Реактивный раздатчик ok.ru/music

Вадим Цесько

Одноклассники

- **71M пользователей** (MAU)
- **7000 машин** в **4 ДЦ**

[1] Без учёта CDN

Одноклассники

- **71M пользователей** (MAU)
- **7000 машин** в **4 ДЦ**
- **2 Тб/с**[1]

[1] Без учёта CDN

Одноклассники

- **71M пользователей** (MAU)
- **7000 машин** в **4 ДЦ**
- **2 Тб/с**[1]
- До **100K запросов/с** на ноду (4 ядра)
- **99%** $<$ **100 мс**

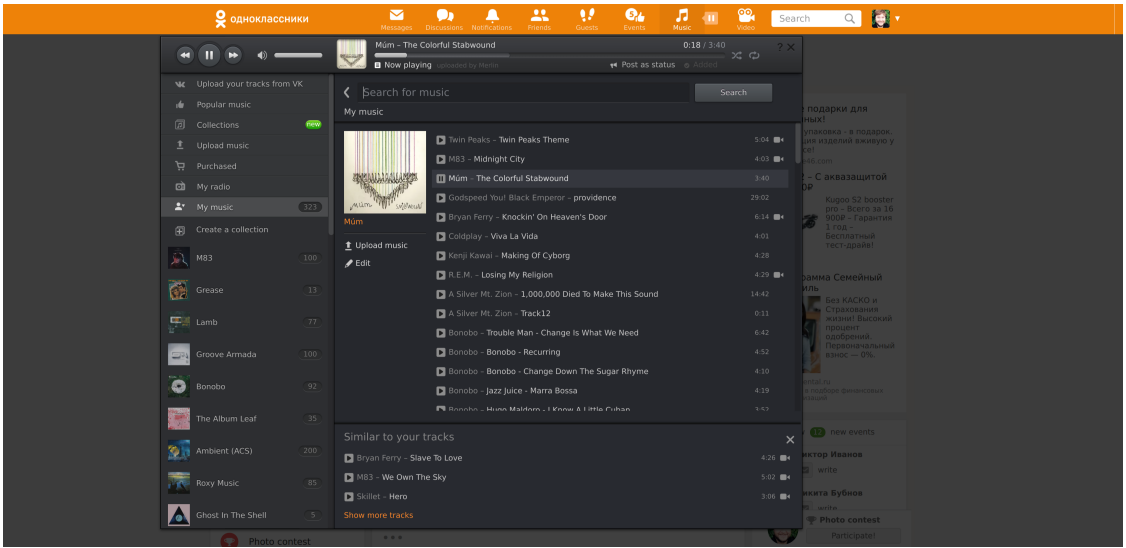
[1] Без учёта CDN

Одноклассники

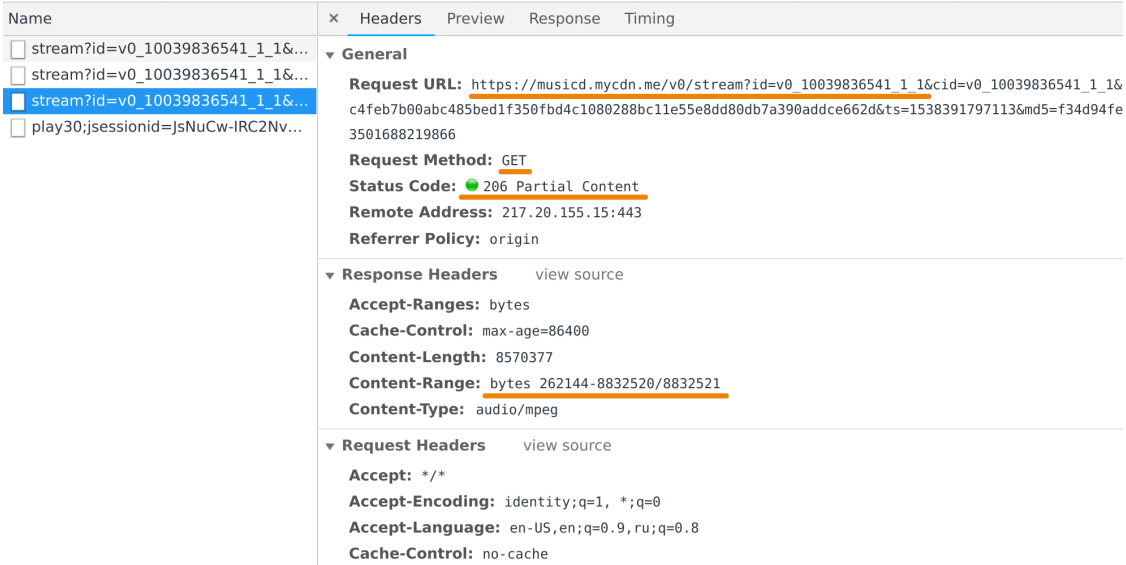
- **71M пользователей** (MAU)
- **7000 машин** в **4 ДЦ**
- **2 Тб/с**[1]
- До **100K запросов/с** на ноду (4 ядра)
- **99%** $<$ **100 мс**
- **Java**

[1] Без учёта CDN

ok.ru/music


одноклассники
Messages
Discussions
Notifications
Friends
Guests
Events
Music
Video
Search
Mùm – The Colorful Stabwound
0:18 / 3:40
Now playing
Post as status
Upload your tracks from VK
Popular music
Collections
Upload music
Purchased
My radio
My music 323
Create a collection
M83 100
Grease 13
Lamb 77
Groove Armada 100
Bonobo 92
The Album Leaf 35
Ambient (ACS) 200
Roxy Music 85
Ghost In The Shell 5
Search for music
Search
My music
Mùm
Upload music
Edit
Twin Peaks – Twin Peaks Theme 5:04
M83 – Midnight City 4:03
Mùm – The Colorful Stabwound 3:40
Godspeed You! Black Emperor – providence 29:02
Bryan Ferry – Knockin' On Heaven's Door 6:14
Coldplay – Viva La Vida 4:01
Kenji Kawai – Making Of Cyborg 4:28
R.E.M. – Losing My Religion 4:29
A Silver Mt. Zion – 1,000,000 Died To Make This Sound 14:42
A Silver Mt. Zion – Track12 0:11
Bonobo – Trouble Man - Change Is What We Need 6:42
Bonobo – Bonobo - Recurring 4:52
Bonobo – Bonobo - Change Down The Sugar Rhyme 4:10
Bonobo – Jazz Juice - Marra Bossa 4:19
Similar to your tracks
Bryan Ferry – Slave To Love 4:26
M83 – We Own The Sky 5:02
Skillet – Hero 3:06
Show more tracks
Photo contest

Раздатчик ok.ru/music



Раздатчик ok.ru/music

- Раздаёт музыку с 2010

[2] http://profyclub.ru/docs/174

[3] http://2017.jokerconf.com/2017/talks/4fasygftomyekgaqaaiauo/

Раздатчик ok.ru/music

- Раздаёт музыку с 2010
- $\leq$ **100 Гб/с**
- $\leq$ **500K соединений**

[2] http://profyclub.ru/docs/174

[3] http://2017.jokerconf.com/2017/talks/4fasygftomyekgaqaaiauo/

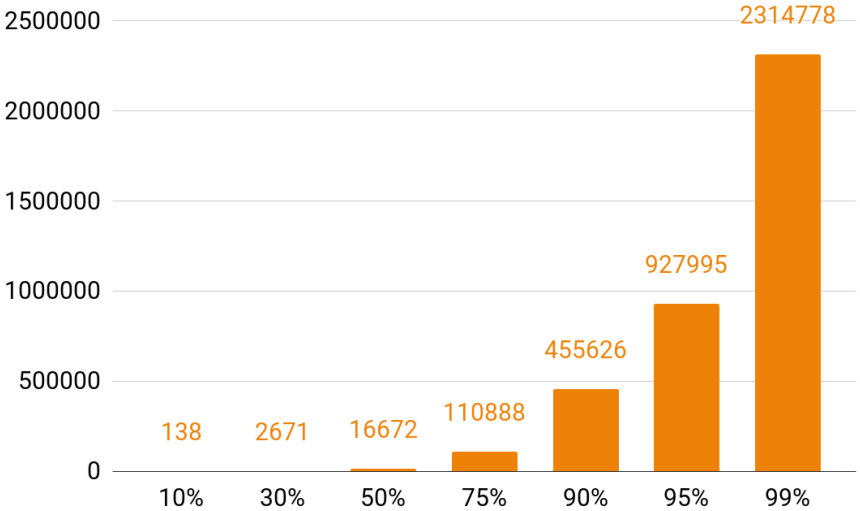
Раздатчик ok.ru/music

- Раздаёт музыку с 2010
- $\leq$ **100 Гб/с**
- $\leq$ **500K соединений**
- Кеширующий фронтенд перед one-blob-storage[2] + one-cold-storage[3]

[2] http://profyclub.ru/docs/174

[3] http://2017.jokerconf.com/2017/talks/4fasygftomyekgaqaaiauo/

Длинный хвост



Must have

- До 100K «медленных» клиентов на ноду
- Масштабируемость
- Отказоустойчивость

Масштабировать

Что:

Масштабировать

Что:

- **Пропускную способность** кластера

Масштабировать

Что:

- **Пропускную способность** кластера
- **Суммарную ёмкость** кеша кластера

Масштабировать

Что:

- **Пропускную способность** кластера
- **Суммарную ёмкость** кеша кластера

Как:

Масштабировать

Что:

- **Пропускную способность** кластера
- **Суммарную ёмкость** кеша кластера

Как:

- **Горизонтально**: ДЦ и машины

Масштабировать

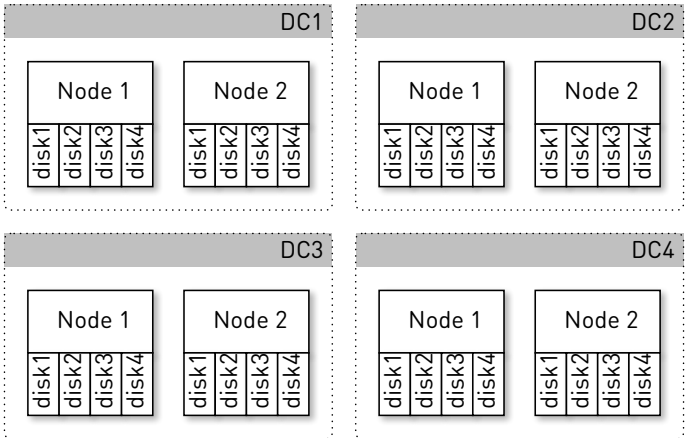
Что:

- **Пропускную способность** кластера
- **Суммарную ёмкость** кеша кластера

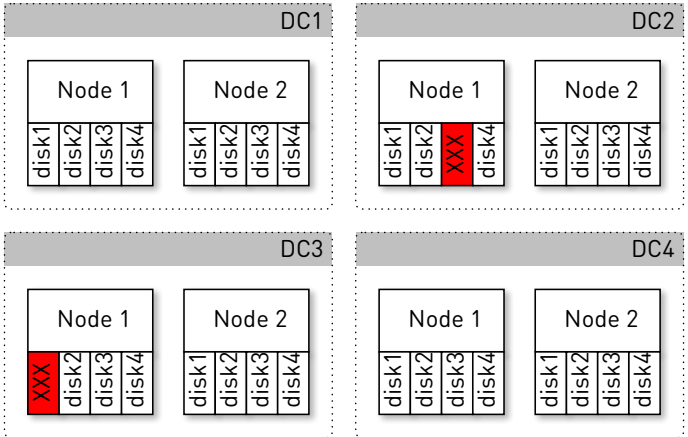
Как:

- **Горизонтально**: ДЦ и машины
- **Вертикально**: диски и RAM

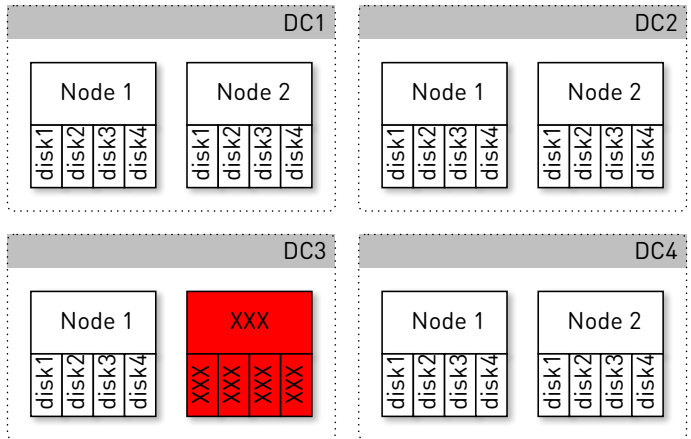
Эффект масштаба



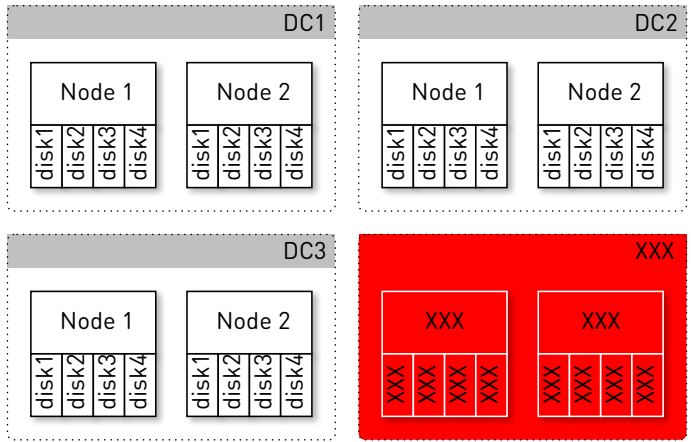
Отказ диска


DC1
Node 1
disk1
disk2
disk3
disk4
Node 2
disk1
disk2
disk3
disk4
DC2
Node 1
disk1
disk2
XXX
disk4
Node 2
disk1
disk2
disk3
disk4
DC3
Node 1
XXX
disk2
disk3
disk4
Node 2
disk1
disk2
disk3
disk4
DC4
Node 1
disk1
disk2
disk3
disk4
Node 2
disk1
disk2
disk3
disk4

Отказ машины



Отказ ДЦ


DC1
Node 1
disk1
disk2
disk3
disk4
Node 2
disk1
disk2
disk3
disk4
DC2
Node 1
disk1
disk2
disk3
disk4
Node 2
disk1
disk2
disk3
disk4
DC3
Node 1
disk1
disk2
disk3
disk4
Node 2
disk1
disk2
disk3
disk4
XXX
XXX
XXX
XXX
XXX
XXX
XXX
XXX
XXX
XXX
XXX
XXX

Всё (с)ломается[4]

- Прорабатываем сценарии отказов
  - ДЦ
  - Машины
  - Диски

[4] https://techtrain.ru/talks/1pqtojnosesumeo00ayccm/

Всё (с)ломается[4]

- Прорабатываем сценарии отказов
  - ДЦ
  - Машины
  - Диски
- Резервируем

[4] https://techtrain.ru/talks/1pqtojnosesumeo00ayccm/

Всё (с)ломается[4]

- Прорабатываем сценарии отказов
  - ДЦ
  - Машины
  - Диски
- Резервируем
  - Реплицируем данные

[4] https://techtrain.ru/talks/1pqtojnosesumeo00ayccm/

Всё (с)ломается[4]

- Прорабатываем сценарии отказов
  - ДЦ
  - Машины
  - Диски
- Резервируем
  - Реплицируем данные
  - Запас по пропускной способности

[4] https://techtrain.ru/talks/1pqtojnosesumeo00ayccm/

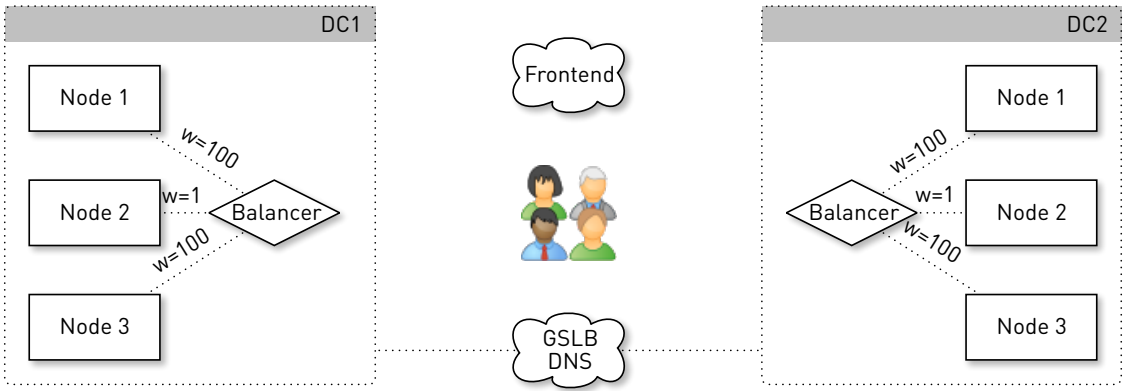
Балансировка запросов

- По **ДЦ**
  - Манёвры
  - Аварии

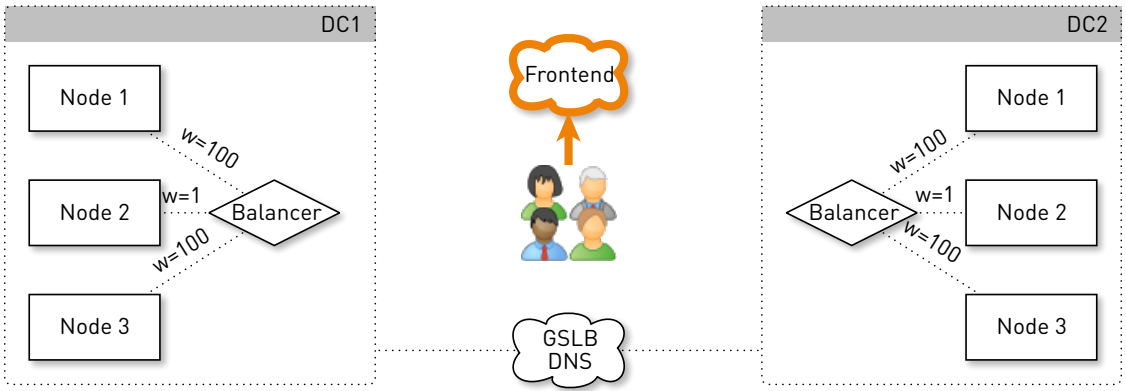
Балансировка запросов

- По **ДЦ**
  - Манёвры
  - Аварии
- По **нодам** внутри ДЦ
  - С **весами**
  - Плавный ввод в ротацию
  - Нагрузочное тестирование

