Лекция Go to memory

Аллокации:

Golang запрашивает большую область памяти (арена),
чтобы не просить у ОС каждый раз по маленькому кусочку памяти.

Структура запрашиваемых арен в разных операционных системах:

```
  (1 << addr bits) = arena size * L1 entries * L2 entries

Currently, we balance these as follows:

      Platform  Addr bits  Arena size  L1 entries   L2 entries
--------------  ---------  ----------  ----------  -----------
      */64-bit         48        64MB           1    4M (32MB)
windows/64-bit         48         4MB          64    1M  (8MB)
     ios/arm64         40         4MB           1  256K  (2MB)
      */32-bit         32         4MB           1  1024  (4KB)
    */mips(le)         31         4MB           1   512  (2KB)
```

Фрагментация арены:

После получения арены от ОС, Go разделяет ее на страницы (pages) по 8 KB.
Всеми страницами управляет структура `heapArena` (`mheap`);

Диаграмма Арены в памяти:

```
┌─────────────────────────────────────────────┐
│                One Heap Arena               │
│          (64 MB continuous memory)          │
├─────────────────────────────────────────────┤
│ mspan[0]   → class 0 (объекты размером 8 B)  │
│ mspan[1]   → class 1 (объекты размером 16 B) │
│ mspan[2]   → class 2 (объекты размером 32 B) │
│ ...                                         │
│ mspan[n]   → объекты бОльших размеров        │
├─────────────────────────────────────────────┤
│ Each mspan contains multiple slots for objs  │
```

```
| of the same size class, tracked by bitmap.    |
└──────────────────────────────────────────────┘
```

Каждая арена состоит из набора `mspan` - блоков памяти фиксированного размера, где каждый `mspan` хранит объекты **одного класса размера** (например, 16 байт, 32 байта, 64 байта и т.д.).

Взаимосвязь структур:

```
┌──────────┐
| mheap    |   ← управляет всеми аренами
└──────────┘
     |
     ▼
┌──────────┐
| arena 0  |
| arena 1  |   ← арены
| ...      |
└──────────┘
     |
     ▼
┌──────────┐
| mspan[]  |   ← делят арену на куски по классам размера
└──────────┘
     |
     ▼
┌──────────┐
| objects  |   ← конкретные аллоцированные объекты Go
└──────────┘
```

Элементы `Heap`:

Элемент	Размер / Значение	Краткое описание
`Arena`	4 MB, либо 64 MB (в зависимости от системы)	Основная единица выделения heap'а
`Page`	8 KB	Минимальная единица внутри `Arena`
`Span`	Несколько страниц	Группа страниц, выделенная под класс размера
`Object`	8 B - 32 KB	Конкретный объект Go в `span`

Связь `mcentral` и `mspan`:

```
mheap
  ├── arena[0x10000000]
  ├── arena[0x14000000]
  │
  ├── mcentral[16B] ──► mspan1 ↔ mspan2 ↔ mspan3
  ├── mcentral[32B] ──► mspan4 ↔ mspan5
  └── mcentral[64B] ──► mspan6 ↔ mspan7 ↔ mspan8
```

Полная структура управления памятью `Go Heap`:

```
mheap
│
├── Arena #0 (64 MB)
│   │
│   ├── mcentral[8B]  ← для объектов размером 8 байт
│   │   │
│   │   ├── mspan #1 (2 pages = 16 KB)
│   │   │   │
│   │   │   ├── Page #0 (8 KB)
│   │   │   │   ├── Obj #0 (8B)
│   │   │   │   ├── Obj #1 (8B)
│   │   │   │   ├── ...
│   │   │   │   └── Obj #1023 (8B)
│   │   │   │
│   │   │   └── Page #1 (8 KB)
│   │   │       ├── Obj #0 (8B)
│   │   │       └── ...
│   │   │
│   │   └── mspan #2 (4 pages = 32 KB)
│   │       └── ... аналогично ...
│   │
│   ├── mcentral[32B]  ← для объектов по 32 байта
│   │   │
│   │   ├── mspan #3 (1 page = 8 KB)
│   │   │   ├── Page #0
│   │   │   │   ├── Obj #0   (32B)
│   │   │   │   ├── Obj #1   (32B)
│   │   │   │   ├── ...
│   │   │   │   └── Obj #255 (32B)
```

```
│   │   │   │      (8 KB / 32 B = 256 объектов)
│   │   │   │
│   │   │   └── ...
│   │   │
│   │   └── mspan #4 (2 pages)
│   │       └── ... аналогично ...
│   │
│
└── Arena #1 (64 MB)
    ├── mcentral[8B]
    ├── mcentral[16B]
    ├── mcentral[32B]
    └── ...
```

Структуры диаграммы:

- `mheap` - главный менеджер, знает обо всех аренах;
- `arena` - большой кусок памяти (64 MB), который ОС отдала во владение Go;
- `mcentral[class]` - менеджер всех `span` для одного размера класса;
- `mspan` - набор последовательных страниц (обычно 8 KB * N),
  где все объекты одного размера;
- `page` - физический участок 8 KB;
- `object` - выделенный пользователем объект Go.

