Tony Bai

一个程序员的心路历程

图解Go内存分配器

- 二月 20, 2020
- 0 条评论

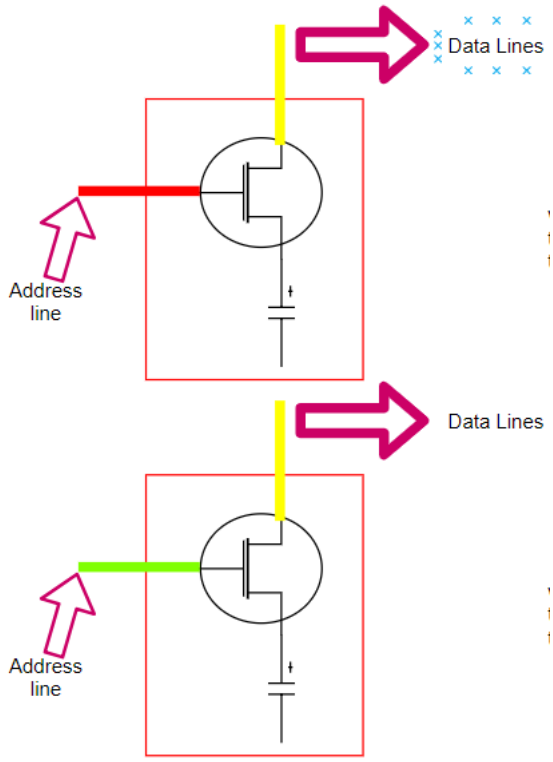
本文翻译自《A visual guide to Go Memory Allocator from scratch (Golang)》。

当我刚开始尝试了解Go的内存分配器时，我发现这真是一件可以令人发疯的事情，因为所有事情似乎都像一个神秘的黑盒(让我无从下手)。由于几乎所有技术魔法都隐藏在抽象之下，因此您需要逐一剥离这些抽象层才能理解它们。

在这篇文章中，我们就来这么做(剥离抽象层去了解隐藏在其下面的技术魔法)。如果您想了解有关Go内存分配器的知识，那么本篇文章正适合您。

一. 物理内存(Physical Memory)和虚拟内存(Virtual Memory)

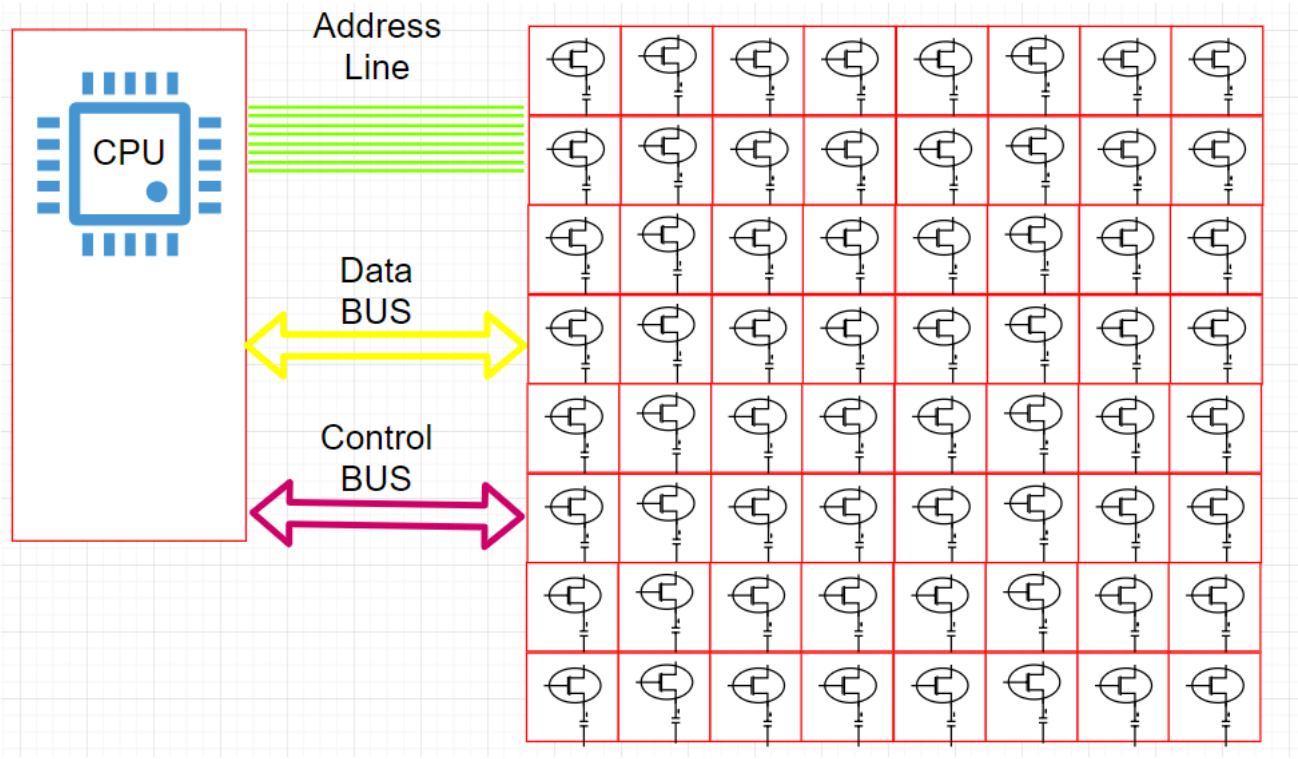
每个内存分配器都需要使用由底层操作系统管理的虚拟内存空间(Virtual Memory Space)。让我们看看它是如何工作的吧。



物理存储单元的简单图示（不精确的表示）

单个存储单元（工作流程）的简要介绍：

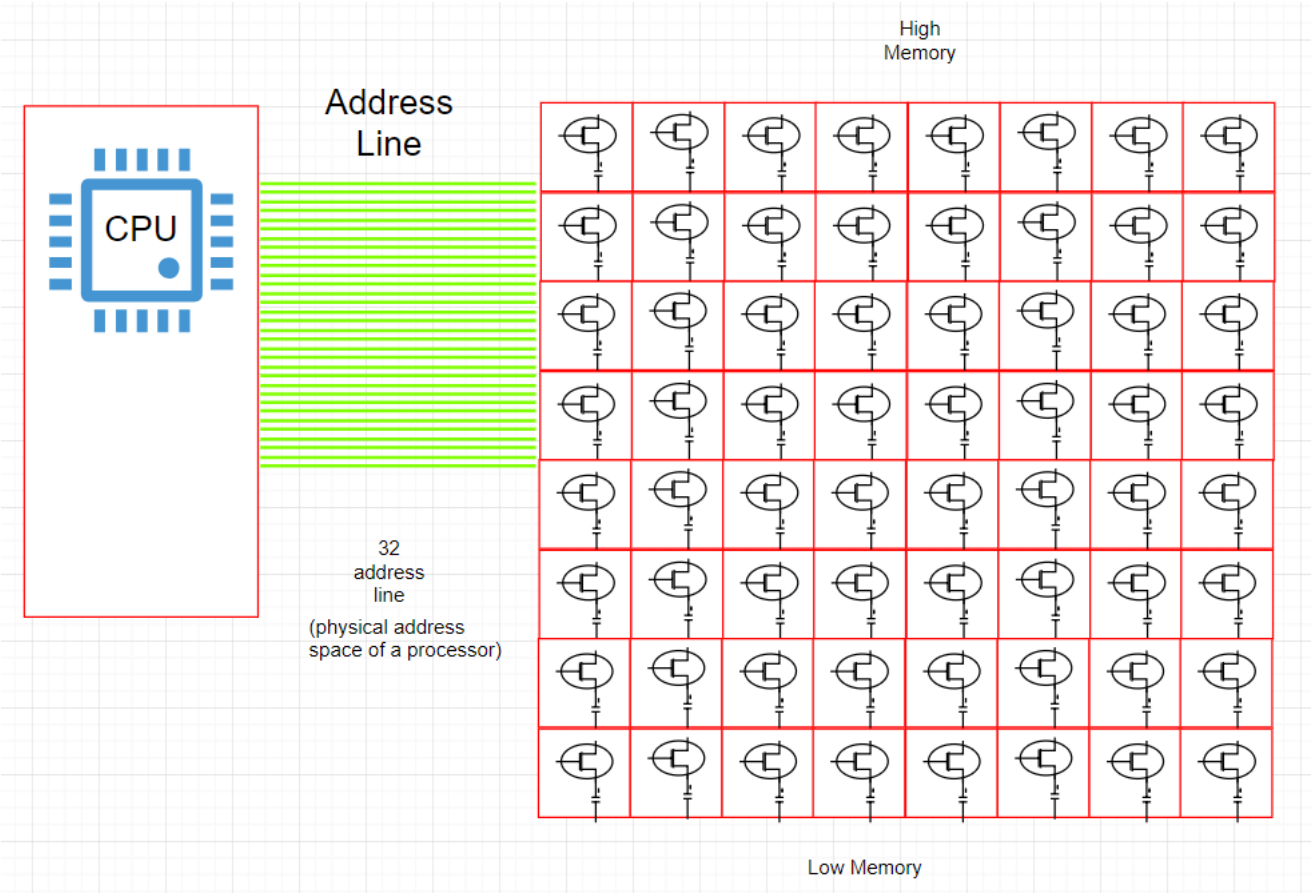
1. 地址线(address line, 作为开关的晶体管)提供了访问电容器的入口(数据到数据线(data line))。
2. 当地址线中有电流流动时（显示为红色），数据线可能会写入电容器，因此电容器已充电，并且存储的逻辑值为“1”。
3. 当地址线没有电流流动（显示为绿色）时，数据线可能不会写入电容器，因此电容器未充电，并且存储的逻辑值为“0”。
4. 当处理器(CPU)需要从内存(RAM)中“读取”一个值时，会沿着“地址线”发送电流（关闭开关）。如果电容器保持电荷，则电流流经“ DATA LINE”（数据线）得到的值为1；否则，没有电流流过数据线，电容器将保持未充电状态，得到的值为0。



