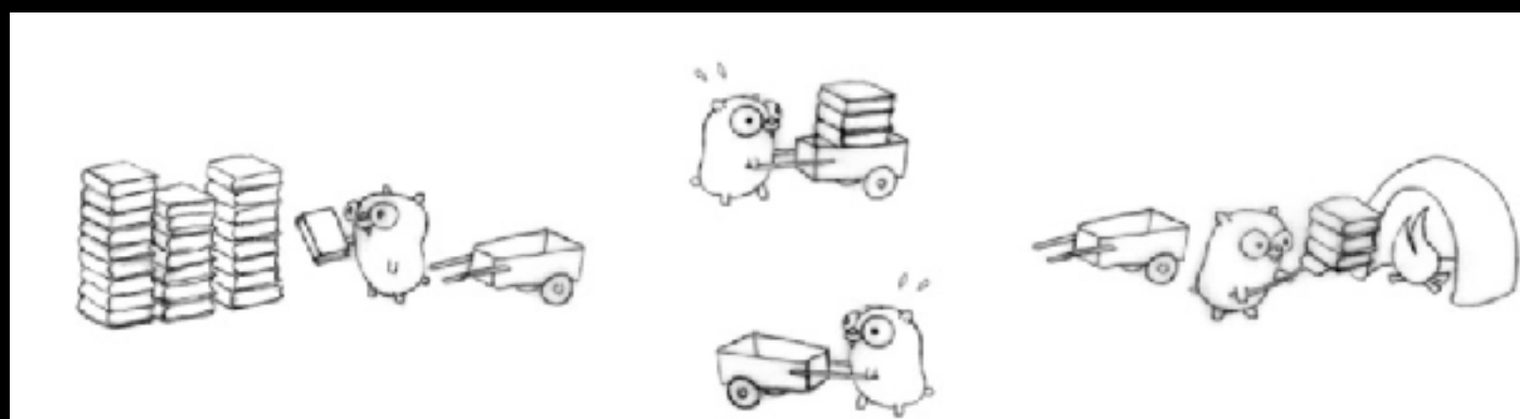
Concurrency in Go



- goroutine
- sync
- channel
- select

- what’s goroutine
- what’s channel
- scheduler

goroutine

a goroutine is a function that is running concurrently.

sync

The sync package contains the concurrency primitives that are most useful for low-level memory access synchronization.

- WaitGroup
- Mutex 、 RwMutex

channel

Channels are one of the synchronization primitives in Go derived from CSP. they are best used to communicate information between goroutines.

channel

- bidirectional channel / unidirectional channel
- buffered channel / unbuffered channel

```
biChan:=make(chan int)
//read only
uniChanRead:=make(<-chan int)
//write only
uniChanWrite:=make(chan<- int)

//unbuffered chan
unbufedChan:=make(chan int)
//buffered chan
bufedChan:=make(chan int,2)
```

channel

operation	channel state	result
read	nil	block
	open and not empty	value
	open and empty	block
	closed	<default>, false
	write only	compilation error
write	nil	block
	open and full	block
	open and not full	write value
	closed	panic
	received only	compilation error
close	nil	panic
	open and not empty	close channel until channel is empty, and read default value
	open and empty	close channel, and read default value
	closed	panic
	received only	compilation error