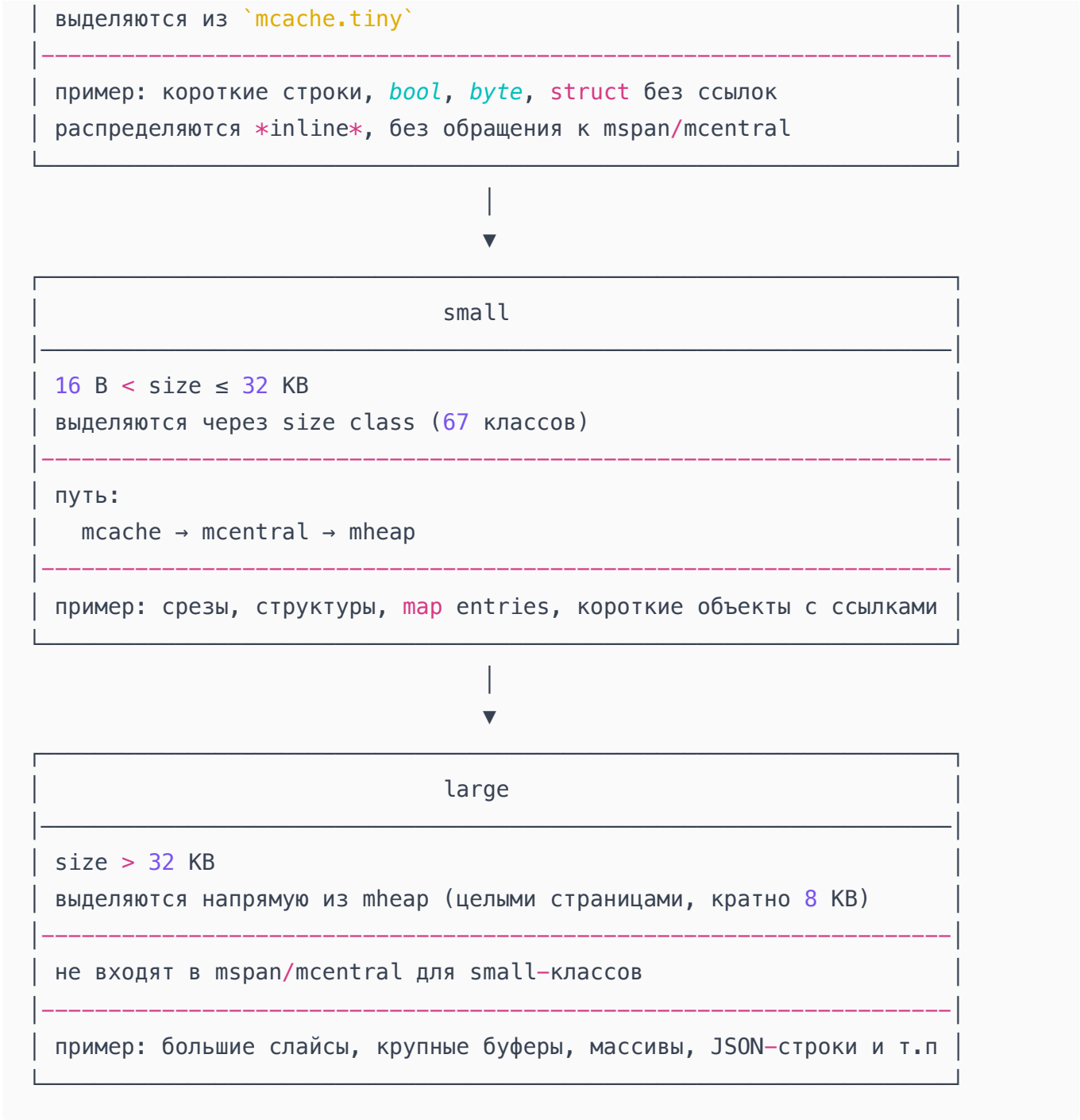
Concurrency:

Если в нашей программе существует несколько тредов GMP,
то при одновременном доступе к общему участку памяти появится состояние гонки (race condition).

Для избежания такой ситуации в Go тред сначала идет в свой блок кэша,
которым управляет `mcache`;

Категории аллокации объектов:

```
┌──────────────────────────────────────────────────────────────┐
│                            tiny                              │
├──────────────────────────────────────────────────────────────┤
│ size ≤ 16 B (без указателей)                                 │
│ shared allocation: несколько tiny-объектов в одном 16-B слоте│
```



Ключевые пороги:

Категория	Размер	Куда идет	Особенности
`Tiny`	≤ 16 B	`macche.tiny`	Без указателей, упаковка нескольких объектов в один слот
`Small`	16 B – 32 KB	`mcache` → `mcentral` → `mheap`	67 size classes, по одному `mcentral` на класс
`Large`	> 32 KB	Напрямую из `mheap`	Кратное страницам (8 KB) выделение

Tiny allocation:

Алгоритм:

- Горутина запрашивает блок менее 16 B;
- Компилятор определяет его в категорию `tiny`,
- Укомплектовывает все блоки в кэше по максимуму;
- Если все `mspan` уже забиты:
  - `lock` нужного `mcentral`;
  - запрос `mspan` с объектами нужного размера;
  - укомплектация `mspan`.

Small allocation:

Алгоритм:

- `lock` нужного `mcentral`;
- запрос `mspan` с объектами нужного размера;
- укомплектация `mspan`.

Large allocatinon:

Алгоритм:

- Ставится `lock`;
- Записывается значение в арену;