物理内存单元如何与CPU交互的简单说明

数据总线(Data Bus)：用于在CPU和物理内存之间传输数据。

让我们讨论一下地址线(Address Line)和可寻址字节(Addressable Bytes)。



CPU和物理内存之间的地址线的表示

1. DRAM中的每个“字节(BYTE)”都被分配有唯一的**数字标识符（地址）**。但“物理字节的表示 != 地址线的数量”。（例如：16位Intel 8088，PAE）
2. 每条“地址线”都可以发送1bit值，因此它可以表示给定字节地址中指定“bit”。
3. 在图中，我们有32条地址线。因此，每个可寻址字节都将拥有一个“32bit”的地址。

```
[ 00000000000000000000000000000000 ] — 低内存地址
[ 11111111111111111111111111111111 ] — 高内存地址
```

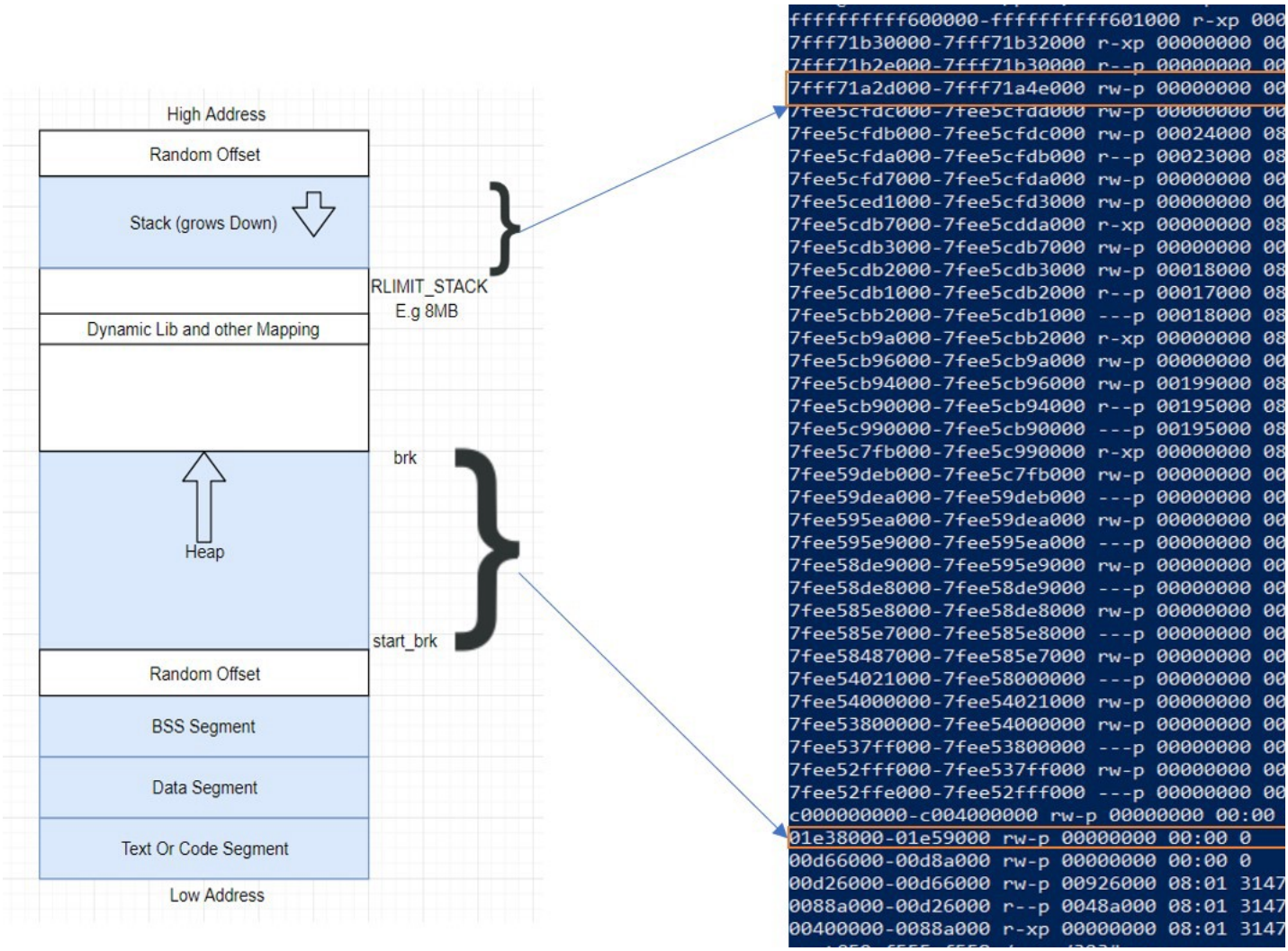
4.由于每个字节都有一个32bit地址，所以我们的地址空间由2的32次方个可寻址字节（即4GB）组成。

因此，可寻址字节取决于地址线的总量，对于64位地址线（x86–64 CPU），其可寻址字节为2的64次方个，但是大多数使用64位指针的体系结构实际上使用48位地址线（AMD64 ）和42位地址线（英特尔），理论上支持256TB的物理RAM（Linux 在x86–64上每个进程支持128TB以及4级页表(page table)和Windows每个进程则支持192TB）

由于实际物理内存的限制，因此每个进程都在其自己的内存沙箱中运行-“虚拟地址空间”，即**虚拟内存**。

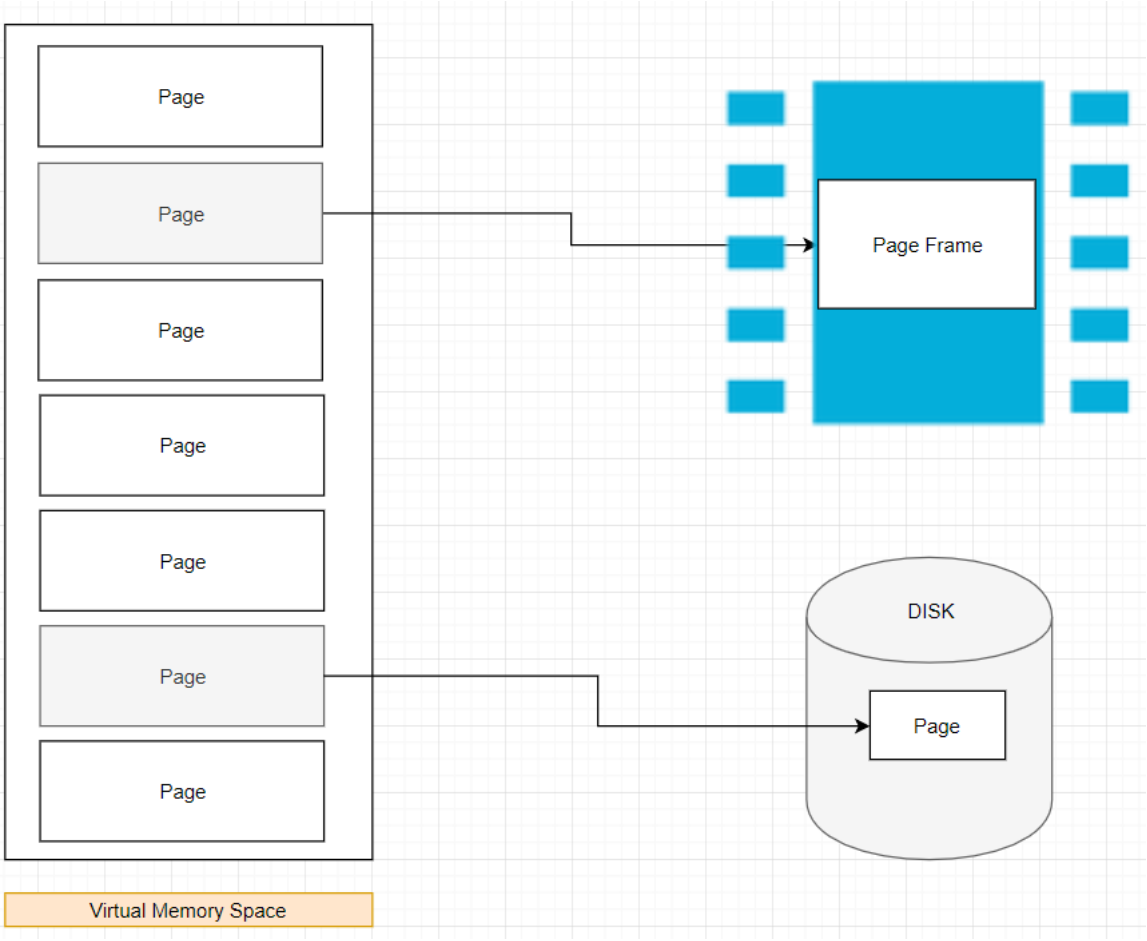
该虚拟地址空间中字节的地址不再与处理器在地址总线上放置的地址相同。因此，必须建立转换数据结构和系统，以将虚拟地址空间中的字节映射到物理内存地址上的字节。

虚拟地址长什么样呢?



