

ООО «ИТСК»

Сизов Сергей Викторович

Telegram - @sudoroot

Для кого данный доклад?

- Вы уже знакомы с докером и используете его.
- Вы уже знакомы с Linux и желательно с LXC (для более глубокого понимания).

Docker под капотом

Сложности CI/CD

Как это работает?

Namespaces

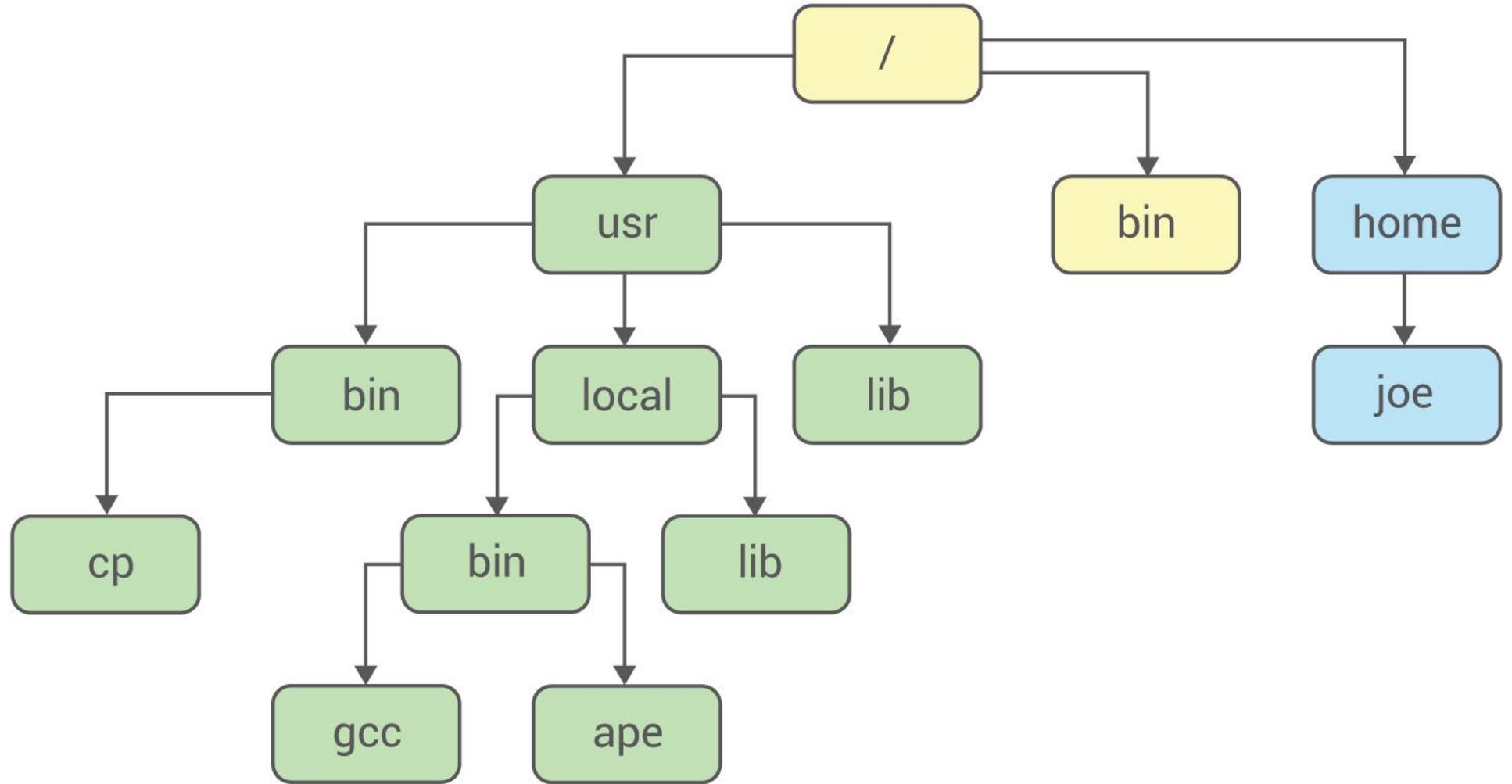
Cgroups

Docker daemon (Автоматизация управления)

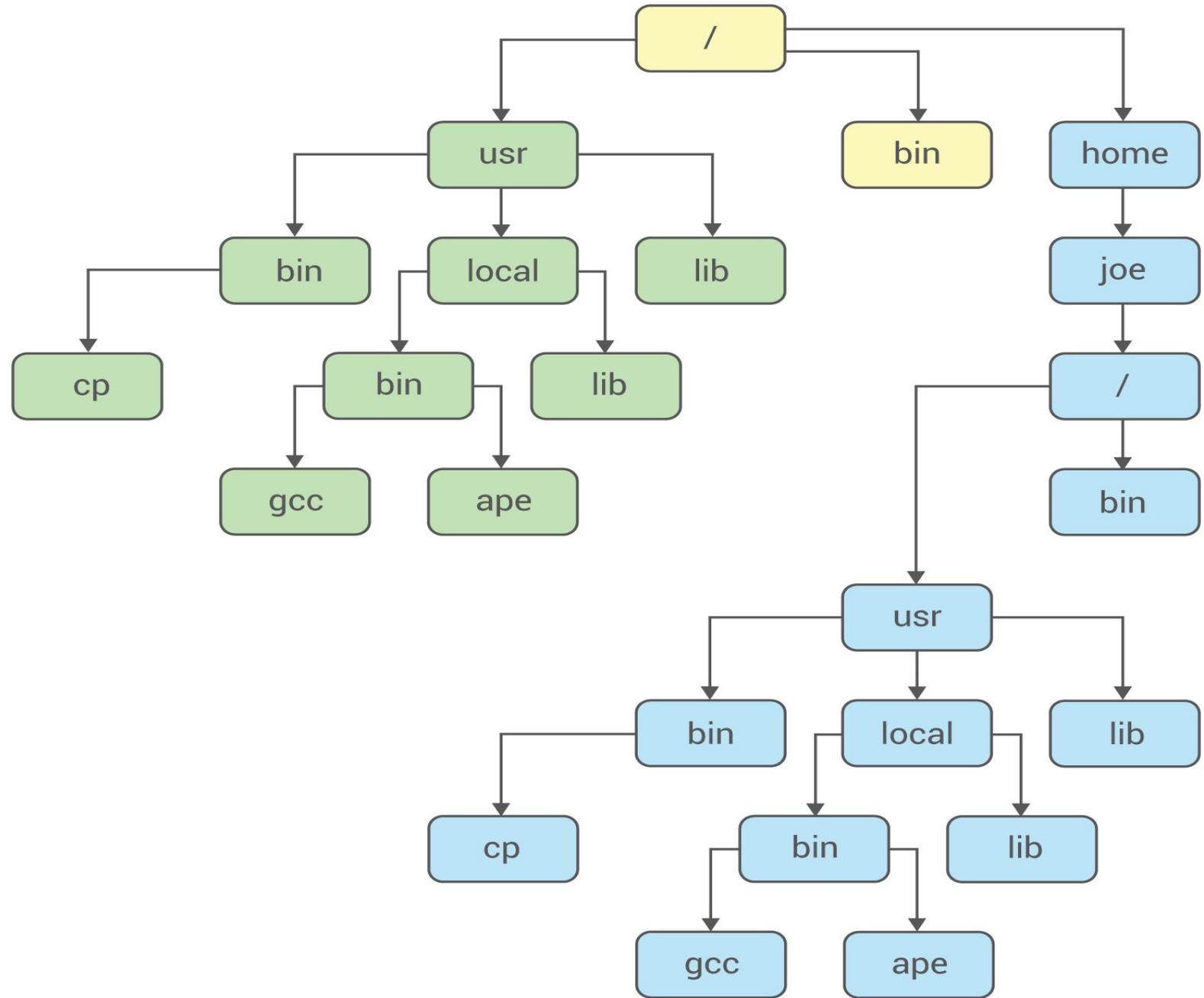
Linux namespaces

- Вначале было слово chroot! (1982 год!)