select

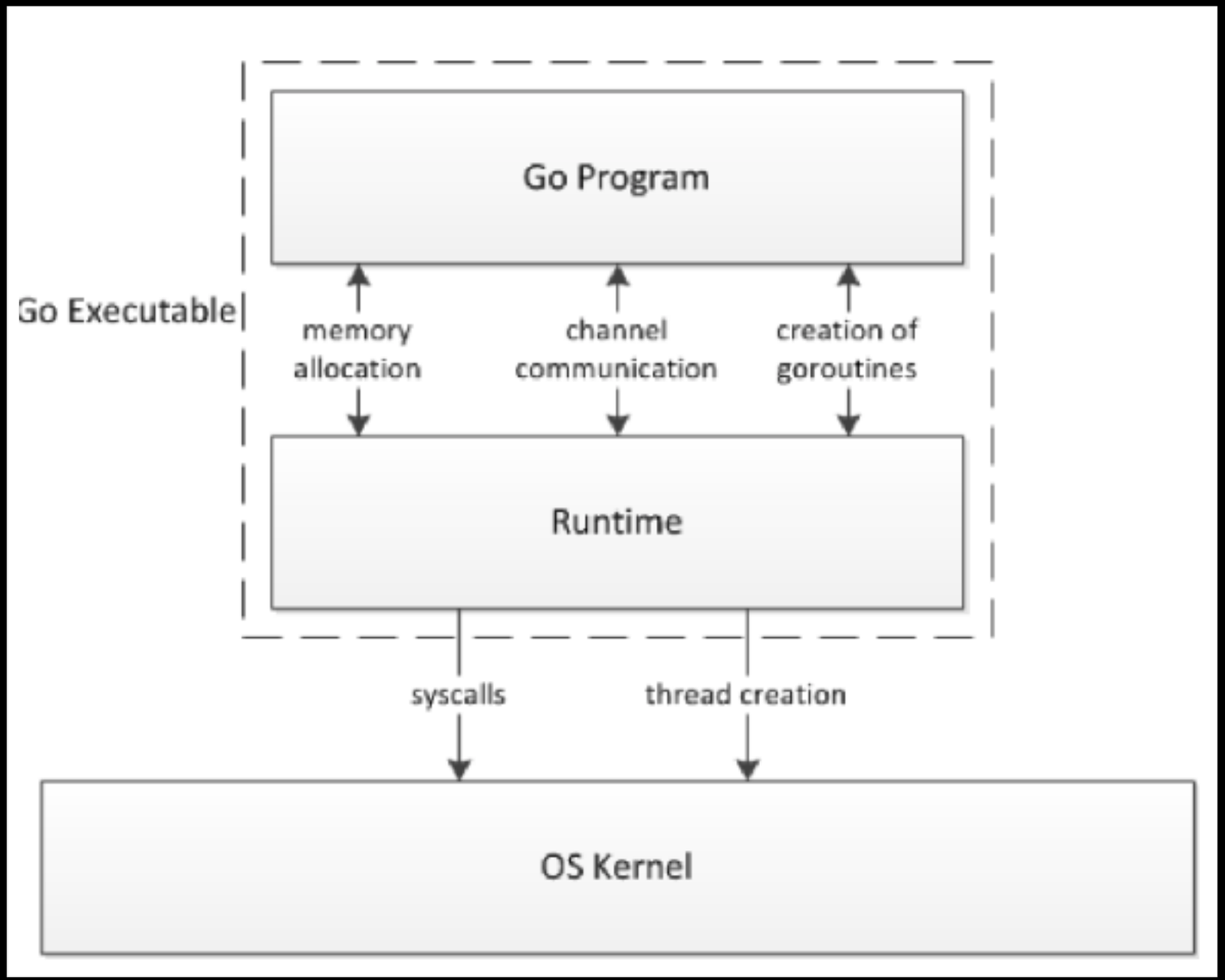
The select statement is the glue that binds channels together .