虚拟地址空间表示

因此，当CPU执行引用内存地址的指令时。第一步是将VMA(virtual memory address)中的**逻辑地址**转换为**线性地址(liner address)**。这个翻译工作由**内存管理单元MMU(Memory Management Unit)**完成。



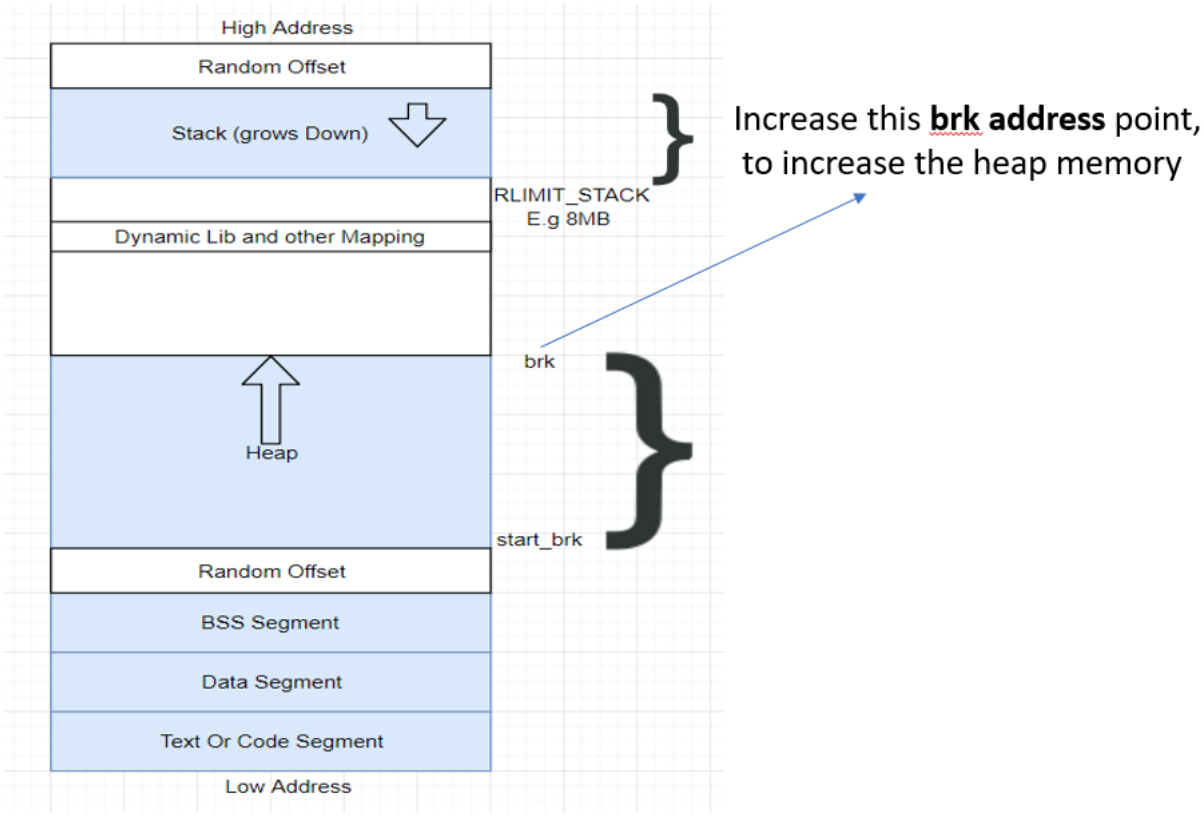
这不是物理图，仅是描述。为了简化，不包括地址翻译过程

由于此逻辑地址太大而无法单独管理（取决于各种因素），因此将通过页(page)对其进行管理。当必要的分页构造被激活后，**虚拟内存空间将被划分为称为页的较小区域（大多数OS上页大小为4KB，可以更改）**。它是虚拟内存中用于内存管理的最小单位。虚拟内存不存储任何内容，仅简单地将程序的地址空间映射到真实的物理内存空间上。

单个进程仅将VMA(虚拟内存地址)视为其地址。**这样，当我们的程序请求更多“堆内存(heap memory)”时会发生什么呢?**

```
_start:
        mov $12, %rax    # brk syscall number
        mov $0, %rdi     # 0 is invalid, want to get current
        syscall
b0:
        mov %rax, %rsi   # rsi now points to start of heap m
        mov %rax, %rdi   # move top of heap to here ...
        add $4, %rdi     # .. plus 4 bytes we allocate
        mov $12, %rax    # brk, again
        syscall
```

一段简单的用户请求更多堆内存的汇编代码



增加堆内存

程序通过brk（sbrk/mmap等）系统调用请求更多内存。但内核实际上仅是更新了堆的VMA。

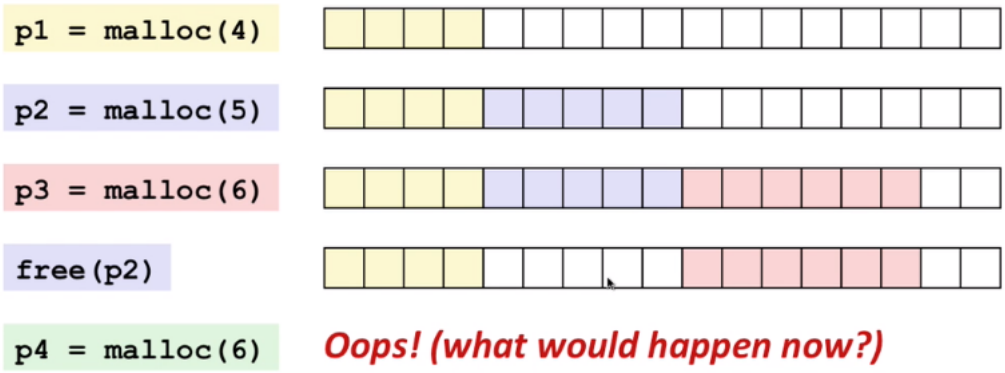
> 注意：此时，实际上并没有分配任何页帧，并且新页面也没有在物理内存存在。这也是VSZ与RSS之间的差异点。

二. 内存分配器

有了“虚拟地址空间”的基本概述以及堆内存增加的理解之后，内存分配器现在变得更容易说明了。

> 如果堆中有足够的空间来满足我们代码中的内存请求，则内存分配器可以在内核不参与的情况下满足该请求，否则它会通过系统调用brk扩大堆，通常会请求大量内存。（默认情况下，对于malloc而言，大量的意思是 > MMAP_THRESHOLD字节-128kB）。

但是，内存分配器的责任不仅仅是更新brk地址。其中一个主要的工作则是如何的降低内外部的内存碎片以及如何快速分配内存块。考虑按p1~p4的顺序，先使用函数malloc在程序中请求连续内存块，然后使用函数free(pointer)释放内存。



外部内存碎片演示

在第4步，即使我们有足够的内存块，我们也无法满足对6个连续内存块分配的请求，从而导致内存碎片。

那么如何减少内存碎片呢？这个问题的答案取决于底层库使用的特定的内存分配算法。

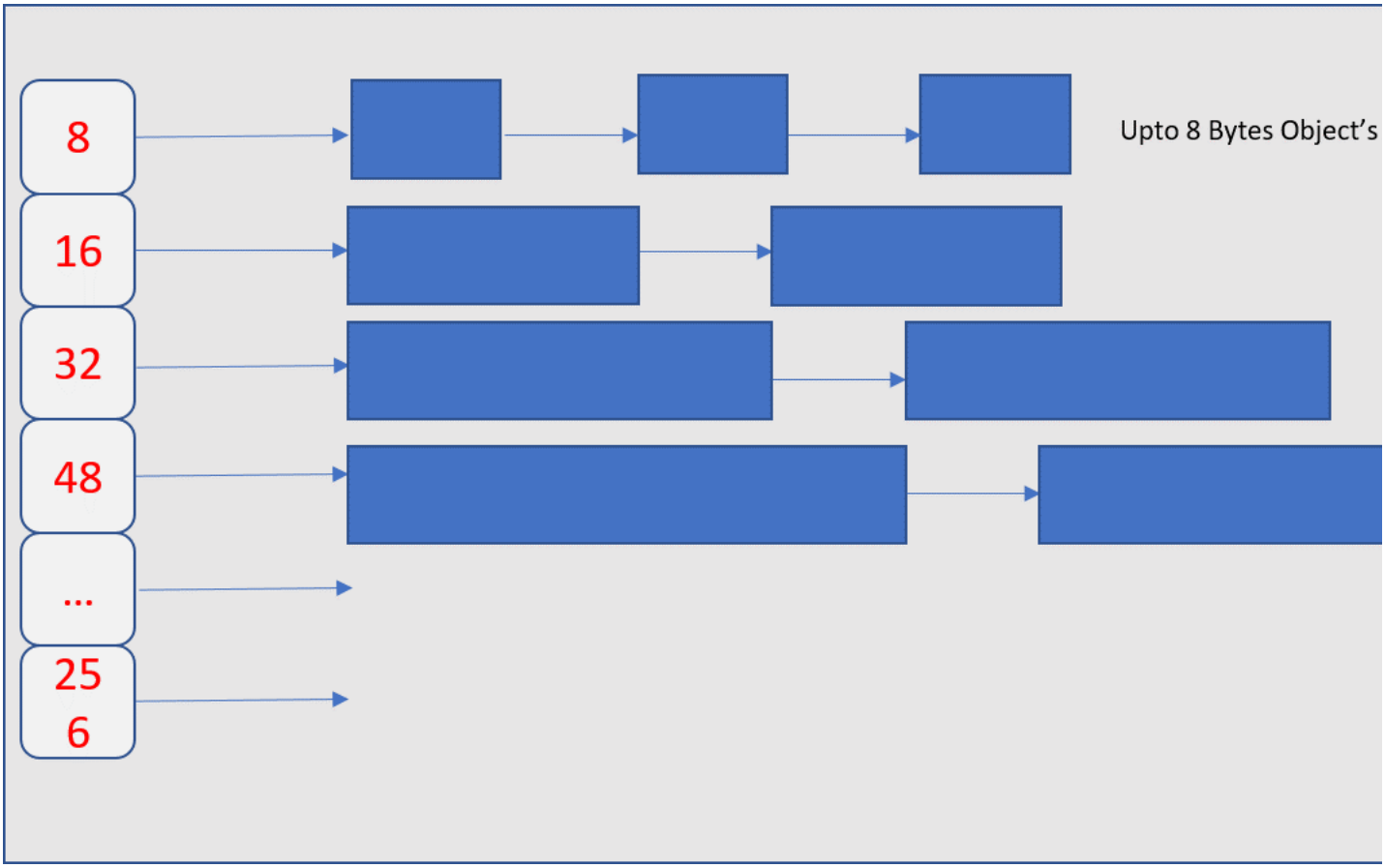
我们将研究TCMalloc内存分配器，Go内存分配器采用的就是该内存分配器模型。