chroot



chroot



Linux namespaces

- Вначале было ~~есть~~ chroot! (1982 год!)
- И создал админ Jail

Linux namespaces

- Вначале было ~~слово~~ chroot!
- И создал админ Jail
- И наконец в Linux появились Namespace

Linux namespaces

UTC

IPC

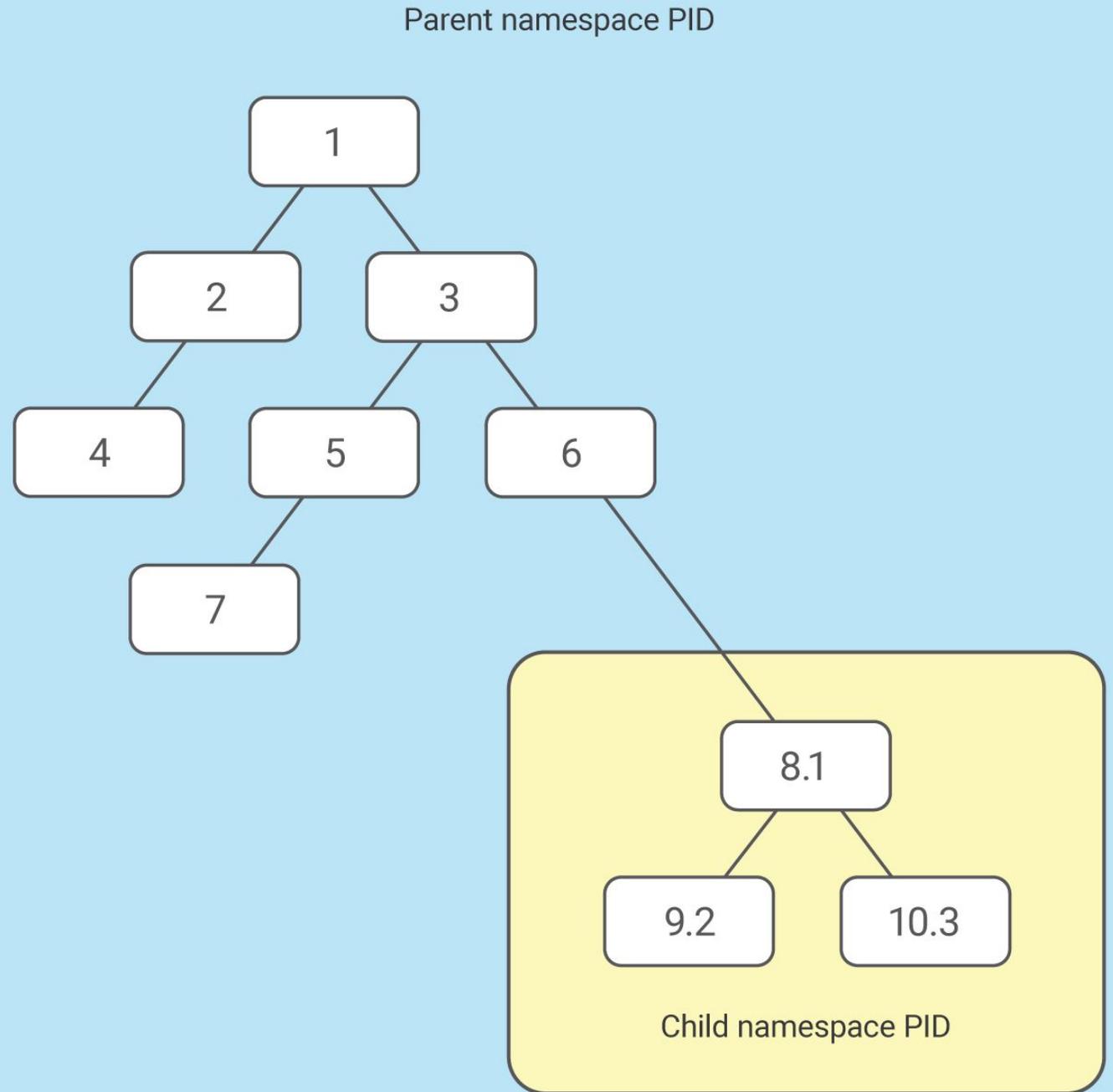
PID

User

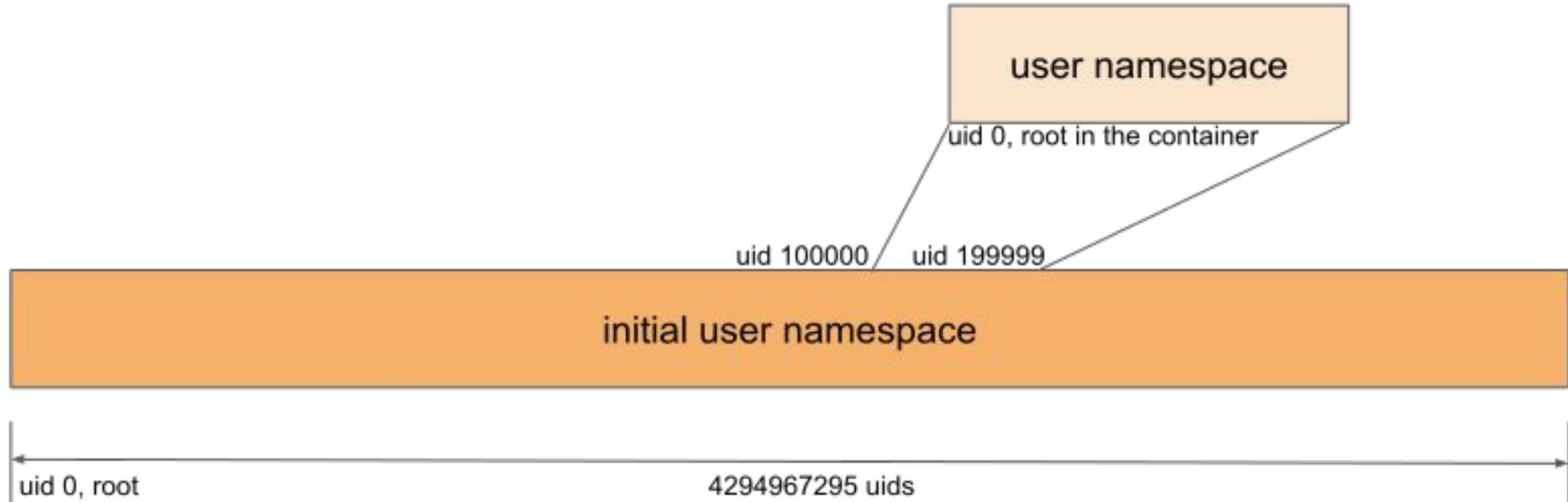
Mount

Network

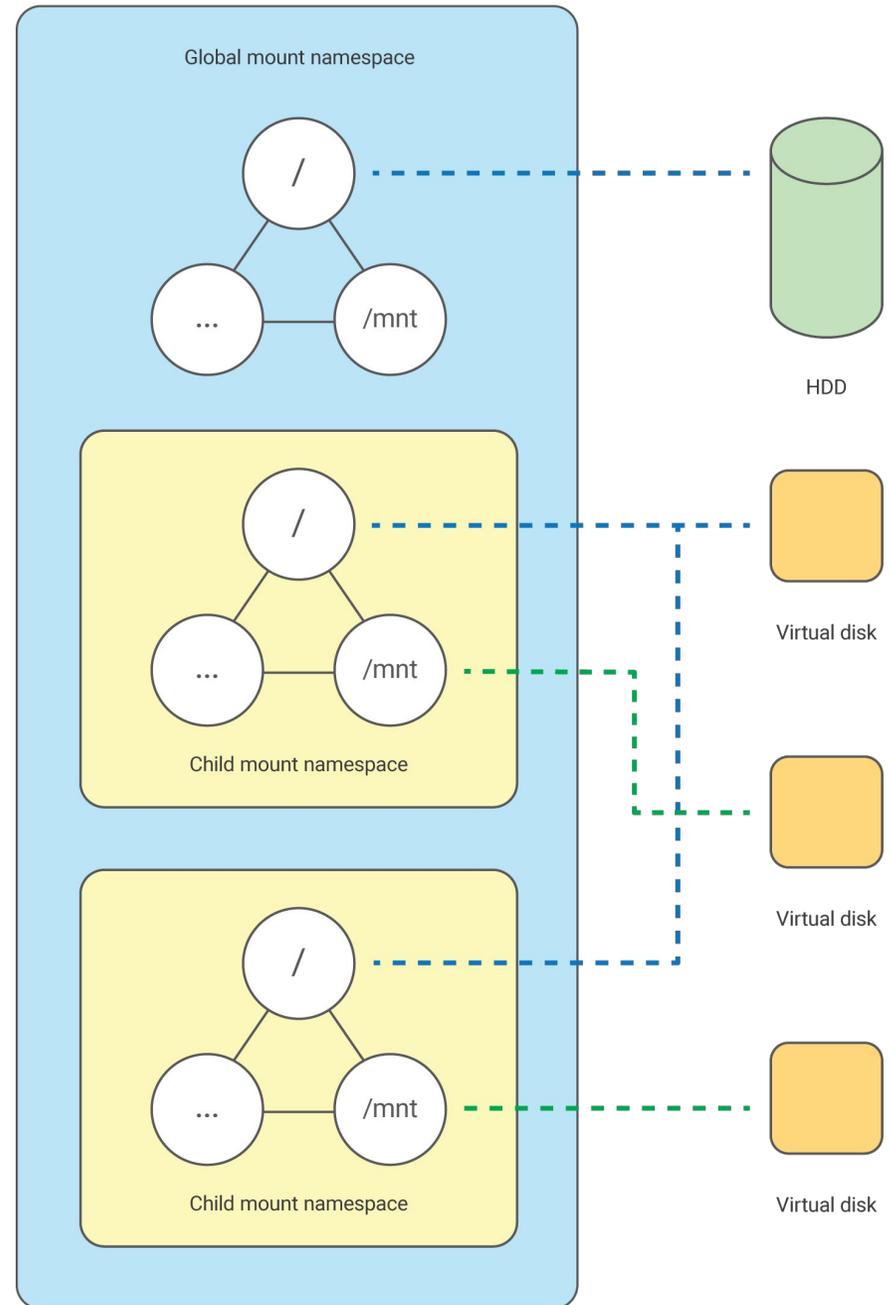
Linux namespaces - PID