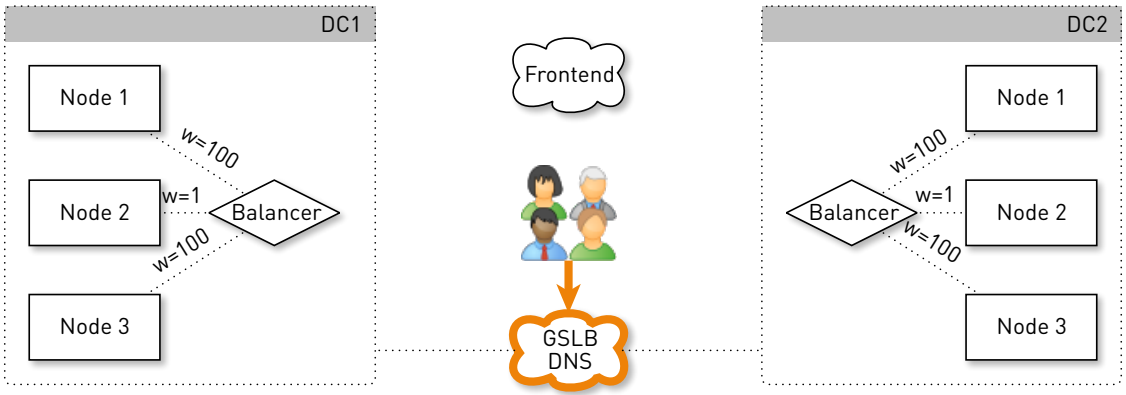
Путь запроса



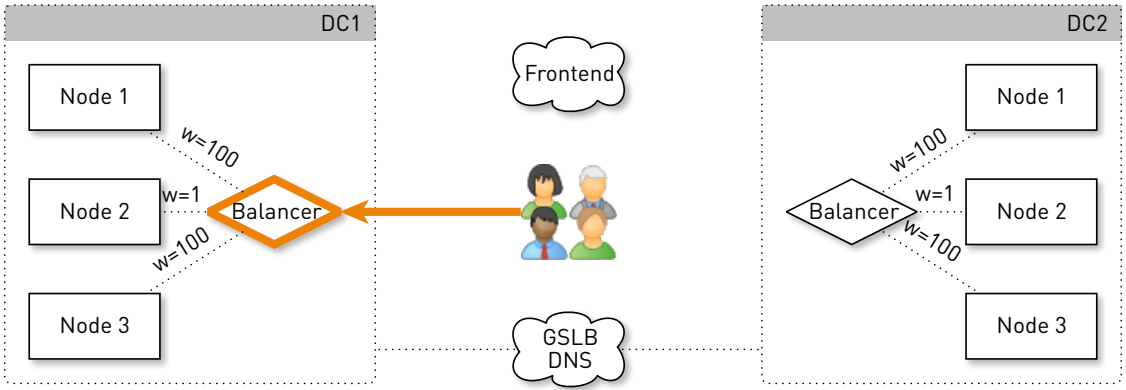
https://ok.ru/music



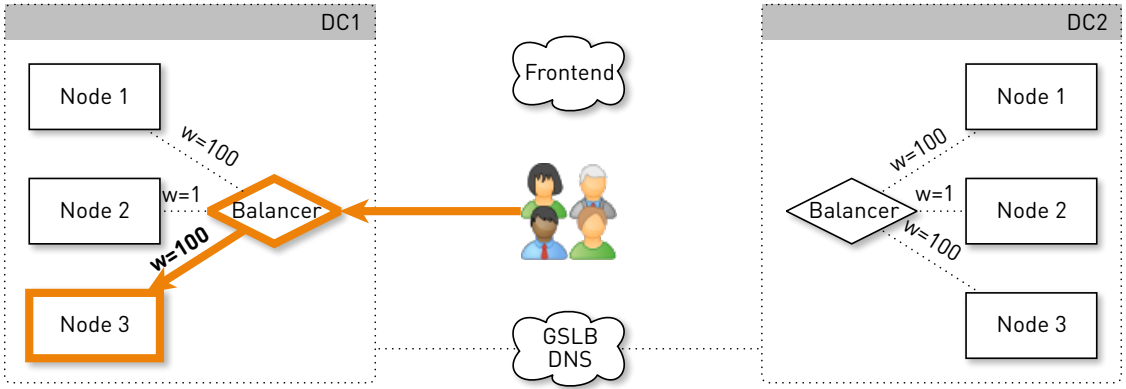
musicd.mycdn.me/v0/stream?id=...



5.61.23.6/v0/stream?id=...

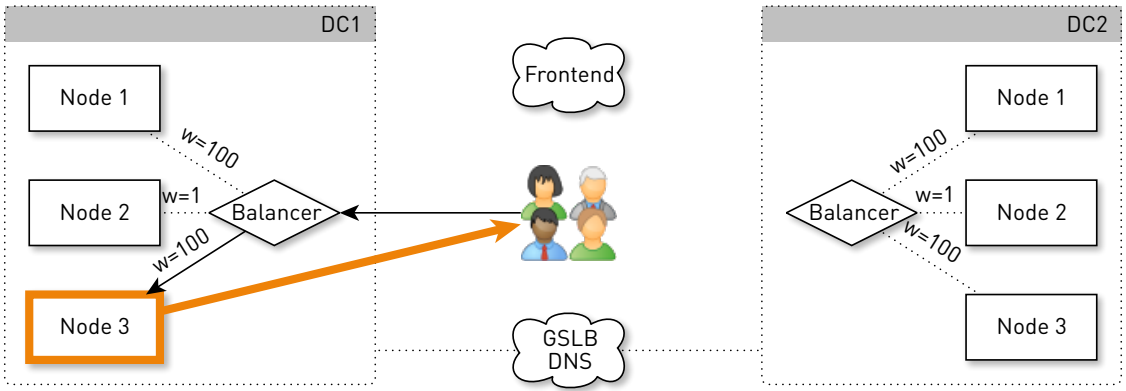


Соединение с нодой[5]

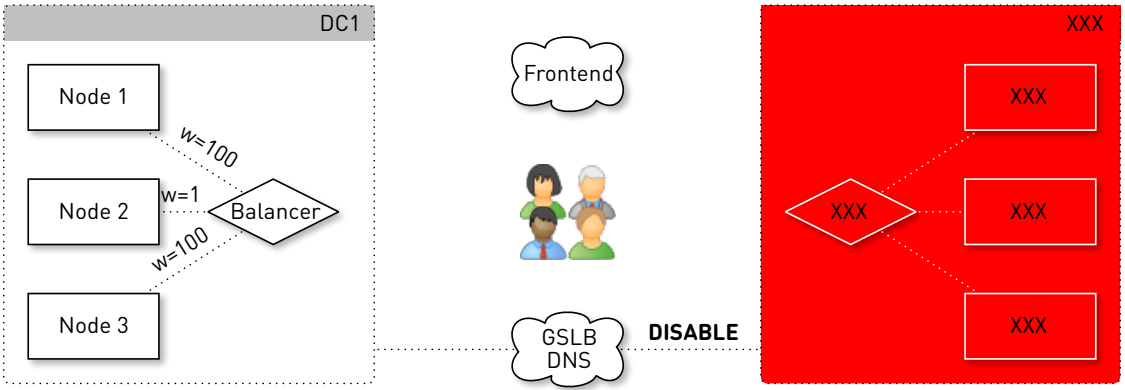


[5] http://www.highload.ru/2017/abstracts/2858.html

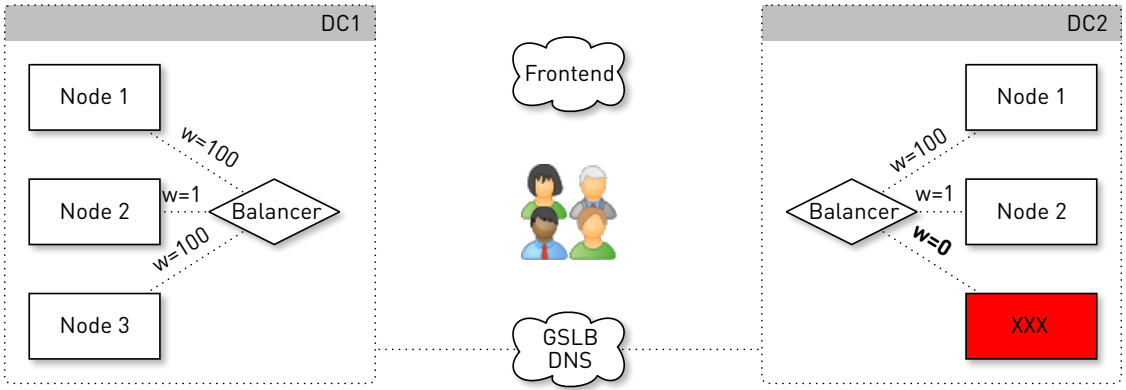
Direct Server Return



Сбой ДЦ


DC1
Node 1
Node 2
Node 3
w=100
w=1
w=100
Balancer
Frontend
GSLB
DNS
DISABLE
XXX
XXX
XXX
XXX
XXX

Сбой ноды


DC1
Node 1
Node 2
Node 3
w=100
w=1
w=100
Balancer
Frontend
GSLB
DNS
DC2
Node 1
Node 2
XXX
w=100
w=1
w=0
Balancer

Балансировка треков по нодам

- Независимые ДЦ

Балансировка треков по нодам

- Независимые ДЦ
- Равномерно по машинам

Балансировка треков по нодам

- Независимые ДЦ
- Равномерно по машинам
- Репликация треков на случай отказов

Балансировка треков по нодам

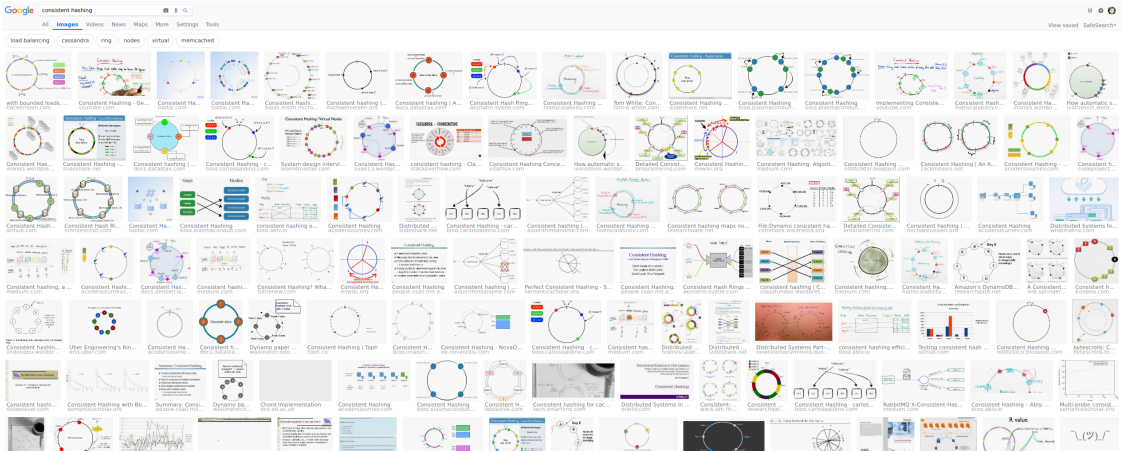
- Независимые ДЦ
- Равномерно по машинам
- Репликация треков на случай отказов
- Перераспределение нагрузки при отказах

Балансировка треков по нодам

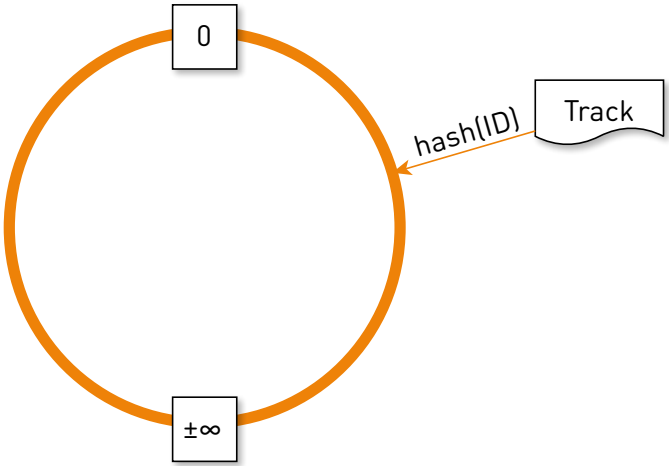
- Независимые ДЦ
- Равномерно по машинам
- Репликация треков на случай отказов
- Перераспределение нагрузки при отказах
- **Consistent hashing**[6]

[6] https://medium.com/@dgryski/consistent-hashing-algorithmic-tradeoffs-ef6b8e2fcae8