三. TCMalloc

TCMalloc（thread cache malloc）的核心思想是将内存划分为多个级别，以减少锁的粒度。在TCMalloc内部，内存管理分为两部分：线程内存和页堆(page heap)。

线程内存(thread memory)

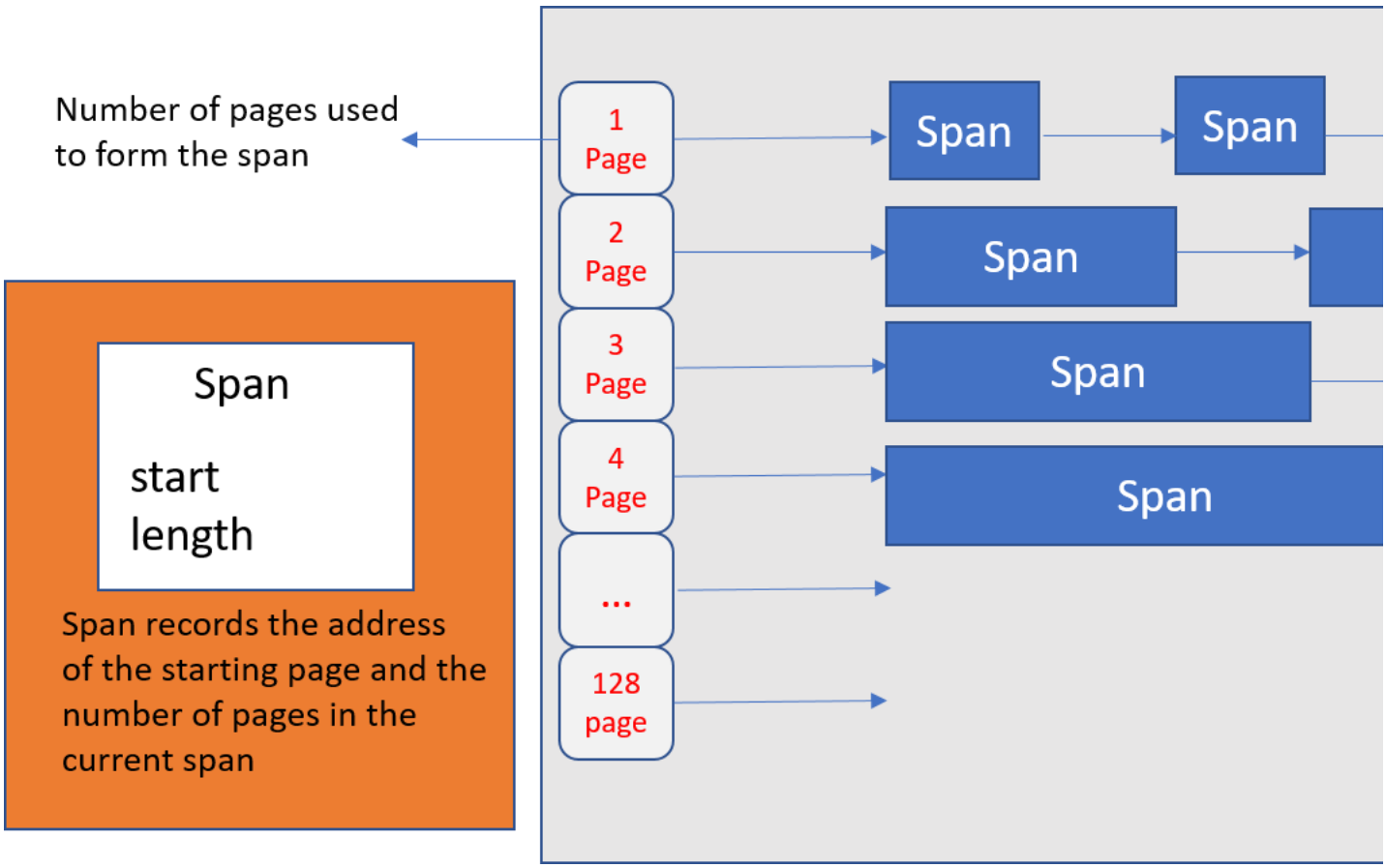
每个内存页分为多级固定大小的“空闲列表”，这有助于**减少碎片**。因此，每个线程都会有一个无锁的小对象缓存，这使得在并行程序下分配小对象（<= 32k）非常高效。



线程缓存（每个线程拥有此线程本地线程缓存）

页堆(page heap)

TCMalloc管理的堆由页集合组成，其中一组连续页的集合可以用span表示。当分配的对象大于32K时，将使用页堆进行分配。



页堆（用于span管理）

如果没有足够的内存来分配小对象，内存分配器就会转到页堆以获取内存。如果还没有足够的内存，页堆将从操作系统中请求更多内存。

由于这种分配模型维护了一个用户空间的内存池，因此极大地提高了内存分配和释放的效率。

注意：尽管go内存分配器最初是基于tcmalloc的，但是现在已经有了很大的不同。

四. Go内存分配器

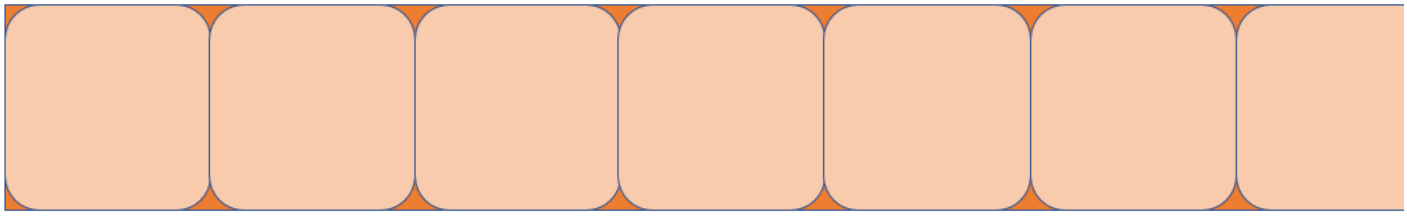
我们知道Go运行时会将Goroutines（G）调度到逻辑处理器（P）上执行。同样，基于TCMalloc模型的Go还将内存页分为67个不同大小级别。

如果您不熟悉Go调度程序，则可以在这里获取关于Go调度程序的相关知识。

```
const _NumSizeClasses = 67
var class_to_size = [_NumSizeClasses]uint16{0, 8, 16
, 32, 48, 64, 80, 96, 112, 128, 144, 160, 176, 192,
208, 224, 240, 256, 288, 320, 352, 384, 416, 448,
480, 512, 576, 640, 704, 768, 896, 1024, 1152, 1280,
 1408, 1536, 1792, 2048, 2304, 2688, 3072, 3200,
3456, 4096, 4864, 5376, 6144, 6528, 6784, 6912, 8192
, 9472, 9728, 10240, 10880, 12288, 13568, 14336,
16384, 18432, 19072, 20480, 21760, 24576, 27264,
28672, 32768}
```

Go中的内存块的大小级别

Go默认采用8192B大小的页。如果这个页被分成大小为1KB的块，我们一共将拿到8块这样的页:



