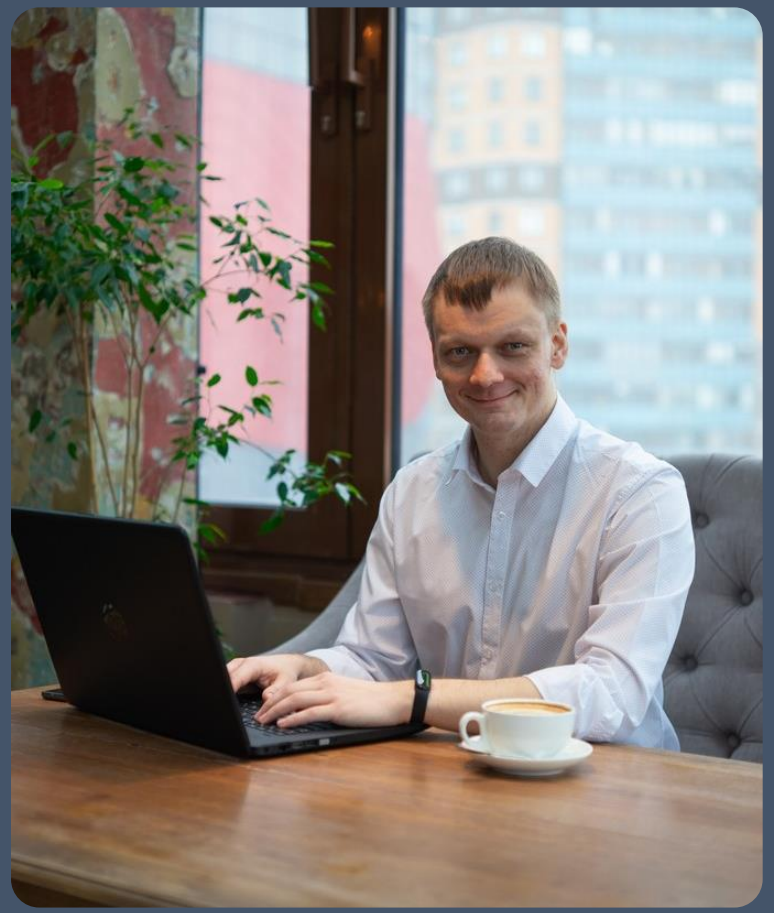
PLATFORM V

# Поиск проблем Java-приложения с 31G heap и 500G off-heap

# Дмитрий Павлов



SBER TECH

Главный эксперт
Лидер команды
внедрения и поддержки
PlatformV DataGrid

apache Ignite™

VP продукта
Apache Ignite

- 20 лет в коммерческой разработке
- 7 лет работаю с in-memory БД и DataGrids

# Agenda

## Что будет в докладе

- Примеры ошибок
- Примеры падений и как устроена отладка в этом случае
- Проблемы с диском при высокой нагрузке
- Производительность: Как действуем, если система не укладывается в SLA

## Чего не будет

- Настроек zabbix и grafana
- Систем анализа логов, вроде ELK
- Best practices для grep