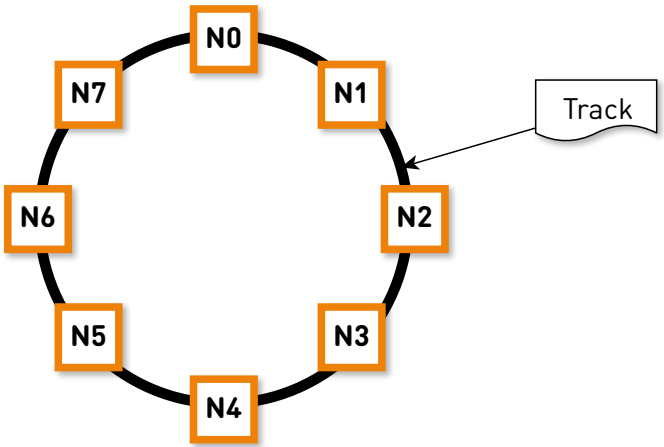
Consistent hashing



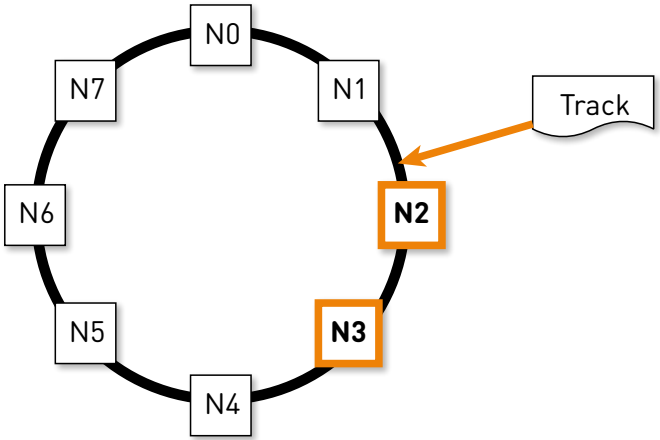
Строим кольцо хешей



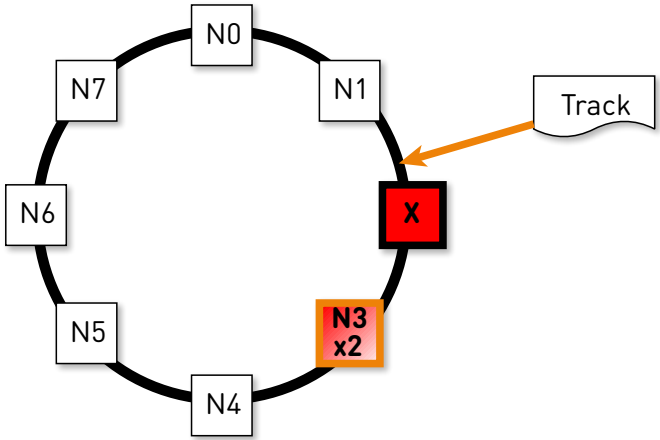
Делим интервалы между нодами



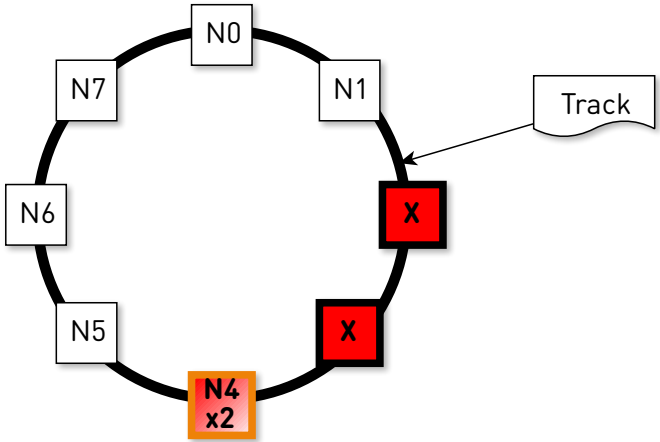
Определяем реплики



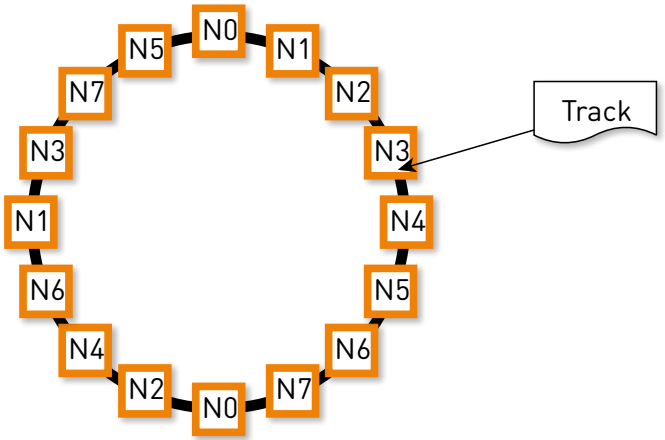
Отказ ноды



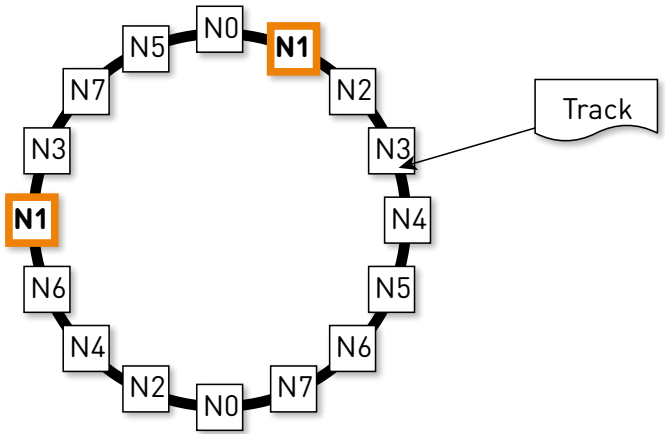
Каскадный отказ



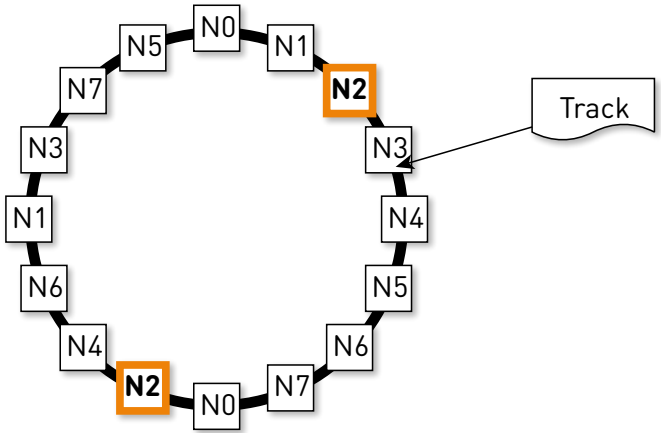
На практике vnodes ⋙ nodes



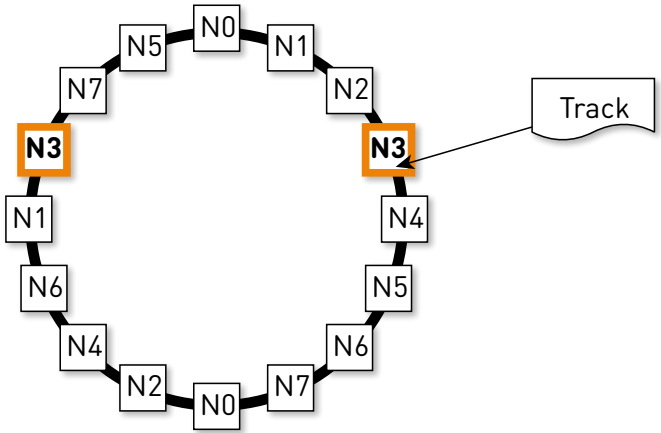
N1



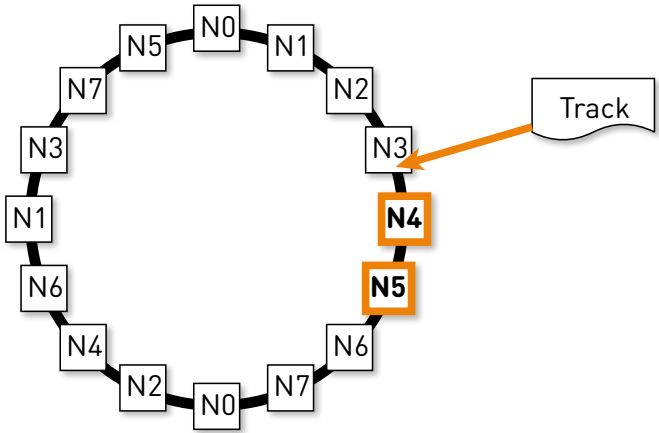
N2



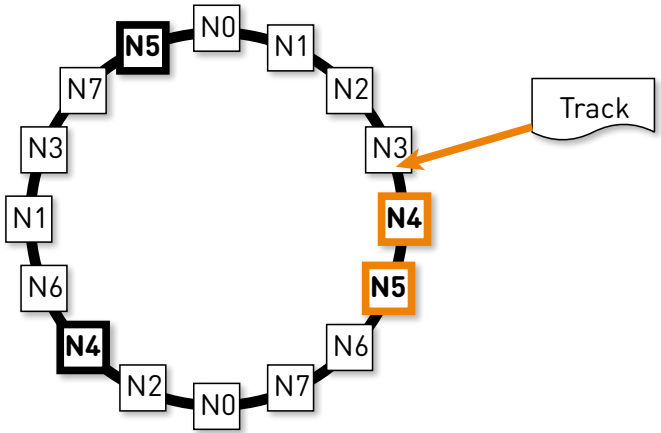
N3



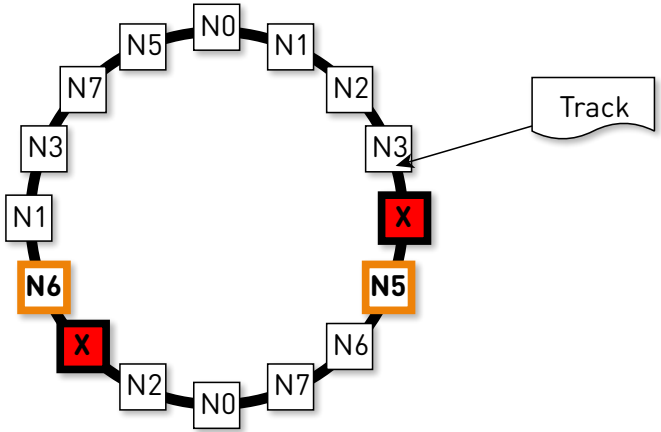
Так же определяем реплики



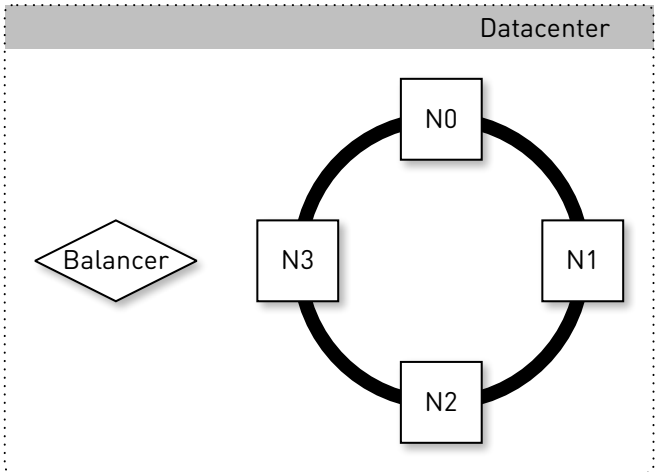
N4 + N5



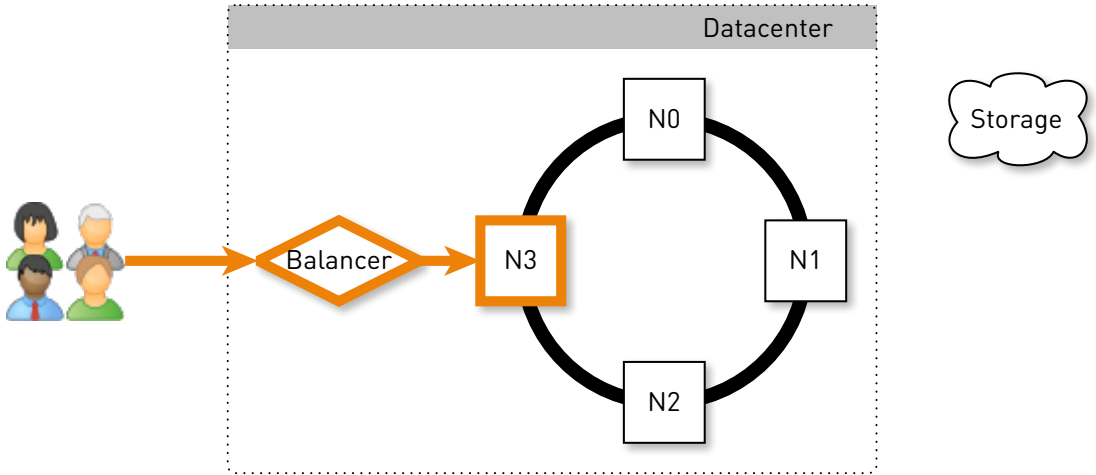
Отказ ноды



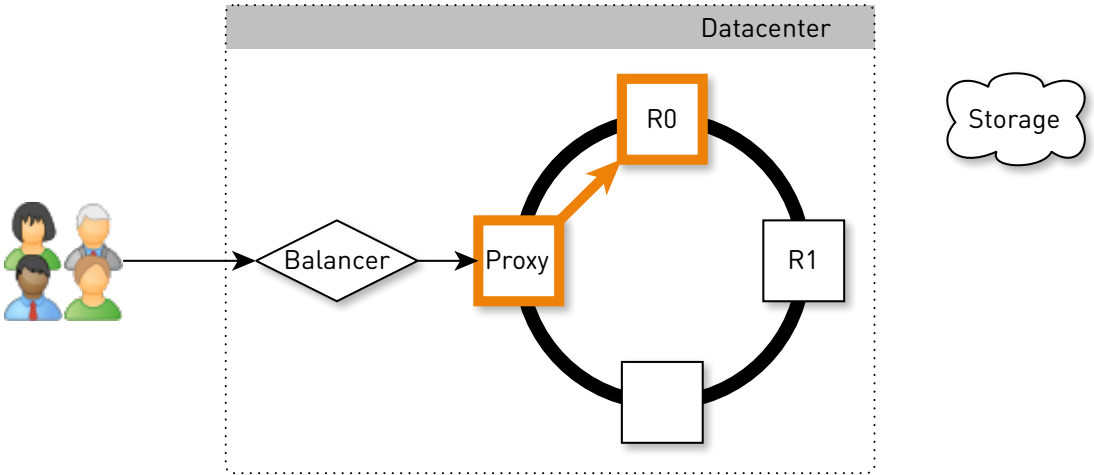
Отдача трека



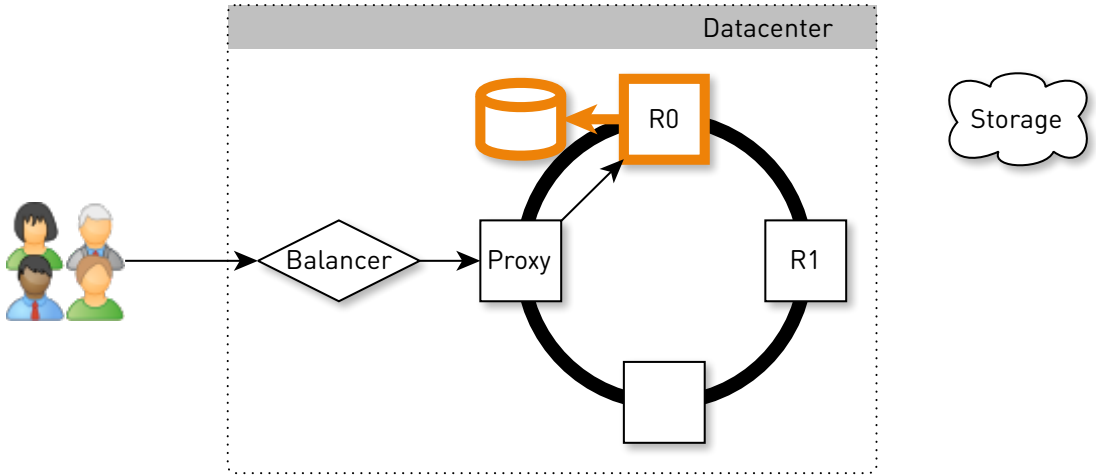
Случайная нода



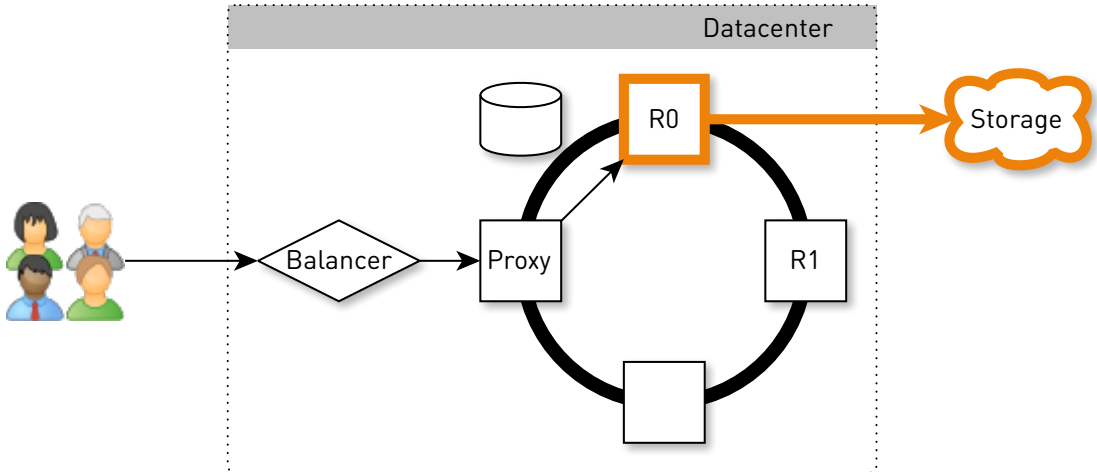
Проксируем



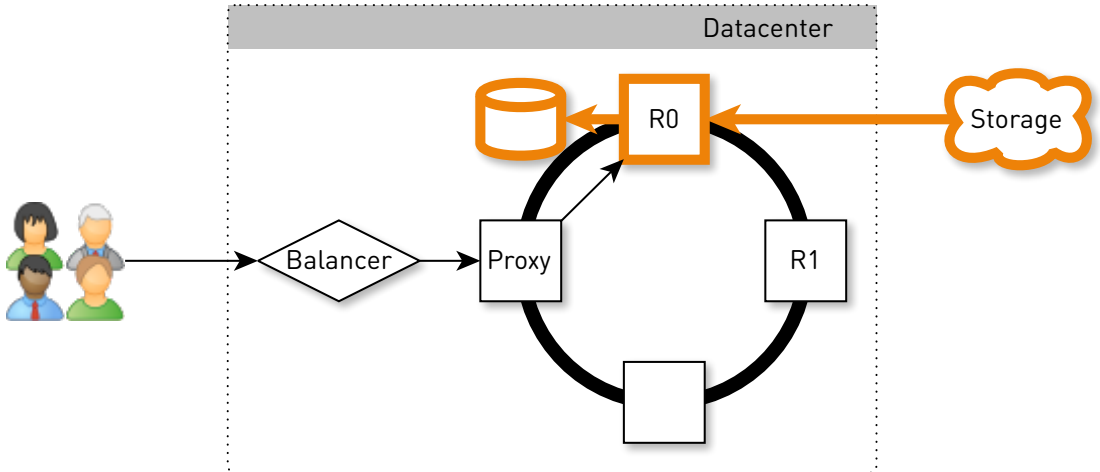
Ищем локально



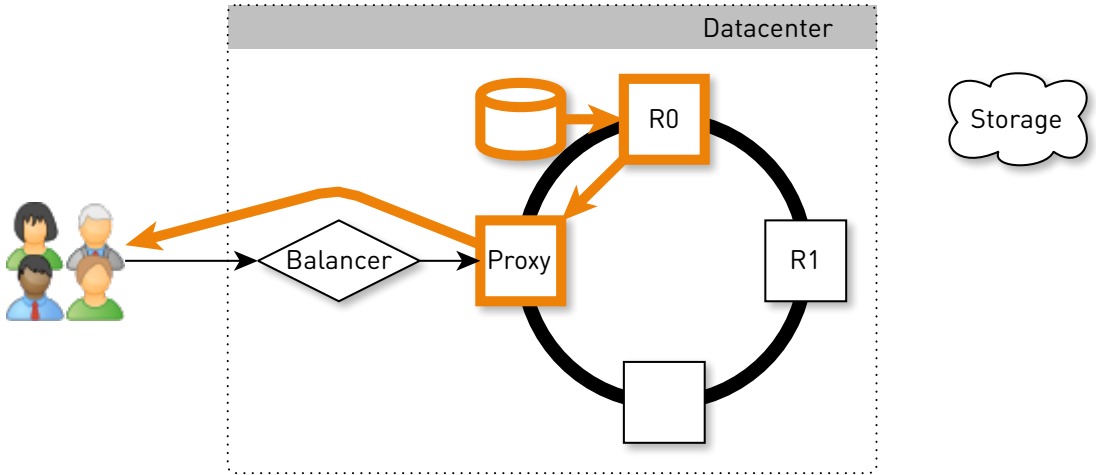
Подтягиваем данные



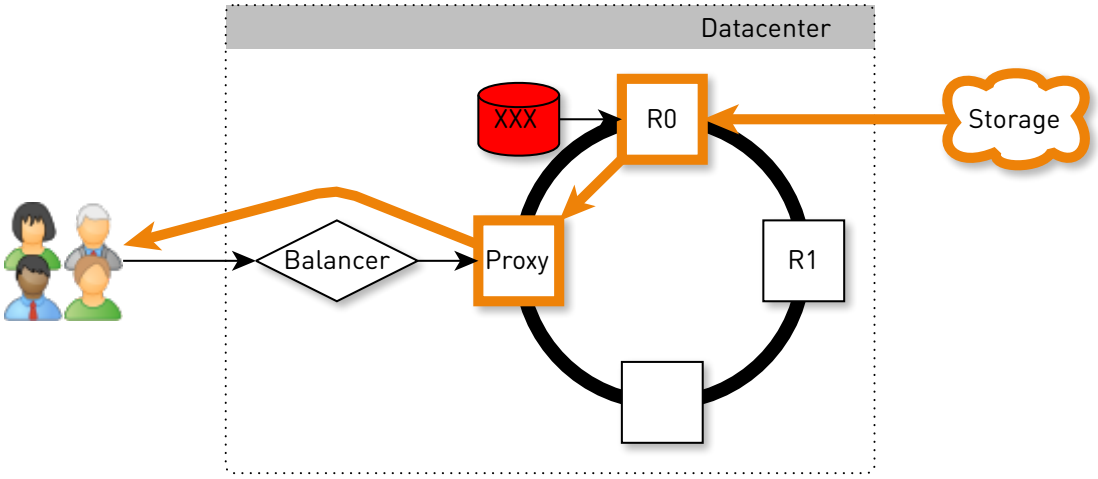
Сохраняем



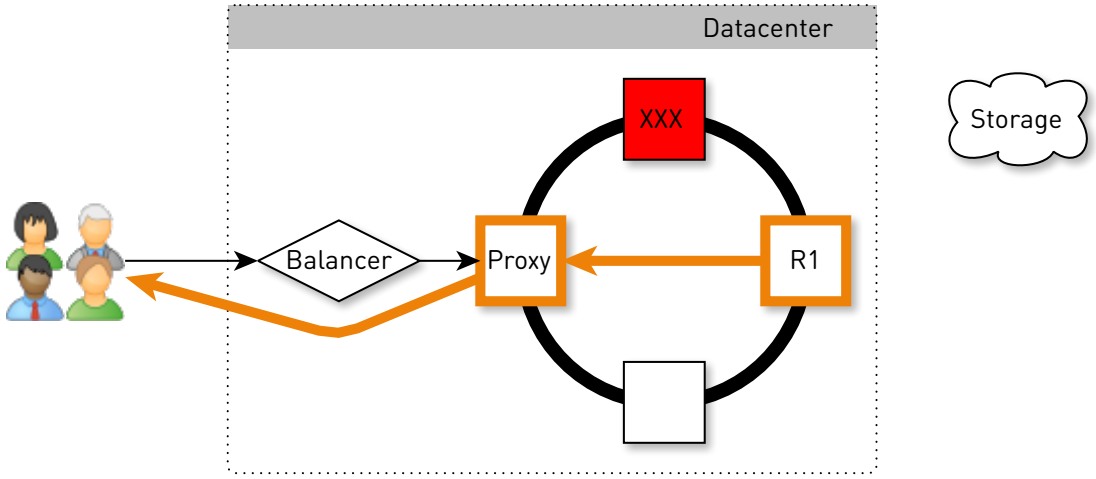
Отдаём с диска



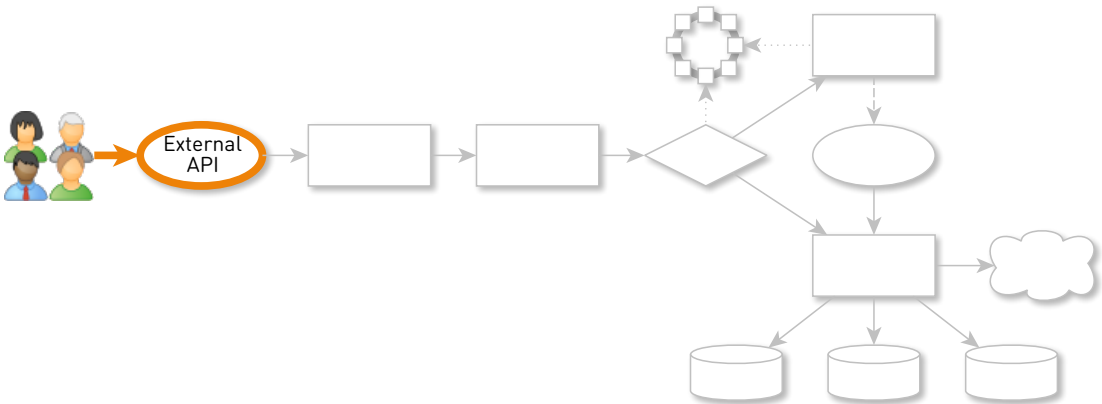
Сбой диска



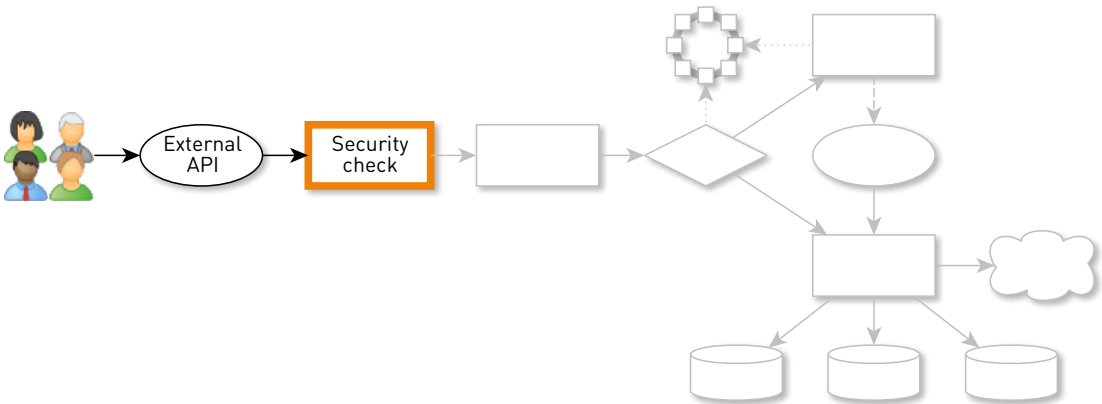
Сбой реплики



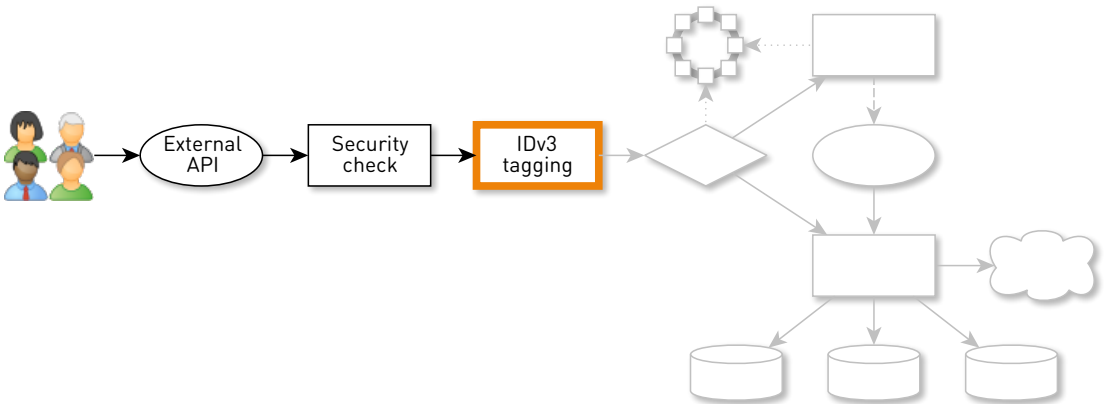
Нода изнутри



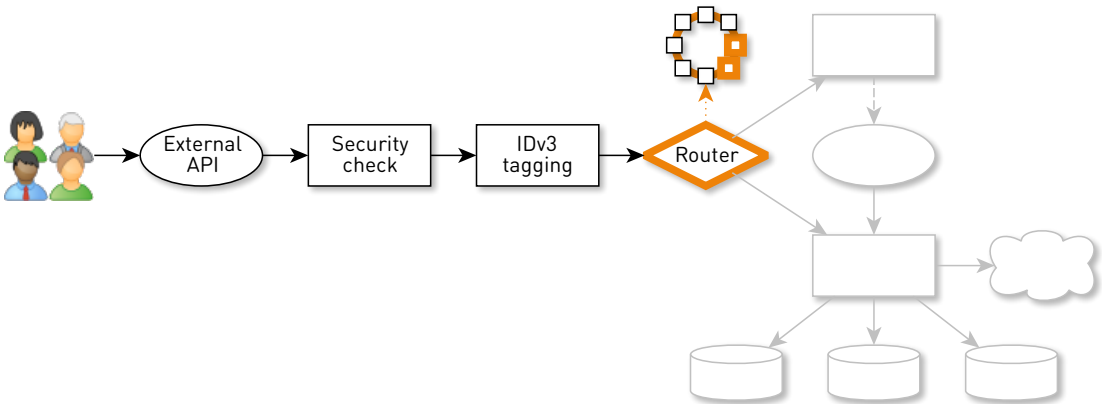
Проверка подписи



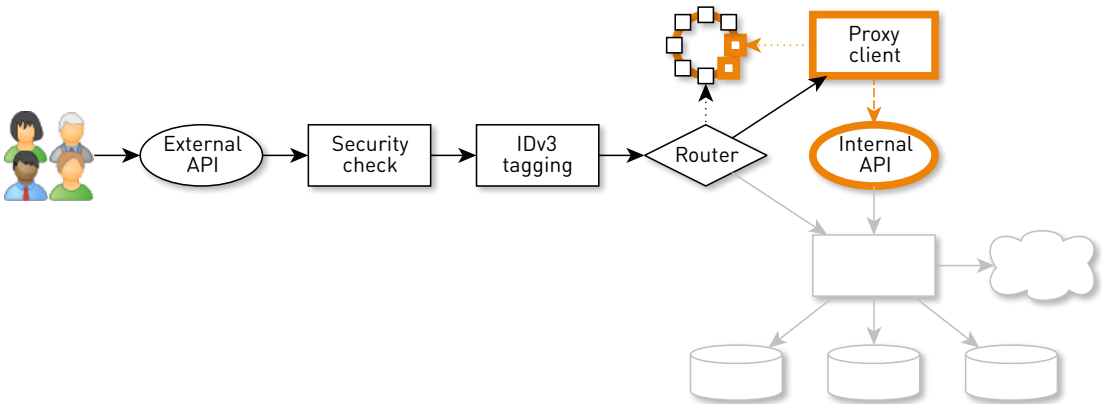
Добавление IDv3



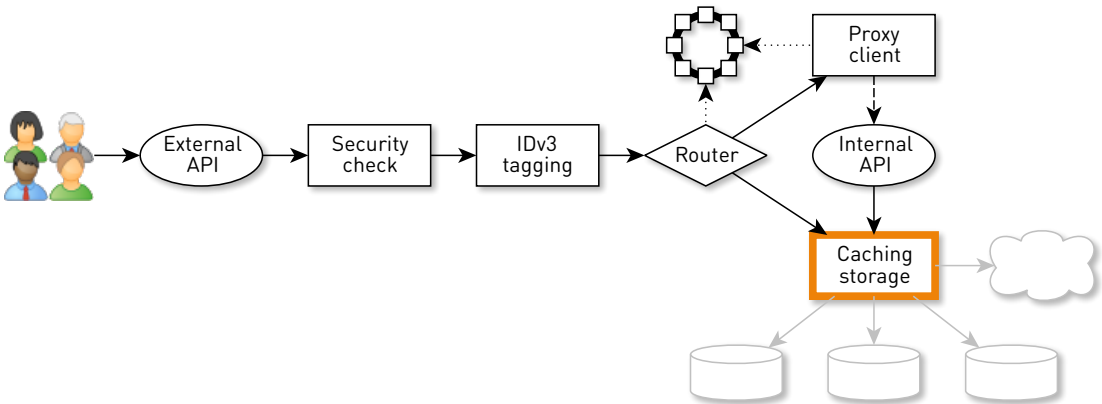
Определение реплик



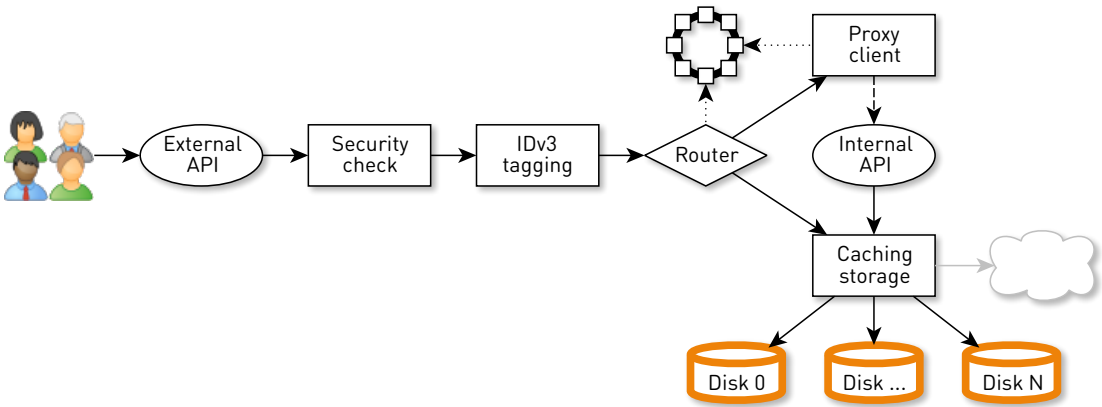
Проксируем с соседа



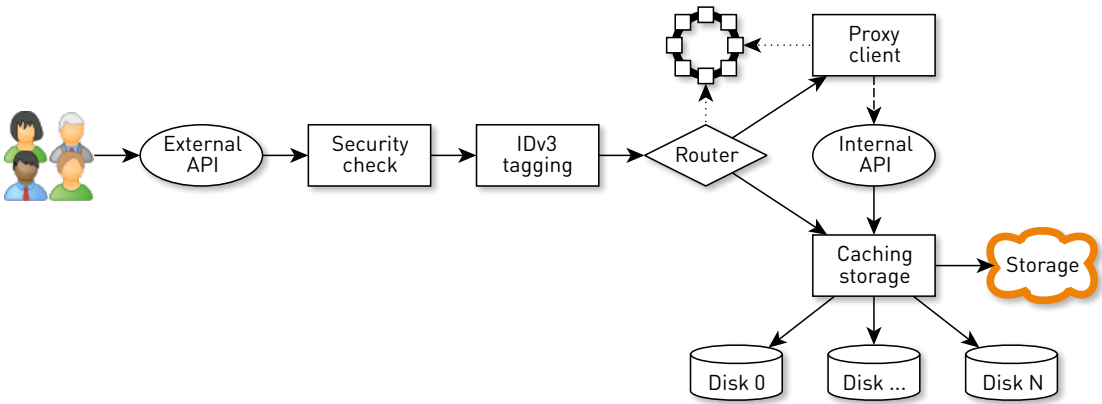
Отдаём локально



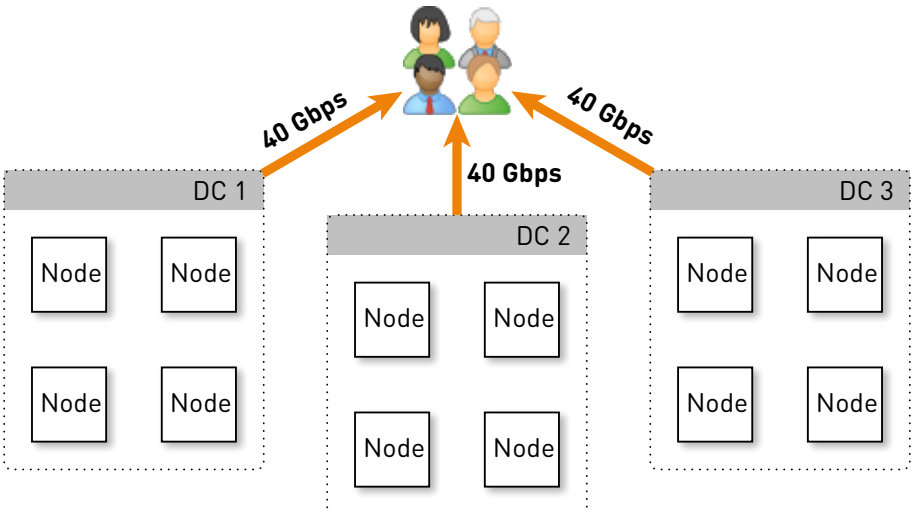
Hit


External API
Security check
IDv3 tagging
Router
Proxy client
Internal API
Caching storage
Disk 0
Disk ...
Disk N

