

# Использование ресурсов Kubernetes Go-приложениями

Нюансы работы Go-приложений с  
GOMAXPROCS на облачных серверах



# О чём будем говорить

- Введение
- Основы
  - CFS
  - Goroutines
  - Cgroups
- Kubernetes
  - Requests
  - Limits
- GOMAXPROCS
- Рекомендации

- **Введение**
- Основы
  - CFS
  - Goroutines
  - Cgroups
- Kubernetes
  - Requests
  - Limits
- GOMAXPROCS
- Рекомендации

# Kubernetes

- Удобство
  - Абстракция над физическими или виртуальными машинами
  - Выкатка приложений одной командой (`kubectl apply`)
- Масштабируемость
  - НРА - горизонтальное масштабирование подов
  - СА - cluster autoscaling - горизонтальное масштабирование нод

# Но иногда Kubernetes - это...

- CPU Throttling
- Container Evicted (MemoryPressure)
- Зашкаливающие 99 персентили по длительности ответа сервера
- Непредсказуемое масштабирование кластера
- Или наоборот - Простаивающие ресурсы на нодах с потреблением CPU, Memory менее 50%

# Что делать?

- Разберемся,
  - как Kubernetes управляет ресурсами через Requests & Limits
  - как эти настройки влияют на утилизацию ресурсов
- Взглянем,
  - как мы тюнили Go-приложение и
    - улучшили 90-ый персентиль длительности ответа на 20%
    - улучшили CPU Usage на 33%
    - улучшили длительность Go GC на 50%

- Введение
- **Основы**
  - **CFS**
  - Goroutines
  - Cgroups
- Kubernetes
  - Requests
  - Limits
- GOMAXPROCS
- Рекомендации

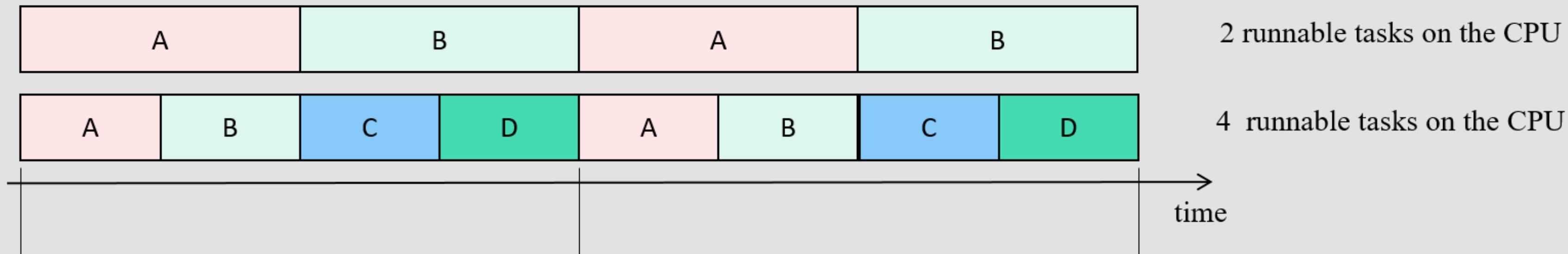
# CFS

- У нас есть один процессор...
- Как бы работал компьютер, если бы он исполнял только 1 приложение в момент времени?
- Современные компьютеры - это всегда многозадачность и многопоточность
- CFS - алгоритм, который реализует многозадачность в рамках процессора

# Completely Fair Scheduler (CFS)

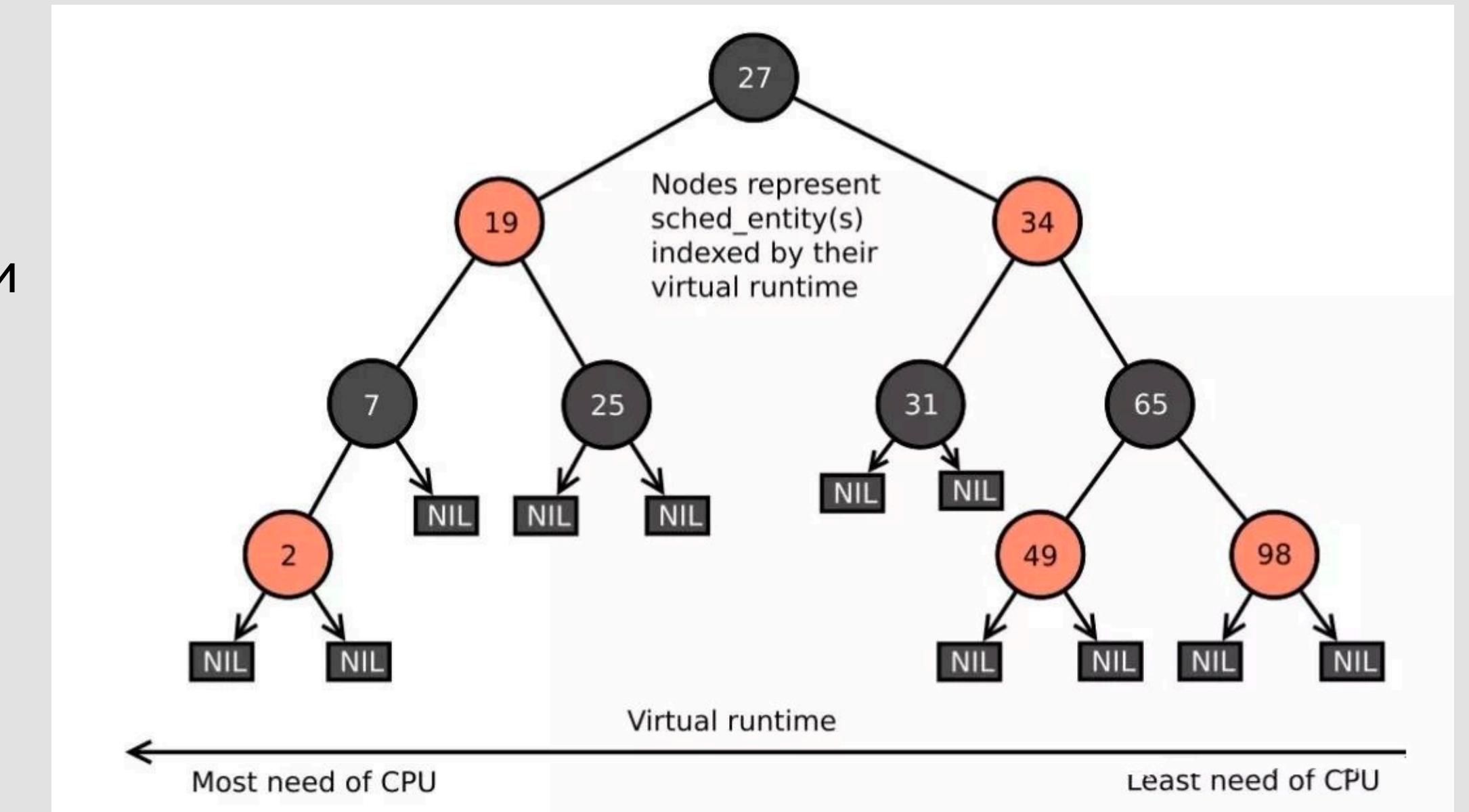
- CFS - моделирует идеально-точную многозадачность CPU
- CPU может выполнять все задачи с равной скоростью
- Время, которое выполнялась задача измеряется в *virtual runtime*