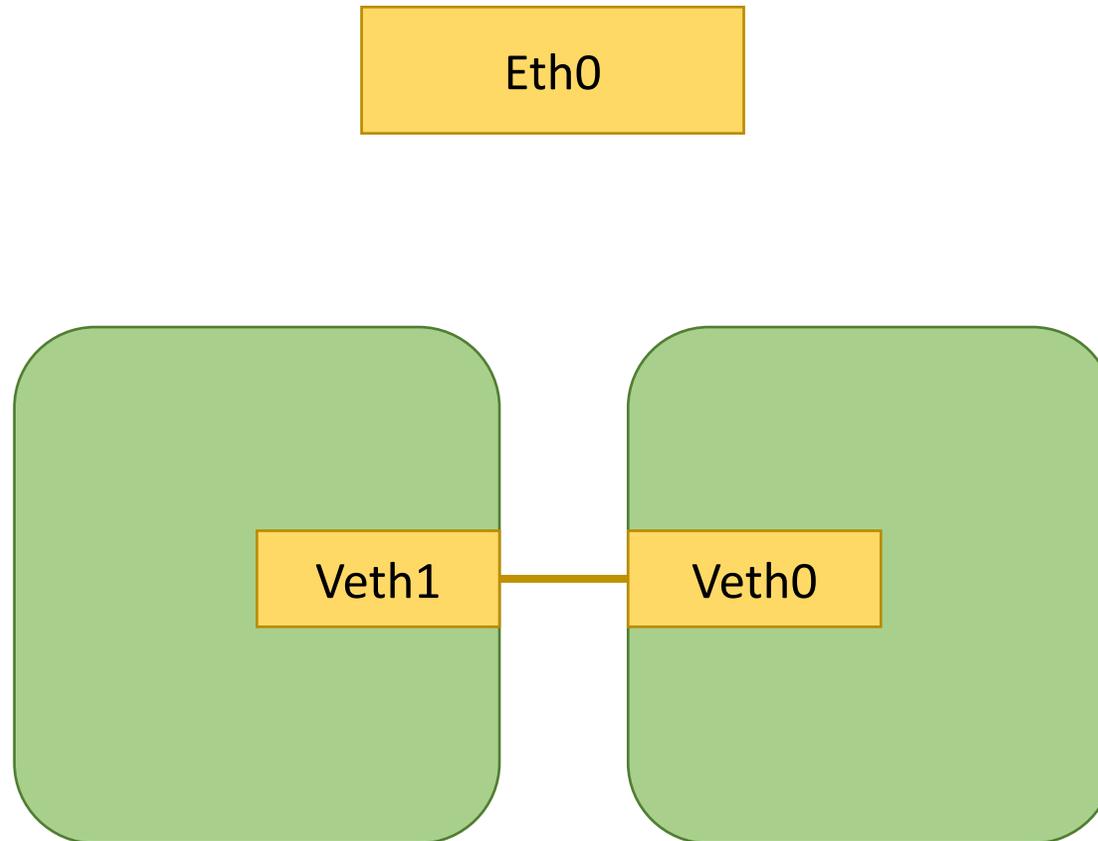
Linux namespaces - User



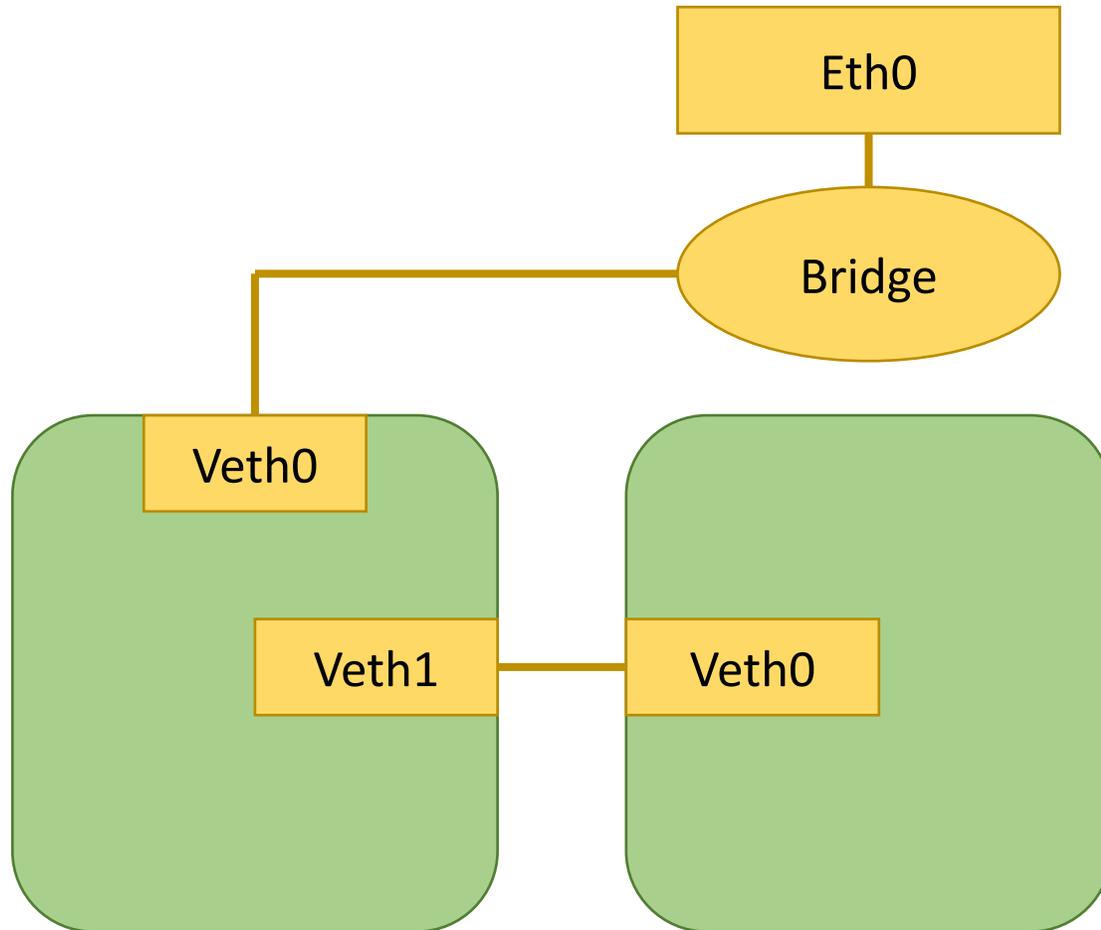
Linux namespaces – Mount



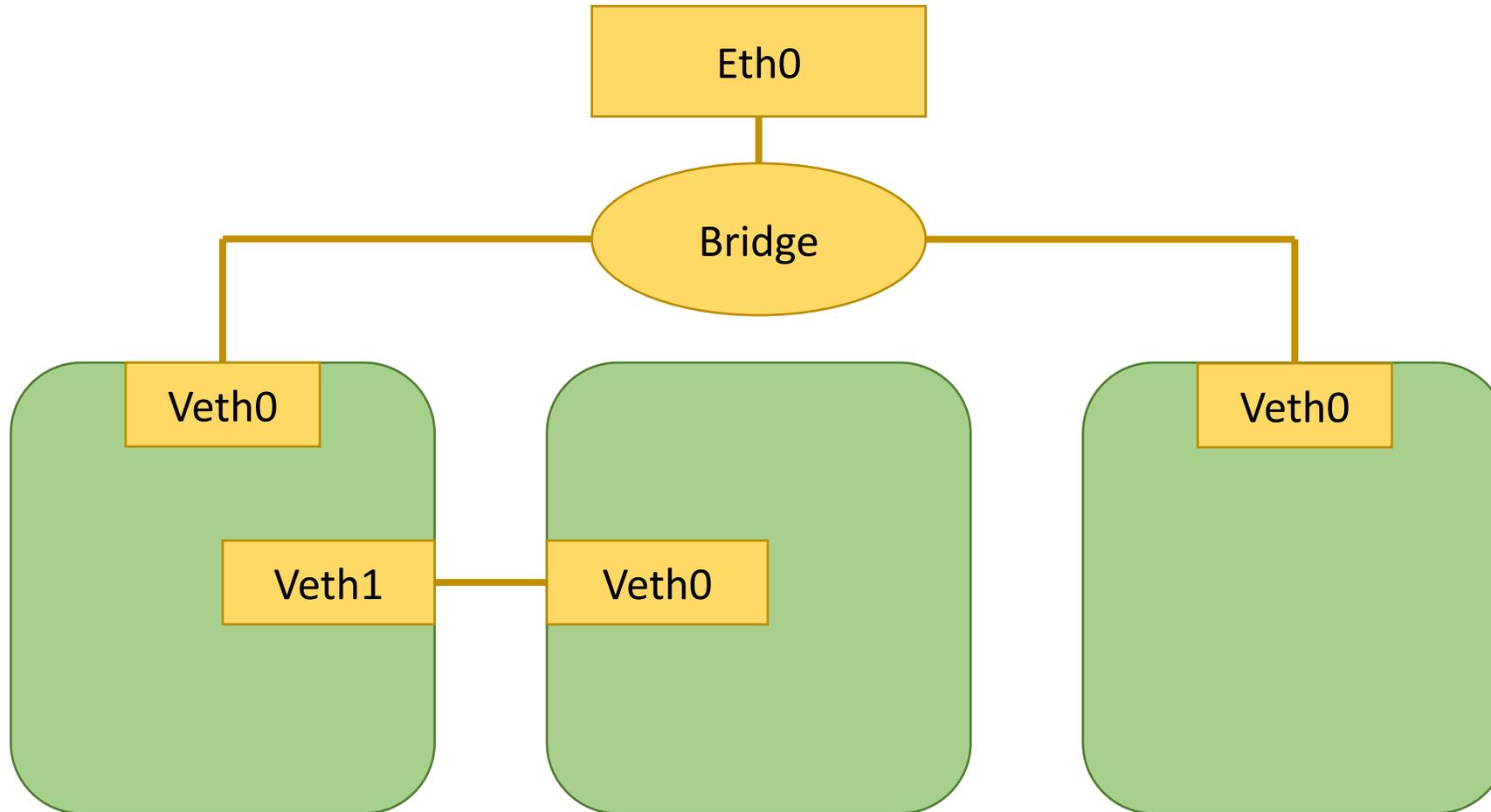
Linux namespaces - Network



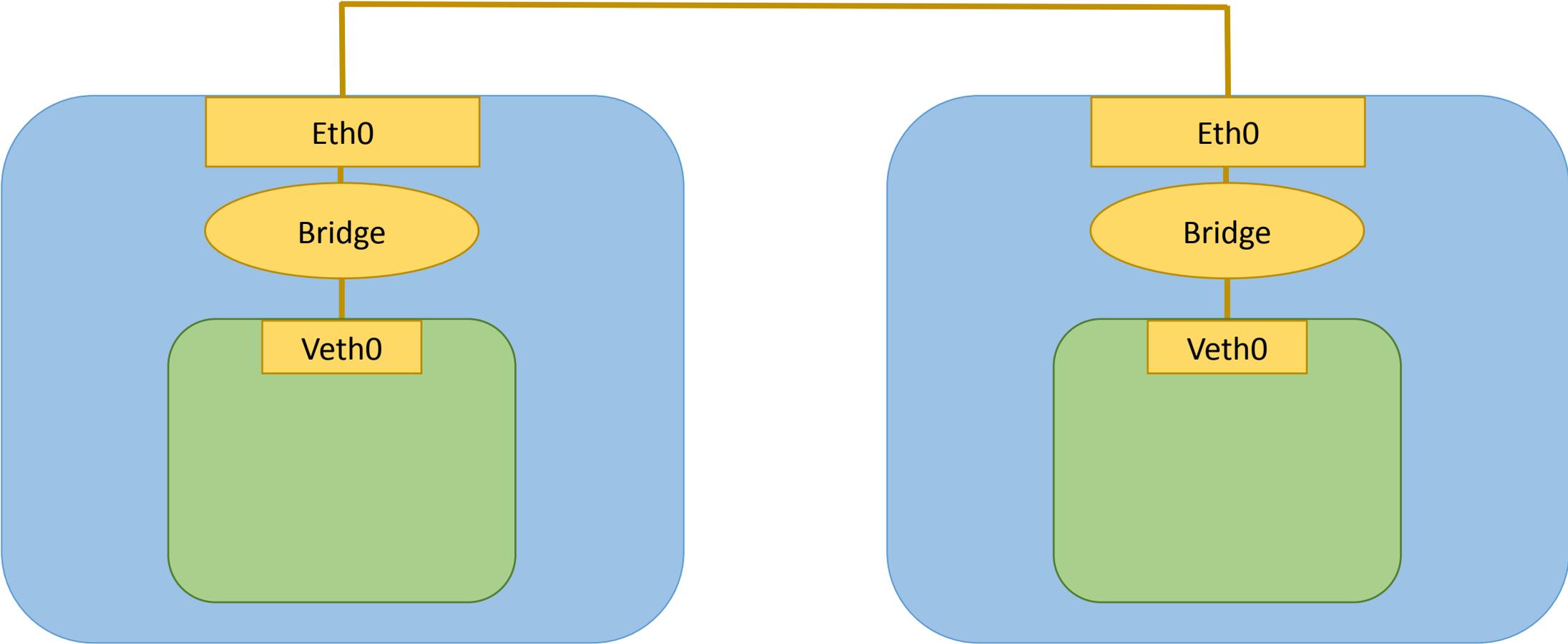
Linux namespaces - Network



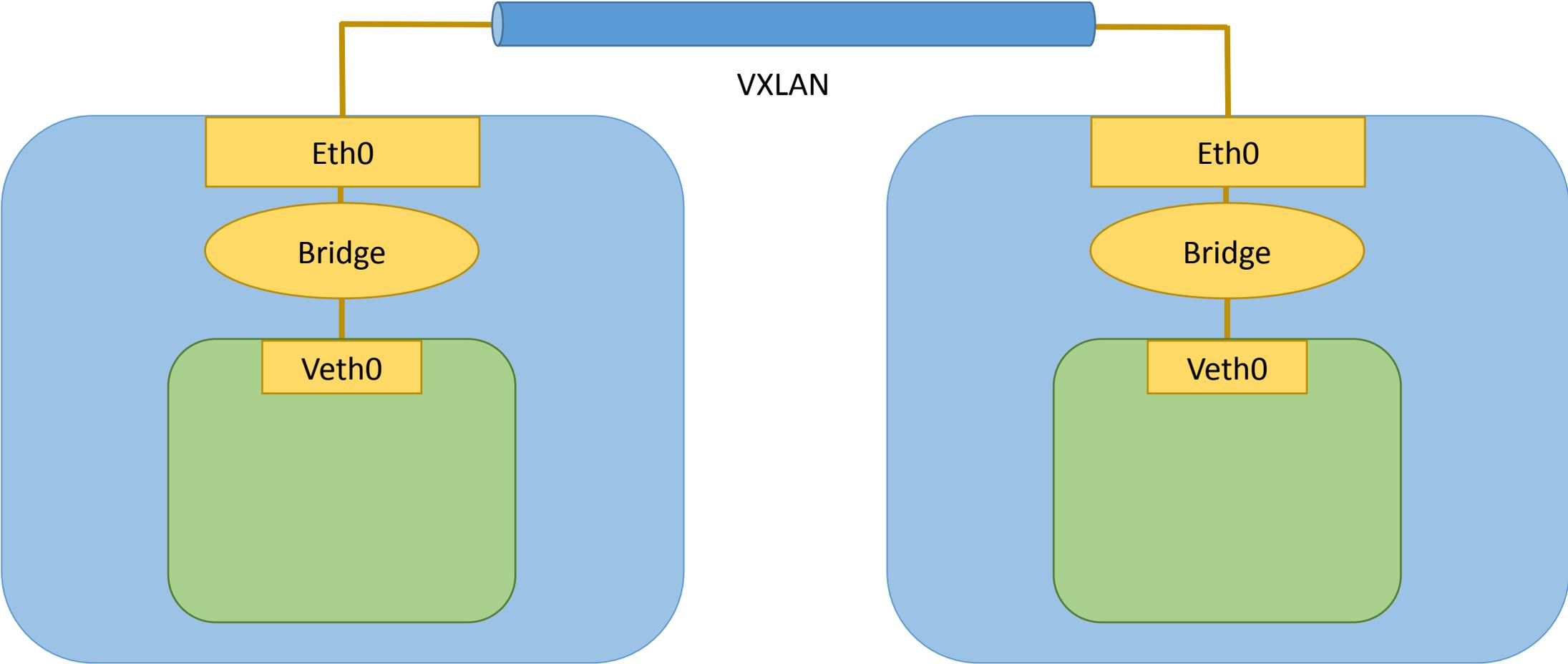
Linux namespaces - Network



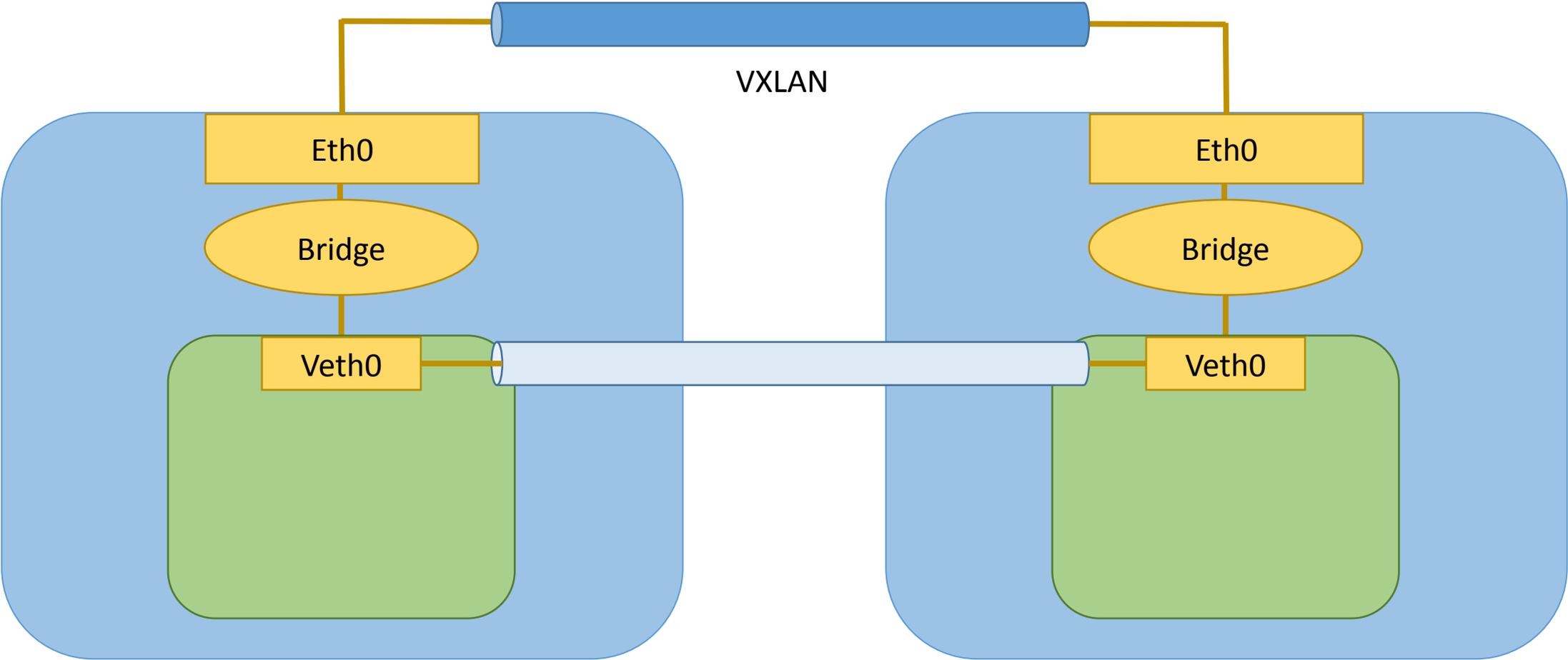
Linux namespaces – Network + VXLAN



Linux namespaces – Network + VXLAN