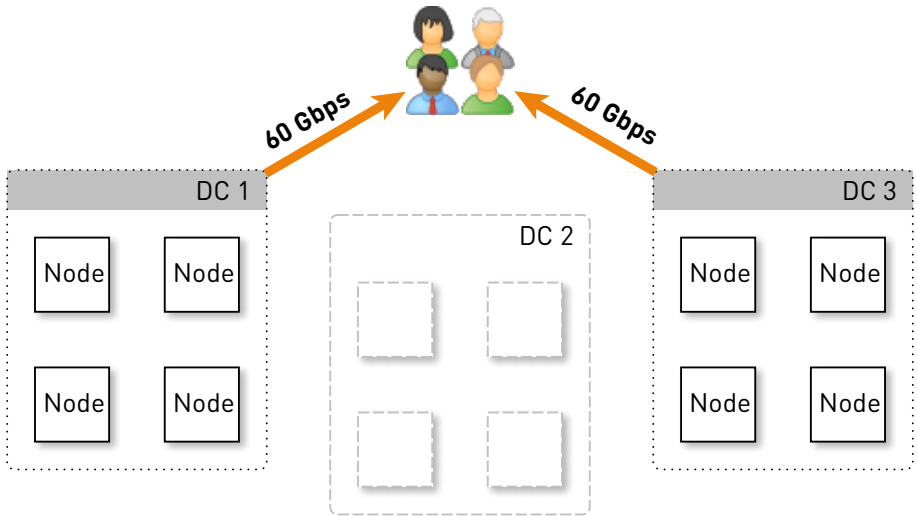
Miss


External API
Security check
IDv3 tagging
Router
Proxy client
Internal API
Caching storage
Storage
Disk 0
Disk ...
Disk N

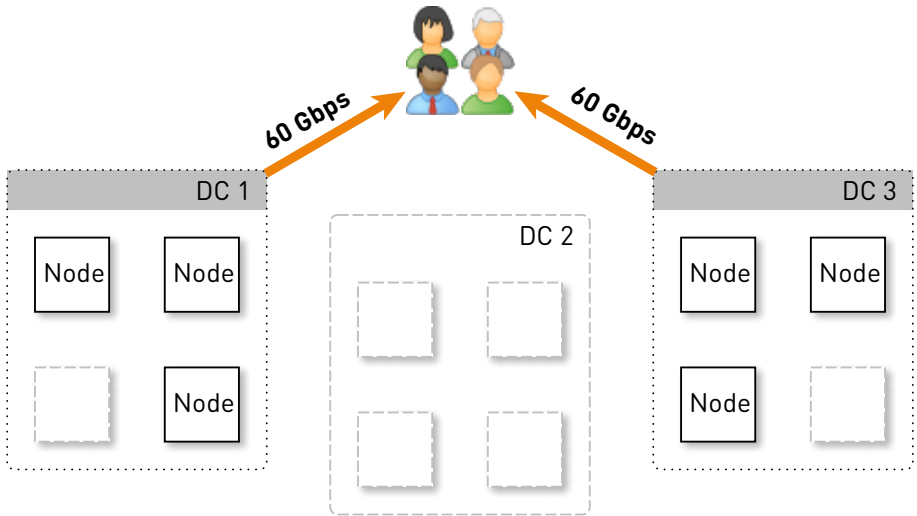
Пусть 120 Гб/с...



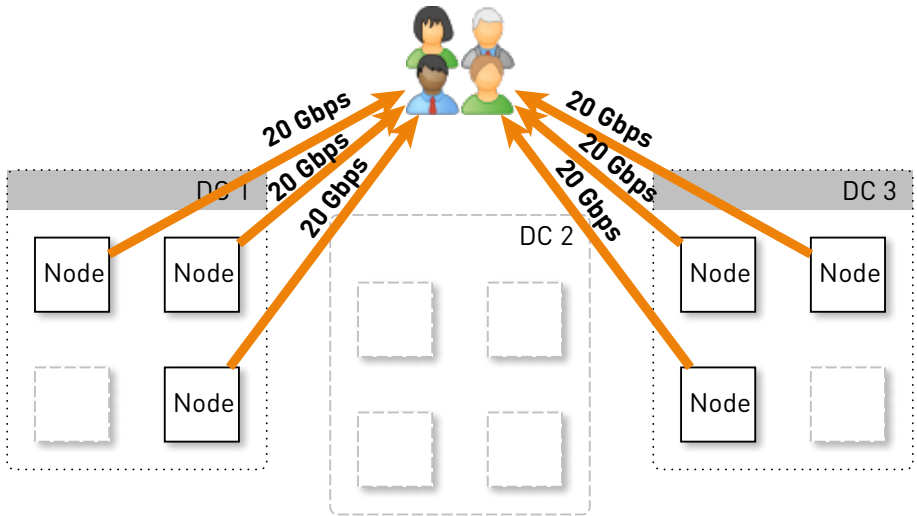
-1 ДЦ


60 Gbps
60 Gbps
DC 1
Node
Node
Node
Node
DC 2
DC 3
Node
Node
Node
Node

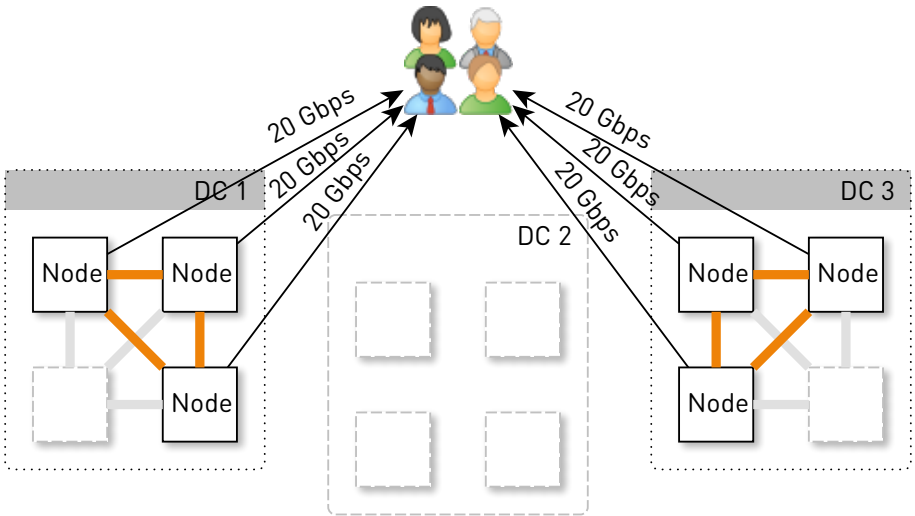
Ещё и релиз



С ноды


20 Gbps
20 Gbps
20 Gbps
20 Gbps
20 Gbps
20 Gbps
DC 1
DC 2
DC 3
Node
Node
Node
Node
Node
Node

$RF = 2 \Rightarrow$ 50% проксируется



Итого одна нода

- Отдаёт пользователям **20 Гб/с**

Итого одна нода

- Отдаёт пользователям **20 Гб/с**
- Из них **10 Гб/с проксирует**

Итого одна нода

- Отдаёт пользователям **20 Гб/с**
- Из них **10 Гб/с проксирует**
- И отдаёт **соседям-прокси 10 Гб/с**

Итого одна нода

- Отдаёт пользователям **20 Гб/с**
- Из них **10 Гб/с проксирует**
- И отдаёт **соседям-прокси 10 Гб/с**
- В итоге **наружу 30 Гб/с**, из них **20 Гб/с с дисков**

Итого одна нода

- Отдаёт пользователям **20 Гб/с**
- Из них **10 Гб/с проксирует**
- И отдаёт **соседям-прокси 10 Гб/с**
- В итоге **наружу 30 Гб/с**, из них **20 Гб/с с дисков**
- **512 ГБ RAM** вместит ≈ **50K** горячих треков

Итого одна нода

- Отдаёт пользователям **20 Гб/с**
- Из них **10 Гб/с проксирует**
- И отдаёт **соседям-прокси 10 Гб/с**
- В итоге **наружу 30 Гб/с**, из них **20 Гб/с с дисков**
- **512 ГБ RAM** вместит ≈ **50K** горячих треков
- ≈ **30-40%** хвоста провалится в диски

Итого одна нода

- Отдаёт пользователям **20 Гб/с**
- Из них **10 Гб/с проксирует**
- И отдаёт **соседям-прокси 10 Гб/с**
- В итоге **наружу 30 Гб/с**, из них **20 Гб/с с дисков**
- **512 ГБ RAM** вместит ≈ **50K** горячих треков
- ≈ **30-40%** хвоста провалится в диски
- ≈ **8 Гб/с** с дисков

Как хранить данные?