Picture from <https://arthurchiao.art>



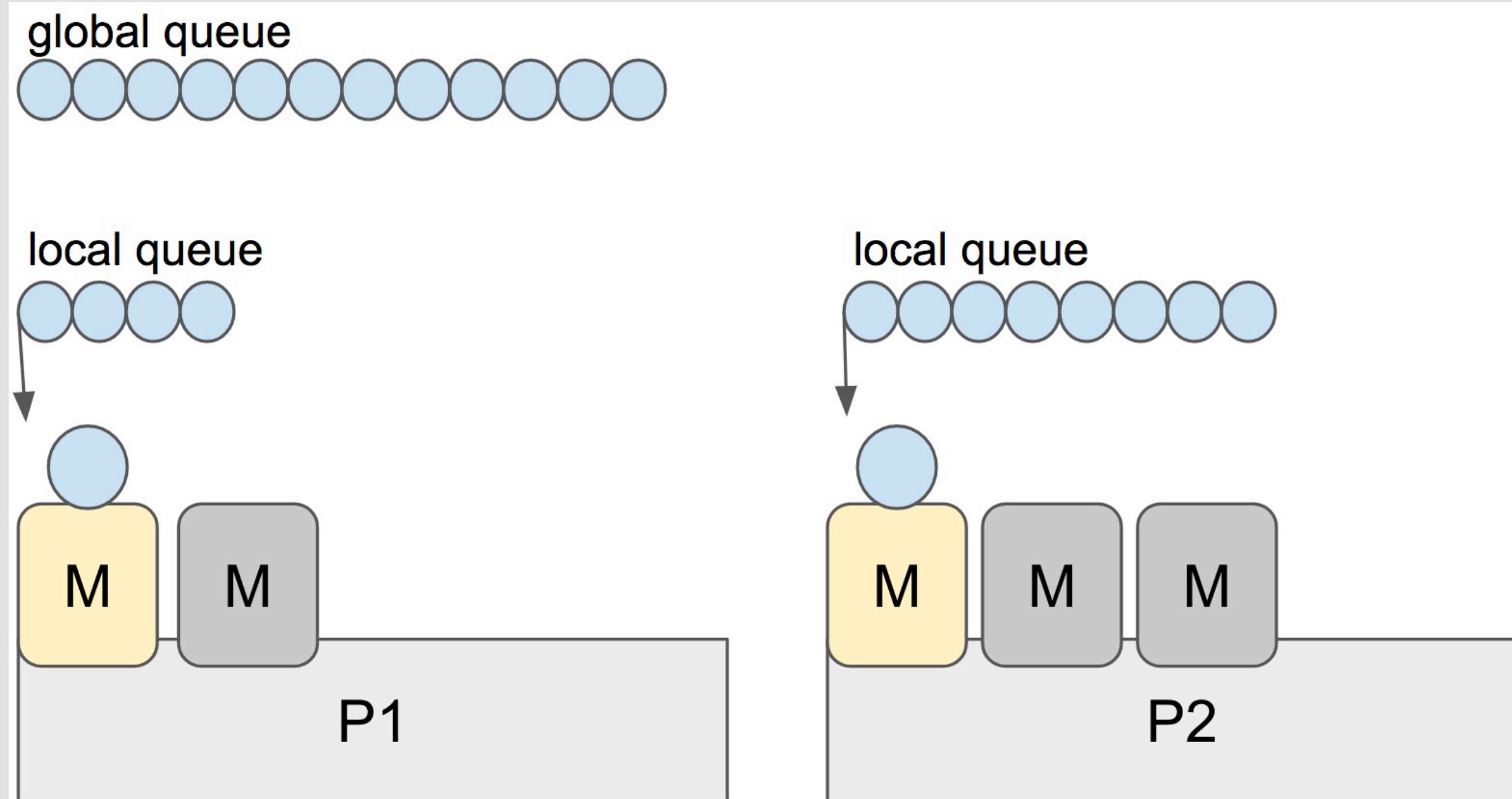
# Completely Fair Scheduler (CFS)