Linux namespaces – Network + VXLAN



Linux namespaces

UTC

IPC

PID

User

Mount

Network

Как это работает?

Namespaces

Cgroups

Docker daemon (Автоматизация управления)

App

The diagram consists of a large light blue rounded rectangle representing a system boundary. Inside this boundary, at the top, are three smaller, darker blue rounded rectangles. From left to right, they are labeled 'App', 'DB', and 'Cache'. The 'App' component is on the left, 'DB' is in the middle, and 'Cache' is on the right. There is a significant amount of empty space below these three components within the system boundary.

DB

Cache

App

DB

Cache

App
exporter

DB exporter

App

DB

Cache

App
exporter

ELK

DB exporter

App

DB

Cache

App
exporter

ELK

DB exporter

App

DB

Cache

App
exporter

ELK

DB exporter

App

DB

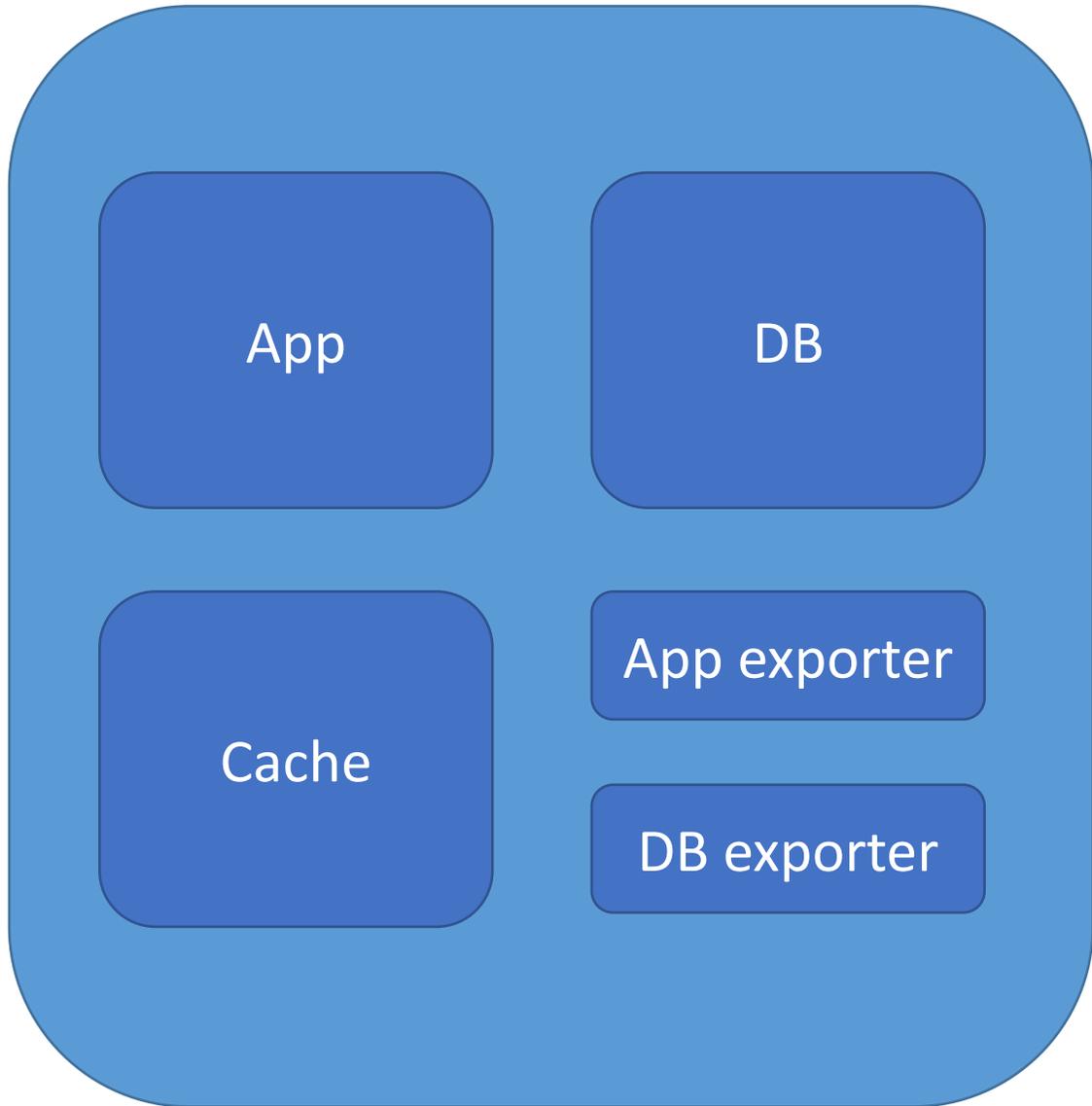
Cache

App
exporter

{ 4 GB Ram }

ELK

DB exporter



Linux control groups

- Block devices (IOPS)
- CPU (Core, and CPU time access)
- RAM + OOM Control
- Devices
- Network packets priority – QoS per network interface

Docker limits и почему это важно?

- CPU
 - Memory
 - GPU
 - Disk IO
-
- Мы знаем сколько потребляет наше приложение!
 - Мы можем выявить нагрузку ко времени!
 - Можем прогнозировать стоимость у облачного провайдера услуг!

OOMy God!