Как хранить данные?

- Трек = файл?
  - **Десятки ТБ**
  - **Миллионы файлов**

Как хранить данные?

- Трек = файл?
  - **Десятки ТБ**
  - **Миллионы файлов**
  - Накладные расходы на **метаданные**

Как хранить данные?

- Трек = файл?
  - **Десятки ТБ**
  - **Миллионы файлов**
  - Накладные расходы на **метаданные**
  - Тянуть треки **только сначала**

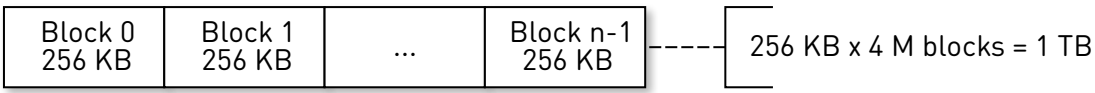
Как хранить данные?

- Трек = файл?
  - **Десятки ТБ**
  - **Миллионы файлов**
  - Накладные расходы на **метаданные**
  - Тянуть треки **только сначала**
  - Хранить треки **только целиком**

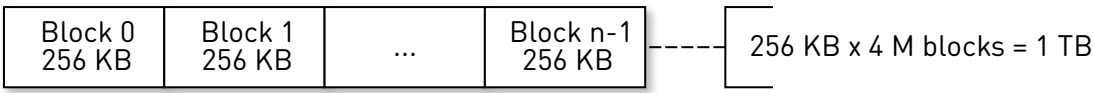
Как хранить данные?

- Трек = файл?
  - **Десятки ТБ**
  - **Миллионы файлов**
  - Накладные расходы на **метаданные**
  - Тянуть треки **только сначала**
  - Хранить треки **только целиком**
- **Так мы не делаем**

Трек = N x 256 КБ блоков (SSD)

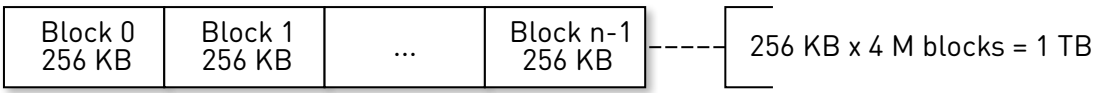


Трек = N x 256 КБ блоков (SSD)



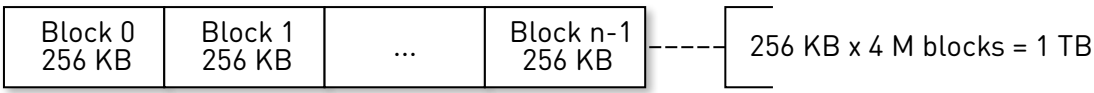
- Каждый диск — независимое хранилище

Трек = N x 256 КБ блоков (SSD)



- Каждый диск — независимое хранилище
- `hash(track, block_num)` чтобы **размазать блоки**

Трек = N x 256 КБ блоков (SSD)



- Каждый диск — независимое хранилище
- `hash(track, block_num)` чтобы **размазать блоки**
- **Page cache** ОС

Как хранить блоки?

- Один файл на XFS размером с диск
  - `xfs_io -c resvsp 0 ...`

Как хранить блоки?

- Один файл на XFS размером с диск
  - `xfs_io -c resvsp 0 ...`
- Сырое блочное устройство
  - `new RandomAccessFile("/dev/sdc", "rw")`

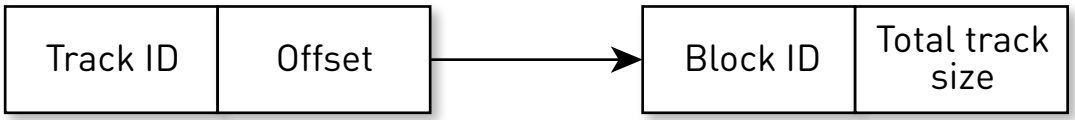
Как хранить блоки?

- Один файл на XFS размером с диск
  - `xfs_io -c resvsp 0 ...`
- Сырое блочное устройство
  - `new RandomAccessFile("/dev/sdc", "rw")`

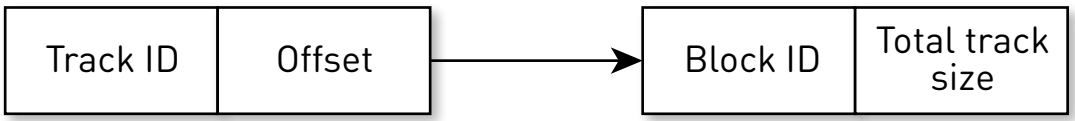
Блочное устройство vs файл на XFS:

- LA **в 2 раза ниже**
- Запись **в 1.5 раза быстрее**
- Время ответа **в 2-3 раза ниже**

Индекс

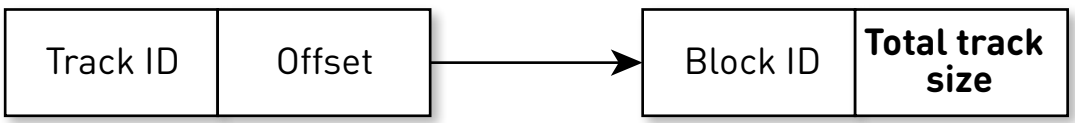


Индекс



- **29 байт** на блок (≈ 1 ГБ на 10 ТБ блоков)

Total track size



- **29 байт** на блок (≈ 1 ГБ на 10 ТБ блоков)

Total track size

▼ **Response Headers** view source

Accept-Ranges: bytes

Cache-Control: max-age=86400

Content-Length: 8570377

Content-Range: bytes 262144-8832520/8832521

Content-Type: audio/mpeg

Как хранить?

- In-memory

[7] https://github.com/ben-manes/caffeine/wiki/Efficiency

Как хранить?

- In-memory
- Компактно

[7] https://github.com/ben-manes/caffeine/wiki/Efficiency

Как хранить?

- In-memory
- Компактно
- Персистентно

[7] https://github.com/ben-manes/caffeine/wiki/Efficiency

Как хранить?

- In-memory
- Компактно
- Персистентно
- Вытеснение LRU[7]

[7] https://github.com/ben-manes/caffeine/wiki/Efficiency

Как хранить?

- In-memory
- Компактно
- Персистентно
- Вытеснение LRU[7]
- Thread-safe

[7] https://github.com/ben-manes/caffeine/wiki/Efficiency

Как хранить?

- In-memory
- Компактно
- Персистентно
- Вытеснение LRU[7]
- Thread-safe

- `SharedMemoryFixedMap`[8] на `tmpfs`

[7] https://github.com/ben-manes/caffeine/wiki/Efficiency
[8] https://github.com/odnoklassniki/one-nio

Durability

- **Рестарт** машины — теряем `tmpfs`

Durability

- **Рестарт** машины — теряем `tmpfs`
- **Падение** процесса — неконсистентность

Durability

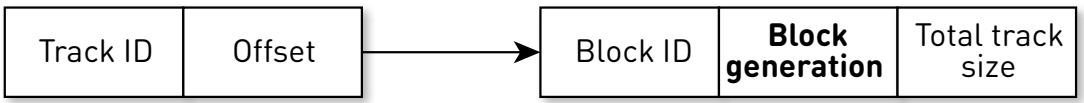
- **Рестарт** машины — теряем `tmpfs`
- **Падение** процесса — неконсистентность
- **Snapshot**

Durability

- **Рестарт** машины — теряем `tmpfs`
- **Падение** процесса — неконсистентность

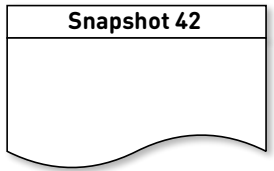
- **Snapshot**
- **WAL**

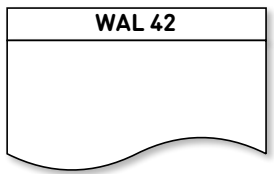
Порядок операций



Snapshot + WAL

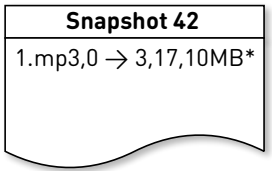
Track ID	Offset	Block ID	Generation	Track size
1.mp3	0	3	17	10 MB
1.mp3	1	7	18	10 MB
2.mp3	0	9	21	15 MB

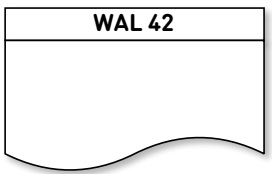




Пишем в snapshot

Track ID	Offset	→	Block ID	Generation	Track size
1.mp3	**0**	→	**3**	**17**	**10 MB**
1.mp3	1	→	7	18	10 MB
2.mp3	0	→	9	21	15 MB





Пишем в snapshot

Track ID	Offset	Block ID	Generation	Track size
1.mp3	0	3	17	10 MB
1.mp3	**1**	**7**	**18**	**10 MB**
2.mp3	0	9	21	15 MB

Snapshot 42

1.mp3,0 → 3,17,10MB
1.mp3,1 → 7,18,10MB*

WAL 42

Вытеснение в WAL

Track ID	Offset
1.mp3	0
2.mp3	0

Block ID	Generation	Track size
3	17	10 MB
7	**18**	
9	21	15 MB

Snapshot 42

1.mp3,0 → 3,17,10MB
1.mp3,1 → 7,18,10MB

WAL 42

7,18 dirty*

Вытеснение в WAL

Track ID	Offset
1.mp3	0

Block ID	Generation	Track size
3	17	10 MB
7	18	
9	**21**	

Snapshot 42

1.mp3,0 → 3,17,10MB
1.mp3,1 → 7,18,10MB

WAL 42

7,18 dirty
9,21 dirty*

Переиспользование блока

Track ID	Offset
1.mp3	0
3.mp3	**0**

Block ID	Generation	Track size
3	17	10 MB
7	18	
9	**22**	**8 MB**

Snapshot 42

1.mp3,0 → 3,17,10MB
1.mp3,1 → 7,18,10MB

WAL 42

7,18 dirty
9,21 dirty

Заканчиваем snapshot

Track ID	Offset	Block ID	Generation	Track size
1.mp3	0	3	17	10 MB
		7	18	
3.mp3	**0**	**9**	**22**	**8 MB**

Snapshot 42

1.mp3,0 → 3,17,10MB
1.mp3,1 → 7,18,10MB
3.mp3,0 → 9,22,8MB*

WAL 42

7,18 dirty
9,21 dirty

В итоге

Track ID	Offset
1.mp3	0
3.mp3	0

Block ID	Generation	Track size
3	17	10 MB
9	22	8 MB

Snapshot 42

1.mp3,0 → 3,17,10MB
1.mp3,1 → 7,18,10MB
3.mp3,0 → 9,22,8MB

WAL 42

7,18 dirty
9,21 dirty

Восстанавливаем

Track ID	Offset	Block ID	Generation	Track size
1.mp3	0	3	17	10 MB
3.mp3	0	9	22	8 MB

Snapshot 42

1.mp3,0 → 3,17,10MB
1.mp3,1 → 7,18,10MB
3.mp3,0 → 9,22,8MB

WAL 42

7,18 dirty
9,21 dirty

Сканируем WAL

Track ID	Offset	→	Block ID	Generation	Track size
1.mp3	0	→	3	17	10 MB
3.mp3	0	→	9	22	8 MB

Snapshot 42

1.mp3,0 → 3,17,10MB
1.mp3,1 → 7,18,10MB
3.mp3,0 → 9,22,8MB

Track ID	Offset	→	Block ID	Generation	Track size

Dirty	
Block ID	**Generation**
7	18
9	21

Обычная запись

Track ID	Offset	Block ID	Generation	Track size
1.mp3	0	3	17	10 MB
3.mp3	0	9	22	8 MB

Snapshot 42

1.mp3,0 → 3,17,10MB*
1.mp3,1 → 7,18,10MB
3.mp3,0 → 9,22,8MB

Track ID	**Offset**	**Block ID**	**Generation**	**Track size**
1.mp3	**0**	**3**	**17**	**10 MB**

Dirty	
Block ID	**Generation**
7	18
9	21

Вытесненная запись

Track ID	Offset	→	Block ID	Generation	Track size
1.mp3	0	→	3	17	10 MB
3.mp3	0	→	9	22	8 MB

Snapshot 42

1.mp3,0 → 3,17,10MB
1.mp3,1 → 7,18,10MB*
3.mp3,0 → 9,22,8MB

Track ID	**Offset**	→	**Block ID**	**Generation**	**Track size**
1.mp3	0	→	3	17	10 MB

Dirty	
Block ID	**Generation**
7	**18**
9	21

Переиспользованный блок

Track ID	Offset
1.mp3	0
3.mp3	0

Block ID	Generation	Track size
3	17	10 MB
9	22	8 MB

Snapshot 42

1.mp3,0 → 3,17,10MB
1.mp3,1 → 7,18,10MB
3.mp3,0 → 9,22,8MB*

Track ID	**Offset**
1.mp3	0
3.mp3	**0**

Block ID	**Generation**	**Track size**
3	17	10 MB
9	**22**	**8 MB**

Dirty	
Block ID	**Generation**
7	18
9	**21**

Успех

Track ID	Offset	Block ID	Generation	Track size
1.mp3	0	3	17	10 MB
3.mp3	0	9	22	8 MB

Snapshot 42

1.mp3,0 → 3,17,10MB
1.mp3,1 → 7,18,10MB
3.mp3,0 → 9,22,8MB

Track ID	Offset	Block ID	Generation	Track size
1.mp3	0	3	17	10 MB
3.mp3	0	9	22	8 MB

Dirty	
Block ID	**Generation**
7	18
9	21

Уберите детей от экранов

Disclaimer

Your mileage **will** vary.

Page cache

- $<$ **20%** попаданий[9]

[9] `fs/cachestat` from `https://github.com/brendangregg/perf-tools`

[10] `https://github.com/odnoklassniki/one-nio`

Page cache

- $<$ **20%** попаданий[9]
- Read ahead?

[9] `fs/cachestat` from `https://github.com/brendangregg/perf-tools`

[10] `https://github.com/odnoklassniki/one-nio`

Page cache

- $<$ **20%** попаданий[9]
- Read ahead?

`Mem.posix_fadvise(..., POSIX_FADV_RANDOM)`[10]:

[9] `fs/cachestat` from `https://github.com/brendangregg/perf-tools`

[10] `https://github.com/odnoklassniki/one-nio`

Page cache

- $<$ **20%** попаданий[9]
- Read ahead?

`Mem.posix_fadvise(..., POSIX_FADV_RANDOM)`[10]:

- $>$ **70%** попаданий
- **В 2 раза** меньше чтений с диска
- HTTP latency **на 20% ниже**

[9] `fs/cachestat` from `https://github.com/brendangregg/perf-tools`

[10] `https://github.com/odnoklassniki/one-nio`

-XX:+UseHugeTLBFS

- -Xmx24g

-XX:+UseHugeTLBFS

- -Xmx24g
- GC Time/Safepoint Total Time **на 20-30% ниже**
- Более **равномерная** загрузка ядер

-XX:+UseHugeTLBFS

- -Xmx24g
- GC Time/Safepoint Total Time **на 20-30% ниже**
- Более **равномерная** загрузка ядер
- Но **нет эффекта** на HTTP latency

Инцидент

1. В саппорте жалобы

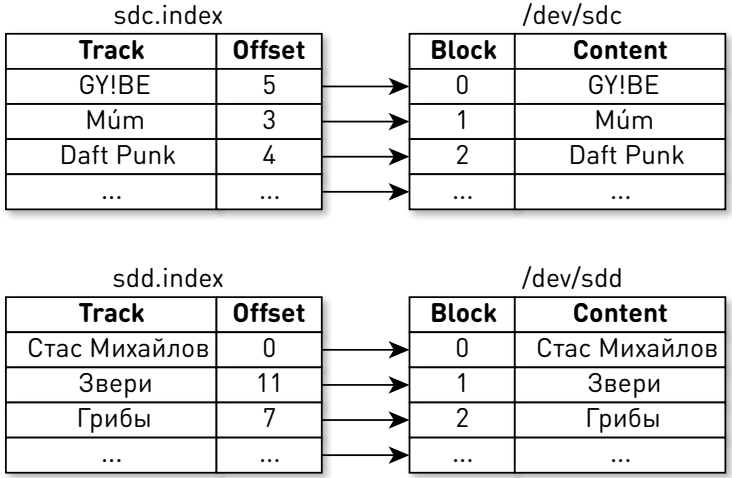
Инцидент

1. В саппорте жалобы
2. Сузили круг поиска до одной машины

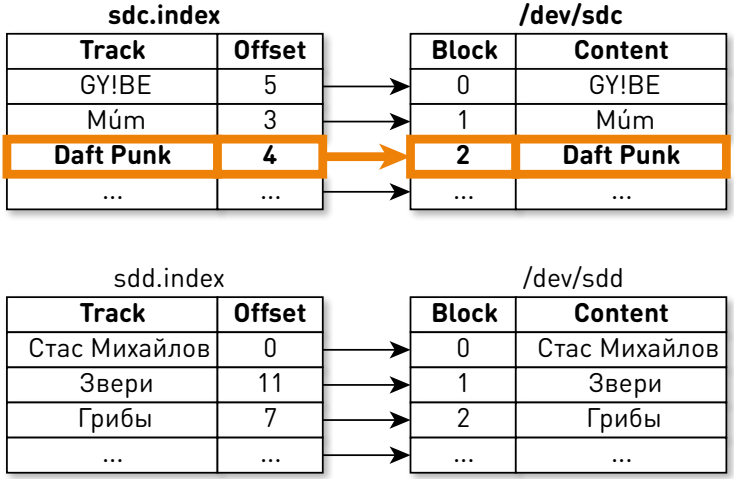
Инцидент

1. В саппорте жалобы
2. Сузили круг поиска до одной машины
3. Машину недавно перезапускали...

