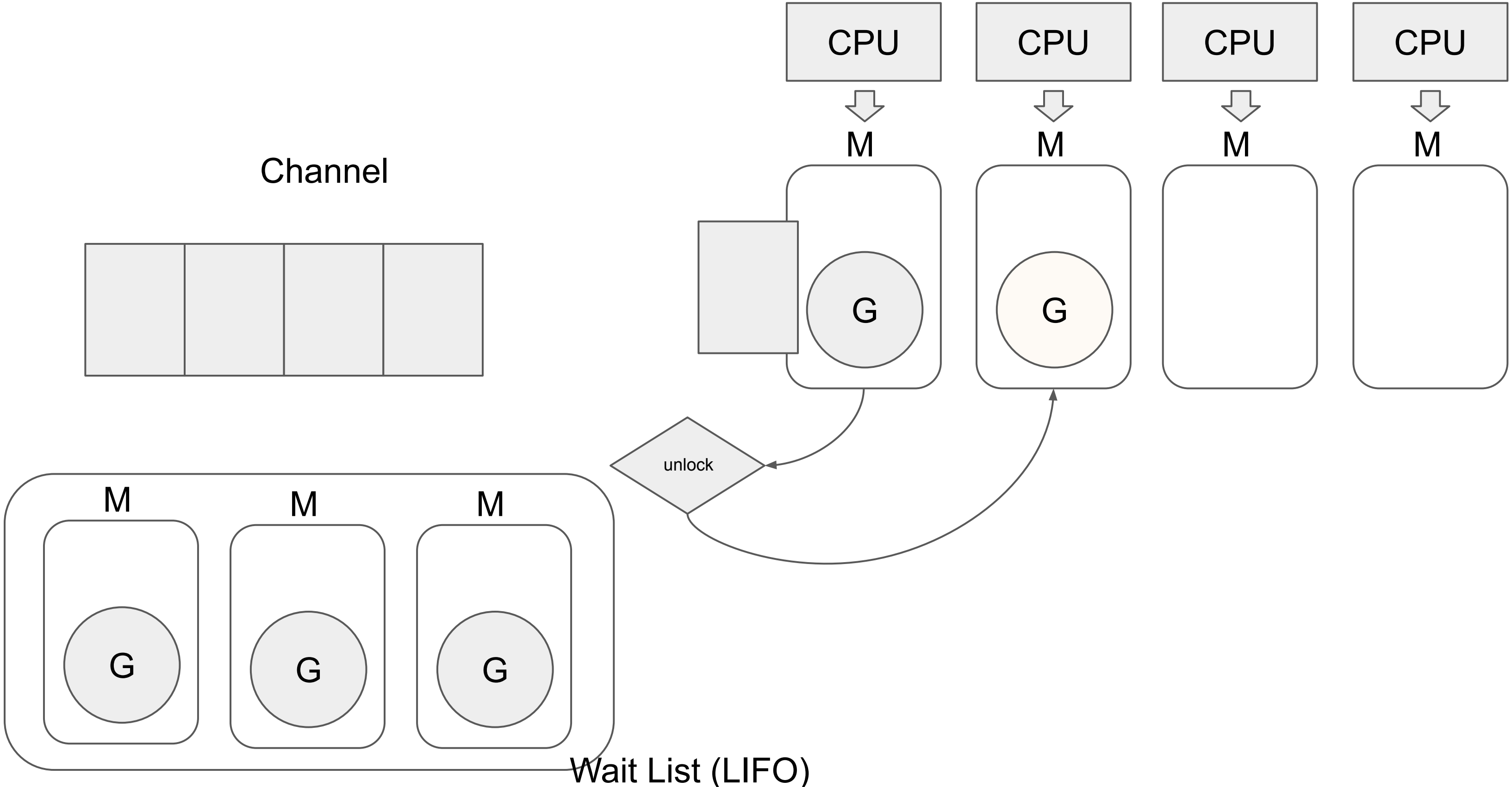
Smarter Locks

A deep dive on the Runtime Mutex Spin Optimisation