- Бинарное дерево, упорядоченное по vruntime
- CFS выбирает задачи из левой части дерева
- По мере выполнения задач, CFS возвращает их в правую часть и берет новые слева



- Введение
- **Основы**
  - CFS
- **Goroutines**
  - Cgroups
- Kubernetes
  - Requests
  - Limits
- GOMAXPROCS
- Рекомендации

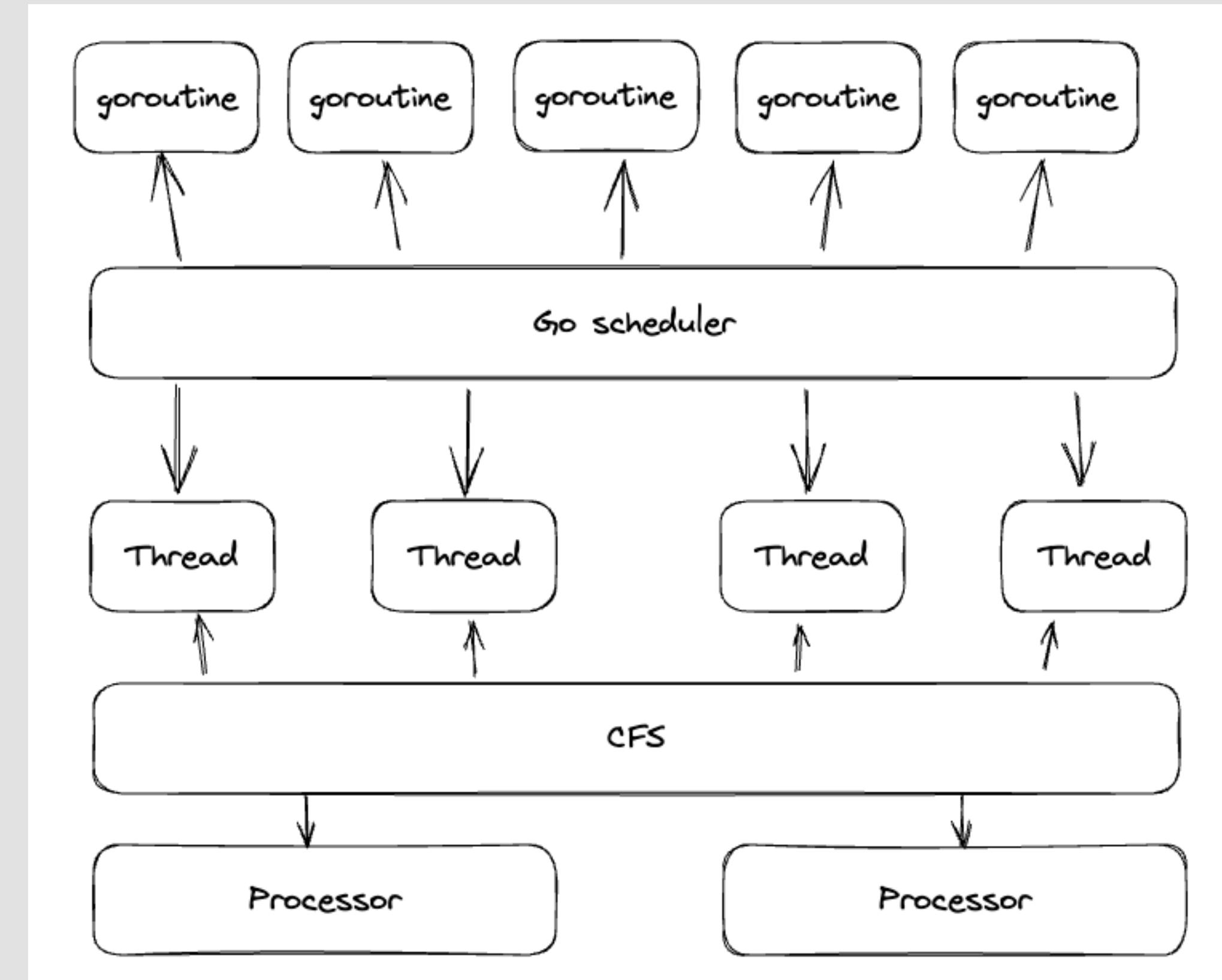
# Goroutine to thread scheduling



- G: goroutine
- M: OS thread  
(machine)
- P: processor

# Goroutines execution

- Go Scheduler приземляет рутины на потоки
- CFS приземляет потоки на процессор



# cgroups

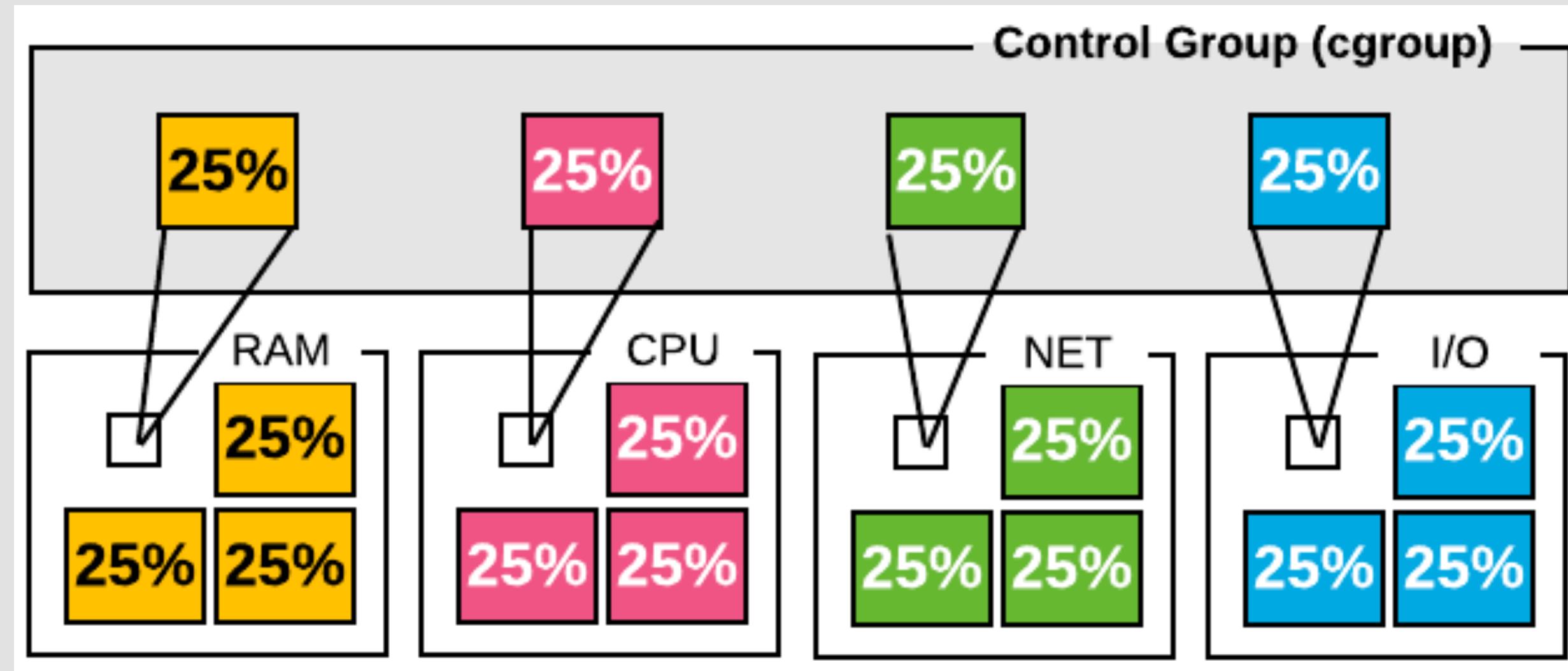
- А если мы не хотим, чтобы все процессы имели одинаковый приоритет?

- Введение
- **Основы**
  - CFS
  - Goroutines
  - **Cgroups**
- Kubernetes
  - Requests
  - Limits
- GOMAXPROCS
- Рекомендации

# Control Groups (cgroups)

- Механизм для:
  - организации процессов в иерархии
  - контролируемого распределения ресурсов по иерархии в соответствии с конфигурацией
- Иерархия групп реализована с помощью дерева
- Каждая группа распоряжается выделенными ей ресурсами
- Основа для контейнеризации приложений (docker, containerd, etc.)

# Control Groups (cgroups)



- Введение
- Основы
  - CFS
  - Goroutines
  - Cgroups
- **Kubernetes**
  - **Requests**
  - Limits
- GOMAXPROCS
- Рекомендации

# Kubernetes Pod Scheduling

- У Ноды есть capacity по каждому типу ресурса
- Планировщик запускает под и вычитает его requests из capacity ноды
- Планировщик не размещает под с requests больше, чем остаточное capacity на ноде