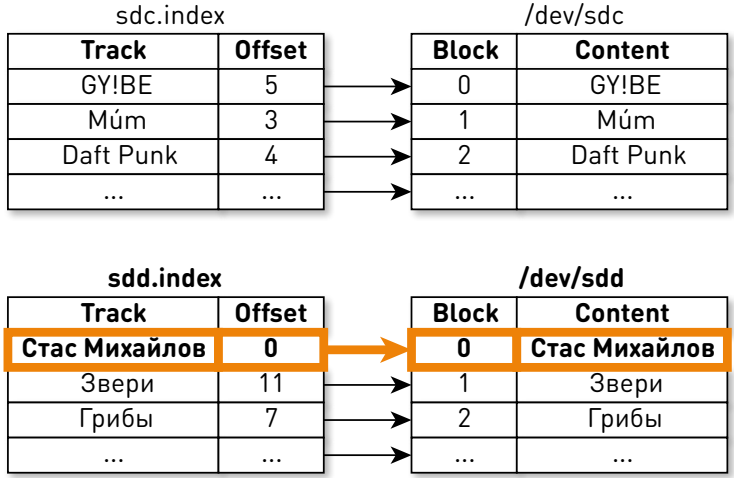
До перезагрузки



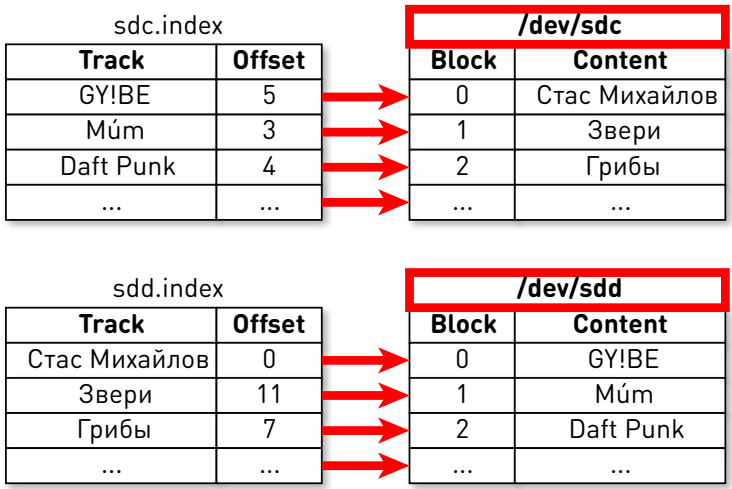
До перезагрузки



До перезагрузки

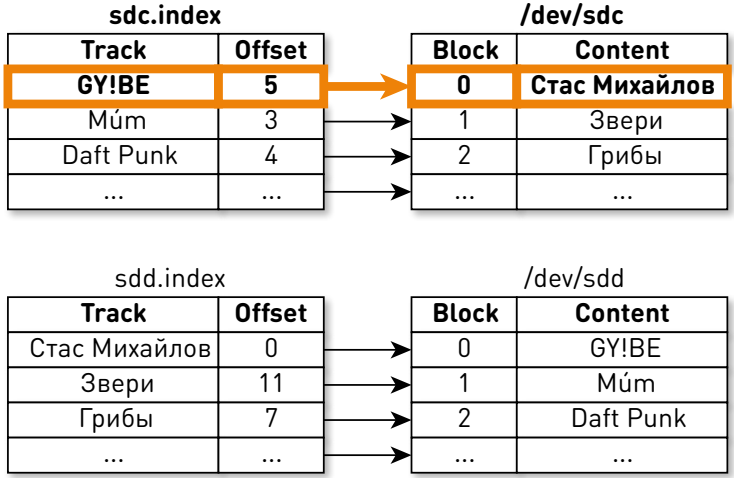


После перезагрузки[11]

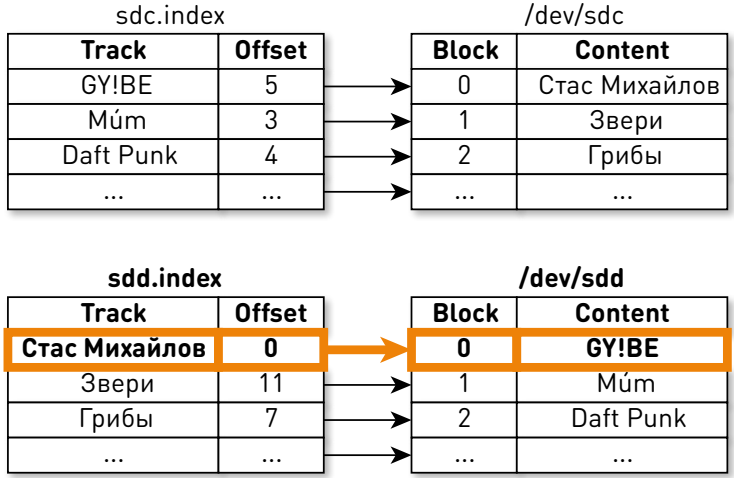


[11] https://wiki.archlinux.org/index.php/Persistent_block_device_naming

Party mode


sdc.index
Track
Offset
GY!BE
5
Ми́м
3
Daft Punk
4
...
...
/dev/sdc
Block
Content
0
Стас Михайлов
1
Звери
2
Грибы
...
...
sdd.index
Track
Offset
Стас Михайлов
0
Звери
11
Грибы
7
...
...
/dev/sdd
Block
Content
0
GY!BE
1
Ми́м
2
Daft Punk
...
...

Party mode


sdc.index
Track
Offset
GY!BE
5
Мúm
3
Daft Punk
4
...
...
/dev/sdc
Block
Content
0
Стас Михайлов
1
Звери
2
Грибы
...
...
sdd.index
Track
Offset
Стас Михайлов
0
Звери
11
Грибы
7
...
...
/dev/sdd
Block
Content
0
GY!BE
1
Мúm
2
Daft Punk
...
...

Fix

- **Персистентный** World Wide Name[12]

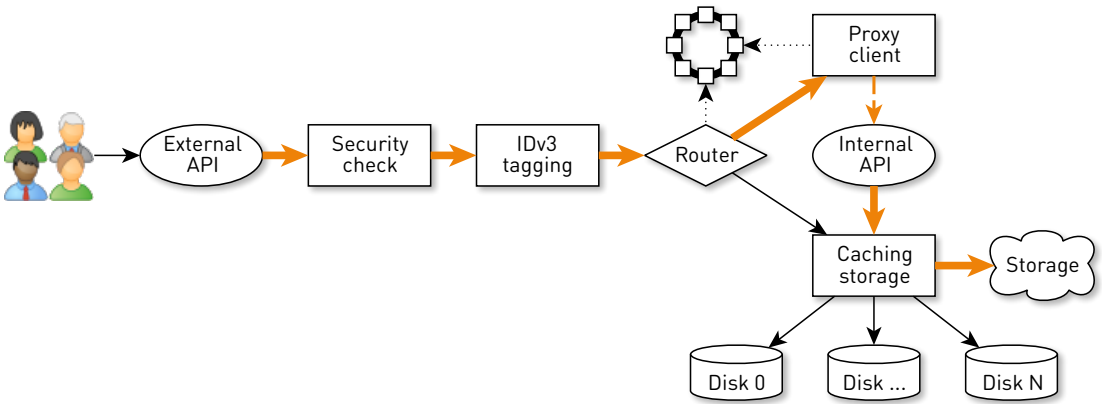
[12] https://en.wikipedia.org/wiki/World_Wide_Name

Fix

- **Персистентный** World Wide Name[12]
- `/dev/disk/by-id/wwn-*` в качестве **ID**
  - Индексов
  - Snapshot
  - WAL

[12] `https://en.wikipedia.org/wiki/World_Wide_Name`

Распределённая отладка



TraceID[13]-метка в запросах

```
1 GET /v0/stream?id=v0_10000051227 HTTP/1.1
2 Range: bytes=10878976-11141119
3 X-OK-Trace-ID: 570752372832:bec:v0_10000051227
```

[13] See OpenTracing/Zipkin

При отладке TraceID в логах

```
1    2018-09-13 17:40:40,147 DEBUG
2      [cache-provider-4] CachingDataProvider:
3 ->   [570752372832:bec:v0_10000051227]
4      Serving chunk from cache:
5      Chunk[10878976-11010047/16346382]
```

Как отправлять данные?

```
1 ByteBuffer buffer = ByteBuffer.allocate(size);
2 int count = fileChannel.read(buffer, position);
3 if (count <= 0) {
4     // ...
5 }
6 buffer.flip();
7 socketChannel.write(buffer);
```

SYNOPSIS

```
#include <sys/sendfile.h>

ssize_t sendfile(int out_fd, int in_fd, off_t *offset, size_t count);
```

DESCRIPTION

sendfile() copies data between one file descriptor and another. Because this copying is done within the kernel, **sendfile**() is more efficient than the combination of read(2) and write(2), which would require transferring data to and from user space.

sendfile()

```
1 final RandomAccessFile blocks;
2
3 int write(Socket socket) {
4     if (socket.getSslContext() == null) {
5         socket.sendFile(blocks, offset, size);
6     }
```

- **Zero-copy**, но только **не SSL**

В случае SSL

```
1 ByteBuffer buffer = ByteBuffer.allocate(size);
2 int count = fileChannel.read(buffer, position);
3 if (count <= 0) {
4     // ...
5 }
6 buffer.flip();
7 socketChannel.write(buffer);
```

IOUtil

```
1  ByteBuffer bb =
       Util.getTemporaryDirectBuffer(dst.remaining());
2  try {
3      int n = readIntoNativeBuffer(fd, bb, position, nd);
4      bb.flip();
5      if (n > 0)
6          dst.put(bb);
7      return n;
8  } finally {
9      Util.offerFirstTemporaryDirectBuffer(bb);
10 }
```

one-nio: DirectMemory и друзья

```
1  final Allocator allocator =
2          new MallocMT(size, concurrency);
3
4
5
6
7
8
9
10 ...
```

Если SSL...

```
1  final Allocator allocator =
2          new MallocMT(size, concurrency);
3
4  int write(Socket socket) {
5      if (socket.getSslContext() != null) {
6
7
8
9
10 ...
```

Выделяем нативный буфер

```
1  final Allocator allocator =
2          new MallocMT(size, concurrency);
3
4  int write(Socket socket) {
5      if (socket.getSslContext() != null) {
6          long address = allocator.malloc(size);
7
8
9
10 ...
```

Оборачиваем в ByteBuffer

```
1  final Allocator allocator =
2          new MallocMT(size, concurrency);
3
4  int write(Socket socket) {
5      if (socket.getSslContext() != null) {
6          long address = allocator.malloc(size);
7          ByteBuffer buf =
8                  DirectMemory.wrap(address, size);
9
10 ...
```

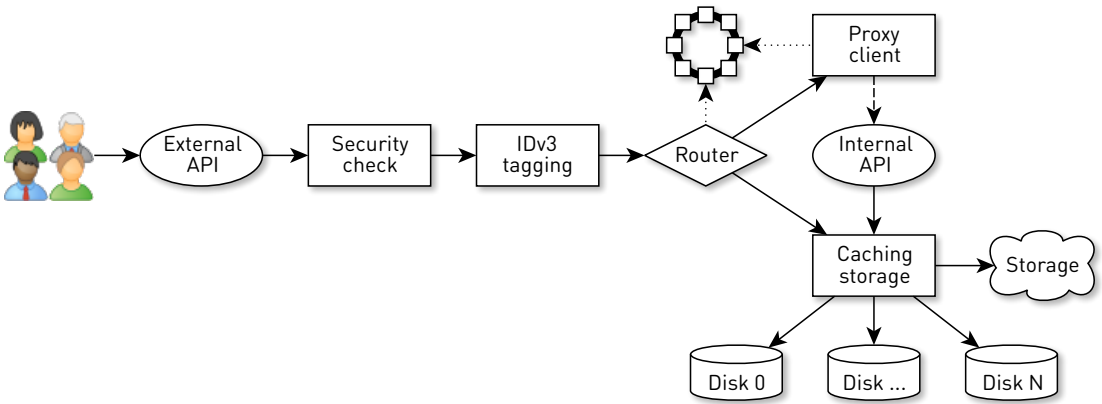
Читаем

```
1  final Allocator allocator =
2          new MallocMT(size, concurrency);
3
4  int write(Socket socket) {
5      if (socket.getSslContext() != null) {
6          long address = allocator.malloc(size);
7          ByteBuffer buf =
8                  DirectMemory.wrap(address, size);
9          int available = channel.read(buf, offset);
10 ...
```

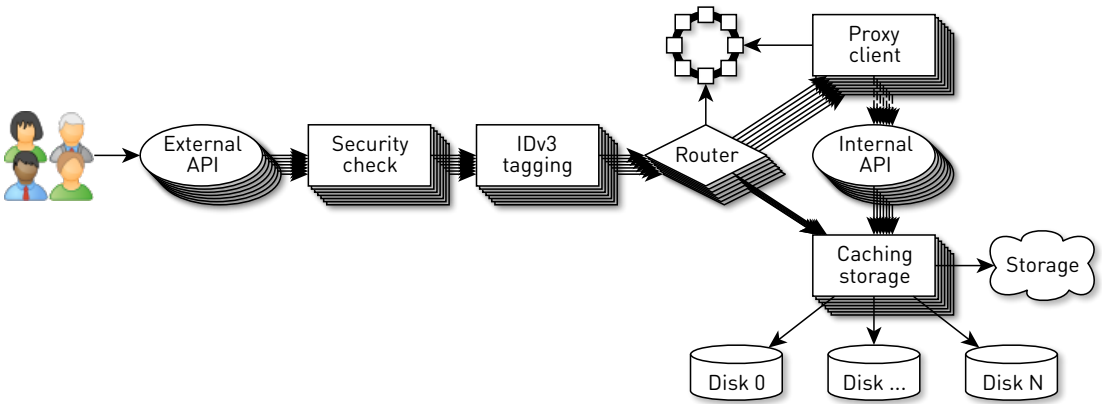
Пишем

```
1  final Allocator allocator =
2          new MallocMT(size, concurrency);
3
4  int write(Socket socket) {
5      if (socket.getSslContext() != null) {
6          long address = allocator.malloc(size);
7          ByteBuffer buf =
8                  DirectMemory.wrap(address, size);
9          int available = channel.read(buf, offset);
10         socket.writeRaw(address, available, flags);
```

One more thing...



Очень много клиентов



Варианты[14]

Клиент = `Thread`:

- **Синхронно**

[14] https://thestrangeloop.com/2017/zuuls-journey-to-non-blocking.html

Варианты[14]

Клиент = `Thread`:

- **Синхронно**
- **100K** потоков?!

[14] `https://thestrangeloop.com/2017/zuuls-journey-to-non-blocking.html`

Варианты[14]

Клиент = `Thread`:

- **Синхронно**
- **100K** потоков?!
- **Нежизнеспособно**

[14] `https://thestrangeloop.com/2017/zuuls-journey-to-non-blocking.html`

Варианты[14]

Клиент = `Thread`:

- **Синхронно**
- **100K** потоков?!
- **Нежизнеспособно**

Клиент = конвейер:

- **Асинхронно**

[14] `https://thestrangeloop.com/2017/zuuls-journey-to-non-blocking.html`

Варианты[14]

Клиент = `Thread`:

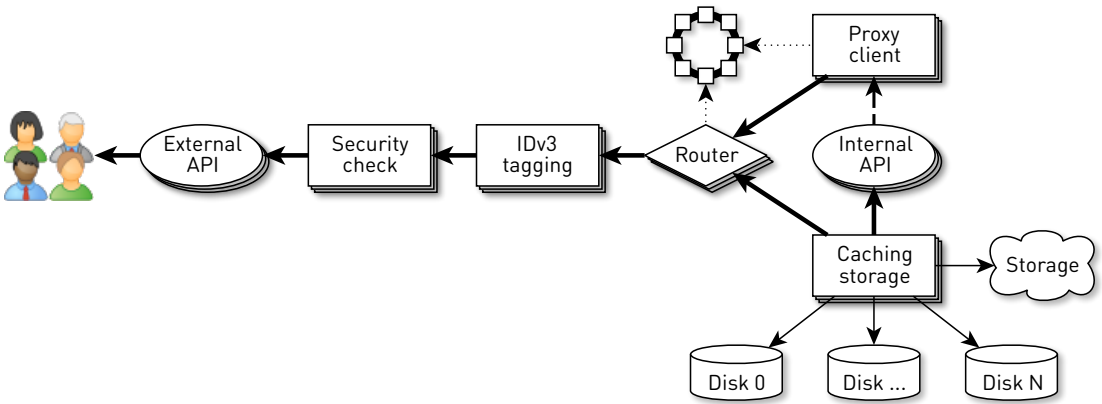
- **Синхронно**
- **100K** потоков?!
- **Нежизнеспособно**

Клиент = конвейер:

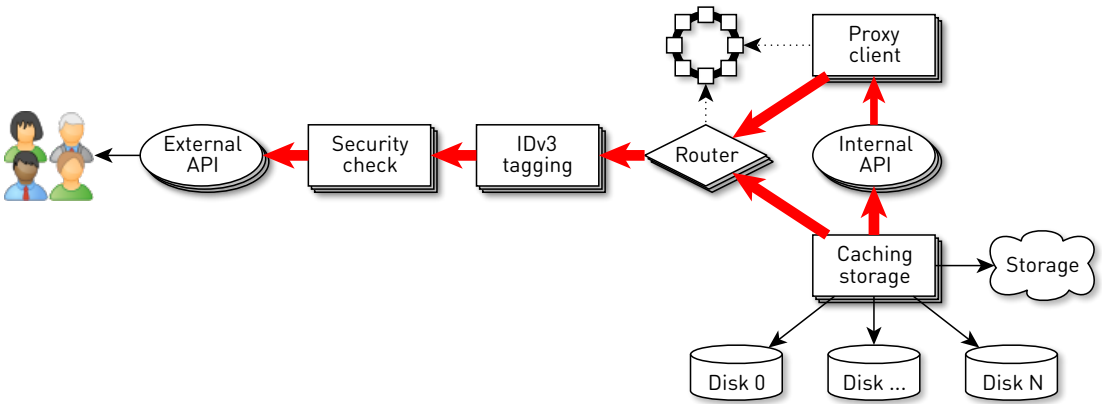
- **Асинхронно**
- Стадия — пул потоков
- **Push** данных клиенту