what’s goroutine

- 轻量级的“线程”
- 非抢占式多任务处理
- 编译器/解释器/虚拟机层面
- 多个goroutine可能在一个或者多个线程上运行

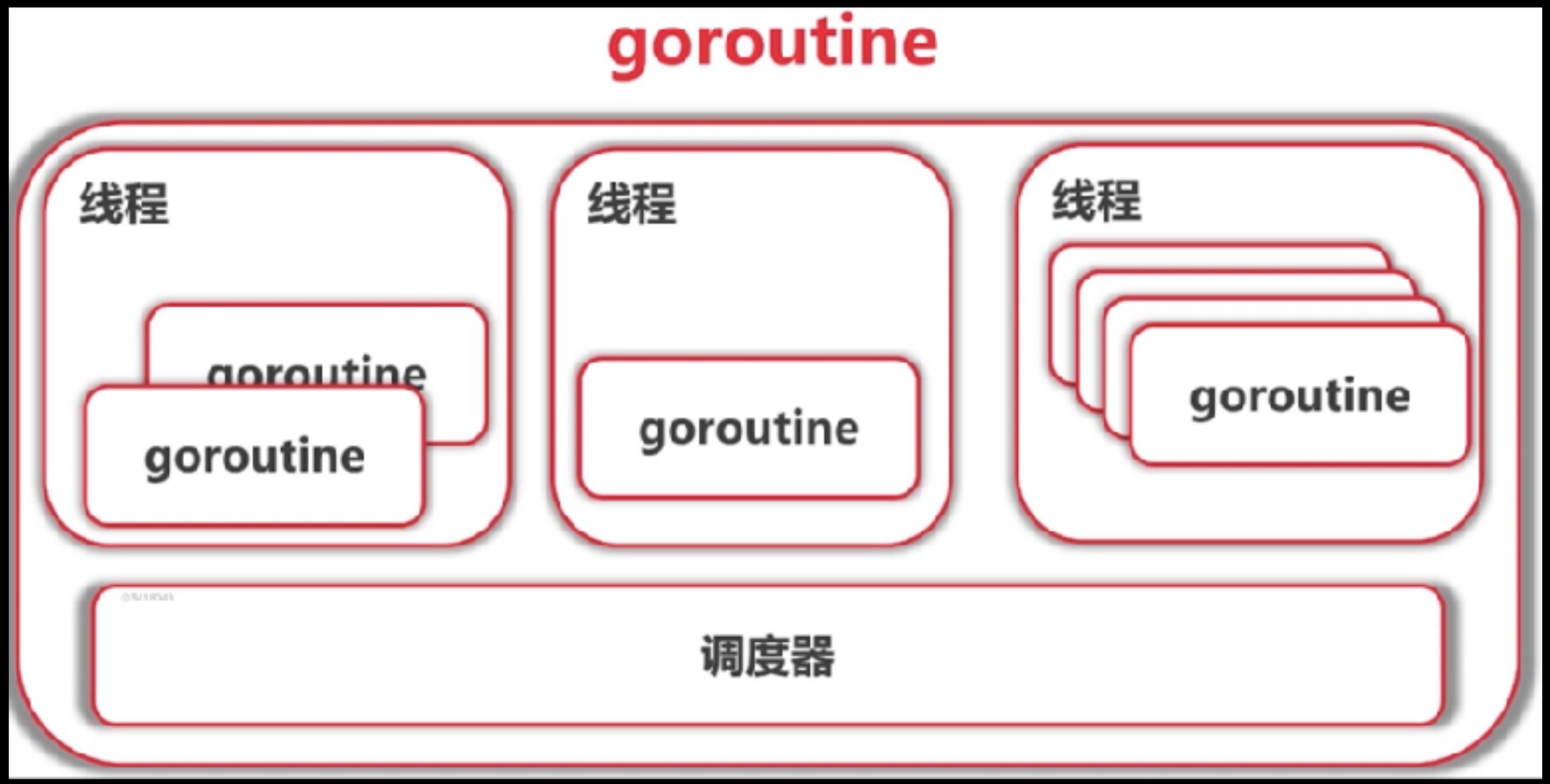
what’s goroutine

- 编译器/解释器/虚拟机层面



what’s goroutine

- 多个goroutine可能在一个或者多个线程上运行



what’s goroutine

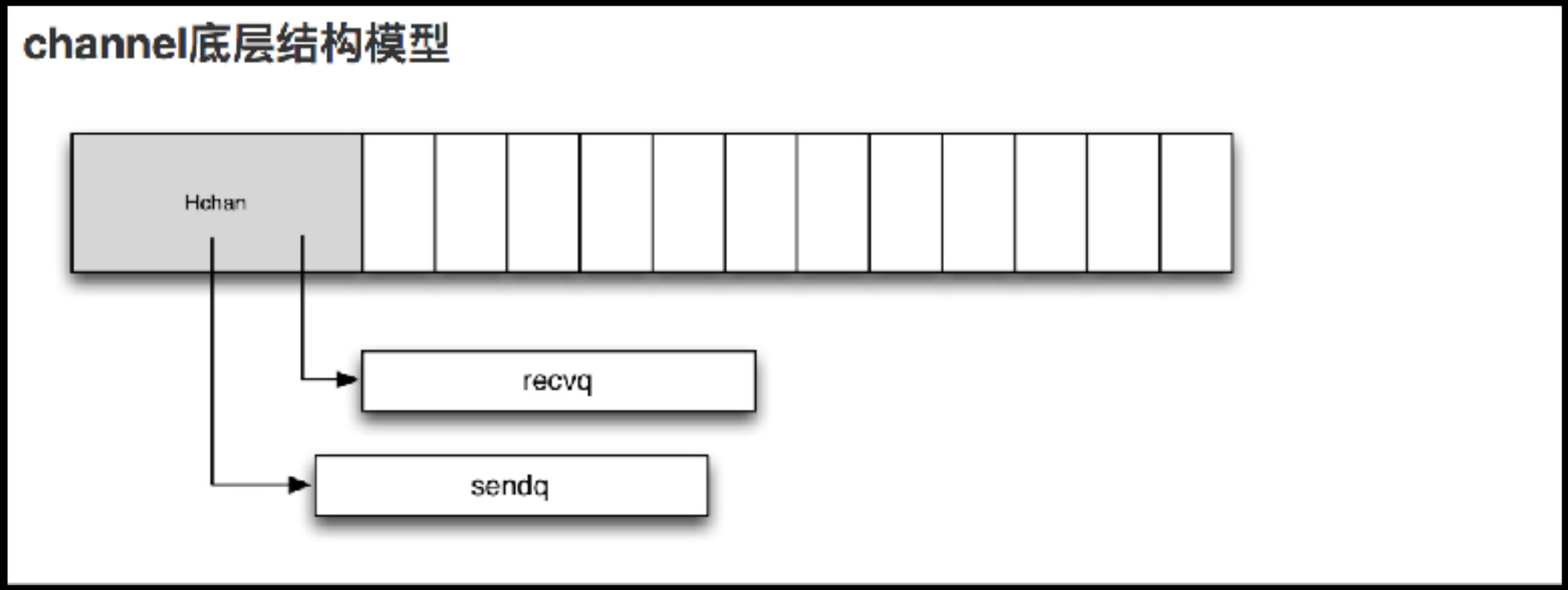
- goroutine可能切换的点

1. I/O
2. channel
3. 等待锁
4. 函数调用
5. runtime.Gosched()

只是参考，不能保证切换，不能保证在其他地方不切换。

what’s channel

- channel 结构