# Platform V DataGrid

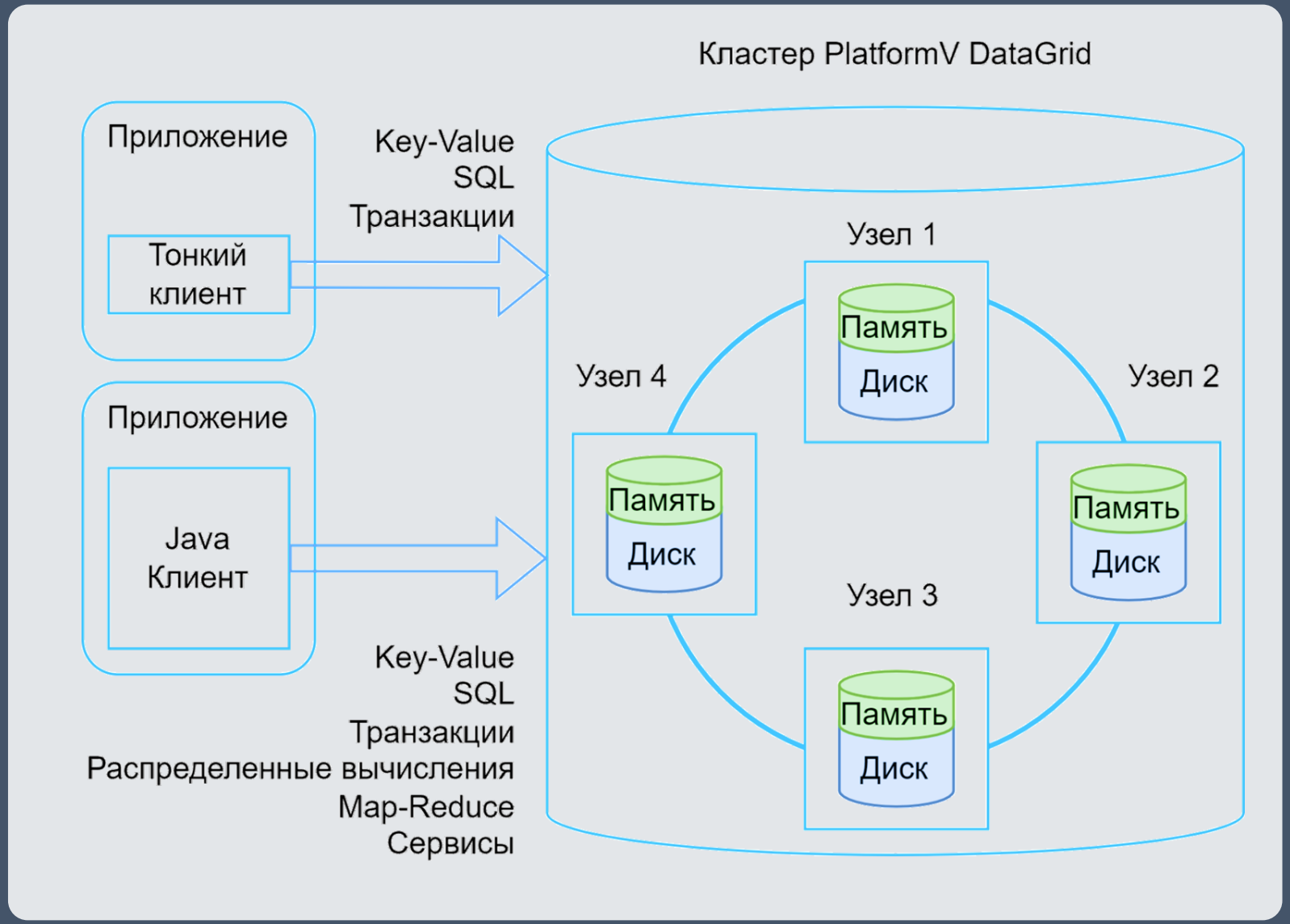
Распределенная БД и IMDG

На Java

На основе Apache Ignite

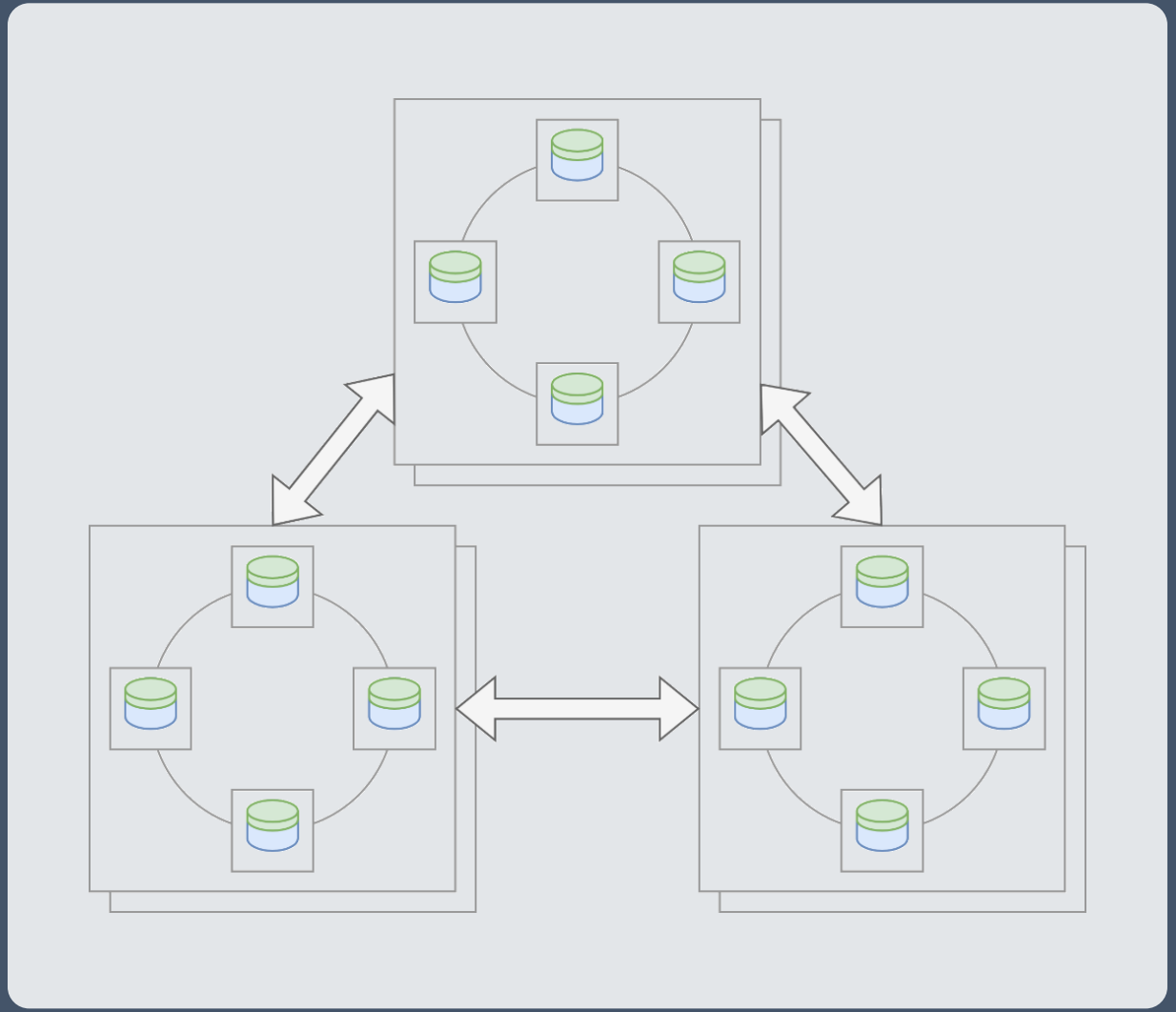
Узел – отдельный сервер

Данные на диске - поверх FS



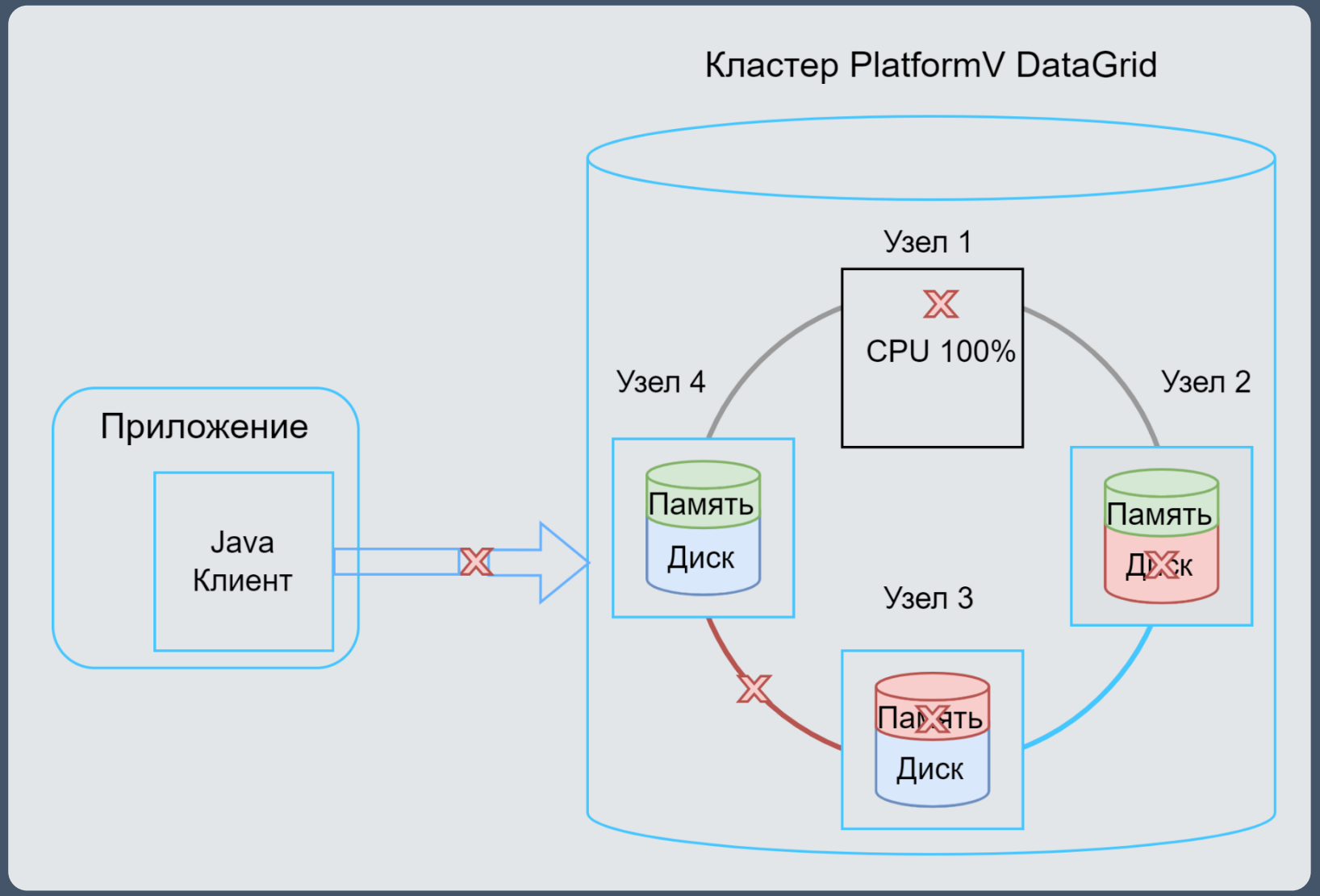
# Процессинг банковских карт

- **Объем:** 3 Tb на узел, 96 Tb на кластер
- **Rps:** 150 000
- **Tps:** 5000 (в тестах выше)
- **Серверов в кластере:** 32
- **Кластеров Active-Active:** 12 в двух ЦОДах
- 1000 групп клиентов, копии в 3х кластерах
- **SLA:** сотни миллисекунд на транзакцию



# Проблемы с железом?

- Переполнение ресурса
- Некорректная работа ресурса



# Disclaimer

Инцидент - не приводит к остановке системы

- Резервирование данных (реплики) ✓
- Резервные кластеры и репликация между ними (CDC) ✓
- Хранение на диске (persistence) ✓
- Создание регулярных снимков данных (бекапы) ✓

Надежность описывается в приложениях A&B (после финального слайда)

