Как месяц искать утечку памяти в СХД

И выяснить, что ее
на самом деле нет

Михаил Мотыленок

Ведущий инженер
по разработке ПО, YADRO

TATLIN.AFA
TATLIN.UNIFIED

Владимир Евграфович Татлин

СХД с блочным и файловым доступом

Клиентские серверы

- Серверы приложений
- БД
- Гипервизоры
- И другие

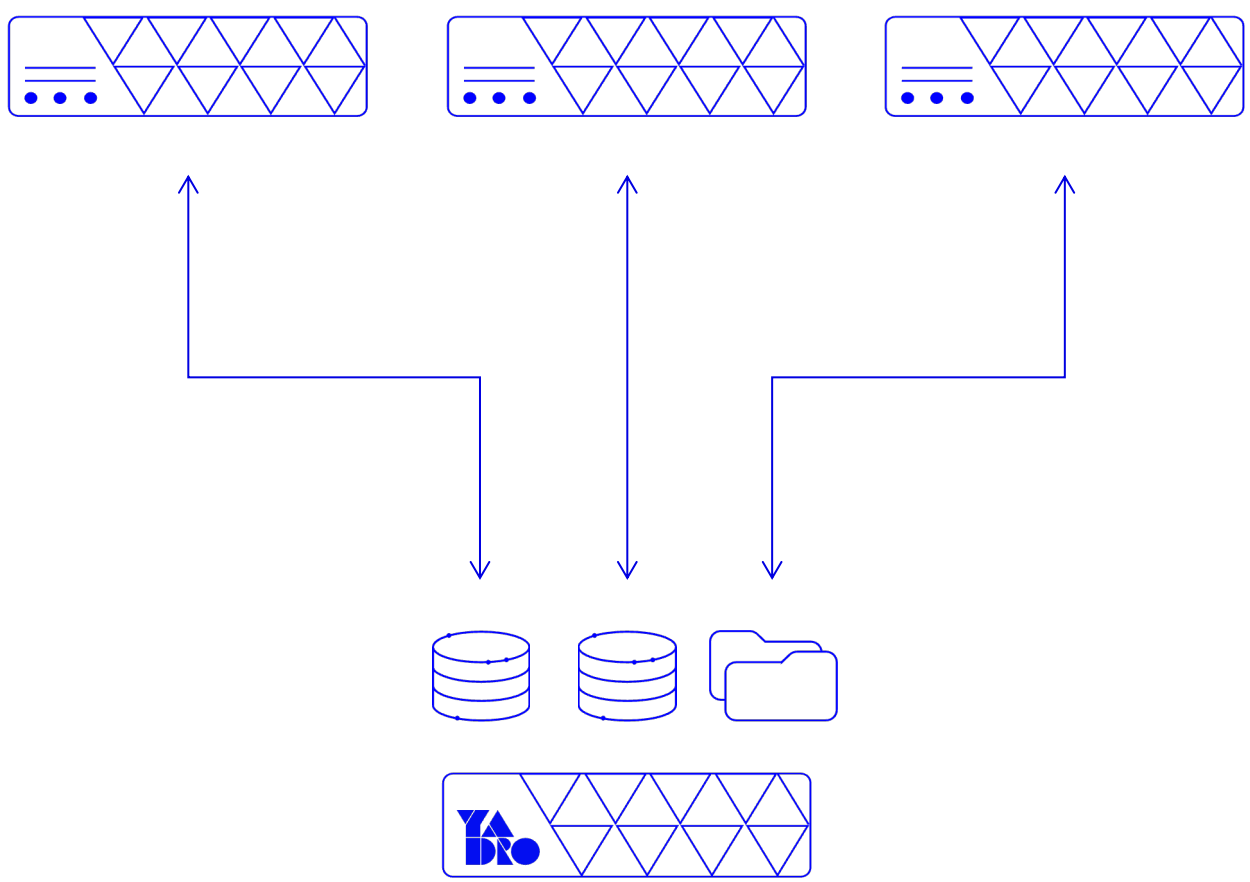
Сеть данных

- FibreChannel / Ethernet
- iSCSI
- NVMeoF (TCP / RoCE)

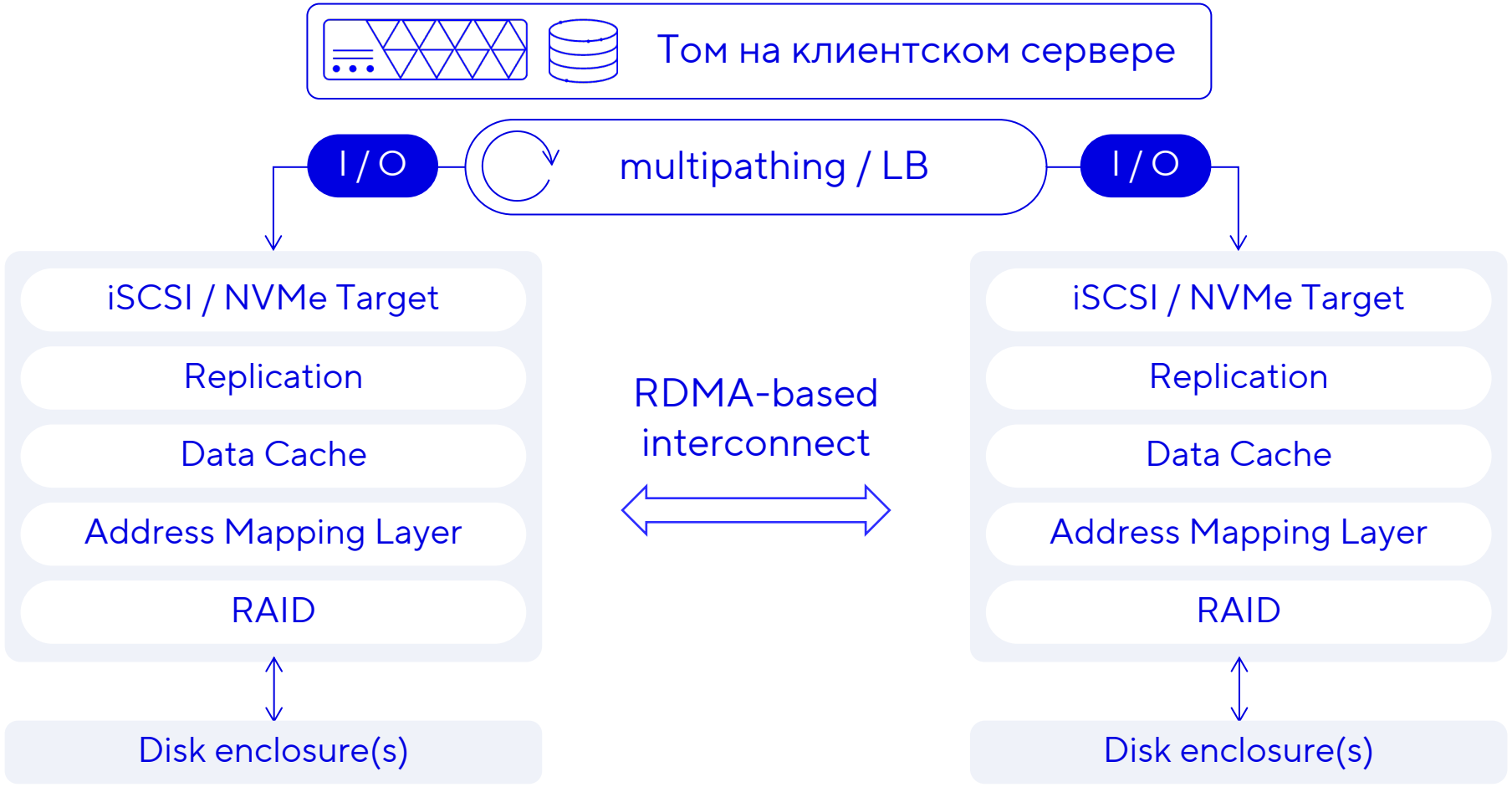
Блочные тома (/dev/mapper/....)

Файловый доступ (SMB / NFS)