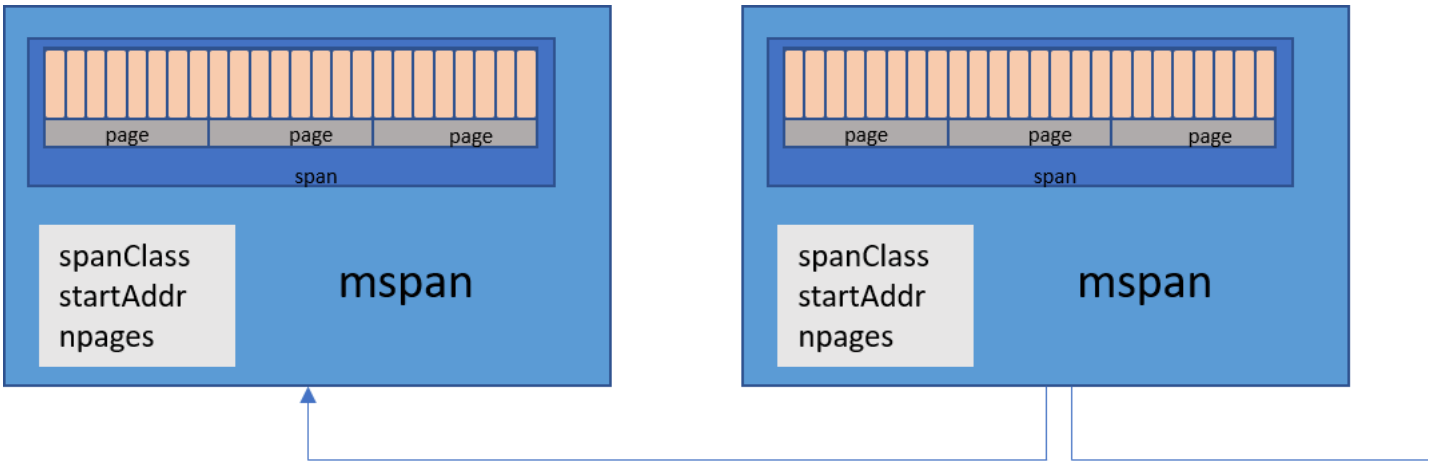
将8 KB页面划分为1KB的大小等级（在Go中，页的粒度保持为8KB）

Go中的这些页面运行也通过称为mspan的结构进行管理。

选择要分配给每个尺寸级别的尺寸类别和页面计数（将页面数分成给定尺寸的对象），以便将分配请求圆整(四舍五入)到下一个尺寸级别最多浪费12.5%

mspan

简而言之，它是一个双向链表对象，其中包含页面的起始地址，它具有的页面的span类以及它包含的页面数。

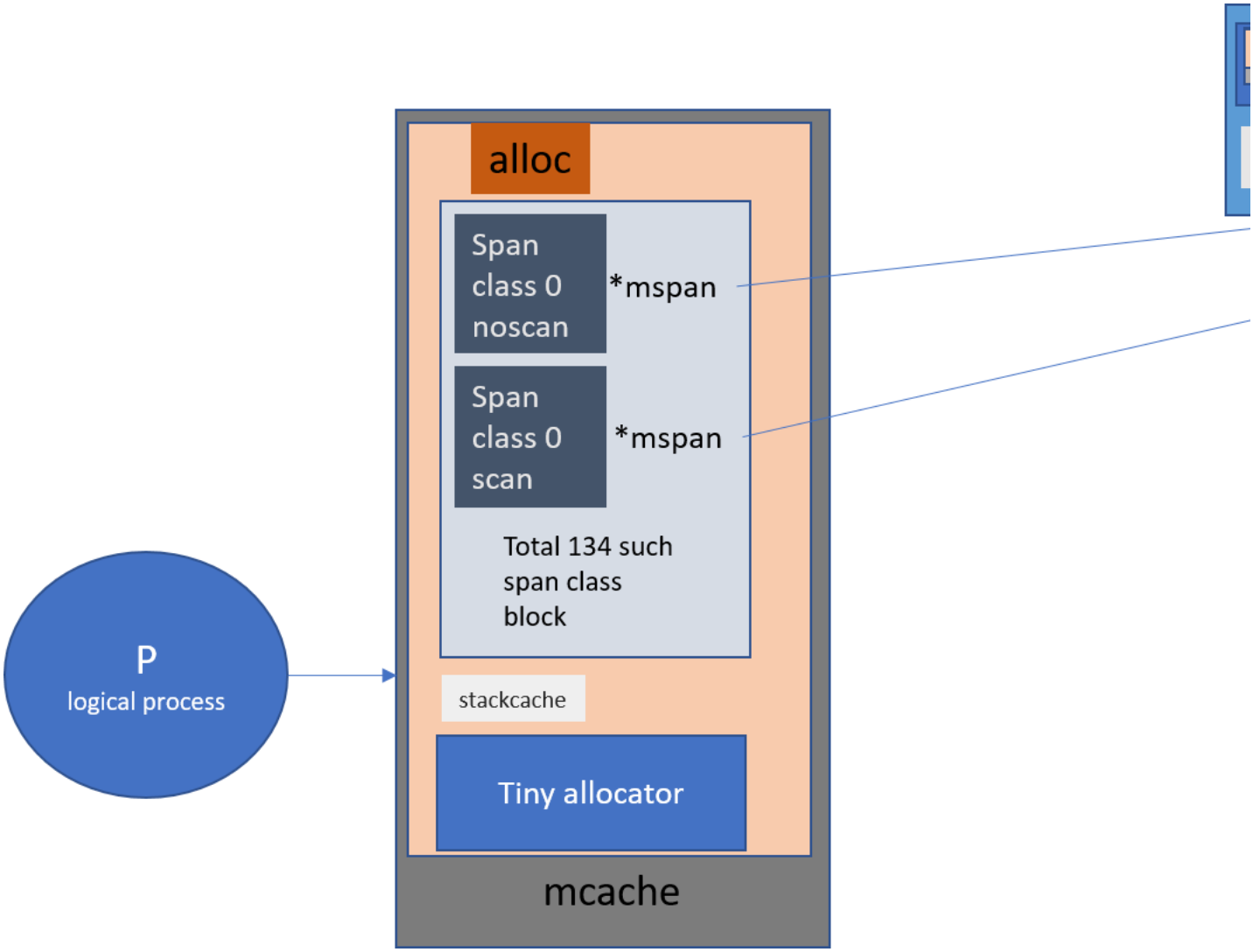


Go内存分配器中mspan的表示形式

mcache

与TCMalloc一样，Go为每个逻辑处理器（P）提供了一个称为**mcache**的本地内存线程缓存，因此，如果Goroutine需要内存，它可以直接从mcache中获取它而无需任何锁，因为在任何时间点只有一个Goroutine在逻辑处理器（P）上运行。

mcache包含所有级别大小的mspan作为缓存。



Go中P，mcache和mspan之间的关系

由于每个P拥有一个mcache，因此从mcache进行分配时无需加锁。

对于每个级别，都有两种类型。
* scan —包含指针的对象。
* noscan —不包含指针的对象。

这种方法的好处之一是在进行垃圾收集时，GC无需遍历noscan对象。

什么Go mcache?

对象大小<= 32K字节的分配将直接交给mcache，后者将使用对应大小级别的mspan应对

当mcache没有可用插槽(slot)时会发生什么?

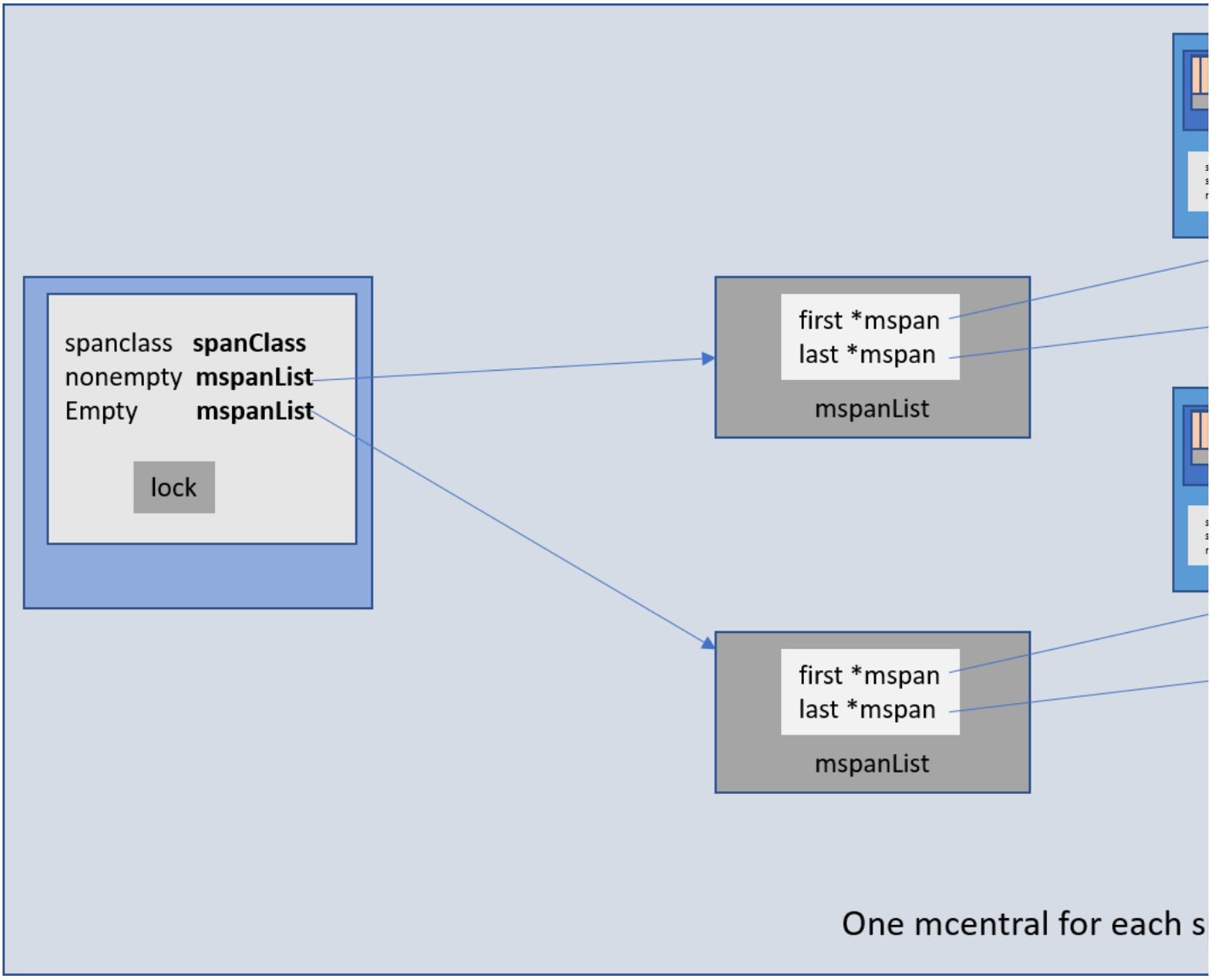
从mcentral mspan list中获取一个对应大小级别的新的mspan。

mcentral

mcentral对象集合了所有给定大小级别的span，每个mcentral是两个mspan列表。

1. 空的mspanList — 没有空闲内存的mspan或缓存在mcache中的mspan的列表
2. 非空mspanList – 仍有空闲内存的span列表。