# Примеры ошибок

- **Вылет узла с исключением или error**

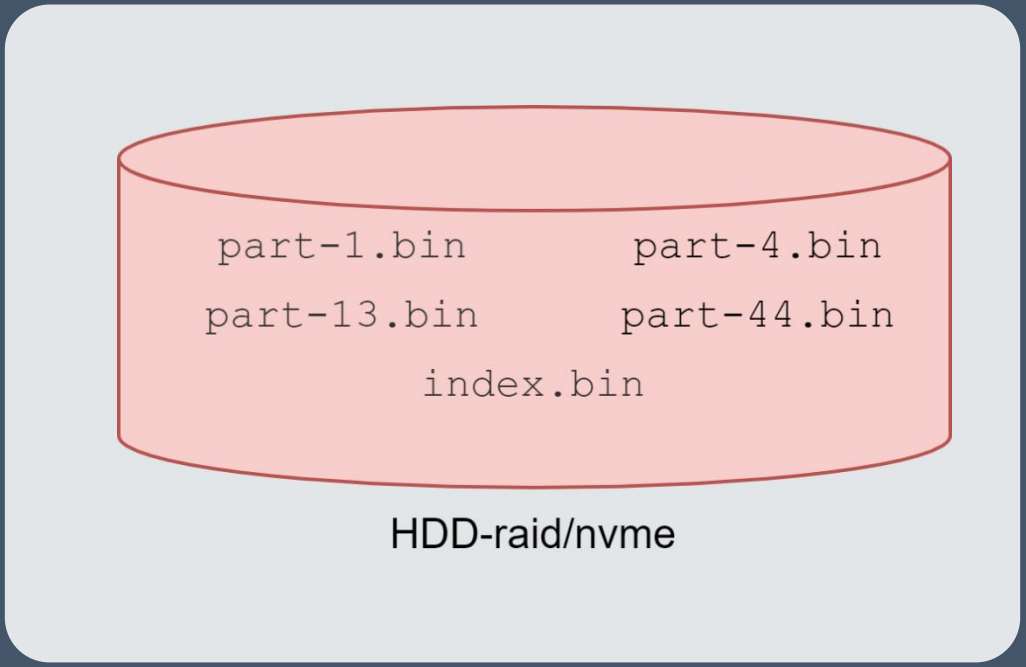
# Куда смотреть? #1

- В лог самого процесса: ignite.log
  - (В Ignite есть разные движки, в основном в ПРОД - log4j2 )
  - `grep exception`
  - `grep error`
  - Проблемный узел/все узлы

```
java.io.IOException: No space left on device
```

# No space left on device

- Только проактивно мониторить: например, df –h
- Маштабироваться вовремя
- Очень давно не случалось




Решение иногда:

- Persistence defragmentation
- Страницы не освобождаются назад
- Requires taking nodes out of their normal operations

# Мониторинг

- Пресеты Zabbix
- Дашборды Grafana
- Параметры оборудования, количество операций, размеры файлов (heap/region там же)
- более интеллектуальный анализ проблем