what’s channel

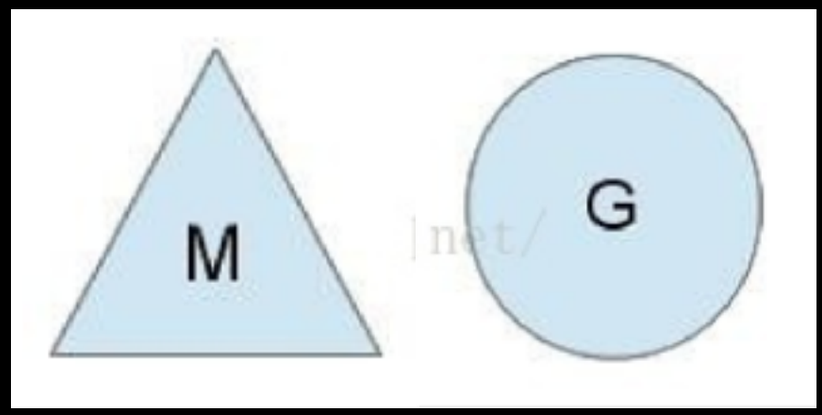
- 读写过程

同步写过程：

1. 对整个channel加锁
2. 判断是否有缓冲区，有缓冲区是异步写，没有缓冲区则是同步写。
3. 从等待读的队列中取出一个goroutine并拷贝数据，设置这个goroutine为ready状态。如果没有等待读的goroutine，写入的元素保存在当前执行写的goroutine的结构里，然后将当前goroutine入队到中并被挂起，等待有goroutine读取元素才会被唤醒。

scheduler

The Go runtime manages scheduling. the scheduler's job is to distribute ready-to-run goroutines over worker threads.