- При нехватке capacity ищет другую ноду или создает новую

# CPU Requests

- Limits & Requests в Kubernetes измеряются в cpu units: 1 cpu = 1000 millicores
- 1 cpu обозначает 1 ядро любого процессора в любом облаке:
  - 1 AWS vCPU
  - 1 GCP Core
  - 1 Azure vCore
  - 1 Hyperthread on a bare-metal Intel processor with Hyperthreading
- При 100% загрузке ноды происходит приоритизация CPU time, используя...

# CPU Shares

- Механизм приоритизации процессорного времени в cgroups
- При 100% загрузке CPU на ноде
- Позволяет на основе весов разделять процессорное время между контейнерами

# Requests

- Контейнеру гарантированы ресурсы, запрошенные в requests
- Контейнер может получить или не получить ресурсы сверх запрошенного
- При 100% загрузке CPU, процессорное время будет распределяться

пропорционально requests (весам)

# Requests

- А что если реальное потребление...
  - Ниже выставленных requests
    - CPU, Memory не заняты эффективно
    - Вы платите за ресурсы, которые не используете

# Requests

- А что если реальное потребление...
  - Выше выставленных requests
    - Замедление производительности приложения
    - Поды могут выселяться при MemoryPressure
    - Непредсказуемое масштабирование кластера (Cluster Autoscaler)

# Requests

- А что если реальное потребление...
  - Соответствует выставленным requests
    - CPU, Memory заняты эффективно
    - Вы используете те ресурсы, за которые платите
    - Лучше корреляция между нагрузкой на кластер и кол-вом нод в кластере