# Куда смотреть? #2

- В лог самого процесса: ignite.log
- Мониторинг

    диск, сеть, память, процессор

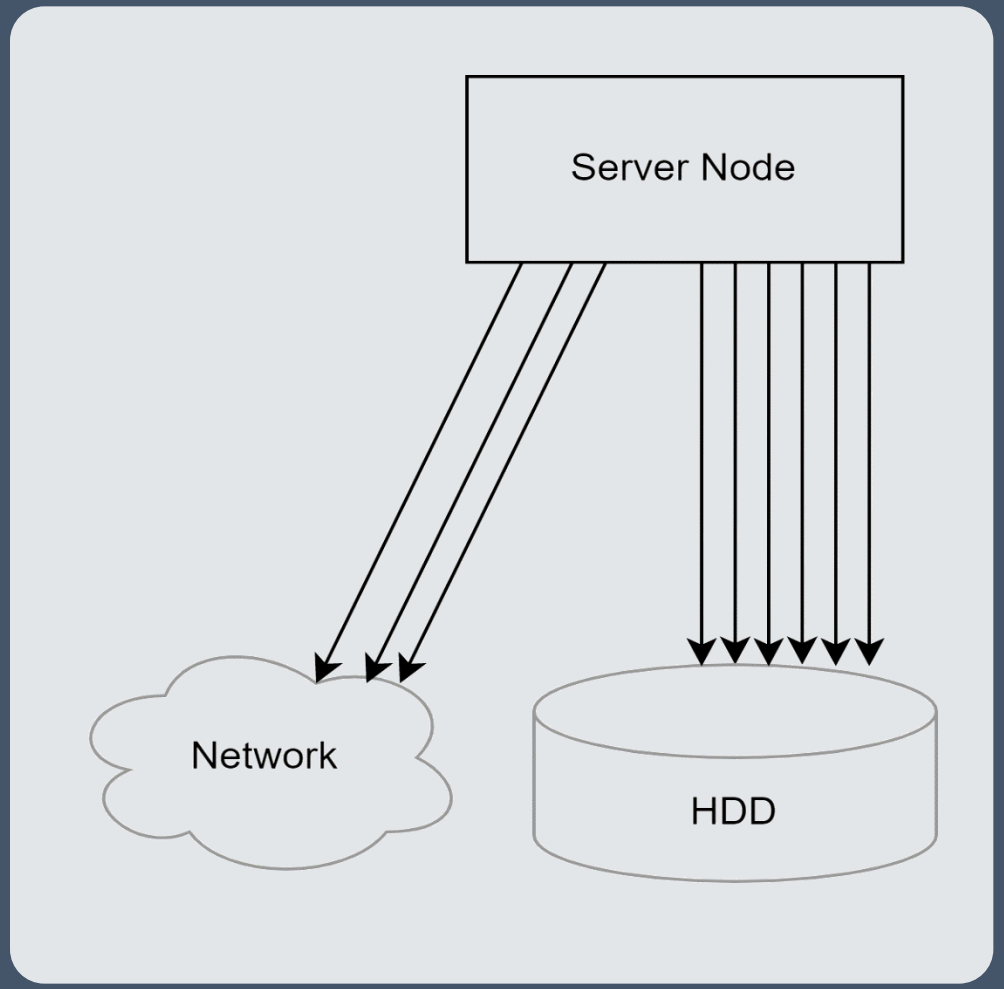
# Too many open files

java.io.FileNotFoundException:
part1021.dat (Too many open files)
Или
java.io.IOException: Too many open files

Каждая партиция каждого кеша – файл

Вывод: увеличиваем лимиты на дескрипторы

```
sysctl: fs.file-max = 300000
```



# Куда смотреть? #3

- В лог самого процесса: ignite.log
- Мониторинг
- Настройки Linux
  - лимиты

# Оперативная память серверного узла

Server RAM ~720Gb

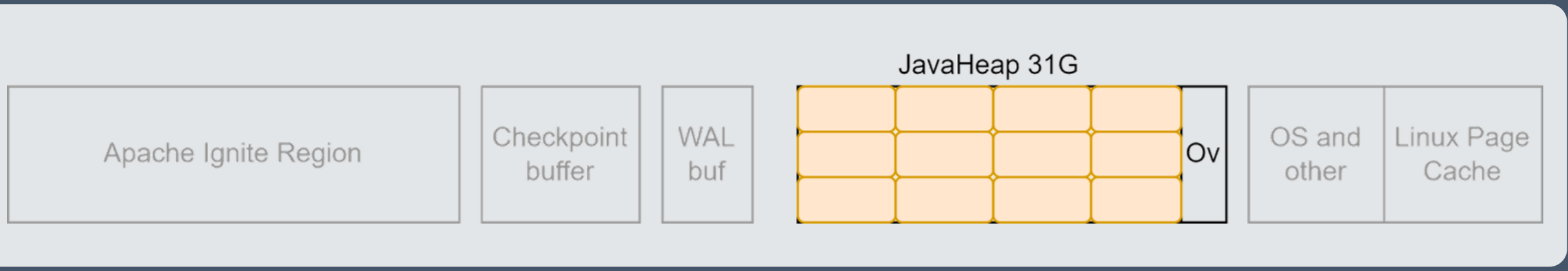
sun.misc.unsafe
4k pages
XML config


JVM ops: -Xmx=31g

JVM > 34g

| Apache Ignite Region ~578Gb | Checkpoint buffer ~8Gb | WAL buf ~1Gb | JavaHeap, ~31G | GC Ov | OS and other | Linux Page Cache |
|---|---|---|---|---|---|---|

# OutOfMemoryError: java heap space

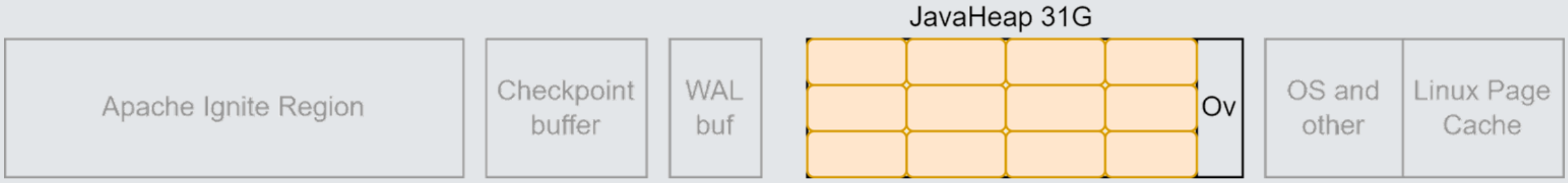



```
java.lang.OutOfMemoryError: java heap space
```

А зачем нам Java Heap:

- batch updates – `putAll/getAll`
- uncommitted transactions – `tx.start()…tx.commit()`
- SQL results – `select * from client order by last_date`

# OutOfMemoryError: что делать?




перезапустить, увеличить xmx и уменьшить batch/fetch sizes

```
jmap -dump,format=b,file=<file-path> <pid>
```

jmap is a tool to print statistics about memory (cmd=dump, format=binary, pid from jps or logs)

```
jcmd <pid> GC.heap_dump <file-path>
```

# OutOfMemoryError: что делать?


Вывод 2

- `-XX:+ExitOnOutOfMemoryError`
- `-XX:+HeapDumpOnOutOfMemoryError`
- `-XX:HeapDumpPath=/opt/ignite/ssd/logs/diag`

# Куда смотреть? #4

- В лог самого процесса: ignite.log
- Мониторинг
- Настройки Linux
- Настройки JVM - `-XX:+HeapDumpOnOutOfMemoryError, -XX:+HeapDumpOnOutOfMemoryError, -XX:HeapDumpPath …`
- **Heapdump (если есть)**