当从mcentral请求新的Span时，它将从非空mspanList列表中获取（如果可用）。这两个列表之间的关系如下：当请求新的span时，该请求从非空列表中得到满足，并且该span被放入空列表中。释放span后，将根据span中空闲对象的数量将其放回非空列表。

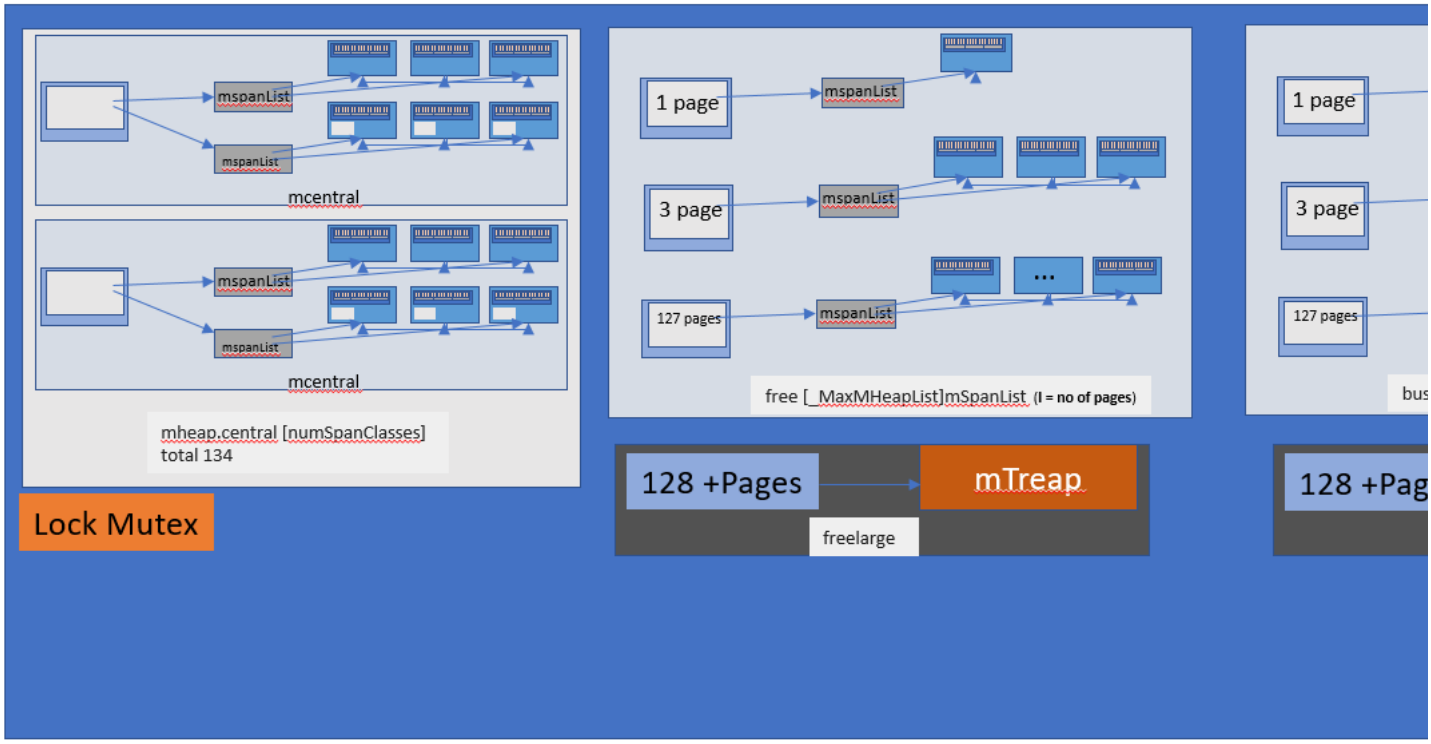


mcentral表示

每个mcentral结构都在mheap中维护。

mheap

mheap是在Go中管理堆的对象，且只有一个全局mheap对象。它拥有虚拟地址空间。



mheap的表示

从上图可以看出，mheap具有一个mcentral数组。此数组包含每个大小级别span的mcentral。

```
central [numSpanClasses]struct {
      mcentral mcentral
        pad      [sys.CacheLineSize unsafe.Sizeof(mcentral{})%sys.CacheLineSize]byte
}
```

由于我们对每个级别的span都有mcentral，因此当mcache从mcentral请求一个mspan时，仅涉及单个mcentral级别的锁，因此其他mache的不同级别mspan的请求也可以同时被处理。

padding确保将MCentrals以CacheLineSize字节间隔开，以便每个MCentral.lock获得自己的缓存行，以避免错误的共享问题。

那么，当该mcentral列表为空时会发生什么？mcentral将从mheap获取页以用于所需大小级别span的分配。

- free [_MaxMHeapList]mSpanList：这是一个spanList数组。每个spanList中的mspan由1～127(_MaxMHeapList-1)页组成。例如，free[3]是包含3个页面的mspan的链接列表。Free表示空闲列表，即尚未进行对象分配。它对应于忙碌列表(busy list)。
- freelarge mSpanList：mspans列表。每个mspan的页数大于127。Go内存分配器以mtreap数据结构来维护它。对应busyLarge。

  大小> 32k的对象是一个大对象，直接从mheap分配。这些较大的请求需要中央锁(central lock)，因此在任何给定的时间点只能满足一个P的请求

五. 对象分配流程

- 大小> 32k是一个大对象，直接从mheap分配。
- 大小<16B，使用mcache的tiny分配器分配
- 大小在16B～32k之间，计算要使用的sizeClass，然后在mcache中使用相应的sizeClass的块分配
- 如果与mcache对应的sizeClass没有可用的块，则向mcentral发起请求。
- 如果mcentral也没有可用的块，则向mheap请求。mheap使用BestFit查找最合适的mspan。如果超出了申请的大小，则会根据需要进行划分，以返回用户所需的页面数。其余页面构成一个新的mspan，并返回mheap空闲列表。
- 如果mheap没有可用的span，请向操作系统申请一组新的页（至少1MB）。

  但是Go在OS级别分配的页面甚至更大（称为arena）。分配大量页面将分摊与操作系统进行对话的成本。

所有请求的堆内存都来自于arena。让我们看看arena是什么。

六. Go虚拟内存

让我们看一个简单go程序的内存。

```
func main () {
    for {}
}
```

```
root@f82921e4c77e:~# ps ux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.1  18196  3228 ?        Ss   08:49   0:00 /bin/bash
root        204  100  0.0 101820   696 ?        Rl   09:05  16:03 ./hello
root        211  0.0  0.1  36640  2816 ?        R+   09:21   0:00 ps ux
root@f82921e4c77e:~# go version
go version go1.11.5 linux/amd64
```

程序的进程状态

因此，即使是简单的go程序，占用的虚拟空间也是大约100MB而RSS只有696kB。让我们尝试首先找出这种差异的原因。

```
00400000-0044f000 r-xp 00000000 08:01 2364609                            /root/hello
0044f000-004b7000 r--p 0004f000 08:01 2364609                            /root/hello
004b7000-004ba000 rw-p 000b7000 08:01 2364609                            /root/hello
004ba000-004d9000 rw-p 00000000 00:00 0
c000000000-c000200000 rw-p 00000000 00:00 0
c000200000-c004000000 rw-p 00000000 00:00 0
7f0b2583e000-7f0b27aae000 rw-p 00000000 00:00 0
7ffc6ba12000-7ffc6ba33000 rw-p 00000000 00:00 0                          [stack]
7ffc6bae8000-7ffc6baea000 r--p 00000000 00:00 0                          [vvar]
7ffc6baea000-7ffc6baec000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
root@f82921e4c77e:~#
root@f82921e4c77e:~# cat /proc/204/smaps | grep  -B 1 -w "Size"
00400000-0044f000 r-xp 00000000 08:01 2364609                            /root/hello
Size:                316 kB
--
0044f000-004b7000 r--p 0004f000 08:01 2364609                            /root/hello
Size:                416 kB
--
004b7000-004ba000 rw-p 000b7000 08:01 2364609                            /root/hello
Size:                 12 kB
--
004ba000-004d9000 rw-p 00000000 00:00 0
Size:                124 kB
--
c000000000-c000200000 rw-p 00000000 00:00 0
Size:               2048 kB
--
c000200000-c004000000 rw-p 00000000 00:00 0
Size:              63488 kB
--
7f0b2583e000-7f0b27aae000 rw-p 00000000 00:00 0
Size:              35264 kB
--
7ffc6ba12000-7ffc6ba33000 rw-p 00000000 00:00 0                          [stack]
Size:                136 kB
--
7ffc6bae8000-7ffc6baea000 r--p 00000000 00:00 0                          [vvar]
Size:                  8 kB
--
7ffc6baea000-7ffc6baec000 r-xp 00000000 00:00 0                          [vdso]
Size:                  8 kB
--
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
Size:                  4 kB
root@f82921e4c77e:~#
```

map和smap统计信息

因此，内存区域的大小约为～2MB, 64MB and 32MB。这些是什么?