СХД TATLIN



Symmetric Active-Active data path



Symmetric Active-Active data path

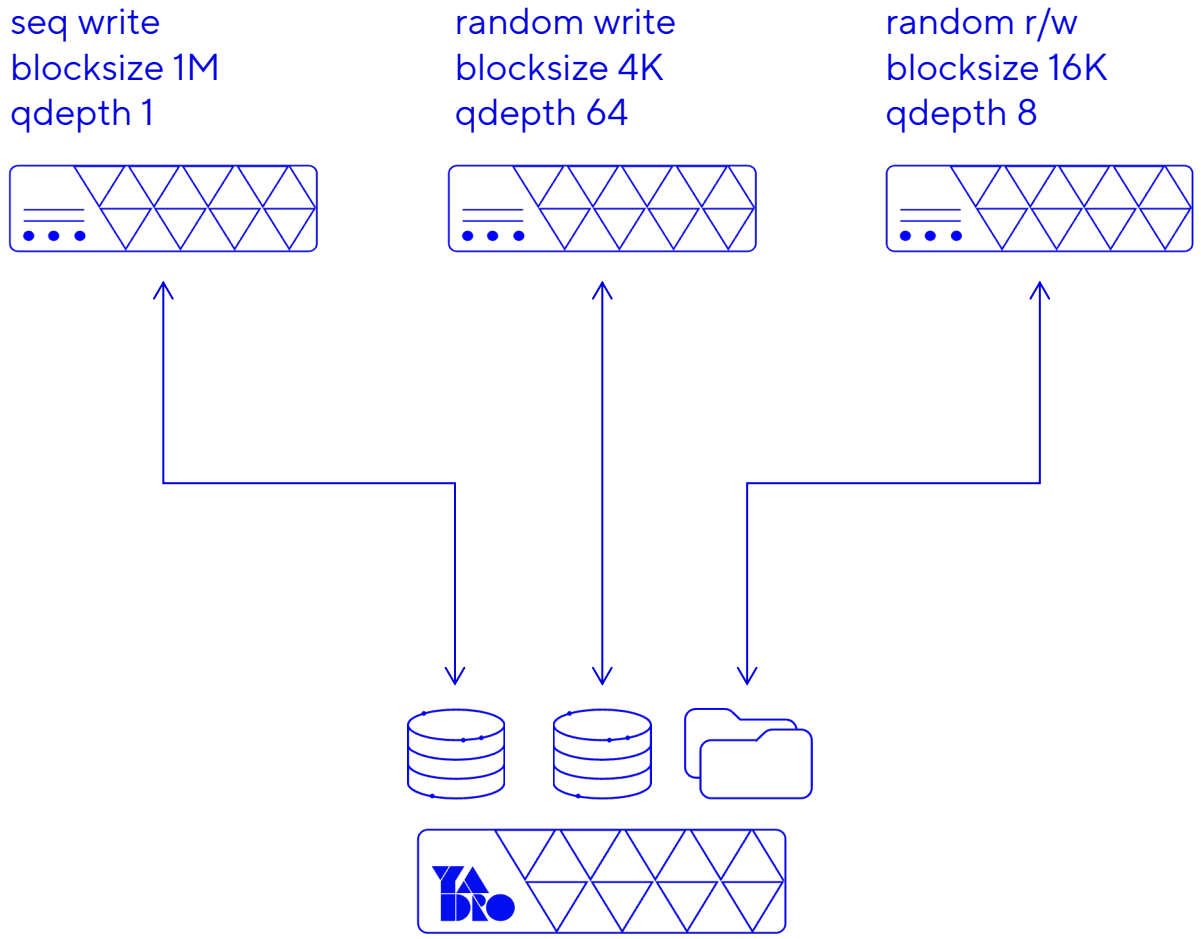
Тестирование СХД

Fio – flexible I/O tester — основной инструмент

- Много настроек
- Различные паттерны
- Верификация данных

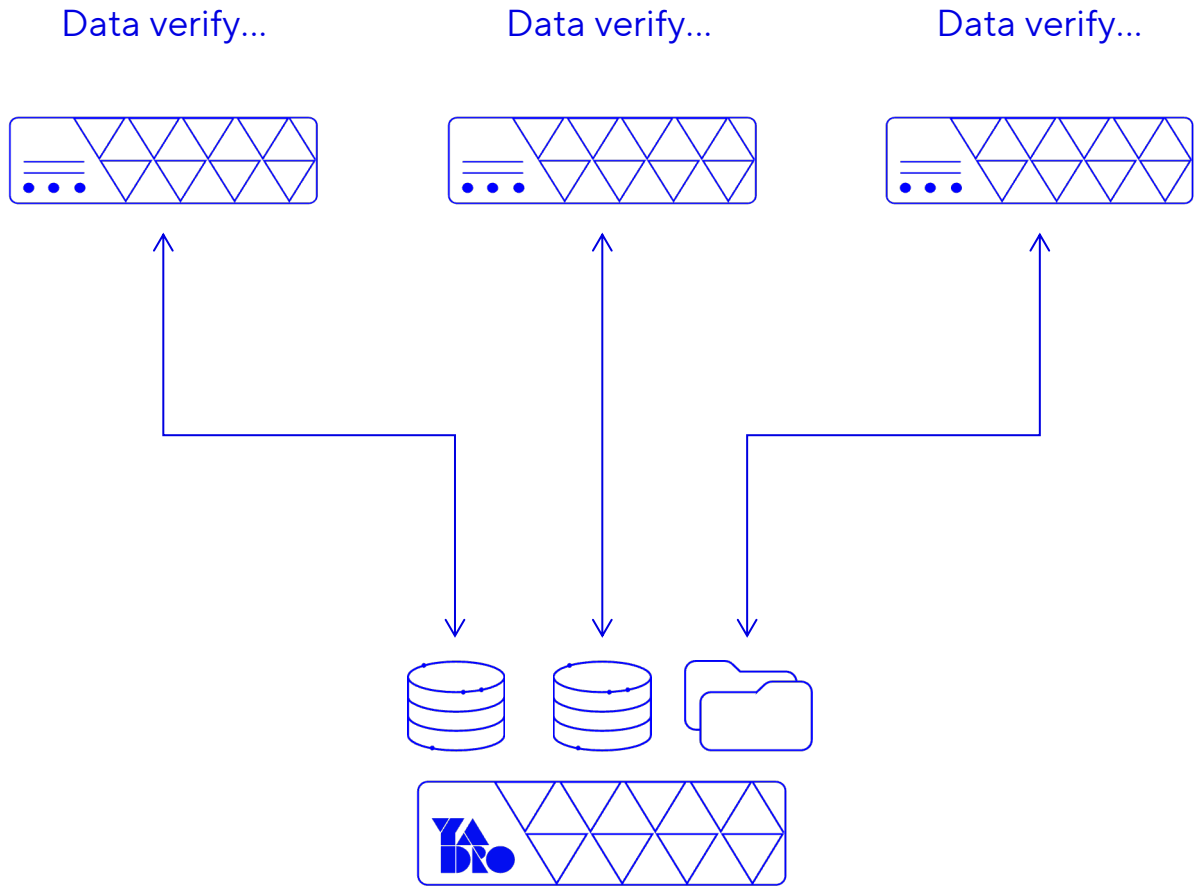
vdbench —

I/O бенчмарк, может генерировать нагрузку с нескольких хостов



Тестирование СХД

В процессе и / или после теста запускается верификация



Тестовые сценарии

Помимо генерации нагрузки, нужен еще сценарий:

Действия пользователя

- Сделали снапшоты
- Добавили / удалили тома
- Сходили в API
- и т.д.

Переконфигурация системы

- Пулы
- Диски
- Хосты
- Порты
- и т.д.

Тестовые сценарии

Помимо генерации нагрузки, нужен еще сценарий:

Действия пользователя

- Сделали снапшоты
- Добавили / удалили тома
- Сходили в API
- и т.д.

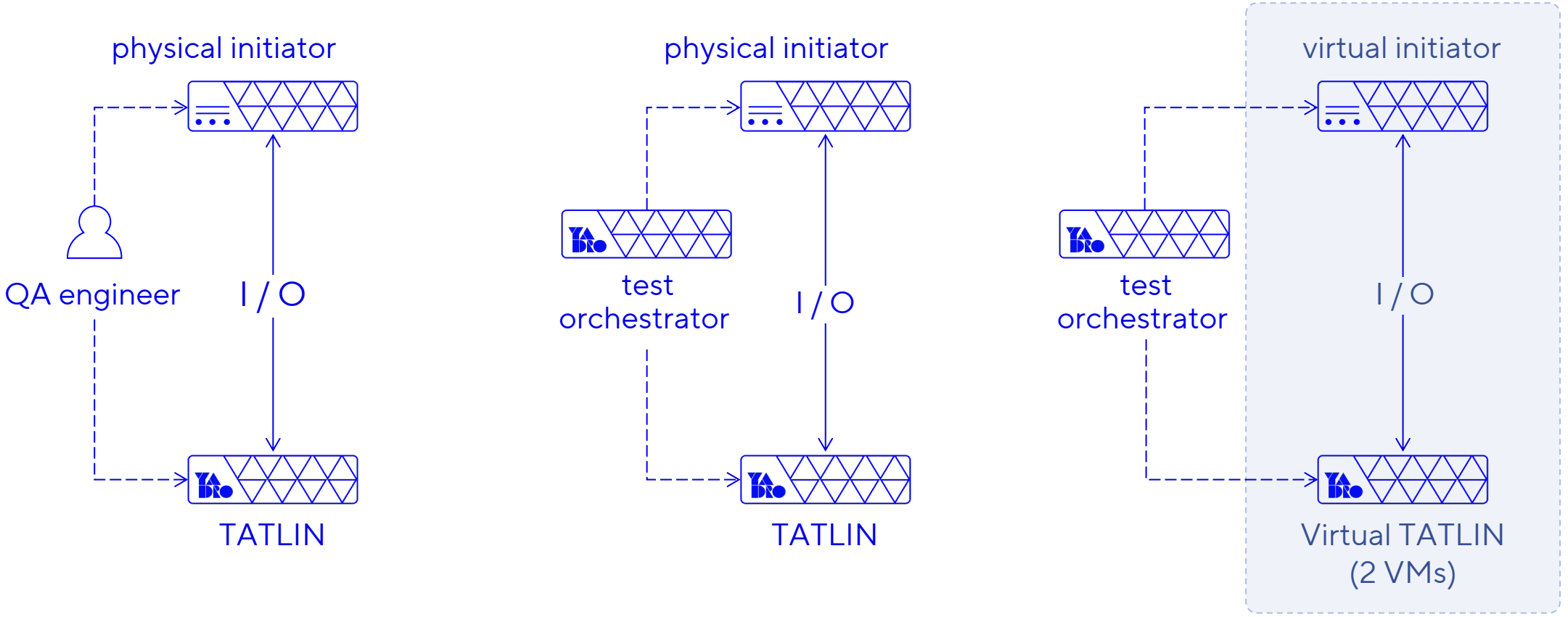
Переконфигурация системы

- Пулы
- Диски
- Хосты
- Порты
- и т.д.