# PCI-DSS and java heap

- Heap Dump оперативки БД может содержать sensitive данные
- невозможен выход за зону PCI-DSS
- И как понять откуда OOME?

```
jmap -histo:live <pid>
(можно снять и с одного узла)
```

-Xlog:gc+classhisto*=trace

# Инстансы классов

```
num     #instances         #bytes  class name (module)
-------------------------------------------------------
 1:         20321        3071824  [B (java.base@11.0.19)
 2:          4197         505752  java.lang.Class (java.base@11.0.19)
 3:         19205         460920  java.lang.String (java.base@11.0.19)
 4:          9258         401896  [Ljava.lang.Object; (java.base@11.0.19)
 5:         10099         323168  java.util.HashMap$Node (java.base@11.0.19)
 6:          1668         267336  [I (java.base@11.0.19)
 7:          7122         227904  java.lang.invoke.LambdaForm$Name (java.base@11.0.19)
 8:          6547         209504  java.util.concurrent.ConcurrentHashMap$Node (java.base@11.0.19)
 9:           661         106672  [Ljava.util.HashMap$Node; (java.base@11.0.19)
10:          1190         104720  java.lang.reflect.Method (java.base@11.0.19)
11:          1493          71664  java.lang.invoke.MemberName (java.base@11.0.19)
12:           933          67176  java.lang.reflect.Field (java.base@11.0.19)
13:           105          66896  [Ljava.util.concurrent.ConcurrentHashMap$Node; (java.base@11.0.19)
```

# Инстансы классов

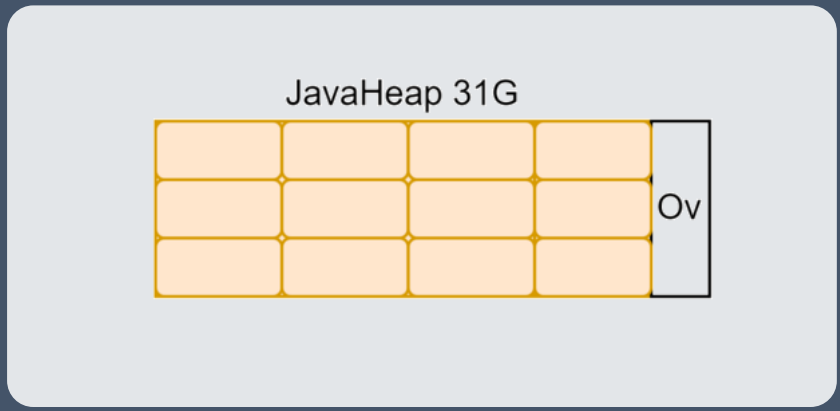
```
num     #instances         #bytes  class name (module)
-------------------------------------------------------
 1:         20321        3071824  [B (java.base@11.0.19)
 2:          4197         505752  java.lang.Class (java.base@11.0.19)
 3:         19205         460920  java.lang.String (java.base@11.0.19)
 4:          9258         401896  [Ljava.lang.Object; (java.base@11.0.19)
 5:         10099         323168  java.util.HashMap$Node (java.base@11.0.19)
 6:          1668         267336  [I (java.base@11.0.19)
 7:          7122         227904  java.lang.invoke.LambdaForm$Name (java.base@11.0.19)
 8:          6547         209504  java.util.concurrent.ConcurrentHashMap$Node (java.base@11.0.19)
 9:           661         106672  [Ljava.util.HashMap$Node; (java.base@11.0.19)
10:          1190         104720  java.lang.reflect.Method (java.base@11.0.19)
11:          1493          71664  java.lang.invoke.MemberName (java.base@11.0.19)
12:           933          67176  java.lang.reflect.Field (java.base@11.0.19)
13:           105          66896  [Ljava.util.concurrent.ConcurrentHashMap$Node; (java.base@11.0.19)
```

# Куда смотреть? #5

- В лог самого процесса: ignite.log
- Мониторинг
- Настройки Linux
- Настройки JVM
- Heapdump (если есть)
- GC logs
  - **Статистика по классам, top 3-5**

# Раннее обнаружение – история провалов

```
-XX:InitiatingHeapOccupancyPercent=30
```

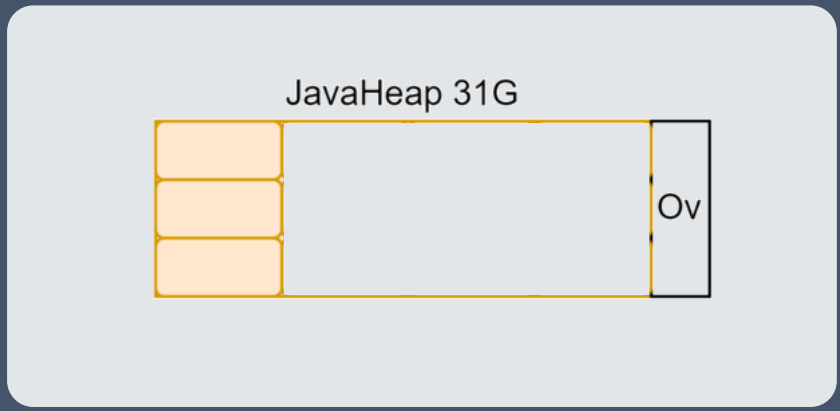



```
MemoryUsage m = ManagementFactory.getMemoryMXBean()
                                 .getHeapMemoryUsage();

if(m.getUsed() > m.getMax() * 0.9) {
    !ALERT!
}
```