[14] https://thestrangeloop.com/2017/zuuls-journey-to-non-blocking.html

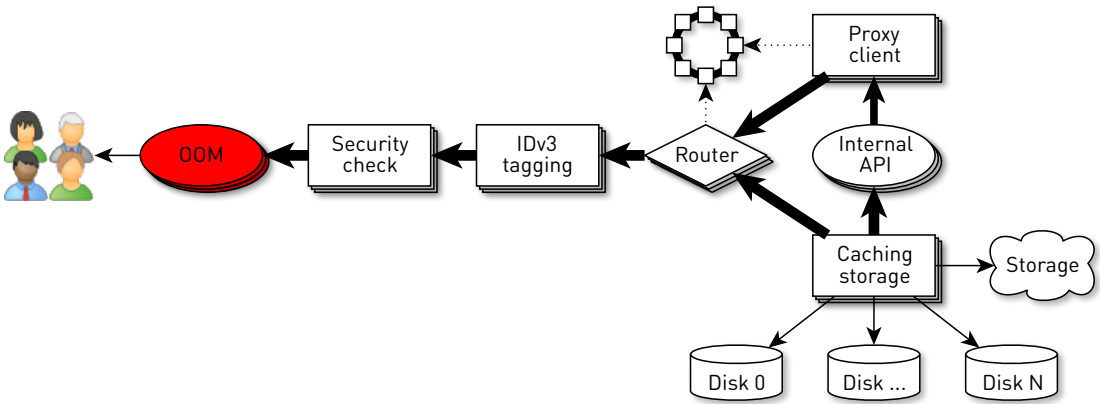
Асинхронный push



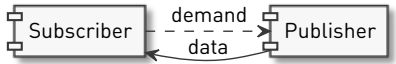
Бекенды гораздо быстрее



Если неограниченные очереди



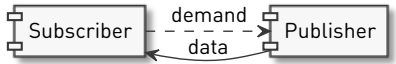
Reactive streams[15] to the rescue



- **Subscriber** управляет скоростью
- Память ограничена **demand**

[15] http://www.reactive-streams.org

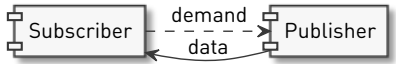
Reactive streams[15] to the rescue



- **Subscriber** управляет скоростью
- Память ограничена **demand**
- Subscriber быстрее — **push**

[15] http://www.reactive-streams.org

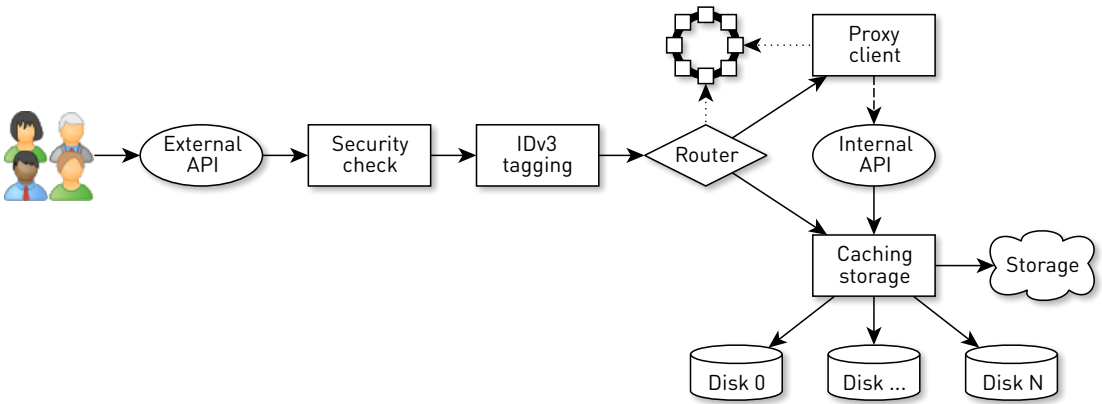
Reactive streams[15] to the rescue



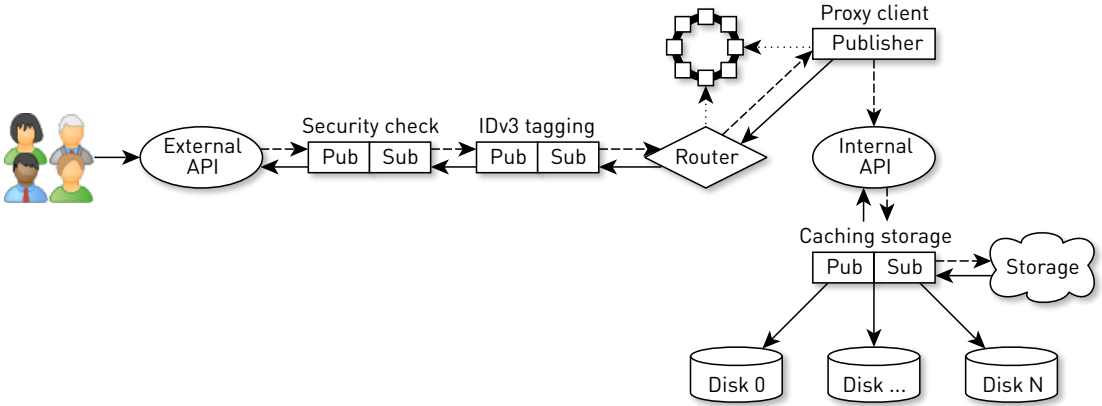
- **Subscriber** управляет скоростью
- Память ограничена **demand**
- Subscriber быстрее — **push**
- Publisher быстрее — **pull**

[15] http://www.reactive-streams.org

Конвейер


External API
Security check
IDv3 tagging
Router
Proxy client
Internal API
Caching storage
Storage
Disk 0
Disk ...
Disk N

Reactive Stream



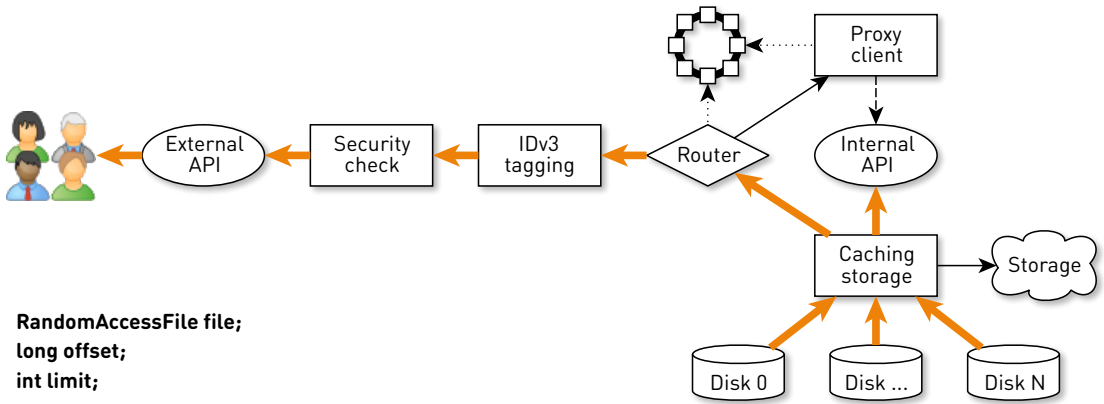
```
1   interface Publisher<T> {
2       void subscribe(Subscriber<? super T> s);
3   }
4   interface Subscriber<T> {
5       void onSubscribe(Subscription s);
6       void onNext(T t);
7       void onError(Throwable t);
8       void onComplete();
9   }
10  interface Subscription {
11      void request(long n);
12      void cancel();
```

Поток Chunkов

```
1 interface Chunk {
2     int read(ByteBuffer dst);
3     int write(Socket socket);
4     void write(FileChannel channel, long offset);
5 }
```

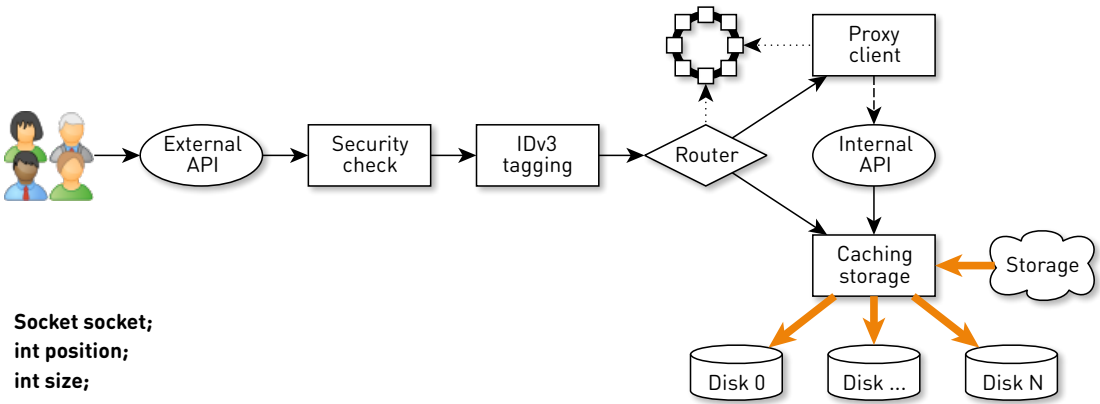
- **Ссылка** на данные
- Ограниченный интерфейс
- Множество реализаций

Chunk over RandomAccessFile



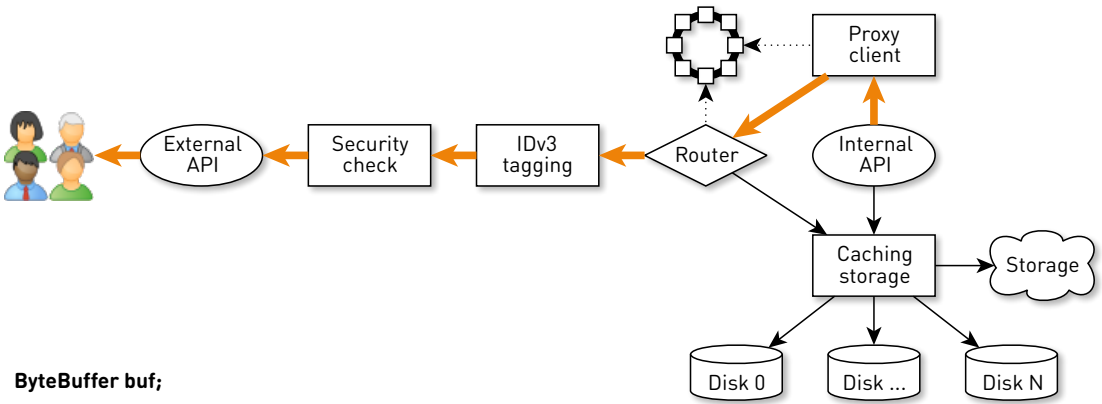
RandomAccessFile file;
long offset;
int limit;

Chunk over Socket



Socket socket;
int position;
int size;

Chunk over ByteBuffer



ByteBuffer buf;

Неблокирующий API[16]

```
1 interface Subscriber<T> {
2     void onSubscribe(Subscription s);
3     void onNext(T t);
4     void onError(Throwable t);
5     void onComplete();
6 }
7 interface Subscription {
8     void request(long n);
9     void cancel();
```

[16] https://jokerconf.com/2018/talks/6wmp33pmjgism04u4wckgy/

Чтение на ночь

[concurrency-interest] Synchronization primitives in Reactive Streams implementations

Pavel Rappo pavel.rappo at gmail.com
Fri Sep 28 08:51:58 EDT 2018

- Previous message (by thread): [concurrency-interest] JLS 17.5 Examples
- Next message (by thread): [concurrency-interest] Synchronization primitives in Reactive Streams implementations
- **Messages sorted by:** [date] [thread] [subject] [author]

```
Hello,

Let me start this discussion as a branch of the "Reactive Streams utility API"
thread here:

  http://mail.openjdk.java.net/pipermail/core-libs-dev/2018-September/055671.html

I would like to talk about a particular synchronization primitive I find to be
repeatedly reinvented in Reactive Streams implementations. I hope that we could
discuss different incarnations of this primitive and maybe extract it into a
reusable component.
```

В духе Typed Actor Model[17]

[17] https://github.com/reactive-streams/reactive-streams-jvm/tree/master/examples

В духе Typed Actor Model[17]

- Вызов метода $\rightarrow$ сообщение
- Сообщение — в `Queue`
- Шедулимся на `Executor`

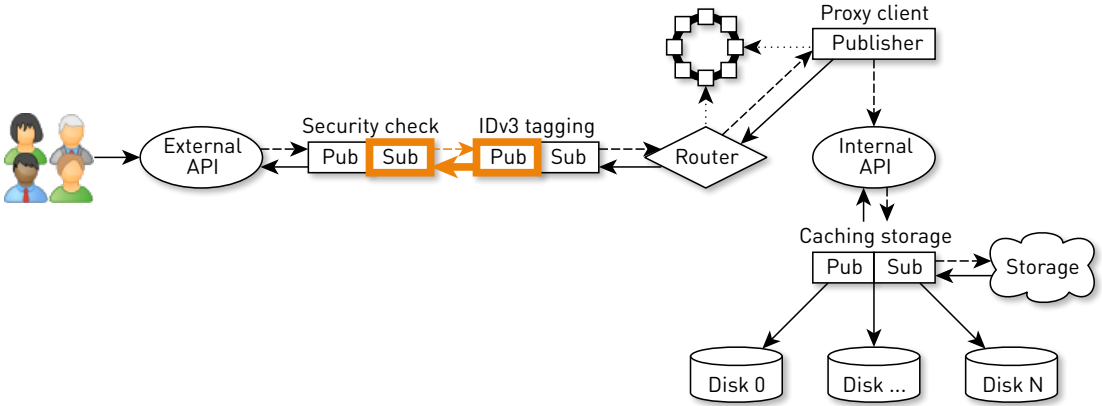
[17] `https://github.com/reactive-streams/reactive-streams-jvm/tree/master/examples`

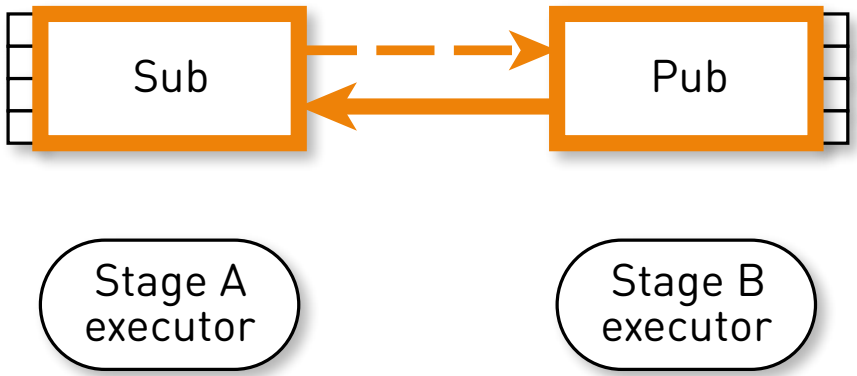
В духе Typed Actor Model[17]

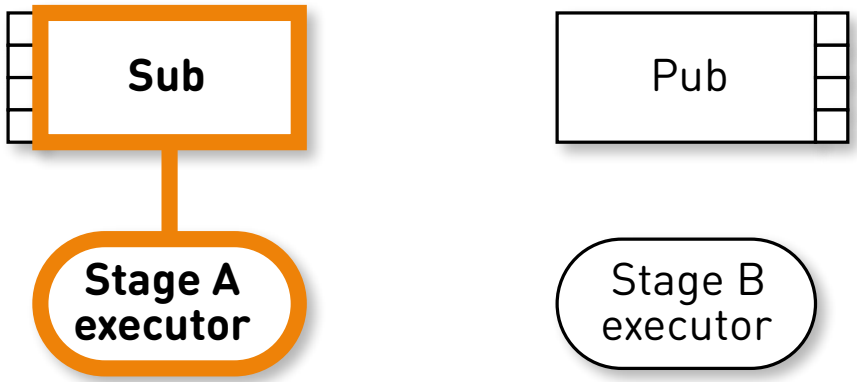
- Вызов метода $\rightarrow$ сообщение
- Сообщение — в `Queue`
- Шедулимся на `Executor`
- Обрабатываем **последовательно**

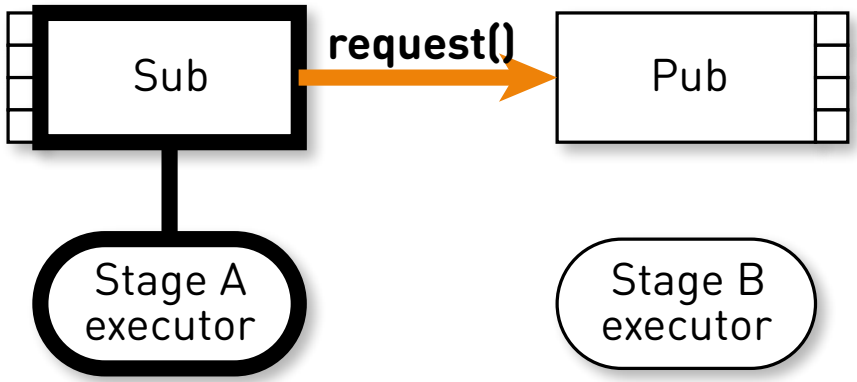
[17] `https://github.com/reactive-streams/reactive-streams-jvm/tree/master/examples`