Сценарии посложнее:

- Power loss (переход на АКБ)
- Failover / Failback
- Отказы дисков
- Отказ дисковой полки / линка или экспандера
- Обновление ПО СХД
- Split-Brain

HW, VM, автотесты



Storage bugs (вид снаружи)

Зависает I/O в один или несколько томов СХД («Застревашка»)

После определенного таймаута пути до СХД отваливаются, клиентское приложение получает ошибки

01

```
lu7-i1:~ # fio profile.fio

Jobs: 1 (f=1): [ _(1) ][19.4%][eta
21h:01m:56s]
```

Не проходит верификация (Data Loss / Data Corrupt)

Хост с fio успешно записал в том данные и не смог прочитать их обратно при верификации

02

```
fio: got pattern '72', wanted 'f8'. Bad bits 3

fio: bad pattern block offset 4096 pattern:
verify failed at file
/dev/mapper/3614529012e5c05044000
800000000003 offset 126877696,
length 4126471154
```

Посыпались I/O-ошибки во время теста (Data Unavailable)

Там, где это не ожидается по сценарию

03

```
fio: io_u error on file
/dev/mapper/36145290038540 1f040008
00000000003: Input/output error: write
offset=8270053376, buflen=32768
```

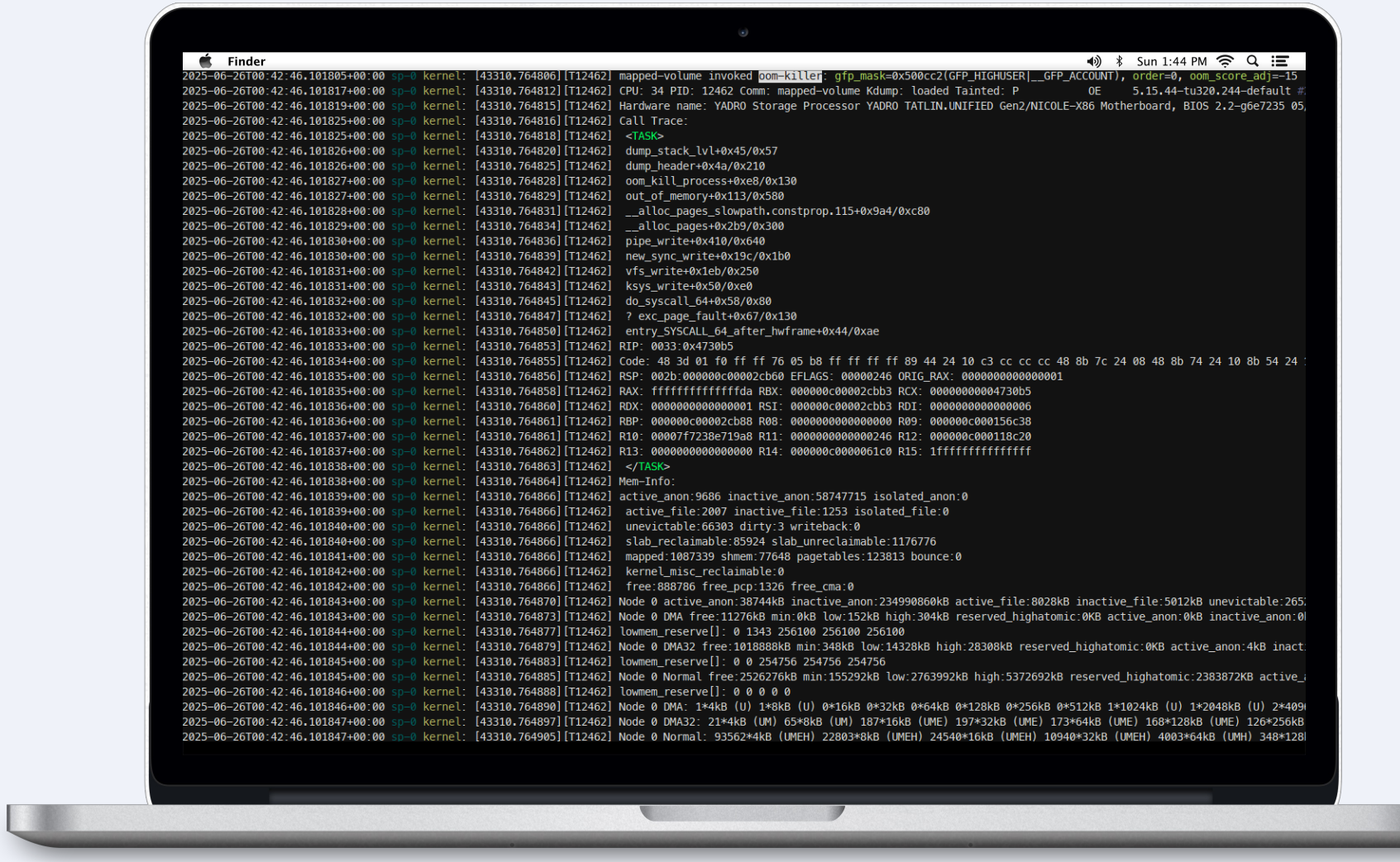
Вид изнутри?

Вид изнутри?

Бывает очень по-разному :)

Кейс

- Out-of-Memory при создании двух тысяч датастор VMware ESXI



?

Зачем

проводить тесты
с решениями
сторонних вендоров?

?

Почему

не достаточно
рандомизированного
трафика?

Зачем?

01

Тестируем интеграцию, все везде экспортируется правильно и работает

02

Там, где есть — интеграцию с API

Например, решения РФ-вендоров

03

Иногда решения сторонних вендоров делают специфичные вещи

Специфика VMware

Требует 512 blocksize 01

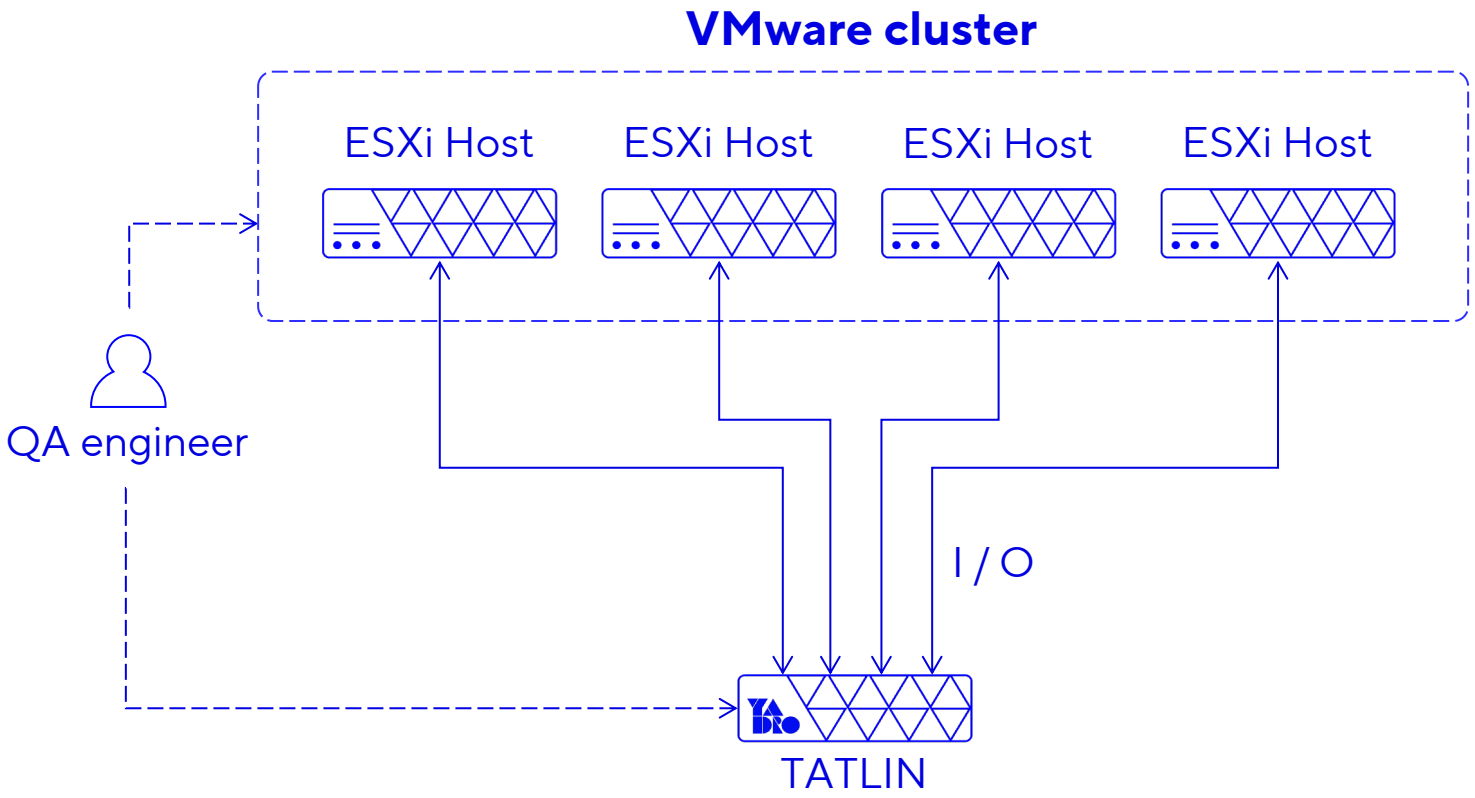
Нужно эмулировать (512e): Read-Modify-Write

Присылает в storage VAAI-примитивы 02

Unmap / WriteZeroes / WriteSame

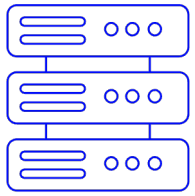
ATS (atomic test and set)