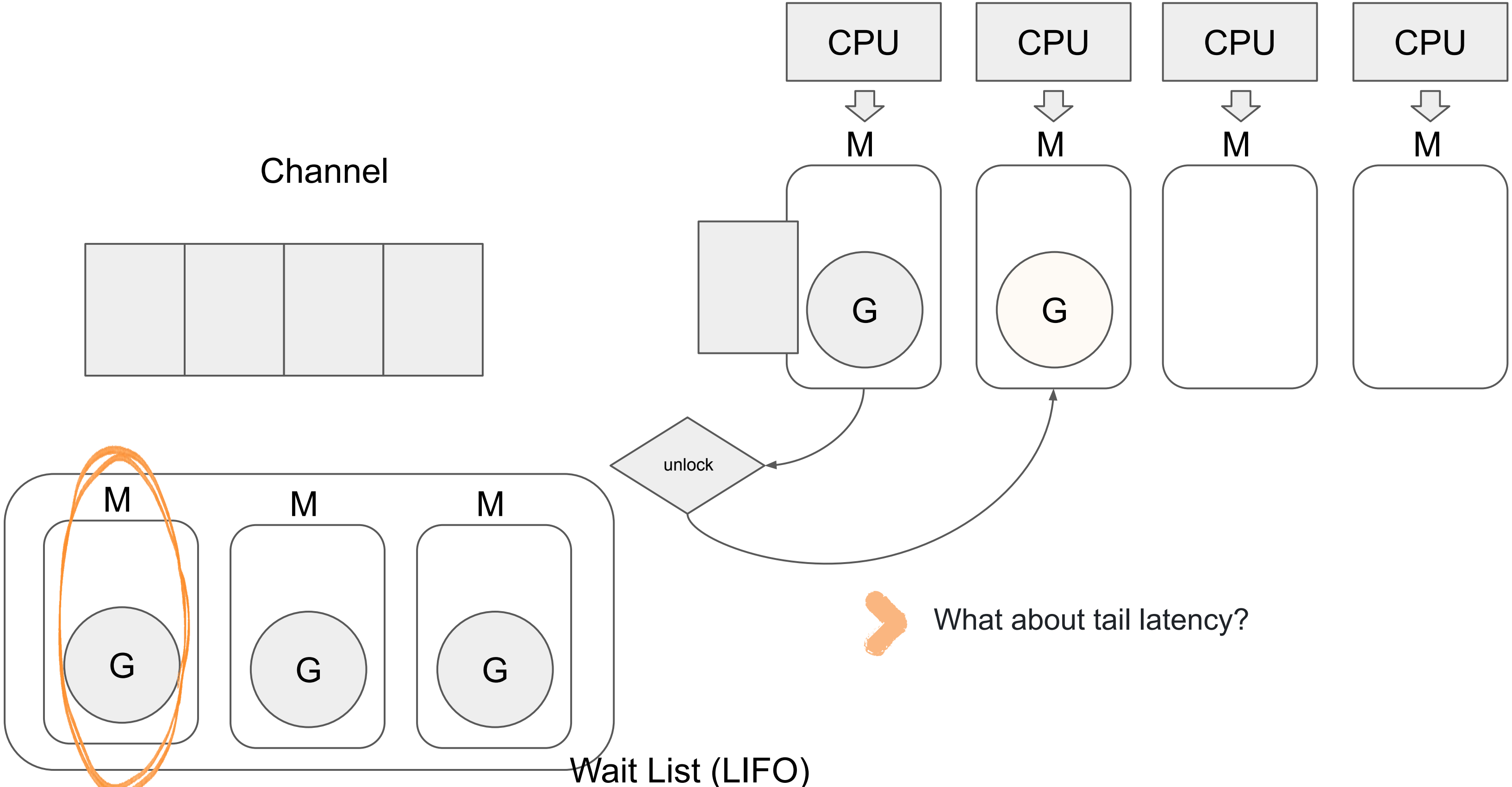
Smarter Locks

A deep dive on the Runtime Mutex Spin Optimisation



Smarter Locks