- M：线程
- G：goroutine

scheduler

缺点：

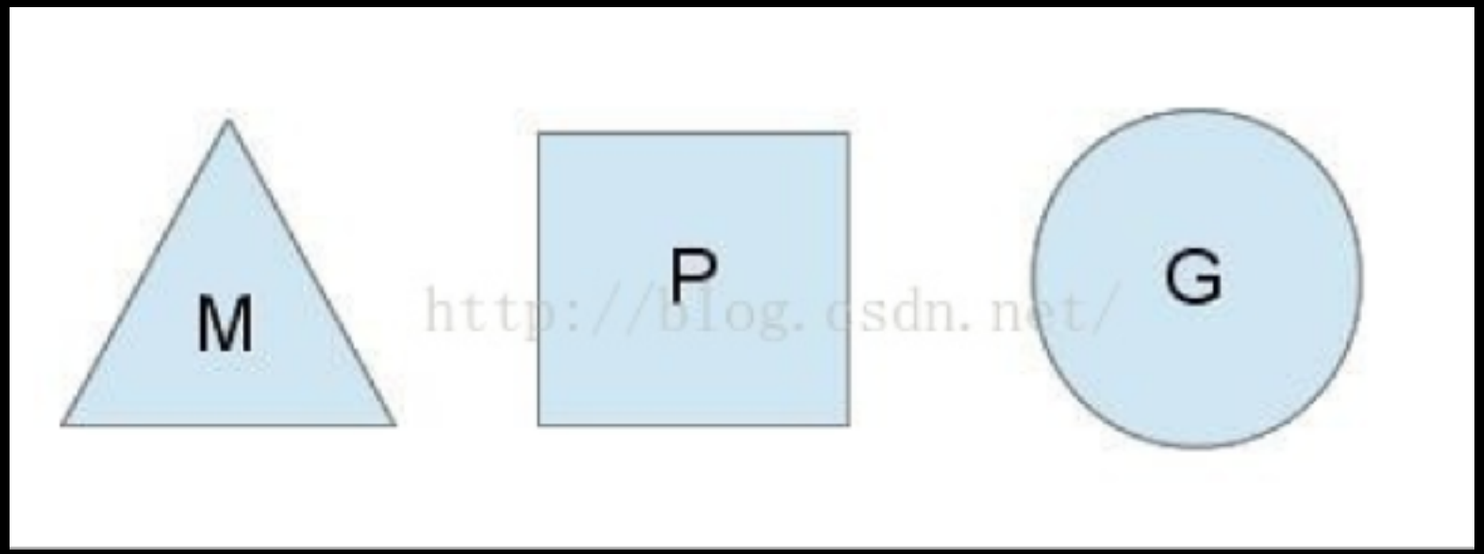
1. 对调度器全局锁的依赖。
2. 当没有代码运行时，也需要为M结构的MCache分配2MB。
3. 系统调用处理不好导致M阻塞，浪费CPU。

```
struct  M
{
  G*        curg;           // current running goroutine
  int32     id;             // unique id
  int32     locks;          // locks held by this M
  MCache    *mcache;        // cache for this thread
  G*        lockedg;        // used for locking M's and G's
  uintptr   createstack[32]; // Stack that created this thread
  M*        nextwaitm;      // next M waiting for lock
   ...
};
```