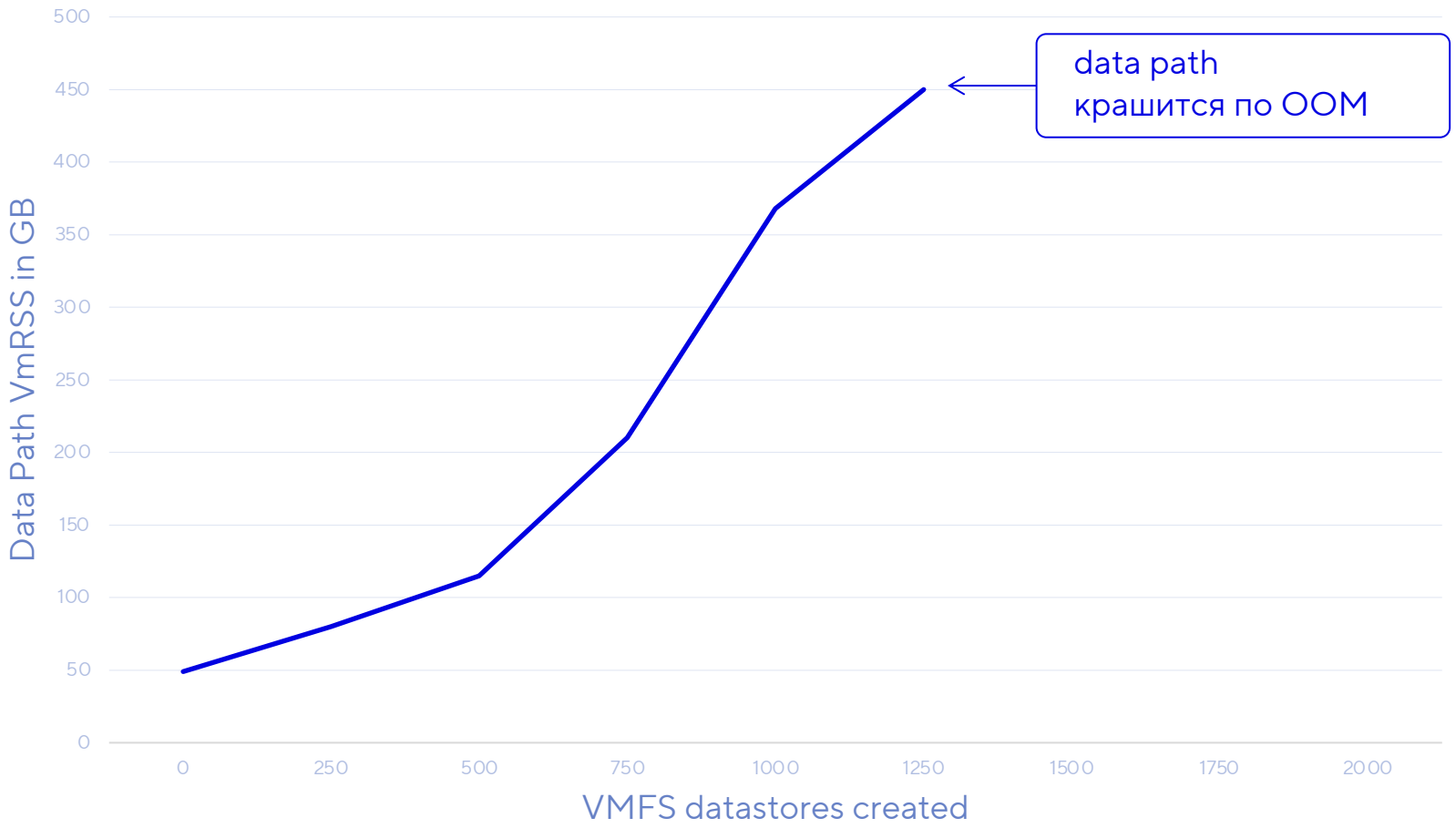
Сценарий


VMware cluster
ESXi Host
ESXi Host
ESXi Host
ESXi Host
QA engineer
I / O
TATLIN

Сценарий

Создаем 2000 VMFS датастор на ресурсах TATLIN.UNIFIED – СХД крашится из-за OOM в datapath





Ищем проблему

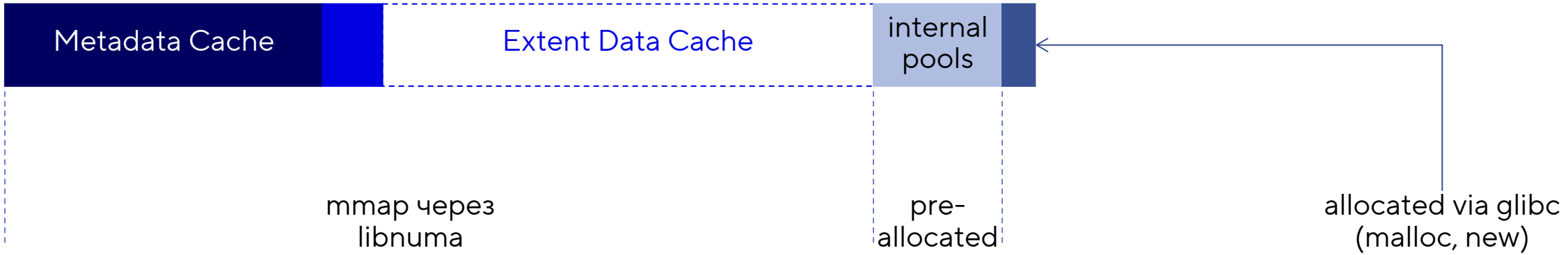
Аллокации в data path???

Кэш и критичные структуры данных

Преалоцированы или
mmap-ятся напрямую

Аллокаций на heap-е остается не так много

- аллокации некоторых сторонних библиотек
- folly future/promise
- tcl::htrie_map hash_nodes
- boost-овые стейт машины



Утечка vs. Фрагментация

У нас сразу же возникают 2 основные гипотезы:

01

Утечка из-за специфики теста

02

Фрагментация

1. Включаем jemalloc и профилируем heap

```
Environment="MALLOC_CONF=prof:true,prof_active:false,prof_accum:true,lg_prof_sample:19"

Environment="LD_PRELOAD=/opt/jemalloc/lib/libjemalloc.so"
```

1. Включаем jemalloc и профилируем heap

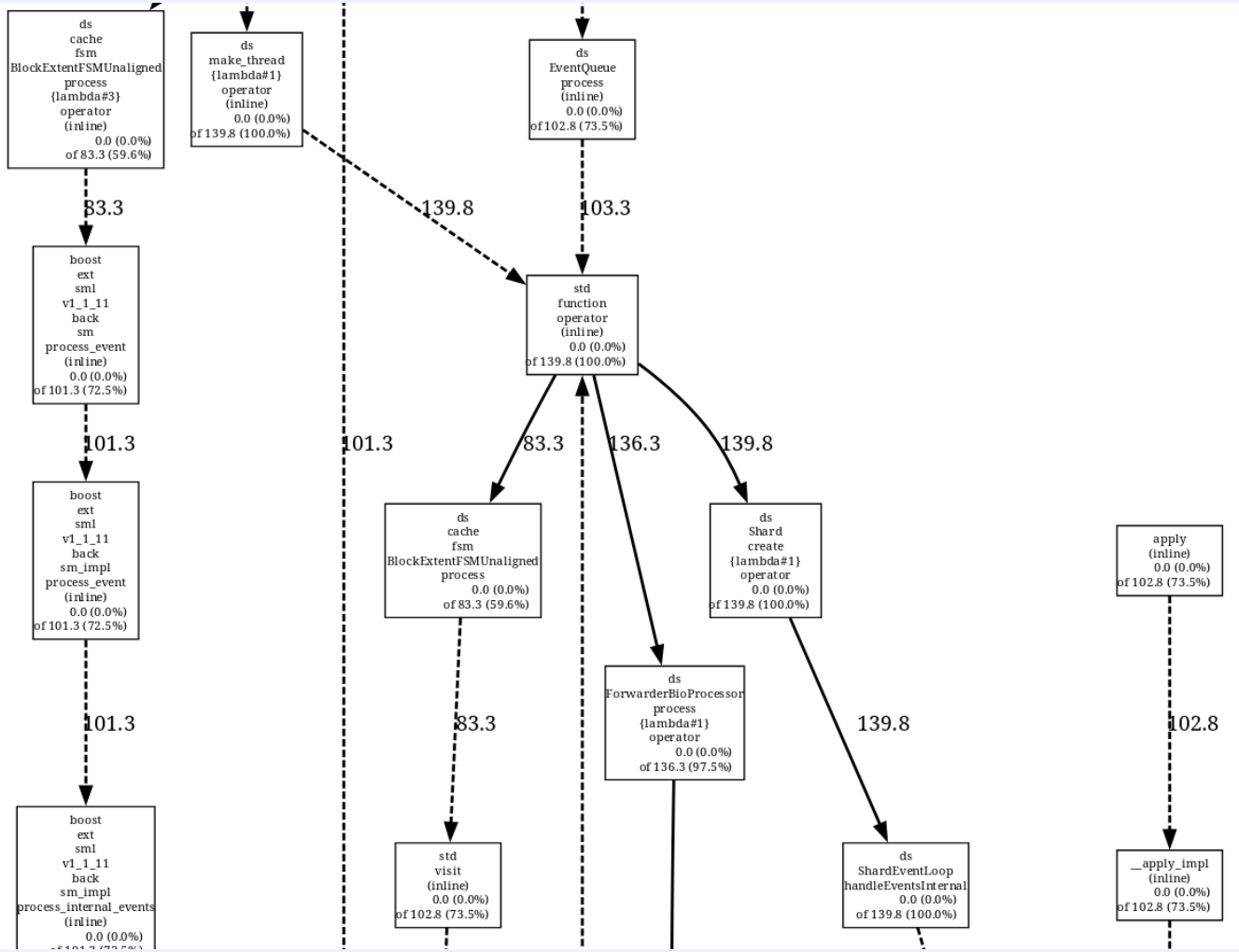
```
Environment="MALLOC_CONF=prof:true,prof_active:false,prof_accum:true,lg_prof_sample:19"

Environment="LD_PRELOAD=/opt/jemalloc/lib/libjemalloc.so"
```

```
"jemalloc":
        "active": "854.67188MB",
        "allocated": "824.62813MB",
        "mapped": "1.55436GB",
        "metadata": "69.83464MB",
        "resident": "1.20178GB",
```

1. Включаем jemalloc и профилируем heap

Можно построить граф аллокаций через jeprof!