Arena

原来，Go中的虚拟内存布局由一组arena组成。初始堆映射是一个arena，即64MB（基于go 1.11.5）。

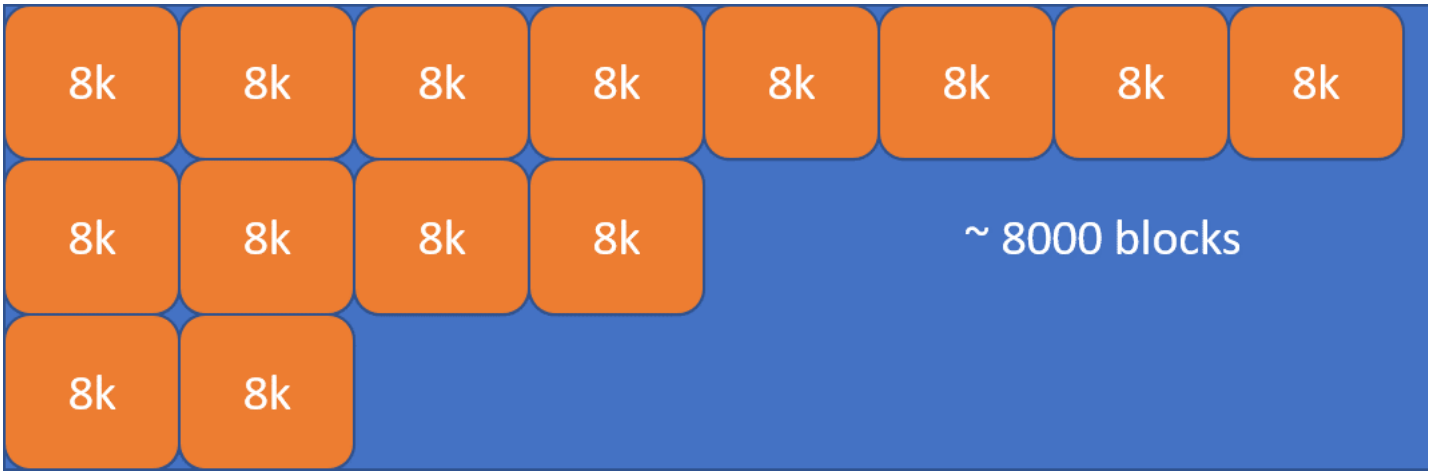
Platform	Addr bits	Arena size	L1 entries	L2 size
*/64-bit	48	64MB	1	32MB
windows/64-bit	48	4MB	64	8MB
*/32-bit	32	4MB	1	4KB
*/mips(le)	31	4MB	1	2KB

当前在不同系统上的arena大小。

因此，当前根据程序需要，内存以较小的增量进行映射，并且它以一个arena（～64MB）开始。

这是可变的。早期的go保留连续的虚拟地址，在64位系统上，arena大小为512 GB。（如果分配足够大并且被mmap拒绝，会发生什么？）

这个arena集合是我们所谓的堆。Go以8192B大小粒度的页面管理每个arena。



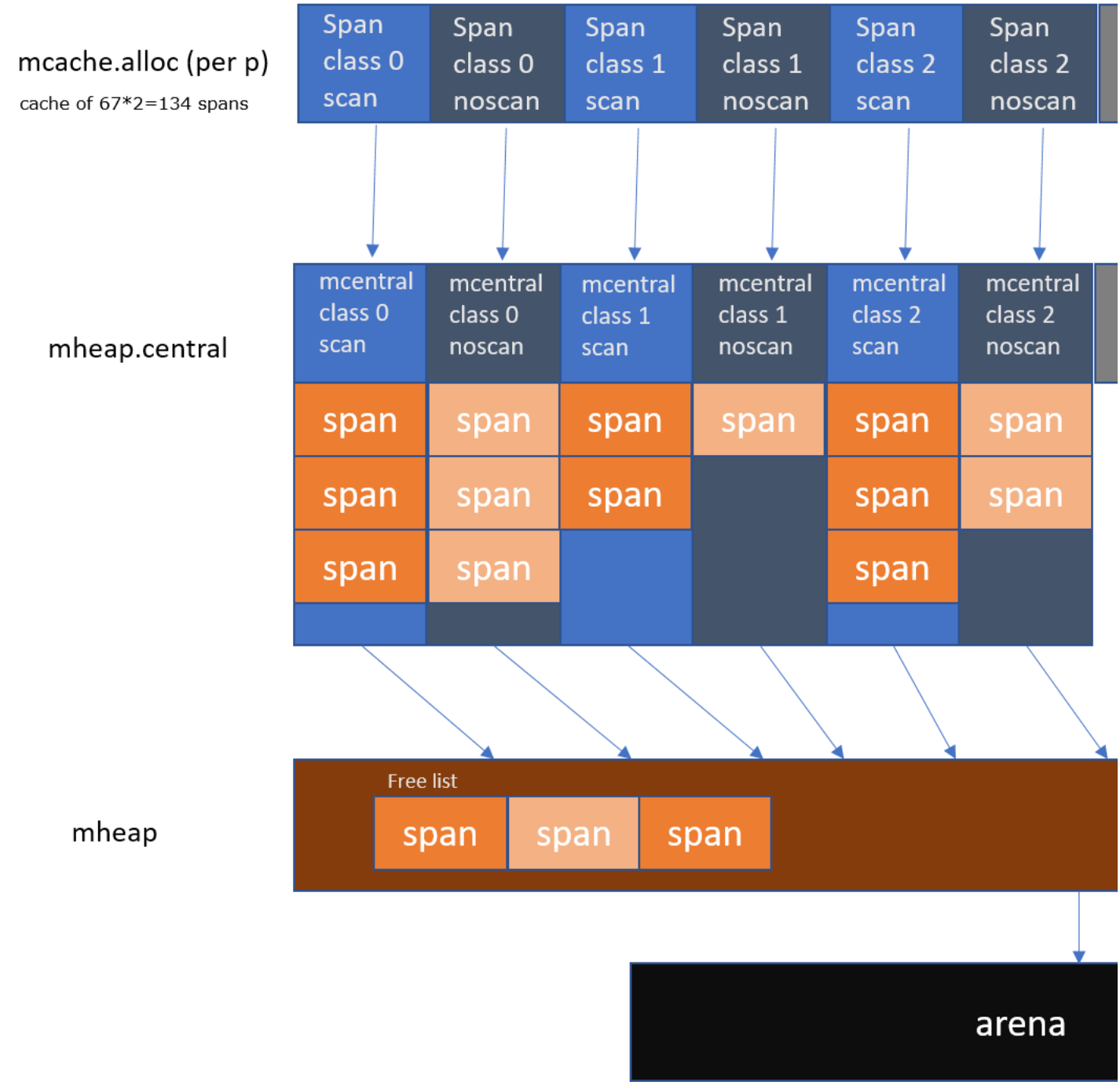
单个arena（64 MB）。

Go还有两个span和bitmap块。它们都在堆外分配，并存储着每个arena的元数据。它主要在垃圾收集期间使用（因此我们现在将其保留）。

我们刚刚讨论过的Go中的内存分配策略，但这些也仅是奇妙多样的内存分配的一些皮毛。

但是，Go内存管理的总体思路是使用不同的内存结构为不同大小的对象使用不同的缓存级别内存来分配内存。将从操作系统接收的单个连续地址块分割为多级缓存以减少锁的使用，从而提高内存分配效率，然后根据指定的大小分配内存分配，从而减少内存碎片，并在内存释放houhou有利于更快的GC。

现在，我将向您提供此Go Memory Allocator的全景图。



运行时内存分配器的可视化全景图。

我的网课“Kubernetes实战：高可用集群搭建、配置、运维与应用”在慕课网上线了，感谢小伙伴们学习支持！

我爱发短信：企业级短信平台定制开发专家 https://51smspush.com/
smspush : 可部署在企业内部的定制化短信平台，三网覆盖，不惧大并发接入，可定制扩展； 短信内容你来定，不再受约束, 接口丰富，支持长短信，签名可选。

著名云主机服务厂商DigitalOcean发布最新的主机计划，入门级Droplet配置升级为：1 core CPU、1G内存、25G高速SSD，价格5$/月。有使用DigitalOcean需求的朋友，可以打开这个链接地址：https://m.do.co/c/bff6eed92687 开启你的DO主机之路。

Gopher Daily(Gopher每日新闻)归档仓库 – https://github.com/bigwhite/gopherdaily

我的联系方式：

微博：https://weibo.com/bigwhite20xx
微信公众号：iamtonybai
博客：tonybai.com
github: https://github.com/bigwhite