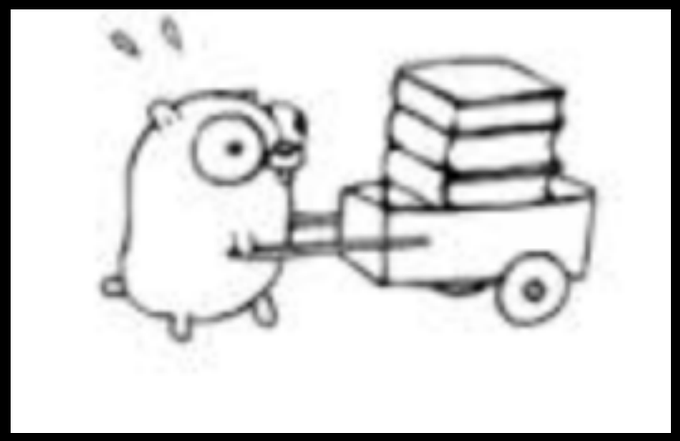
```
struct Sched {
  Lock;             // global sched lock.
                    // must be held to edit G or M queues

  G *gfree;         // available g's (status == Gdead)
  G *ghead;         // g's waiting to run queue
  G *gtail;         // tail of g's waiting to run queue
  int32 gwait;      // number of g's waiting to run
  int32 gcount;     // number of g's that are alive
  int32 grunning;   // number of g's running on cpu
                    // or in syscall

  M *mhead;         // m's waiting for work
  int32 mwait;      // number of m's waiting for work
  int32 mcount;     // number of m's that have been created
   ...
};
```