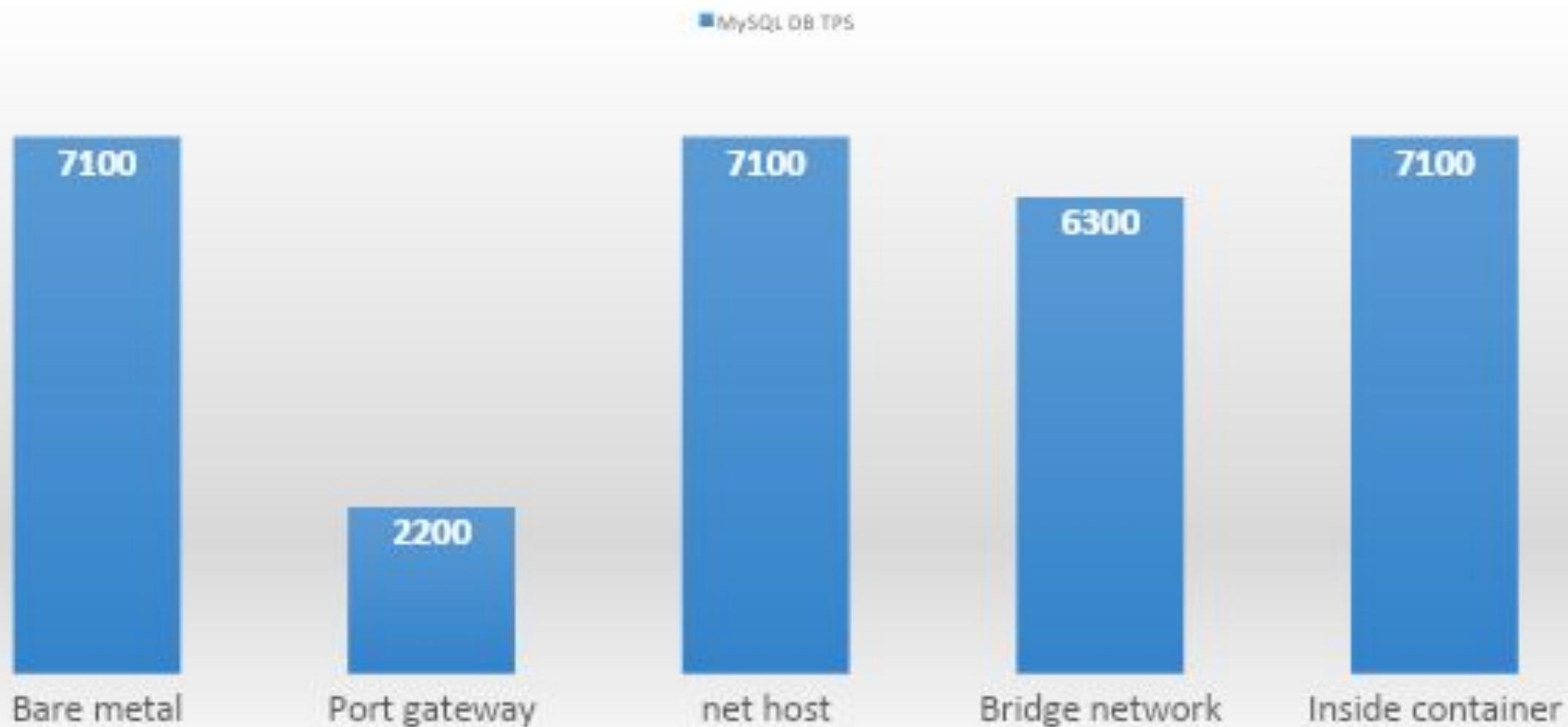
- Не доводить до OOM!
- Выставить ограничения по потреблению RAM.
- В крайнем случае запретить OOM убивать важный сервис.

Overhead?

- CPU – Нет
- RAM – Нет
- Network – зависит от конфигурации
- Disk IO - Нет

2016
год!

Overhead: network



4.4.0-148-generic #174-Ubuntu; Docker 18.09

Docker expose port

| | |
|---------------------------------|---------------------------------|
| Transactions: | 465325 hits |
| Availability: | 100.00 % |
| Elapsed time: | 49.78 secs |
| Data transferred: | 271.59 MB |
| Response time: | 0.00 secs |
| <u>Transaction rate:</u> | <u>9347.63 trans/sec</u> |
| Throughput: | 5.46 MB/sec |
| Concurrency: | 19.92 |
| Successful transactions: | 465325 |
| Failed transactions: | 0 |
| Longest transaction: | 0.04 |
| Shortest transaction: | 0.00 |

Docker network HOST

| | |
|---------------------------------|----------------------------------|
| Transactions: | 523481 hits |
| Availability: | 100.00 % |
| Elapsed time: | 49.62 secs |
| Data transferred: | 305.53 MB |
| Response time: | 0.00 secs |
| <u>Transaction rate:</u> | <u>10549.80 trans/sec</u> |
| Throughput: | 6.16 MB/sec |
| Concurrency: | 19.90 |
| Successful transactions: | 523481 |
| Failed transactions: | 0 |
| Longest transaction: | 0.06 |
| Shortest transaction: | 0.00 |

Разница ~

20%

4.4.0-148-generic #174-Ubuntu; Docker 18.09

Docker expose port

MYSQL Inserts

131498 за 20 секунд!

Docker network HOST

MYSQL Inserts

139550 за 20 секунд!

Разница ~

6%

4.4.0-148-generic #174-Ubuntu; Docker 18.09

Docker expose port

MYSQL Inserts

348177 за 50 секунд!

Docker network HOST

MYSQL Inserts

359070 за 50 секунд!