# ИТОГ

- Requests нужны для:
  - Учет и планирование ресурсов ноды
  - При 100% утилизации CPU ноды для приоритизации процессорного времени
- При 100% утилизации Memory - в каком порядке останавливать контейнеры (Evicted, MemoryPressure)

- Введение
- Основы
  - CFS
  - Goroutines
  - Cgroups
- **Kubernetes**
  - Requests
  - Limits
- GOMAXPROCS
- Рекомендации

# Kubernetes Limits

- По умолчанию контейнер имеет неограниченный доступ к ресурсам
- Kubernetes троллит контейнер, в случае если он упирается в лимит по CPU
- Kubernetes останавливает контейнер, в случае если он упирается в лимит по Memory (статус OOMKilled)
- Kubernetes использует CFS Quota для реализации CPU лимитов

# CFS Bandwidth Control

- CFS позволяет определять потолок CPU, используемого группой
- Группе выделяется quota процессорного времени за отрезок времени period
- Когда вся квота использована, потоки в группе ограничиваются (throttle)
- Потоки, исчерпавшие квоту будут запущены только в следующий period

# Примеры

1. Limit a group to 1 CPU worth of runtime.

```
# echo 250000 > cpu.cfs_quota_us /* quota = 250ms */
# echo 250000 > cpu.cfs_period_us /* period = 250ms */
```

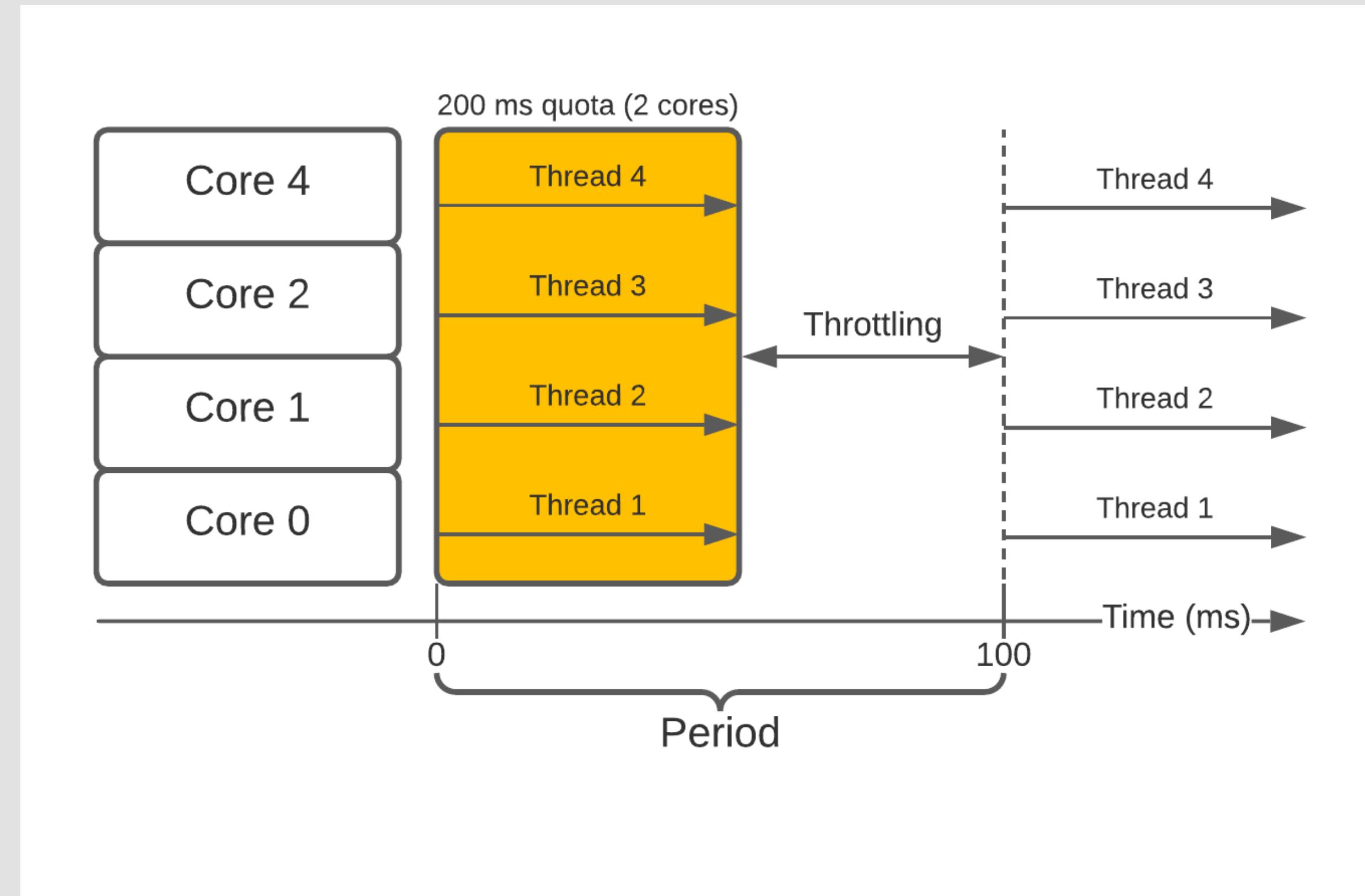
2. Limit a group to 2 CPUs worth of runtime on a multi-CPU machine.

```
# echo 1000000 > cpu.cfs_quota_us /* quota = 1000ms */
# echo 500000 > cpu.cfs_period_us /* period = 500ms */
```