scheduler

改进

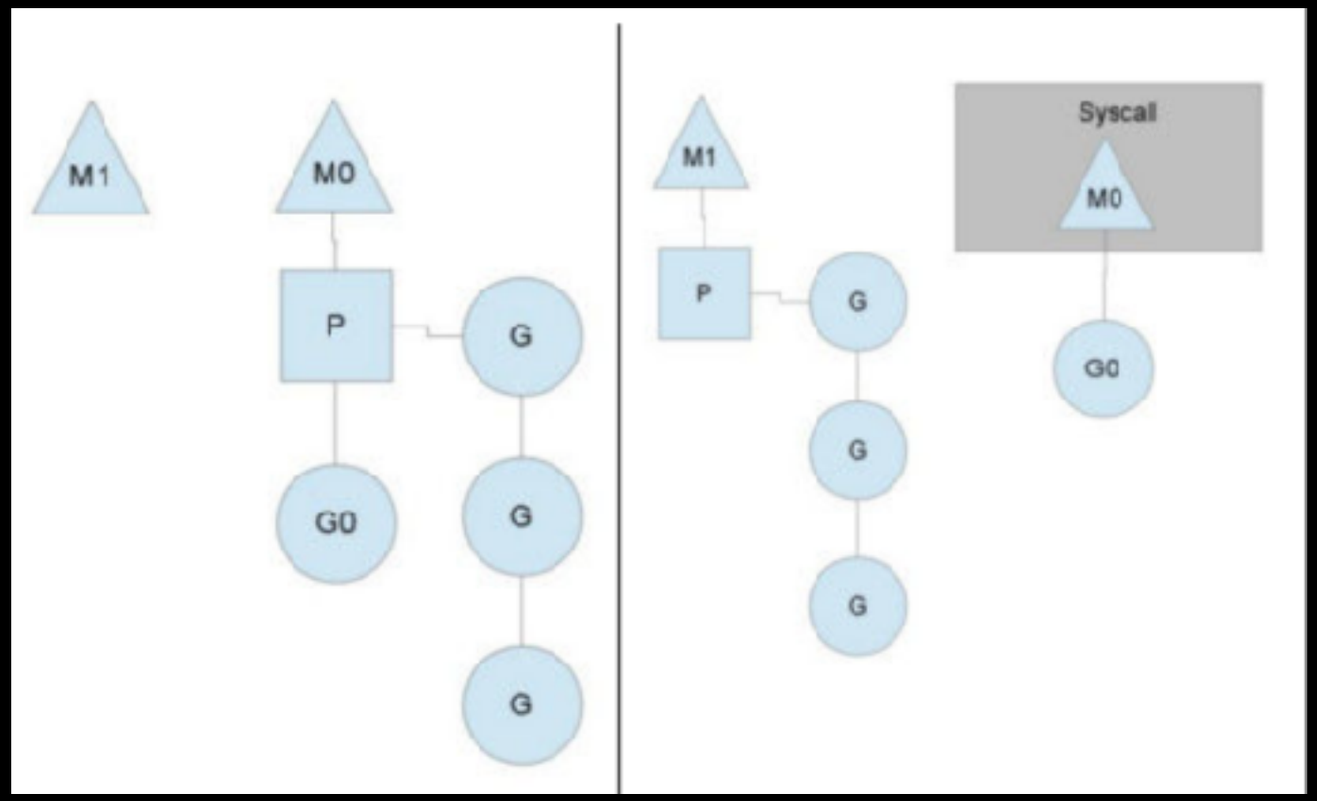


- P:调度的上下文，可以把它看做一个局部的调度器



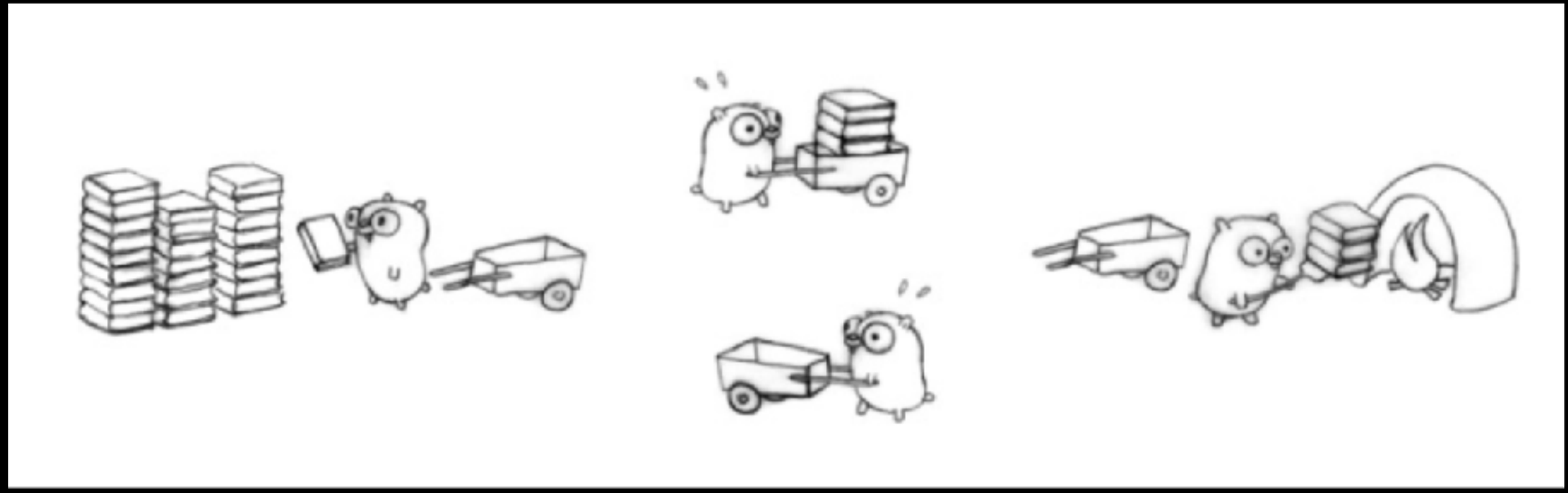
scheduler

1. 根据GOMAXPROCS的值创建P，不超过256个.
2. MCache被移动到P结构中
3. 每一个P结构含有一个可运行G队列
4. P可以在各个M之间切换
5. 同时存在全局的G队列，P也会去检查队列中的任务



scheduler

1. 调度器为M分配一个空闲的P，M使用P执行任务G。
2. P如果自己队列中没有任务G，也会周期性的从全局队列中获取任务。
3. 如果全局队列中没有任务G，则从另一个P的队列中偷取一半的任务。
4. 如果获取任务失败，则M持有使用P，M进入睡眠状态。
5. 如果有任务则执行任务。
6. 当任务太多时，并且有空闲的P，调度器会创建M来获取P执行任务。
7. 当M进行系统调用返回时，没有P可以使用，则将任务G放回全局任务队列，自己睡眠



M（地鼠）、P（小车）、G（砖）