Но на нем чего-либо интересного не видно

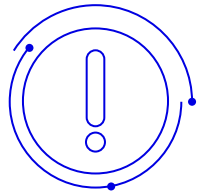


Ищем утечку памяти

Инструмент	Результат
01 Jemalloc + jeprof	Не воспроизводится ✖

2. Смотрим mallinfo

Проблема воспроизводится, однако, glibc 2.31 содержит только **старый mallinfo с известными багами**

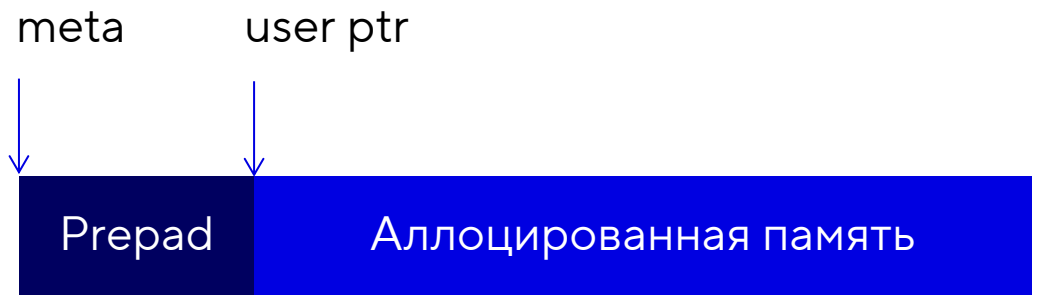


```
"arena": -5332992, // <======== garbage?
"fordblks": -2050959328,  // <======== garbage?
"fsmblks": 1735040,
"hblkhd": 504512512,
"hblks": 103,
"keepcost": 131472,
"ordblks": 155074,
"smblks": 24261,
"uordblks": 2045626336, // <======== in use
"usmblks": 0

"total": "221.57049GB", // <========== RSS
"total_bytes": 237909499904
```

Ищем утечку памяти

Инструмент	Результат
01 Jemalloc + jeprof	Не воспроизводится ✖
02 Mallinfo	Ненадежный результат ✖

3. Пишем link-time wrapper-ы




```
target_link_options(core PUBLIC "-Wl,--wrap,malloc")
target_link_options(core PUBLIC "-Wl,--wrap,calloc")
target_link_options(core PUBLIC "-Wl,--wrap,free")
target_link_options(core PUBLIC "-Wl,--wrap,posix_memalign")
…
```

3. Пишем link-time wrapper-ы

```
E20250729 12:16:04.467710 SP1 PID:6162  6352 malloc_stats.cpp:141] 120 sec. report: MMM: in-use stats:
malloc = 1.35290GB, numa = 72.55408GB, align = 88.44939MB, live allocs count = 9949951, unknown dealloc
count = 0
```

Статистика полученная
с собственной инструментации
полностью согласуется с jemalloc

Проблема перестала
воспроизводиться

Ищем утечку памяти

Инструмент		Результат	
01	Jemalloc + jeprof	→	Не воспроизводится ✖
02	Mallinfo	→	Ненадежный результат ✖
03	Link-time wrappers	→	Не воспроизводится ✖

4. Во всем виноват posix_memalign?

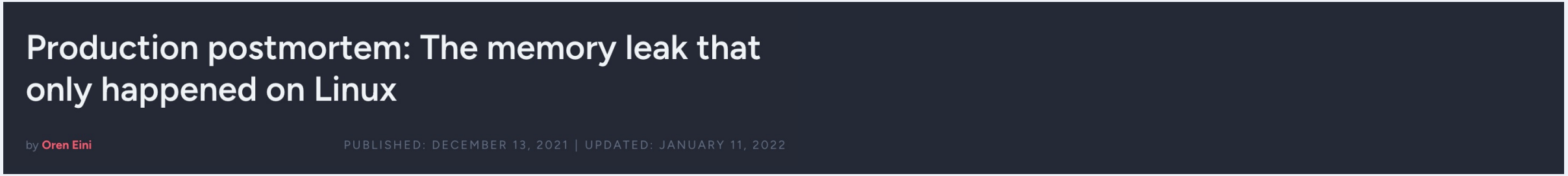
У jemalloc и link-time wrapper-ов со статистиками есть общая вещь →

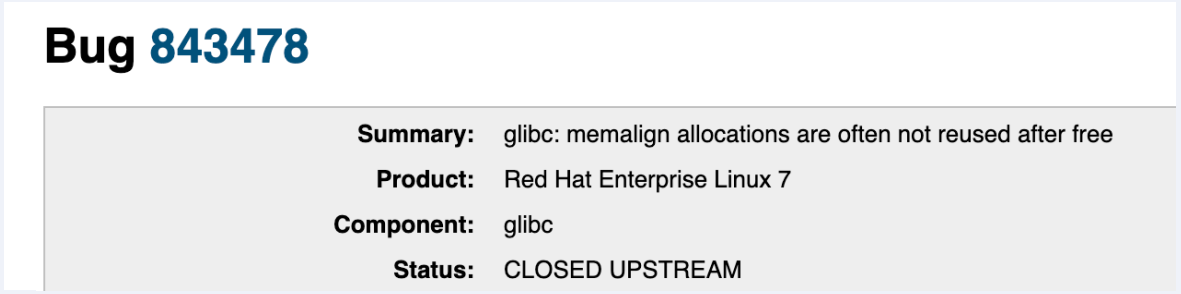
Кастомная имплементация posix_memalign()

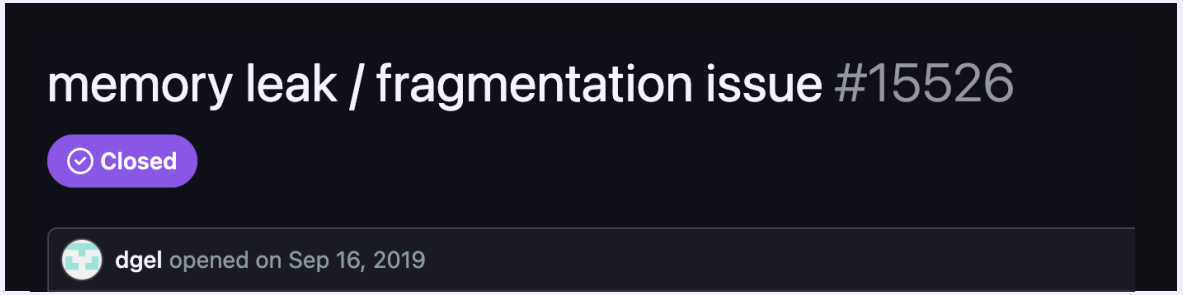
У glibc довольно простой алгоритм для posix_memalign →

- **Мапить больше чем нужно**
- **Лишнее отправить во внутренний freelist**

4. Во всем виноват posix_memalign?







4. Во всем виноват posix_memalign?

Убираем **posix_memalign**
из сценария вообще,
модифицируя код приложения

4. Во всем виноват posix_memalign?

Убираем **posix_memalign**
из сценария вообще,
модифицируя код приложения

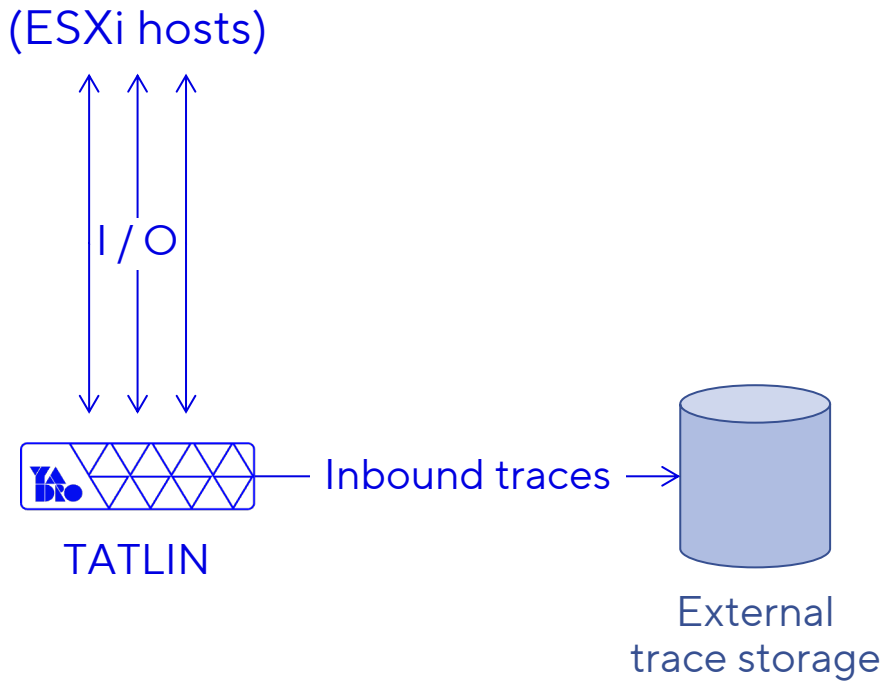
Ничего не меняется,
память течет с той же скоростью

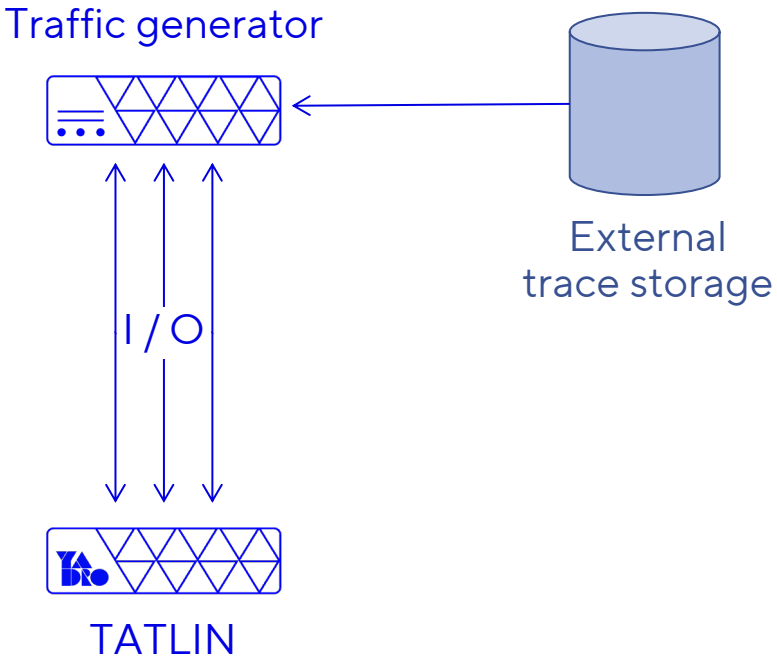
Ищем утечку памяти:

	Инструмент	Результат
01	Jemalloc + jeprof	Не воспроизводится ✖
02	Mallinfo	Ненадежный результат ✖
03	Link-time wrappers	Не воспроизводится ✖
04	Posix_memalign	Влияние исключили ✖

5. Trace + Replay?

Трейсим I/O в data path и пытаемся зареплеить ее в TATLIN,
чтобы избавиться от долгих тестов с полноценным кластером





5. Trace + Replay?

Запускаем тесты
с такой нагрузкой

Проблема
не воспроизводится

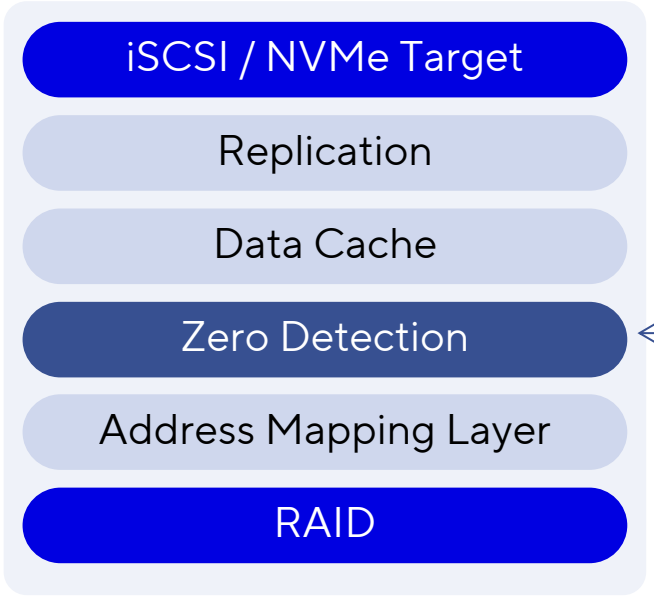
Ищем утечку памяти:

	Инструмент	Результат
01	Jemalloc + jeprof	Не воспроизводится ✖
02	Mallinfo	Ненадежный результат ✖
03	Link-time wrappers	Не воспроизводится ✖
04	Posix_memalign	Влияние исключили ✖
05	Traffic replay	Не воспроизводится ✖

6. Zero Detection

Может быть, специфика не в том, как пишет ESXi
в массив при форматировании VMFS, а что именно он пишет?