3. Limit a group to 20% of 1 CPU.

```
# echo 10000 > cpu.cfs_quota_us /* quota = 10ms */
# echo 50000 > cpu.cfs_period_us /* period = 50ms */
```

# Примеры



# Важно понимать

- CPU
  - Может выделяться сверх requests
  - Может отниматься, если другой контейнер нуждается в CPU
  - Процессорное время сжимаемое (увеличивается, уменьшается)
- Memory
  - Выделенная память не может быть отобрана у контейнера
  - При достижении limit контейнер останавливается с ООМKilled
  - На потребление памяти нельзя повлиять иначе

# ИТОГ

## - **Limits**

- Выставление жестких порогов утилизации ресурсов
- При достижении порога
  - CPU: Контейнер тротлится
  - Memory: Контейнер останавливается (OOM Kill)

- Введение
- Основы
  - CFS
  - Goroutines
  - Cgroups
- Kubernetes
  - Requests
  - Limits
- **GOMAXPROCS**
- Рекомендации

80 ядер

vs

4 ядра



# GOMAXPROCS=?



# GOMAXPROCS

- Облачные сервера имеют десятки и сотни CPU
- По умолчанию Go использует все доступные

# GOMAXPROCS

- Когда используются все доступные CPU:
- Имеют место высокие накладные расходы на переключение контекста CPU

# GOMAXPROCS

- Если ограничивать приложение с помощью низких CPU Limits:
  - Лимиты размазываются по всем ядрам (CPU Quota делится на кол-во CPU)
  - Может наблюдаться Throttling

# GOMAXPROCS

- При GOMAXPROCS >> Container CPU Limit
  - может наблюдаться CPU Throttling
- При GOMAXPROCS = Container CPU Limit
  - вероятность CPU Throttling минимальная

# Automaxprocs

- <https://github.com/uber-go/automaxprocs>
- Библиотека привязывает GOMAXPROCS к Container CPU Limit
- Рекомендуют выставлять целые числа в CPU Limit