微信赞赏：

商务合作方式：撰稿、出书、培训、在线课程、合伙创业、咨询、广告合作。

Related posts:

1. Go语言是如何处理栈的
2. 也谈goroutine调度器
3. Go语言回顾：从Go 1.0到Go 1.13
4. Go 1.6中值得关注的几个变化
5. Goroutine是如何工作的

添加新评论

称呼

邮箱

网站 http://example.com

提交评论

Capatcha

输入关键字搜索 搜索

欢迎使用邮件订阅我的博客

输入邮箱订阅本站，只要有新文章发布，就会第一时间发送邮件通知你哦！

名字： 请输入您的名字

邮箱: 请输入您的电子邮箱

马上订阅

这里是 Tony Bai的个人Blog，欢迎访问、订阅和留言！ 订阅Feed请点击上面图片。

如果您觉得这里的文章对您有帮助，请扫描上方二维码进行捐赠 ，加油后的Tony Bai将会为您呈现更多精彩的文章，谢谢！

如果您希望通过微信捐赠，请用微信客户端扫描下方赞赏码：

如果您希望通过比特币或以太币捐赠，可以扫描下方二维码：

比特币：

以太币：

如果您喜欢通过微信浏览本站内容，可以扫描下方二维码，订阅本站官方微信订阅号“iamtonybai”；点击二维码，可直达本人官方微博主页^_^：

我的业余项目

- smspush短信发送平台

文章

- Go经典阻塞式TCP协议流解析的实践
- 一文搞懂Go语言的plugin
- 一文告诉你如何用好uber开源的zap日志库
- 使用section.key的形式读取ini配置项
- 使用go-metrics在Go应用中增加度量
- 通过实例理解Go Execution Tracer
- 使用functrace辅助进行Go项目源码分析
- 通过实例理解Go逃逸分析
- minikube v1.20.0版本的一个bug
- Go标准库http与fasthttp服务端性能比较

评论

- 正在加载...
-
-

分类

- 光影汇 (7)
- 影音坊 (36)
- 思考控 (66)
- 技术志 (671)
- 教育记 (2)
- 杂货铺 (75)
- 生活簿 (157)
- 职场录 (14)
- 读书吧 (14)
- 运动迷 (107)
- 驴友秀 (40)

标签

Blog Blogger C Cpp docker GCC github GNU Go Golang Google Gopher goroutine http Java k8s Kernel Kubernetes Linux M10 Opensource Programmer Python Solaris Subversion Ubuntu Unix Windows 博客 学习 容器 工作 巴萨 开源 思考 感悟 摄影 旅游 标准库 梅西 生活 程序员 编译器 西甲 足球

归档

- 2021 年七月 (5)
- 2021 年六月 (2)
- 2021 年五月 (2)
- 2021 年四月 (7)
- 2021 年三月 (7)
- 2021 年二月 (5)
- 2021 年一月 (4)
- 2020 年十二月 (11)
- 2020 年十一月 (9)
- 2020 年十月 (1)
- 2020 年九月 (1)
- 2020 年八月 (1)
- 2020 年七月 (1)
- 2020 年六月 (4)
- 2020 年五月 (3)
- 2020 年四月 (2)
- 2020 年三月 (6)
- 2020 年二月 (2)
- 2019 年十二月 (2)
- 2019 年十一月 (6)
- 2019 年十月 (5)

- 2019 年九月 (4)
- 2019 年八月 (5)
- 2019 年七月 (1)
- 2019 年六月 (2)
- 2019 年五月 (1)
- 2019 年四月 (4)
- 2019 年三月 (2)
- 2019 年二月 (1)
- 2019 年一月 (2)
- 2018 年十一月 (3)
- 2018 年十月 (1)
- 2018 年九月 (1)
- 2018 年七月 (1)
- 2018 年六月 (4)
- 2018 年五月 (2)
- 2018 年四月 (1)
- 2018 年三月 (3)
- 2018 年二月 (3)
- 2018 年一月 (7)
- 2017 年十二月 (5)
- 2017 年十一月 (4)
- 2017 年十月 (3)
- 2017 年九月 (2)
- 2017 年八月 (3)
- 2017 年七月 (4)
- 2017 年六月 (8)
- 2017 年五月 (5)
- 2017 年四月 (3)
- 2017 年三月 (2)
- 2017 年二月 (5)
- 2017 年一月 (7)
- 2016 年十二月 (7)
- 2016 年十一月 (7)
- 2016 年十月 (3)
- 2016 年九月 (2)
- 2016 年八月 (1)
- 2016 年六月 (2)
- 2016 年五月 (2)
- 2016 年四月 (2)
- 2016 年三月 (2)
- 2016 年二月 (3)
- 2016 年一月 (2)
- 2015 年十二月 (1)
- 2015 年十一月 (1)
- 2015 年十月 (1)
- 2015 年九月 (3)
- 2015 年八月 (5)
- 2015 年七月 (6)
- 2015 年六月 (4)
- 2015 年五月 (1)
- 2015 年四月 (2)
- 2015 年三月 (2)
- 2015 年一月 (2)
- 2014 年十二月 (5)
- 2014 年十一月 (8)
- 2014 年十月 (9)
- 2014 年九月 (2)
- 2014 年八月 (1)
- 2014 年七月 (1)
- 2014 年五月 (2)
- 2014 年四月 (5)
- 2014 年三月 (4)
- 2014 年二月 (1)
- 2014 年一月 (1)
- 2013 年十二月 (3)

- 2013 年十一月 (5)
- 2013 年十月 (6)
- 2013 年九月 (4)
- 2013 年八月 (5)
- 2013 年七月 (6)
- 2013 年六月 (2)
- 2013 年五月 (6)
- 2013 年四月 (3)
- 2013 年三月 (7)
- 2013 年二月 (4)
- 2013 年一月 (6)
- 2012 年十二月 (8)
- 2012 年十一月 (10)
- 2012 年十月 (5)
- 2012 年九月 (3)
- 2012 年八月 (10)
- 2012 年七月 (4)
- 2012 年六月 (2)
- 2012 年五月 (4)
- 2012 年四月 (10)
- 2012 年三月 (8)
- 2012 年二月 (6)
- 2012 年一月 (6)
- 2011 年十二月 (4)
- 2011 年十一月 (4)
- 2011 年十月 (5)
- 2011 年九月 (8)
- 2011 年八月 (7)
- 2011 年七月 (6)
- 2011 年六月 (7)
- 2011 年五月 (8)
- 2011 年四月 (6)
- 2011 年三月 (10)
- 2011 年二月 (7)
- 2011 年一月 (10)
- 2010 年十二月 (7)
- 2010 年十一月 (6)
- 2010 年十月 (7)
- 2010 年九月 (12)
- 2010 年八月 (8)
- 2010 年七月 (3)
- 2010 年六月 (5)
- 2010 年五月 (4)
- 2010 年四月 (2)
- 2010 年三月 (6)
- 2010 年二月 (4)
- 2010 年一月 (6)
- 2009 年十二月 (6)
- 2009 年十一月 (6)
- 2009 年十月 (5)
- 2009 年九月 (8)
- 2009 年八月 (8)
- 2009 年七月 (8)
- 2009 年六月 (2)
- 2009 年五月 (5)
- 2009 年四月 (7)
- 2009 年三月 (12)
- 2009 年二月 (9)
- 2009 年一月 (15)
- 2008 年十二月 (9)
- 2008 年十一月 (5)
- 2008 年十月 (10)
- 2008 年九月 (13)
- 2008 年八月 (13)
- 2008 年七月 (3)

- 2008 年六月 (1)
- 2008 年五月 (7)
- 2008 年四月 (4)
- 2008 年三月 (9)
- 2008 年二月 (11)
- 2008 年一月 (15)
- 2007 年十二月 (11)
- 2007 年十一月 (14)
- 2007 年十月 (4)
- 2007 年九月 (5)
- 2007 年八月 (1)
- 2007 年七月 (10)
- 2007 年六月 (10)
- 2007 年五月 (10)
- 2007 年四月 (8)
- 2007 年三月 (15)
- 2007 年二月 (4)
- 2007 年一月 (17)
- 2006 年十二月 (18)
- 2006 年十一月 (9)
- 2006 年十月 (11)
- 2006 年九月 (6)
- 2006 年八月 (5)
- 2006 年七月 (22)
- 2006 年六月 (35)
- 2006 年五月 (24)
- 2006 年四月 (26)
- 2006 年三月 (25)
- 2006 年二月 (18)
- 2006 年一月 (15)
- 2005 年十二月 (10)
- 2005 年十一月 (10)
- 2005 年九月 (13)
- 2005 年八月 (11)
- 2005 年七月 (6)
- 2005 年六月 (2)
- 2005 年五月 (3)
- 2005 年四月 (6)
- 2005 年三月 (1)
- 2005 年一月 (15)
- 2004 年十二月 (9)
- 2004 年十一月 (14)
- 2004 年十月 (2)
- 2004 年九月 (2)

私人

- 我的二女儿
- 我的大女儿

链接

- @douban
- @flickr
- @github
- @googlecode
- @picasa
- @slideshare
- @twitter
- @weibo
- Hoterran
- Lionel Messi
- Puras He
- 梦想风暴
- 磊磊落落的博客
- 过眼云烟

开源项目

- buildc
- cbehave
- lcut

翻译项目

- C语言编码风格和标准
- 《Programming in Haskell》中文翻译项目