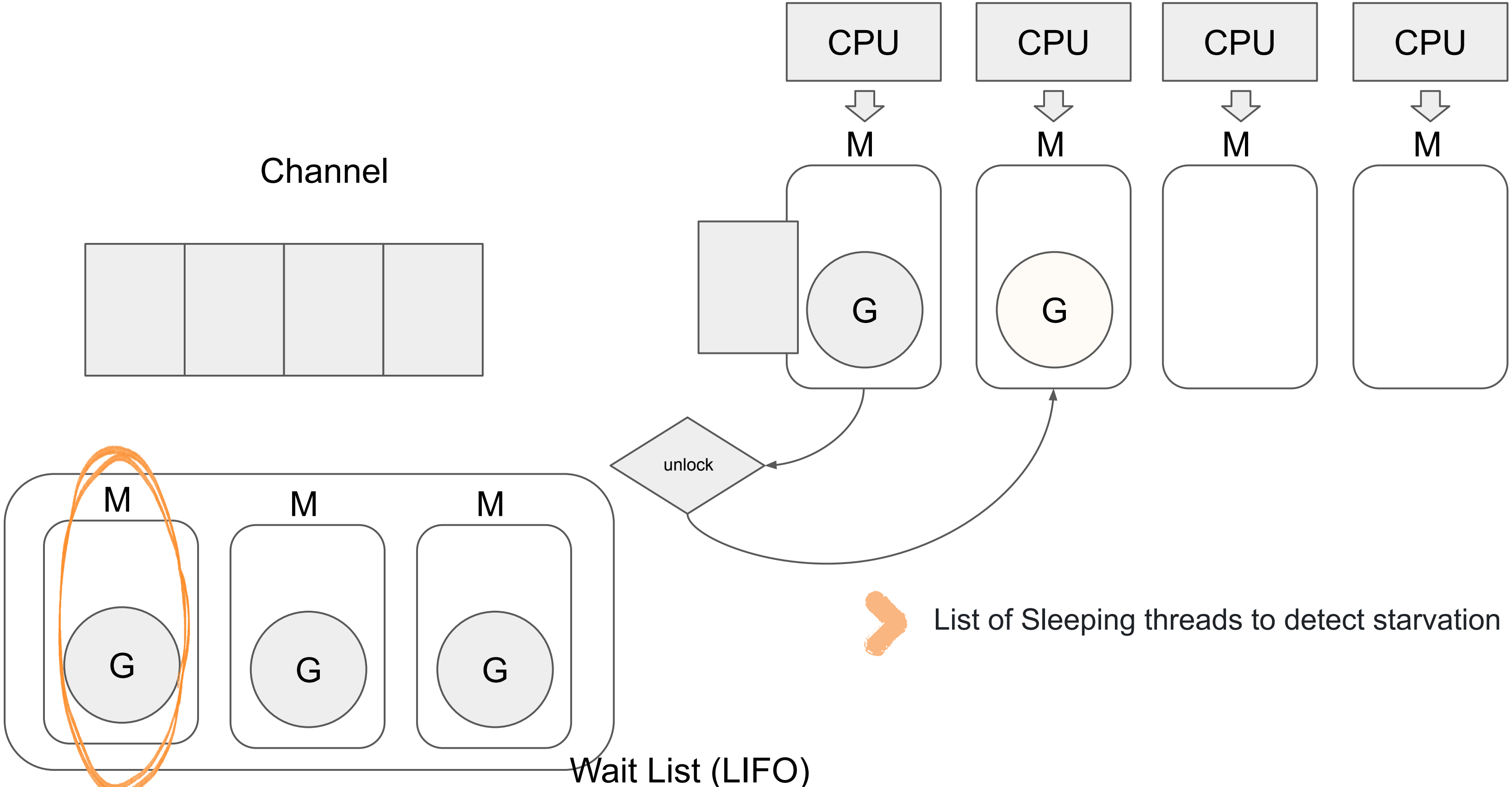
A deep dive on the Runtime Mutex Spin Optimisation



What about tail latency?