А судя по трассам пишет он нули

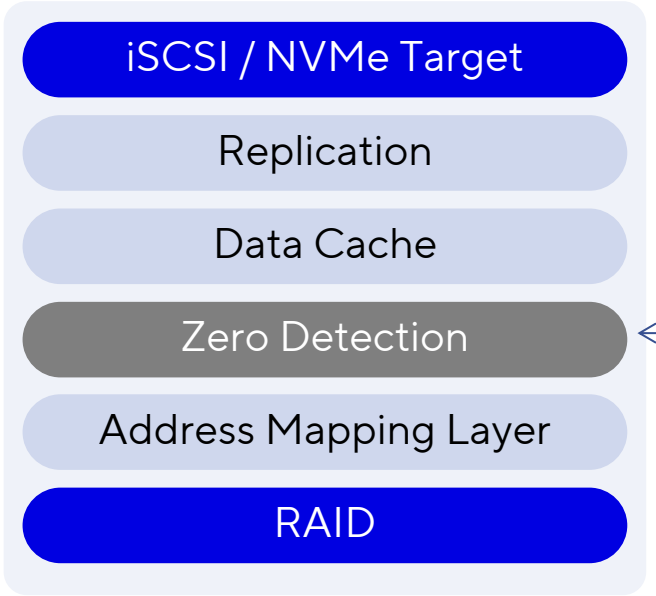


При вытеснении из Write-кэша,
находит 64k-блоки полностью заполненные
нулями → конвертирует их в Unmap

6. Zero Detection

Может быть, специфика не в том, как пишет ESXi
в массив при форматировании VMFS, а что именно он пишет?

А судя по трассам пишет он нули



Выключаем его специальным фича-флагом,
но утечка все равно остается

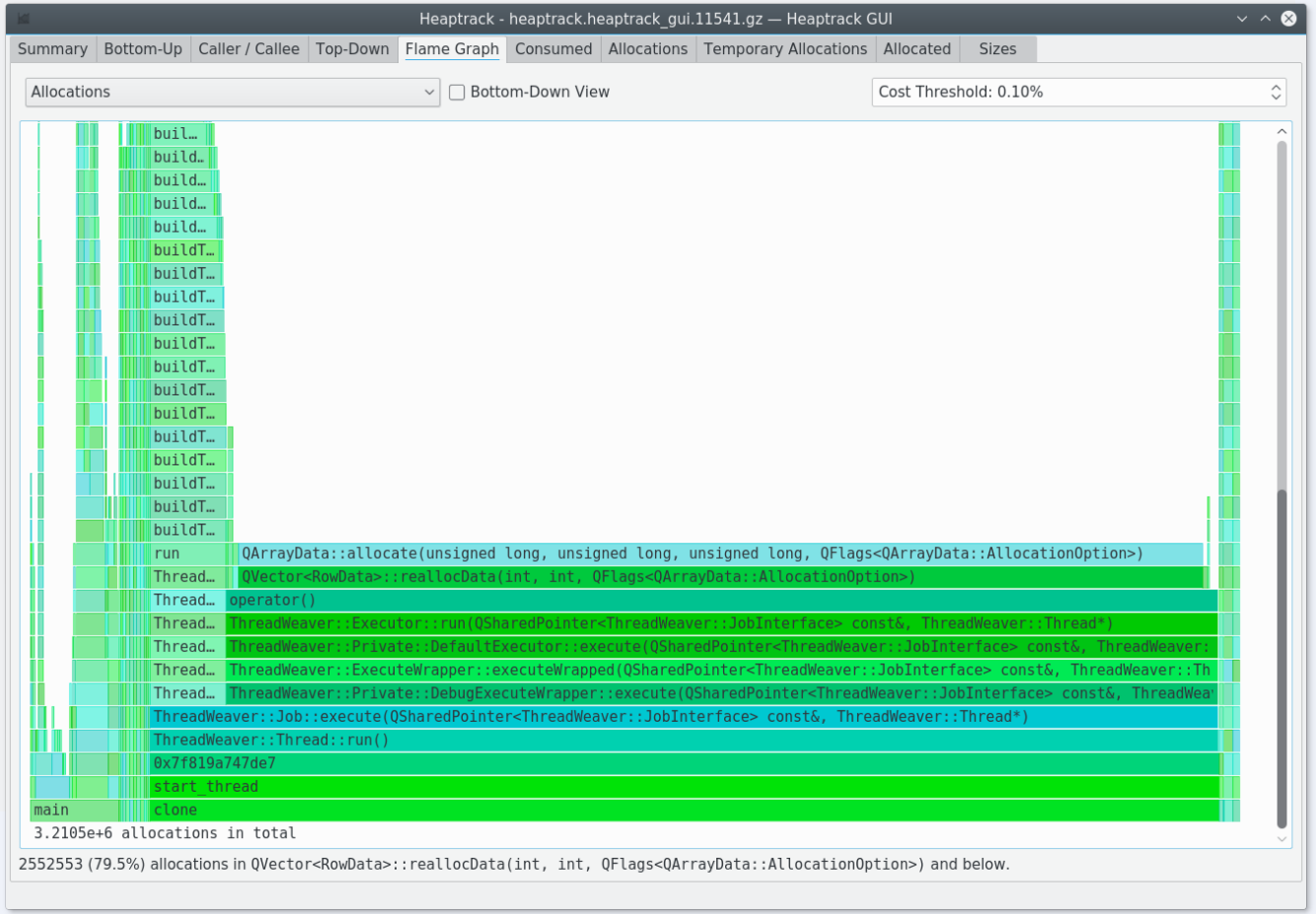
Ищем утечку памяти

	Инструмент	Результат
01	Jemalloc + jeprof	Не воспроизводится ✖
02	Mallinfo	Ненадежный результат ✖
03	Link-time wrappers	Не воспроизводится ✖
04	Posix_memalign	Влияние исключили ✖
05	Traffic replay	Не воспроизводится ✖
06	Zero detection feature	Влияние исключили ✖

7. Пробуем tool-ы для поиска утечек

Runtime тулы:

- Leak Sanitizer
- Valgrind
- Heaptrack



7. Пробуем tool-ы для поиска утечек

Тулы трекающие аллокации вносят оверхед

Под реальной нагрузкой на железе это приводит:

- К задержкам в data path
- Таймаутам
- Ошибкам
- Крешам

Достоверных результатов снять в рантайме не получилось

Ищем утечку памяти

	Инструмент	Результат
01	Jemalloc + jeprof	Не воспроизводится ✕
02	Mallinfo	Ненадежный результат ✕
03	Link-time wrappers	Не воспроизводится ✕
04	Posix_memalign	Влияние исключили ✕
05	Traffic replay	Не воспроизводится ✕
06	Zero detection feature	Влияние исключили ✕
07	Runtime tooling	Ненадежный результат ✕

8. Tool, который в итоге нас спас

Core Analyzer — расширение к gdb,
позволяющее анализировать heap в дампе:

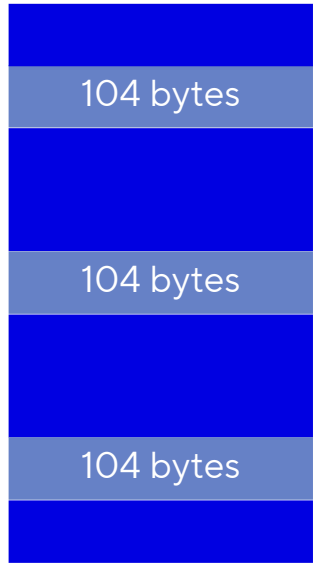
```
There are 228 arenas and 107 mmap-ed memory blocks Total 77.2GB
        Total 9195329 blocks in-use of 1.4GB
        Total 104581 blocks free of 75.9GB
```

Heap-блоки в динамических аренах
сильно фрагментированы:

```
....

[0x7fa94c000030 - 0x7fa94ff49000] Total 63MB in-use 14(2KB) free 11(63MB)
[0x7fa9dc000030 - 0x7fa9dff49000] Total 63MB in-use 18(3KB) free 10(63MB)
[0x7faa68000030 - 0x7faa6bf49000] Total 63MB in-use 15(2KB) free 13(63MB)
[0x7faac8000030 - 0x7faacbf49000] Total 63MB in-use 19(2KB) free 11(63MB)

.....
```