Разница ~

1%

CI/CD и возникающие СЛОЖНОСТИ

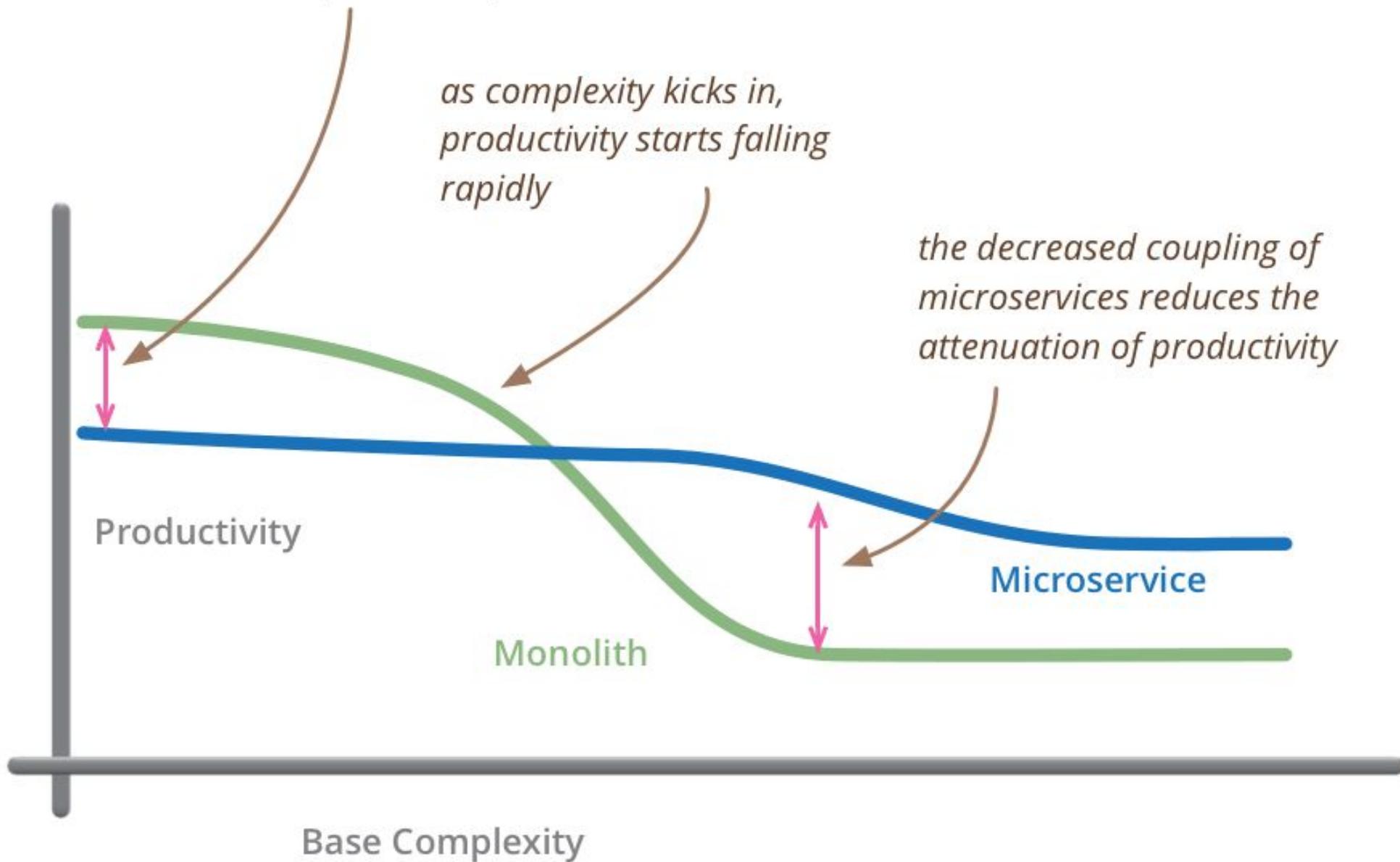
**Когда нужны
микросервисы?**

for less-complex systems, the extra baggage required to manage microservices reduces productivity

as complexity kicks in, productivity starts falling rapidly

the decreased coupling of microservices reduces the attenuation of productivity

Martin Fowler



**Если код изначально
плохой, то..**

Monolithic vs Microservices



Monolithic



Microservices

**Давайте СІ/СDшить
МОНОЛИТ!**

«Типичный» монолит (суровый пример)

- Написан под Windows
- Язык – Дельфи + VBS скрипты.
- Куча legacy кода
- Большое количество зависимостей
- Долгое время старта
- Много интеграций
- Плохая отказоустойчивость
- Плохое масштабирование

И что, вот так и оставить?

- Автоматизация задач, которые принесут пользу, а не вред.
- По возможности и необходимости выносить отдельные части приложения в микросервисы.
- Не переусердствовать.

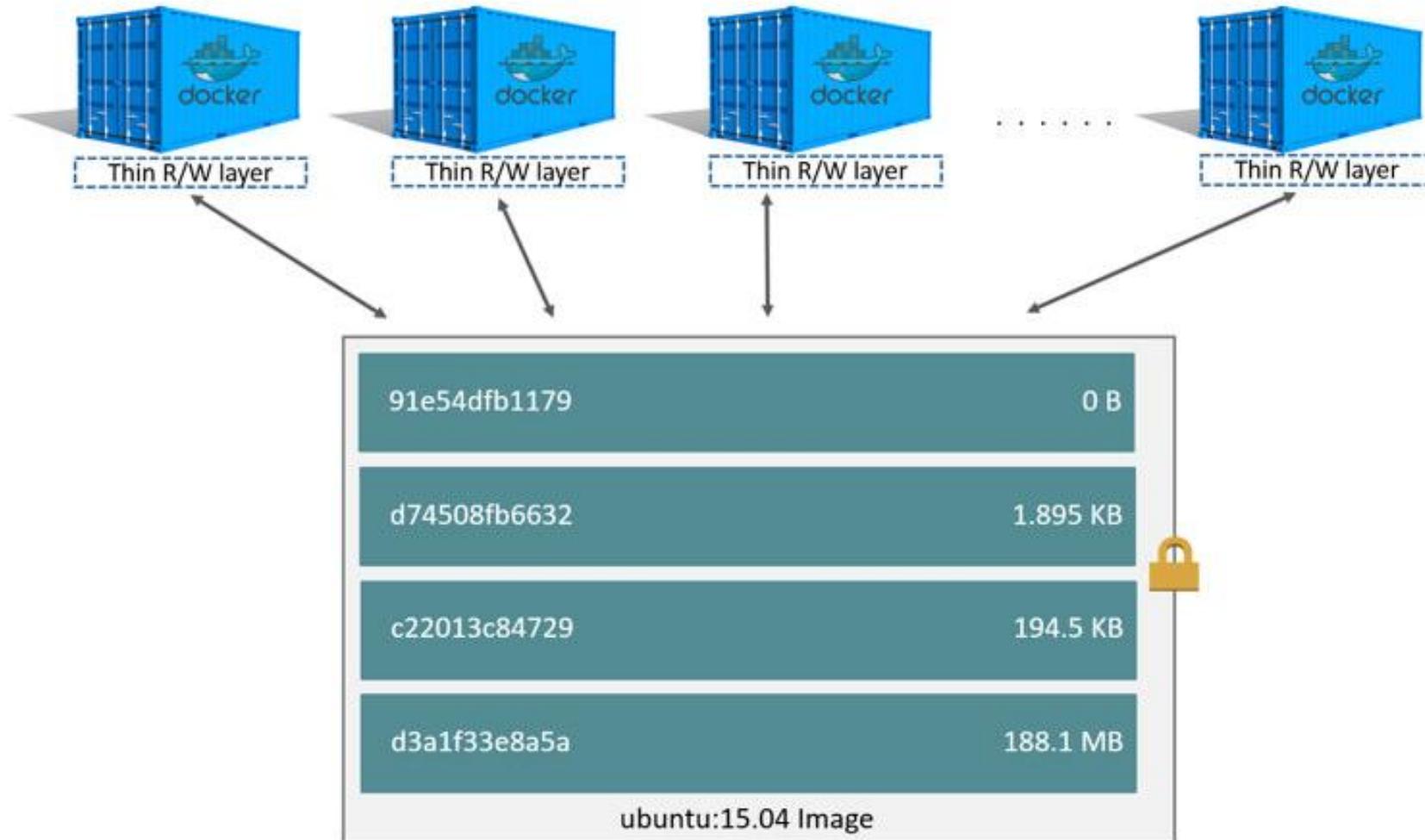
А как быть с паролями?

- Хранить прямо в Git вместе с кодом проекта
- Подкладывать из другого приватного репозитория при сборке
- ENV переменные на средах
- Использовать готовые решения, например – Docker Secret(Swarm), Vault от Hashicorp

Docker registry, а оно нам надо?

- Когда `docker-save --copy -- docker-load` удобнее? (BitBucket CI/CD)
- Какие проблемы может принести Docker-registry?

Union File System



А когда нужен Swarm, K8S ?

- Нам необходимо разнести контейнеры по нодам (напр. Вынести ELK)
- Нам необходимо иметь отказоустойчивость
- Нам необходимо балансировать нагрузку
- Хотим утилизировать имеющиеся мощности на полную

ООО «ИТСК»

Сизов Сергей Викторович

Telegram - @sudoroot