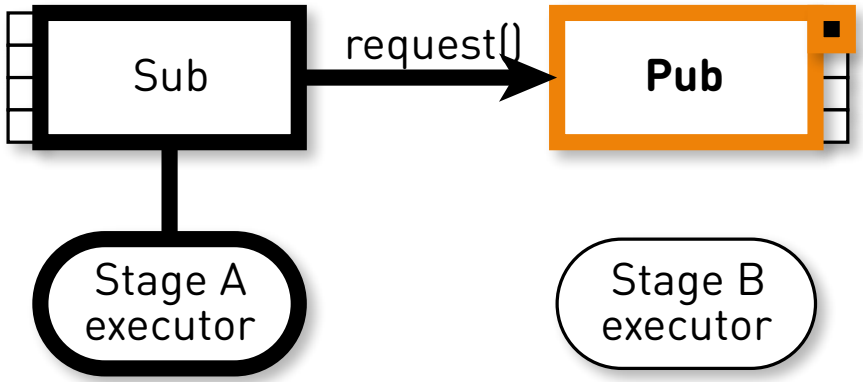
Reactive Stream

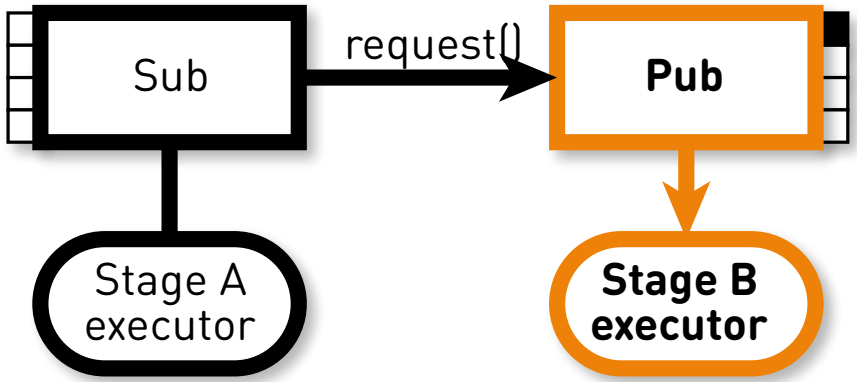


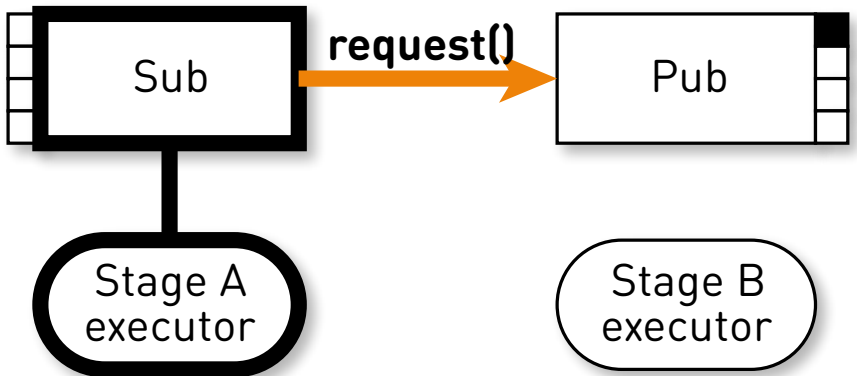

Sub
Pub
Stage A executor
Stage B executor

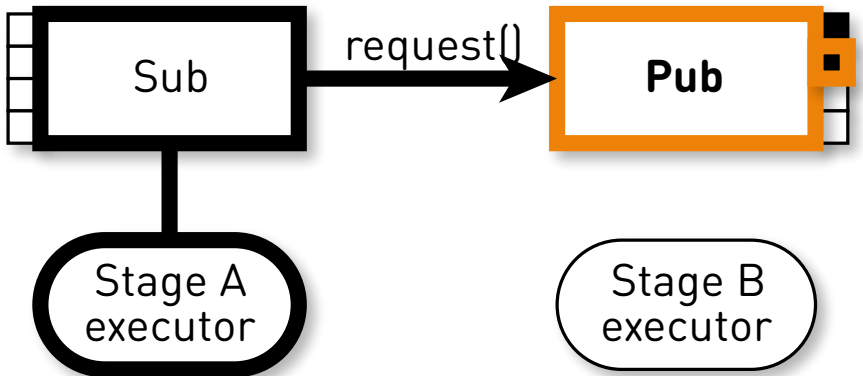

Sub
Pub
Stage A executor
Stage B executor

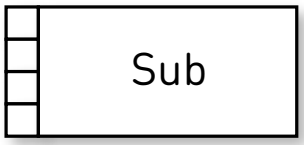

Sub
request()
Pub
Stage A
executor
Stage B
executor

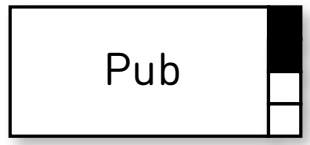

Sub
request()
Pub
Stage A
executor
Stage B
executor


Sub
request()
Pub
Stage A
executor
Stage B
executor

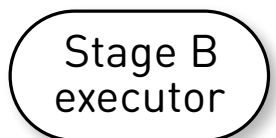

Sub
request()
Pub
Stage A
executor
Stage B
executor

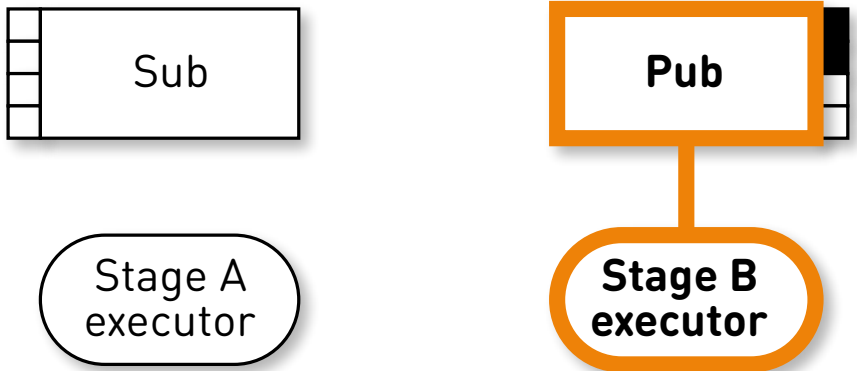

Sub
request()
Pub
Stage A
executor
Stage B
executor

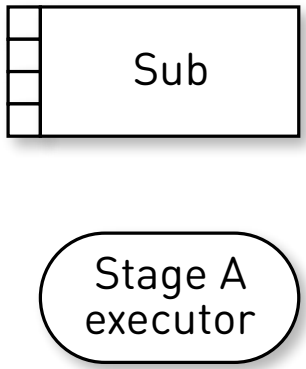

Sub

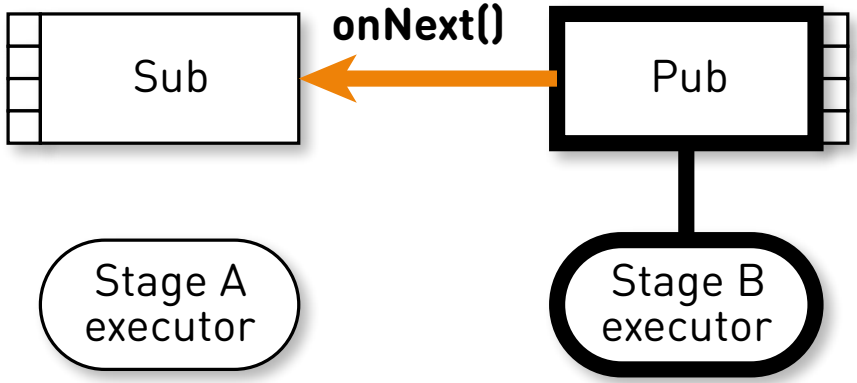

Pub

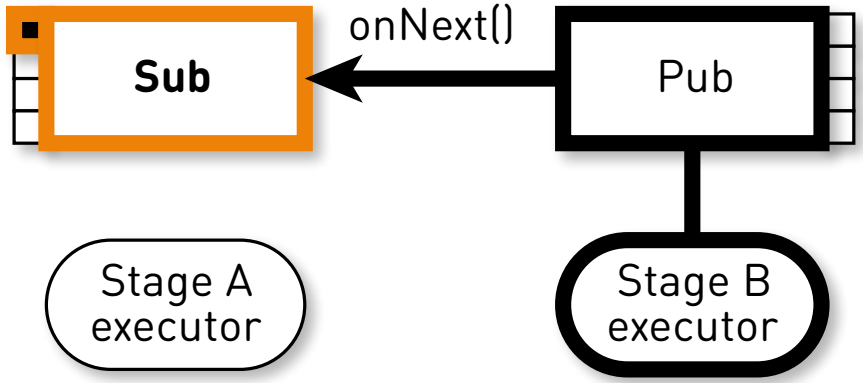

Stage A
executor

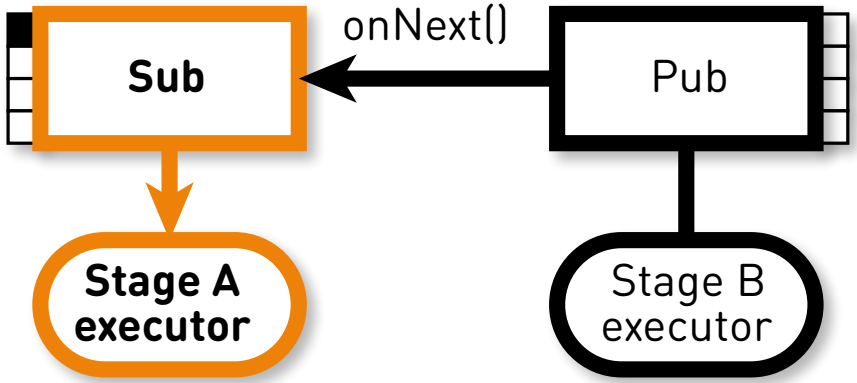

Stage B
executor

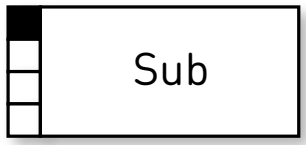

Sub
Pub
Stage A executor
Stage B executor

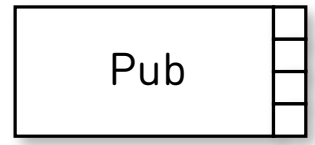

Sub
Stage A executor
Pub
Stage B executor


Sub
onNext()
Pub
Stage A
executor
Stage B
executor


Sub
onNext()
Pub
Stage A
executor
Stage B
executor


Sub
onNext()
Pub
Stage A
executor
Stage B
executor


Sub


Pub


Stage A
executor


Stage B
executor

Состояние

```
1 // Incoming messages
2 final Queue<M> mailbox;
3
4 // Message processing works here
5 final Executor executor;
6
7 // To ensure HB relationship between runs
8 final AtomicBoolean on = new AtomicBoolean();
```

Асинхронизация

```
1 @Override
2 void request(final long n) {
3     enqueue(new Request(n));
4 }
5
6 void enqueue(M message) {
7     mailbox.offer(message);
8     trySchedule ToExecute();
9 }
```

trySheduleToExecute()

```
1 if (on.compareAndSet(false, true)) {
2     try {
3         executor.execute(this);
4     } catch (Exception e) {
5         ...
6     }
7 }
```

run()

```
1  if (on.get())
2      try {
3          dequeueAndProcess();
4      } finally {
5          on.set(false);
6          if (!messages.isEmpty()) {
7              trыScheduleToExecute();
8          }
9      }
10 }
```

dequeueAndProcess()

```
1 M message;
2 while ((message = mailbox.poll()) != null) {
3     // Pattern match
4     if (message instanceof Request) {
5         doRequest(((Request) message).n);
6     } else {
7         ...
8     }
9 }
```

В итоге

- **Неблокирующая** реализация

В итоге

- **Неблокирующая** реализация
- Простой **последовательный** код

В итоге

- **Неблокирующая** реализация
- Простой **последовательный** код
- Никакого contention

В итоге

- **Неблокирующая** реализация
- Простой **последовательный** код
- Никакого contention
- **Ограниченное** количество потоков

Production

- **12 машин** (**2x** запас)

Production

- **12 машин** (**2x** запас)
- До **20 Гб/с** с машины через 100K соединений

Production

- **12 машин** (**2x** запас)
- До **20 Гб/с** с машины через 100K соединений
- Масштабируемость
  - Пропускная способность
  - Ёмкость

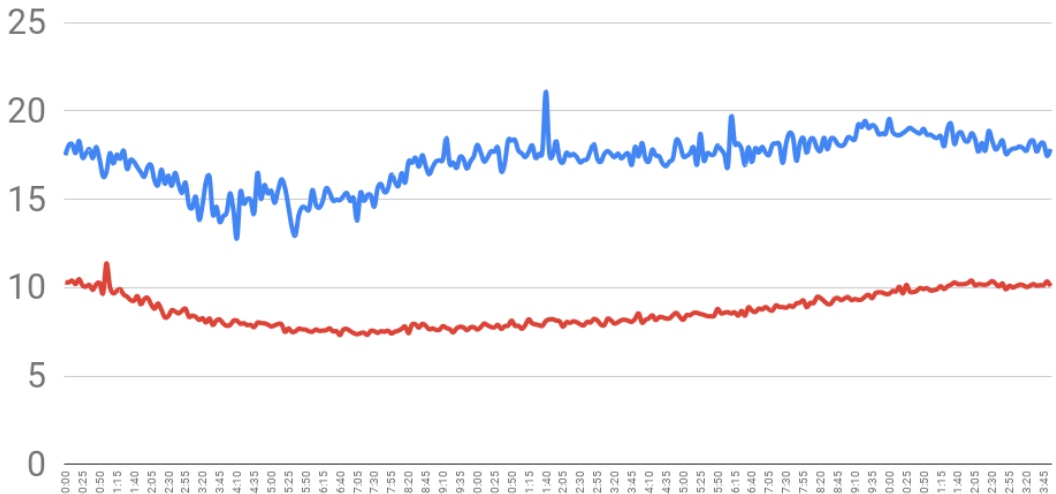
Production

- **12 машин** (**2x** запас)
- До **20 Гб/с** с машины через 100K соединений
- Масштабируемость
  - Пропускная способность
  - Ёмкость
- Отказоустойчивость
  - ДЦ
  - Машины
  - Диски

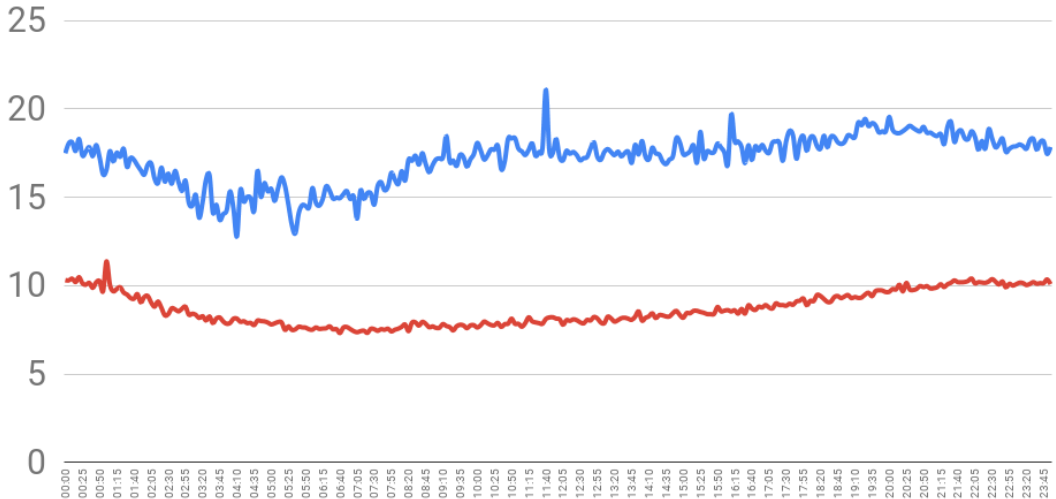
Production

- **12 машин** (**2x** запас)
- До **20 Гб/с** с машины через 100K соединений
- Масштабируемость
  - Пропускная способность
  - Ёмкость
- Отказоустойчивость
  - ДЦ
  - Машины
  - Диски
- Java + `one-nio`

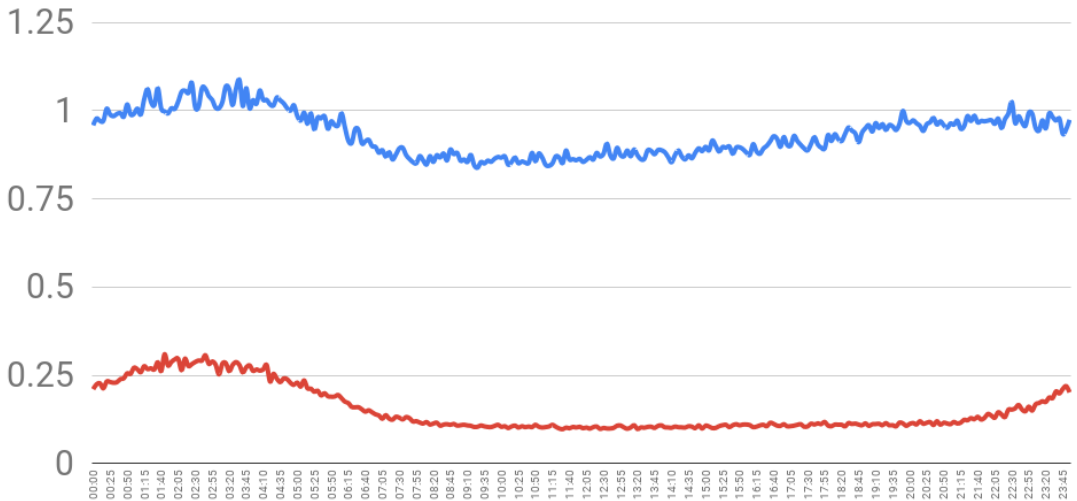
99%, мс


25
20
15
10
5
0
00:00
00:25
00:50
01:15
01:40
02:05
02:30
02:55
03:20
03:45
04:10
04:35
05:00
05:25
05:50
06:15
06:40
07:05
07:30
07:55
08:20
08:45
09:10
09:35
10:00
10:25
10:50
11:15
11:40
12:05
12:30
12:55
13:20
13:45
14:10
14:35
15:00
15:25
15:50
16:15
16:40
17:05
17:30
17:55
18:20
18:45
19:10
19:35
20:00
20:25
20:50
21:15
21:40
22:05
22:30
22:55
23:20
23:45

99%, мс (upstream)



75%, мс (с дисков)



High Load

+

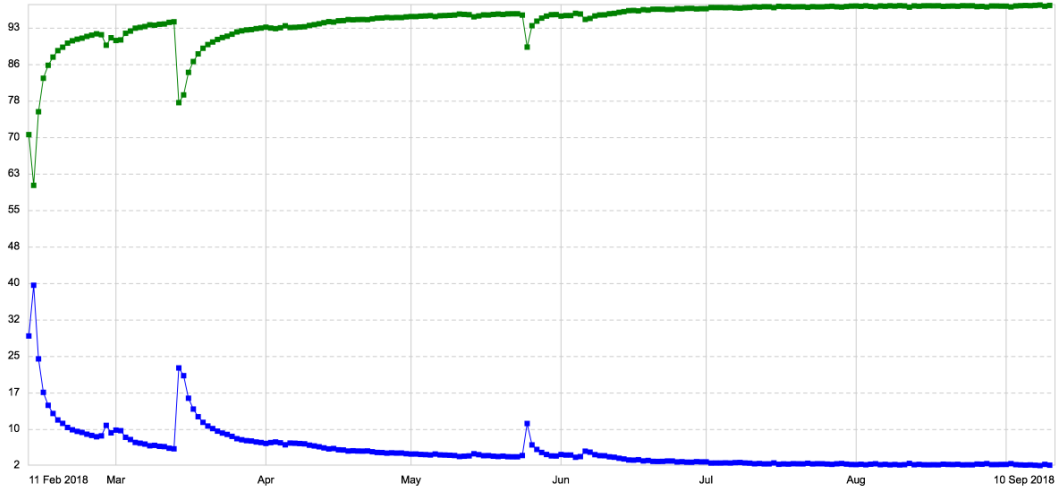
Reactive Streams

=

Вопросы?

Вадим Цесько
Одноклассники
https://v.ok.ru
https://incubos.org

CDN cache hit



Backlog

- Адаптивный фактор репликации

[18] https://en.wikipedia.org/wiki/Rendezvous_hashing

Backlog

- Адаптивный фактор репликации
- Read repair

[18] https://en.wikipedia.org/wiki/Rendezvous_hashing

Backlog

- Адаптивный фактор репликации
- Read repair
- Touch/prefetch

[18] https://en.wikipedia.org/wiki/Rendezvous_hashing

Backlog

- Адаптивный фактор репликации
- Read repair
- Touch/prefetch
- Hash ring поверх дисков

[18] https://en.wikipedia.org/wiki/Rendezvous_hashing

Backlog

- Адаптивный фактор репликации
- Read repair
- Touch/prefetch
- Hash ring поверх дисков
- Rendezvous hashing[18]

[18] https://en.wikipedia.org/wiki/Rendezvous_hashing

Backlog

- Адаптивный фактор репликации
- Read repair
- Touch/prefetch
- Hash ring поверх дисков
- Rendezvous hashing[18]
- Асинхронный HTTP клиент

[18] https://en.wikipedia.org/wiki/Rendezvous_hashing

Backlog

- Адаптивный фактор репликации
- Read repair
- Touch/prefetch
- Hash ring поверх дисков
- Rendezvous hashing[18]
- Асинхронный HTTP клиент
- Open-source `one-download`

[18] `https://en.wikipedia.org/wiki/Rendezvous_hashing`