8. Tool, который в итоге нас спас

```
(gdb) heap /c 0x7fb0f8000030
        Dynamic arena (0x7fce48000020): [0x7fb0f8000020 - 0x7fb0fbf49000]
                        [0x7fb0f8000030 - 0x7fb0f8000078] 72 bytes inuse
                [0x7fb0f8000080 - 0x7fb0f87e9088] 8294408 bytes free
                        [0x7fb0f87e9090 - 0x7fb0f87e90f8] 104 bytes inuse
                [0x7fb0f87e9100 - 0x7fb0f8fd2108] 8294408 bytes free
                        [0x7fb0f8fd2110 - 0x7fb0f8fd2178] 104 bytes inuse
                [0x7fb0f8fd2180 - 0x7fb0f97bb188] 8294408 bytes free
                        [0x7fb0f97bb190 - 0x7fb0f97bb1f8] 104 bytes inuse
                [0x7fb0f97bb200 - 0x7fb0f9fa4208] 8294408 bytes free
                        [0x7fb0f9fa4210 - 0x7fb0f9fa4278] 104 bytes inuse
                [0x7fb0f9fa4280 - 0x7fb0fa78d288] 8294408 bytes free

. . . .

Total inuse 13 blocks 3064 bytes
Total free 10 blocks 66356008 bytes
```



64 MiB heap

Ищем утечку памяти

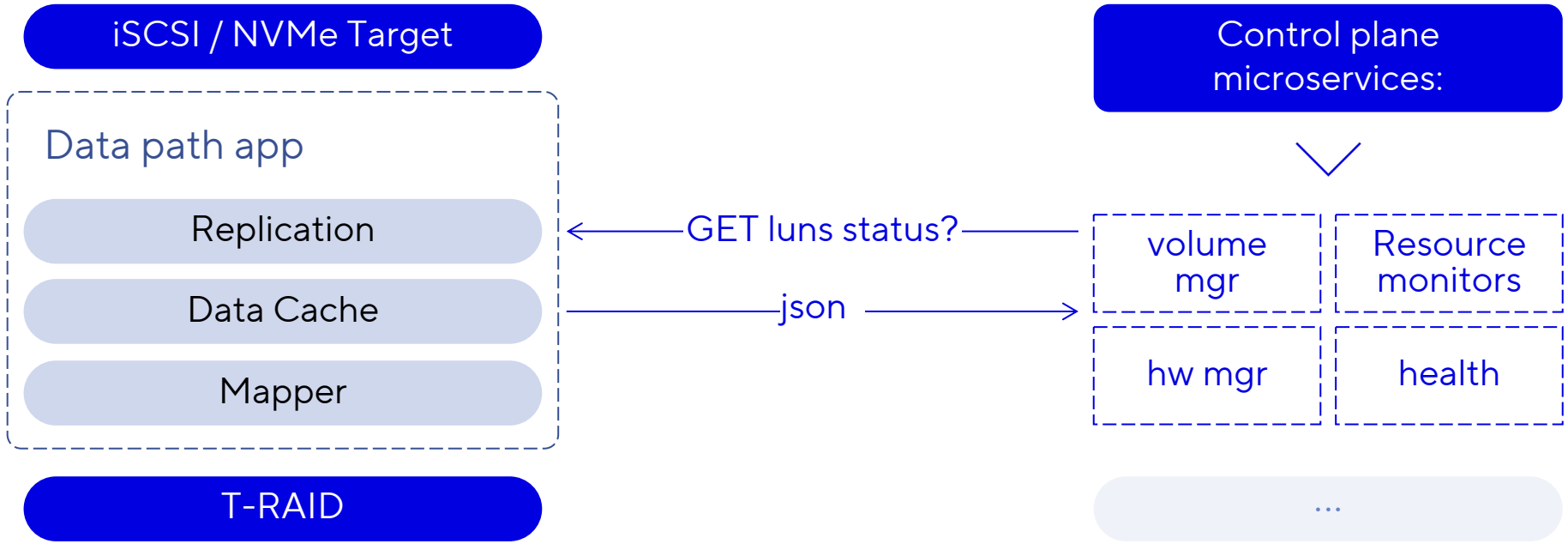
	Инструмент	Результат
01	Jemalloc + jeprof	Не воспроизводится ✕
02	Mallinfo	Ненадежный результат ✕
03	Link-time wrappers	Не воспроизводится ✕
04	Posix_memalign	Влияние исключили ✕
05	Traffic replay	Не воспроизводится ✕
06	Zero detection feature	Влияние исключили ✕
07	Runtime tooling	Ненадежный результат ✕
08	**Core Analyzer**	**Win!** +

В это трудно поверить, но надо признаться

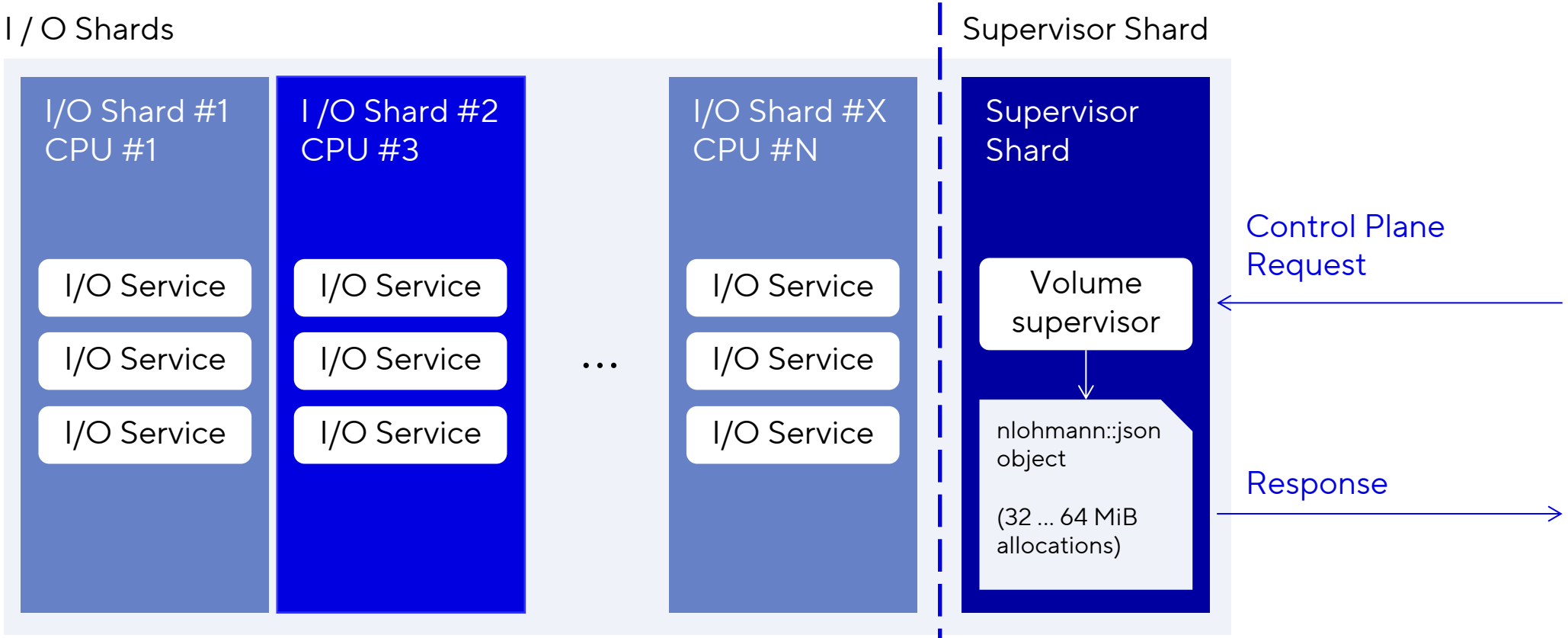
На самом деле, каждый эксперимент показывает, что потребление памяти не вызвано утечкой аллокаций в самом приложении

Мы имеем дело именно с фрагментацией heap-а!

В чем было дело?



В чем было дело?



В чем было дело?

```
MALLOC_MMAP_THRESHOLD = 131072
MALLOC_TRIM_THRESHOLD = 131072
```

Mmap Threshold

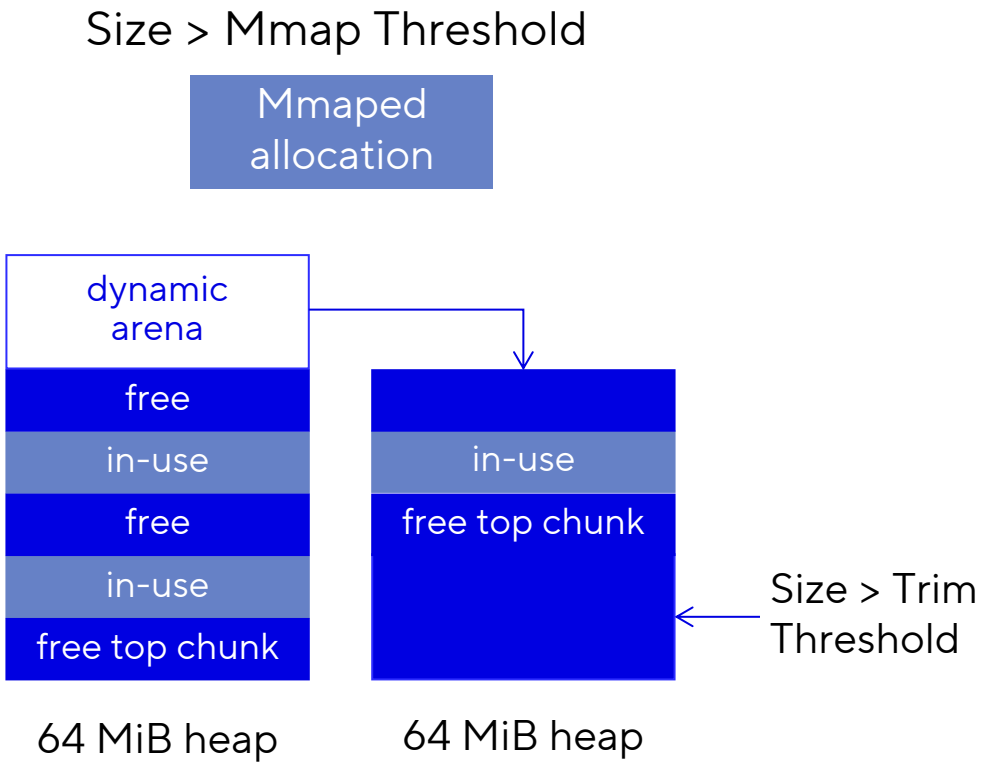
Размер аллокации в байтах, выше которого malloc не пытается искать место в heap-е, а сразу идет просить отдельную память у ОС через mmap

Trim Threshold

Размер top chunk-а, превысив который аллокатором будет вызван **heap_trim():**

→ Сделает madvise (или brk для главной арены)

→ Вернет память в ОС



В чем было дело?