**GOMAXPROCS=80**  
**CPU Limit=0**

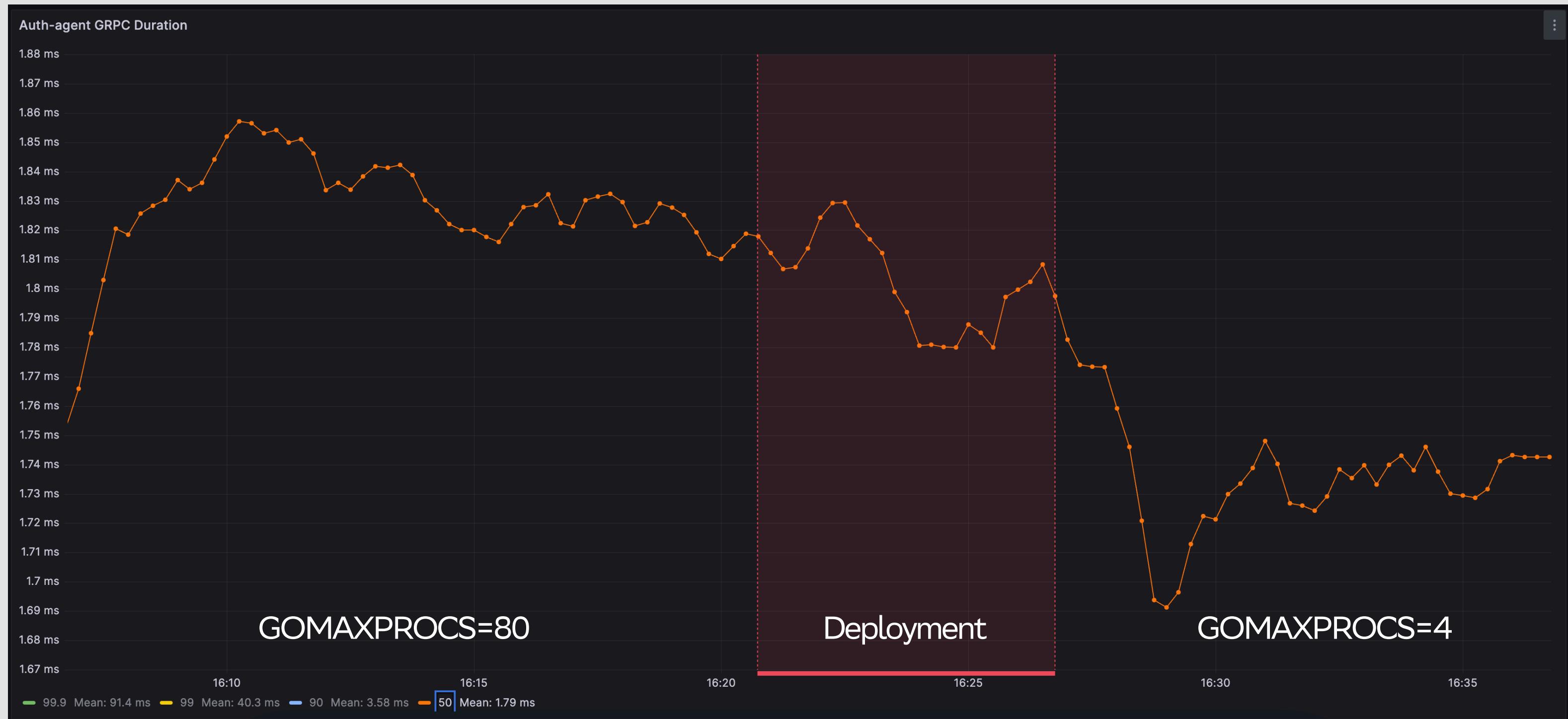
**VS**

**GOMAXPROCS=4**  
**CPU Limit=4**



# GOMAXPROCS

Длительность ответа GRPC 50-ый персентиль



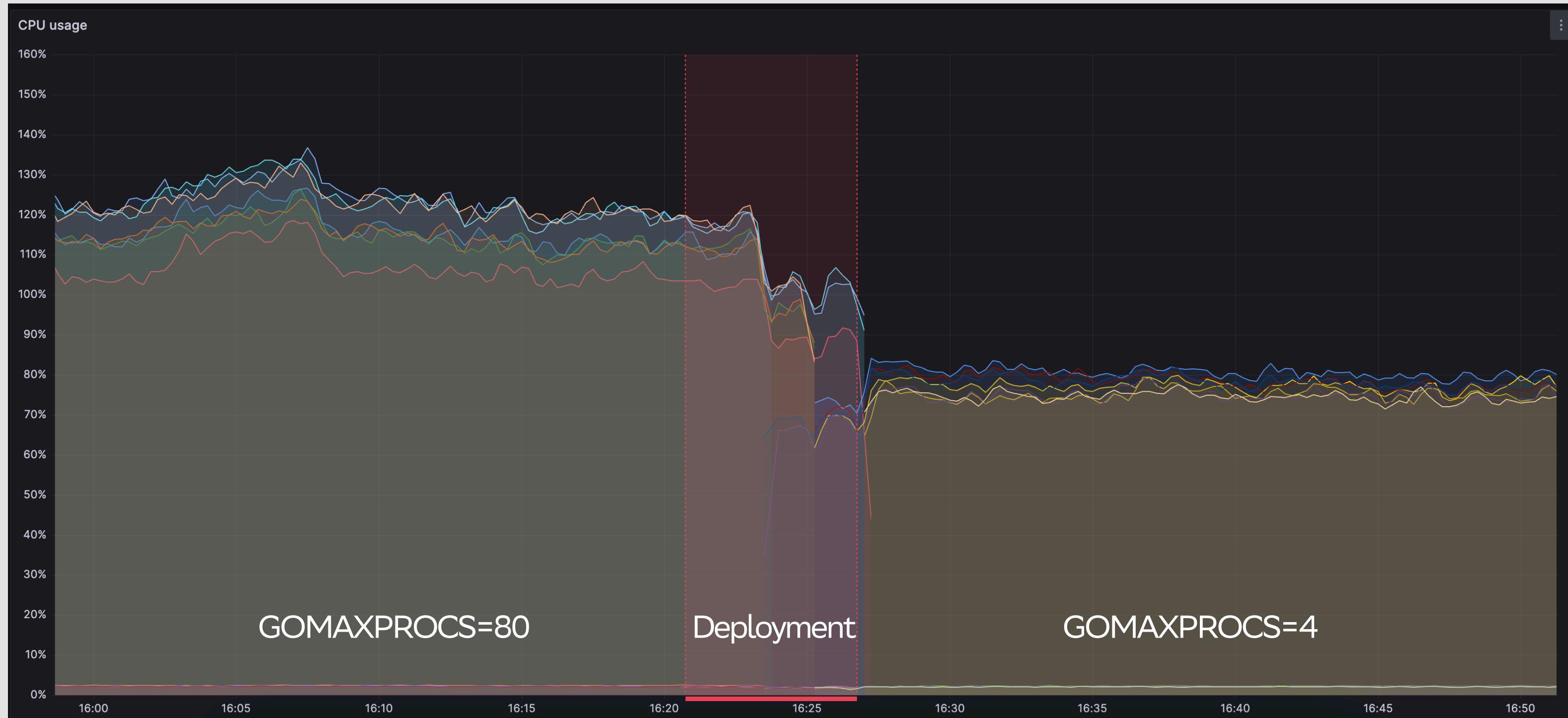
# GOMAXPROCS

Длительность ответа GRPC 90-ый персентиль



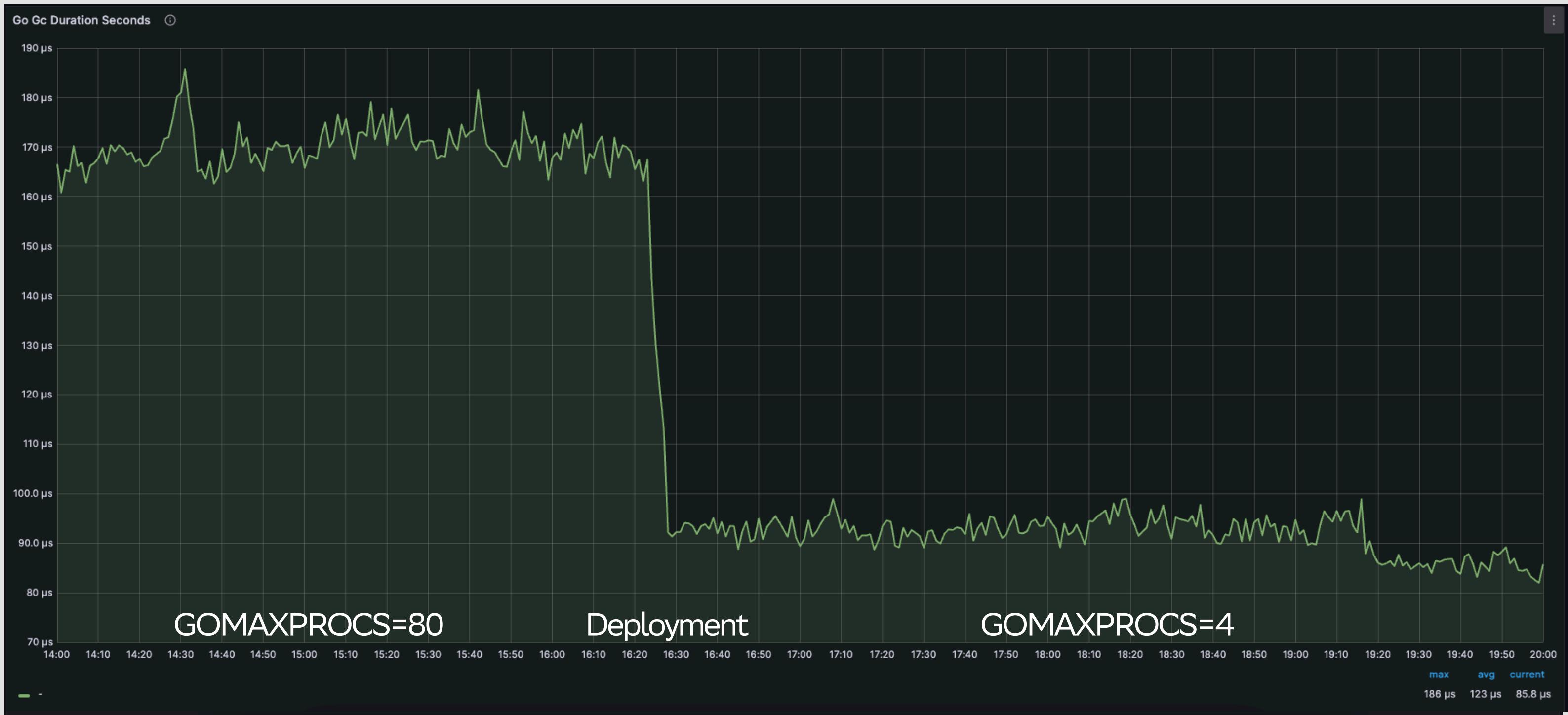
# GOMAXPROCS

## CPU Usage



# GOMAXPROCS

Длительность GC, 50-ый персентиль



# Результаты

- Улучшили 90-ый персентиль длительности ответа на 20%
- Улучшили CPU Usage на 33%
- Улучшили длительность Go GC на 50%

# ИТОГ

## - **GOMAXPROCS**

- Определяет сколько потоков использовать для приложения
- Помогает снизить накладные расходы на переключение контекста CPU

## - **Библиотека Automaxprocs**

- Привязывает GOMAXPROCS к CPU Limits
- Помогает снизить переключение контекста CPU и Throttling

- Введение
- Основы
  - CFS
  - Goroutines
  - Cgroups
- Kubernetes
  - Requests
  - Limits
- GOMAXPROCS
- Рекомендации

# Рекомендации

- Requests: Всегда определяйте CPU, Memory requests
- CPU limits: использовать лимиты в связи с GOMAXPROCS
- Memory limits: выставлять равными requests
- Мониторить CPU Throttling ваших контейнеров

# Рекомендации

- Всегда контролируйте значение GOMAXPROCS
- Оптимальный GOMAXPROCS всегда зависит от вашего контекста
- Пробуйте разные значения, пока не найдете оптимальную комбинацию:
  - Server Latency (50, 90, 99, 99.9 перцентили)
  - CPU Usage
  - GC Duration

## Оптимальные значения:

1. оптимизация бюджета на kubernetes
2. повышение производительности приложений