# Раннее обнаружение – история провалов

```
-XX:InitiatingHeapOccupancyPercent=30
-XX:-G1UseAdaptiveIHOP
```

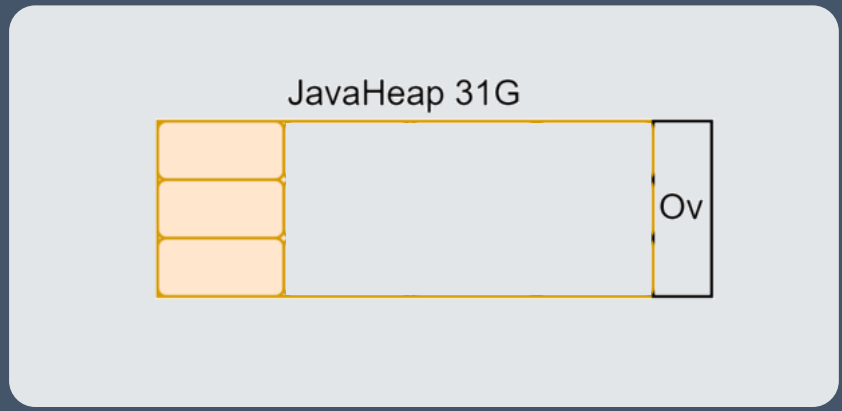



```
MemoryUsage m = ManagementFactory.getMemoryMXBean()
                                 .getHeapMemoryUsage();

if(m.getUsed() > m.getMax() * 0.9) {
    !ALERT!
}
```

# Раннее обнаружение – история провалов

```
if(mu.getUsed() > mu.getMax() * 0.9) {
    !ALERT!
}
```

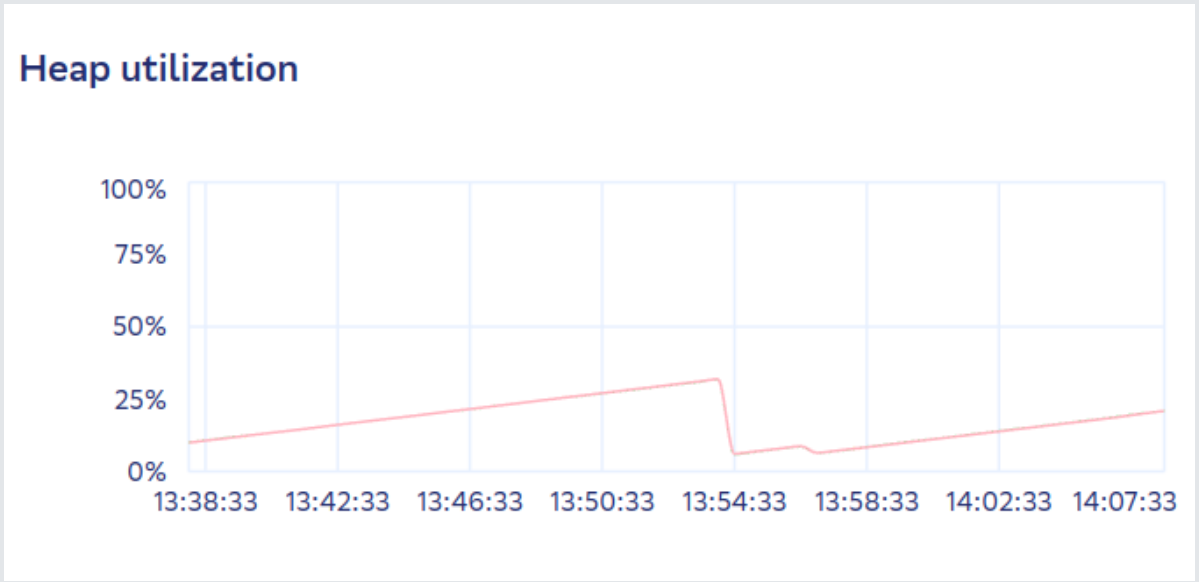



- `mu.getUsed()` - Правдиво одну миллисекунду после получения
- `mu.getUsed()` - Ничего не говорит о достижимости объекта

Не делайте так

# PlatformV GridCenter

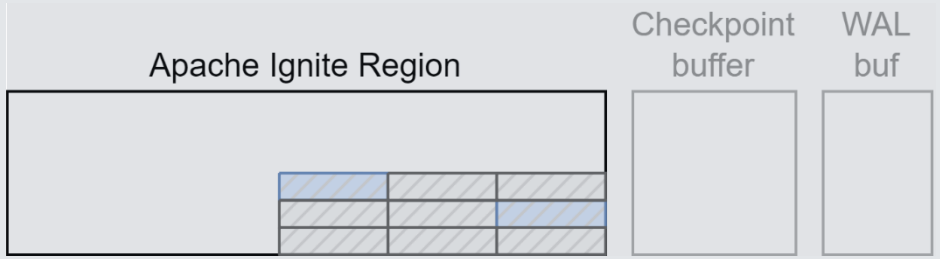
- UI-интерфейс + средство администрирования
- Отдельный продукт для кластеров Apache Ignite и PlatformV DataGrid
- Количество tx, cache put/get/remove
- Метрики memory.heap.*