Smarter Locks

A deep dive on the Runtime Mutex Spin Optimisation



- List of Sleeping threads to detect starvation

Smarter Locks

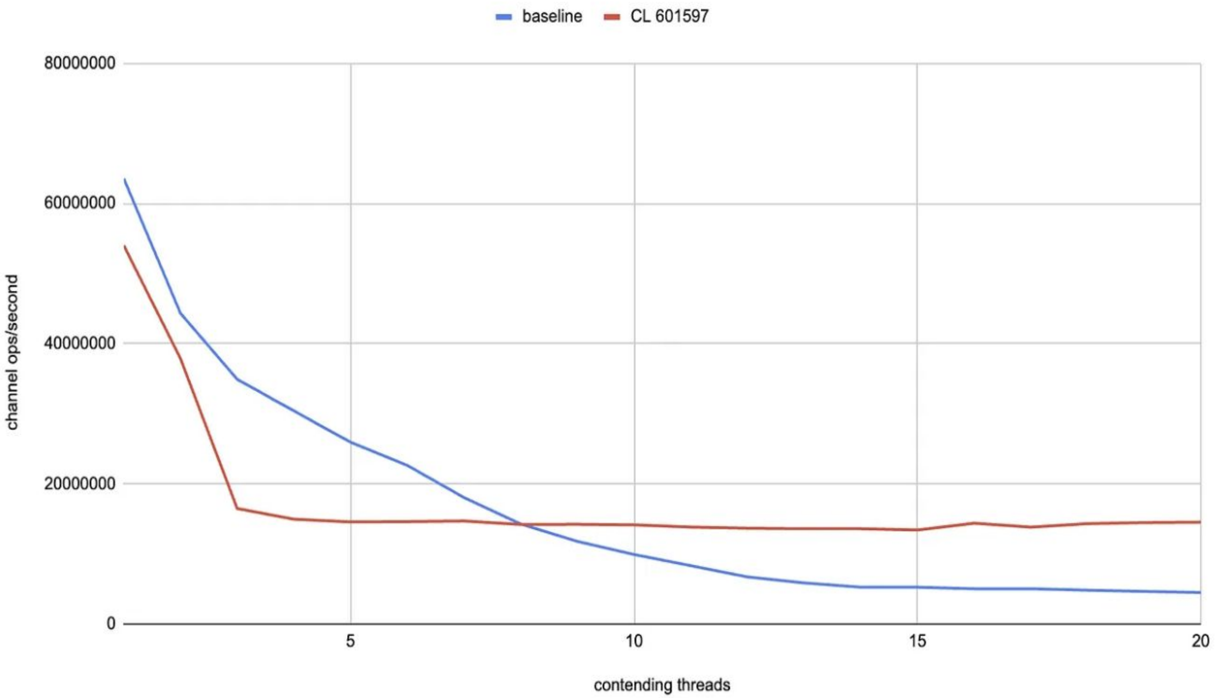
A deep dive on the Runtime Mutex Spin Optimisation

- "Minimizing wakeups of waiting threads".
- "Expand the mutex state word to include a new flag: spinning".
- "If there is one thread spinning, other threads needing the lock go immediately to sleep state".
- "If a thread is sleeping for too long wake it up so it starts to spin or reposition itself on the waiting stack"

https://github.com/golang/proposal/blob/master/design/68578-mutex-spinbit.md

Smarter Locks

A deep dive on the Runtime Mutex Spin Optimisation



CL 601597 gives horizontal scaling

Versus baseline, where each additional thread makes the lock even slower

"Several performance improvements to the runtime have decreased CPU overheads by 2–3% on average across a suite of representative benchmarks."

THANK YOU