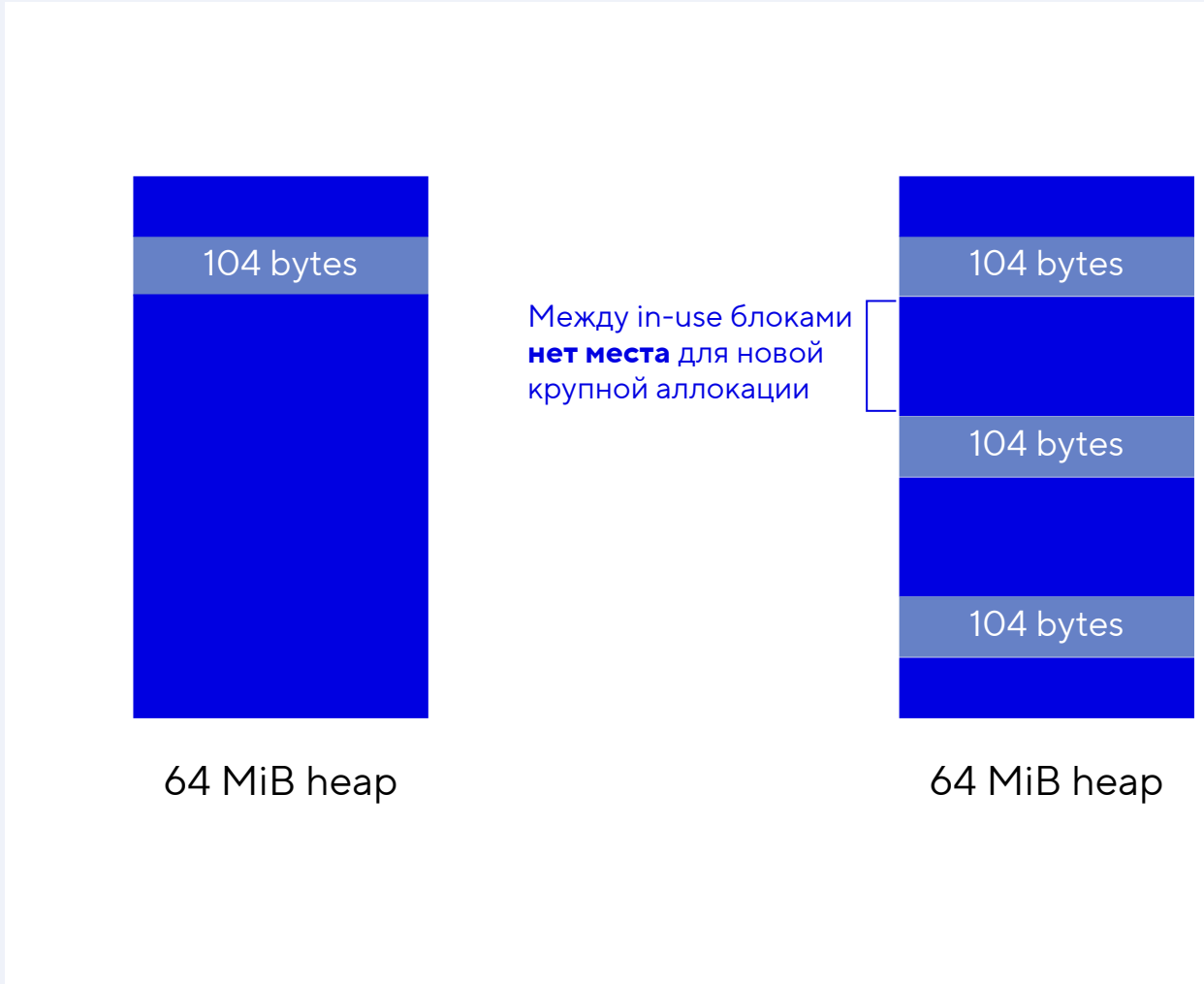
```
void __libc_free (void *mem)
{
  // …
  p = mem2chunk (mem);
  if (chunk_is_mmapped (p)) // при аллокации чанк был больше порога и аллокация делалась через mmap
    {
         if (!mp_.no_dyn_threshold                          // включен adjustment лимитов (default)
            && chunksize_nomask (p) > mp_.mmap_threshold    // аллокация была больше текущего порога
            && chunksize_nomask (p) <= DEFAULT_MMAP_THRESHOLD_MAX // но меньше или равно 32 MiB
            && !DUMPED_MAIN_ARENA_CHUNK (p))
        {
          mp_.mmap_threshold = chunksize (p); // <==== mmap порог = размеру освобождаемой аллокации
          mp_.trim_threshold = 2 * mp_.mmap_threshold; // <==== trim порог в два раза больше
         }
      munmap_chunk (p);
      return;
    }
// …
```

```
(gdb) p mp_
$7 = {trim_threshold = 62922752, .... , mmap_threshold = 31461376, ... , no_dyn_threshold = 0,
```

В чем было дело?

С такими настройками легко получить классическую проблему фрагментации heap-а

Паттерн нагрузки от Vmware провоцирует Data Cache чередовать короткоживущие большие аллокации и маленькие, но долгоживущие



Fix

Auto-adjustment можно выключить, просто задав свои кастомные значения параметрам

```
MALLOC_MMAP_THRESHOLD = 131072
MALLOC_TRIM_THRESHOLD = 262144
```

Провели perf-замеры: ›

```
Environment="MALLOC_MMAP_THRESHOLD_=4194304"
Environment="MALLOC_TRIM_THRESHOLD_=8388608"
```

Выводы

01

Важно тестировать совместимость с продуктами сторонних вендоров, они могут делать неожиданные вещи

02

Важно стараться «не искать под фонарем» — при отладке нужно полагаться на факты и эксперименты, а не на специфику и вероятности

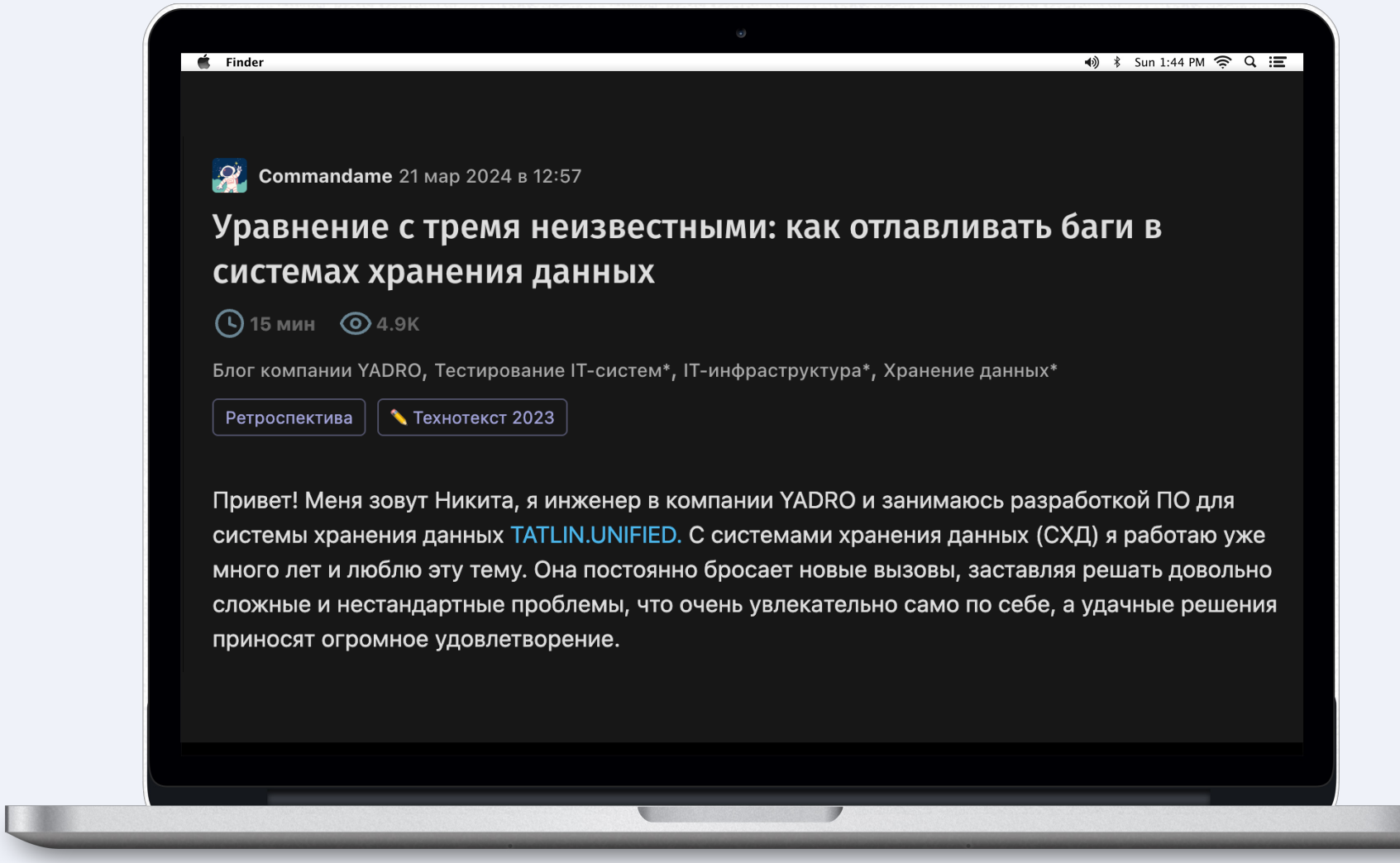
03

Важно в такой ситуации пробовать разные тулы и в случае успеха — описывать их использование во внутренней документации

Это помогло коллегам пофиксить несколько других багов, и улучшить качество продукта

Еще про баги в TATLIN

- **Интересная история**
  про data corrupt
  в Postgres



Спасибо!

Будем на связи!

БУДУЩЕЕ
В НАШИХ РУКАХ