# Вопросы



# Ресурсы

- <https://www.kernel.org/doc/Documentation/scheduler/sched-design-CFS.txt>
- <https://www.kernel.org/doc/html/latest/scheduler/sched-bwc.html>
- <https://www.kernel.org/doc/Documentation/scheduler/sched-bwc.txt>
- <https://programmer.group/the-concept-of-threads.html>
- <https://rakyll.org/scheduler/>
- <https://erickhun.com/posts/kubernetes-faster-services-no-cpu-limits/>
- [https://docs.docker.com/config/containers/resource\\_constraints/](https://docs.docker.com/config/containers/resource_constraints/)
- <https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#how-pods-with-resource-limits-are-run>
- <https://www.redhat.com/sysadmin/cgroups-part-two>
- <https://stackoverflow.com/questions/61851751/multi-threading-with-millicores-in-kubernetes>
- <https://engineering.squarespace.com/blog/2017/understanding-linux-container-scheduling>

# Ресурсы

- <https://github.com/golang/go/issues/33803>
- <https://levelup.gitconnected.com/know-gomaxprocs-before-deploying-your-go-app-to-kubernetes-7a458fb63af1>
- <https://kubernetes.io/docs/tasks/configure-pod-container/assign-memory-resource/#exceed-a-container-s-memory-limit>
- [https://www.youtube.com/watch?v=x9\\_9iaVsfpM](https://www.youtube.com/watch?v=x9_9iaVsfpM)
- <https://kubernetes.io/docs/concepts/architecture/cgroups/#check-cgroup-version>
- <https://mechpen.github.io/posts/2020-07-25-k8s-cpu/index.html>
- <https://kubernetes.io/docs/tasks/administer-cluster/cpu-management-policies/>

# Спасибо



Антон Жуков

Руководитель группы разработки в платформе СберМаркет