# Примеры падений

- **Процесс был и нет**

# Регион “небезопасный”




`UNSAFE.allocateMemory(size)`

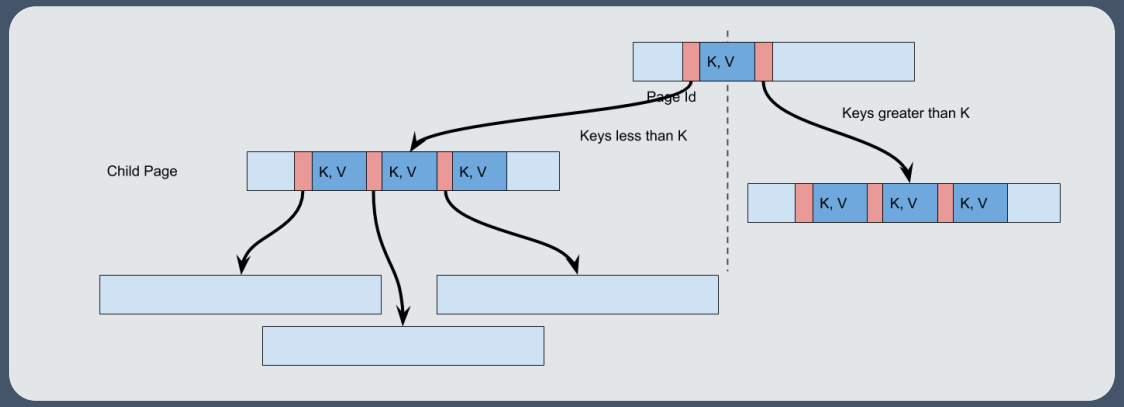
В основном страницы 4K

Основное хранилище – часть Multitier storage

Работаем с указателями напрямую

# Структуры в unmanaged памяти

Доступ к unsafe надо разрешать

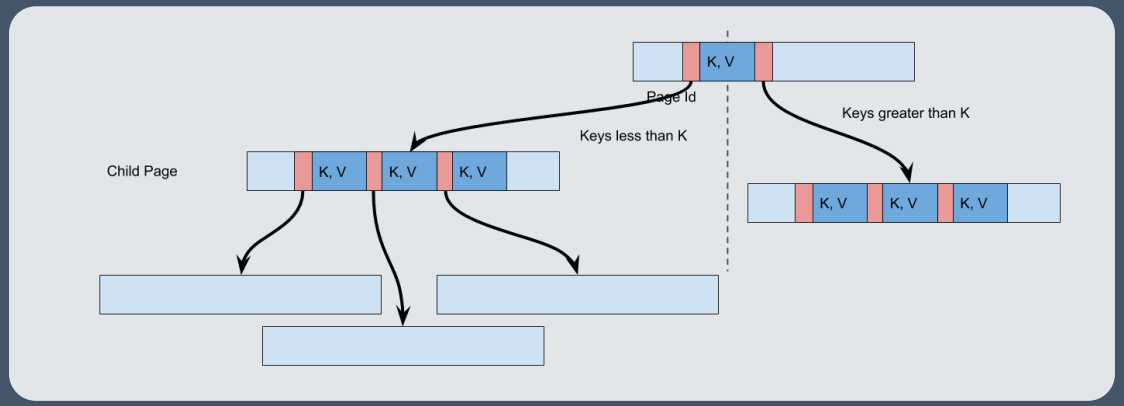



Пример для Java 11:

```
--add-exports=java.base/jdk.internal.misc=ALL-UNNAMED
--add-exports=java.base/sun.nio.ch=ALL-UNNAMED
--add-exports=java.management/com.sun.jmx.mbeanserver=ALL-UNNAMED
--add-exports=jdk.internal.jvmstat/sun.jvmstat.monitor=ALL-UNNAMED
--add-exports=java.base/sun.reflect.generics.reflectiveObjects=ALL-UNNAMED
--add-opens=jdk.management/com.sun.management.internal=ALL-UNNAMED
--add-opens=java.base/jdk.internal.access=ALL-UNNAMED
--illegal-access=permit
```

# Структуры в unmanaged памяти

И при запуске будет warning
– это нормально




```
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.ignite.internal.util.GridUnsafe$2
```

О чем нас предупреждает VM?

при выходах за границы мы получим не ArrayIndexOutOfBounds

# JVM crash

```
-XX:ErrorFile=/opt/ignite/logs/diag/crash-dump-%p.err
```

```
#
# A fatal error has been detected by the Java Runtime Environment:
#
#  SIGSEGV (0xb) at pc=0x00007f99307fd2af, pid=3672351, tid=3672842
#
# JRE version: OpenJDK Runtime Environment Temurin-11.0.19+7 (11.0.19+7) (build 11.0.19+7)
# Java VM: OpenJDK 64-Bit Server VM Temurin-11.0.19+7 (11.0.19+7, mixed mode, tiered,
compressed oops, g1 gc, linux-amd64)
# Problematic frame:
# J 22616 c2
```

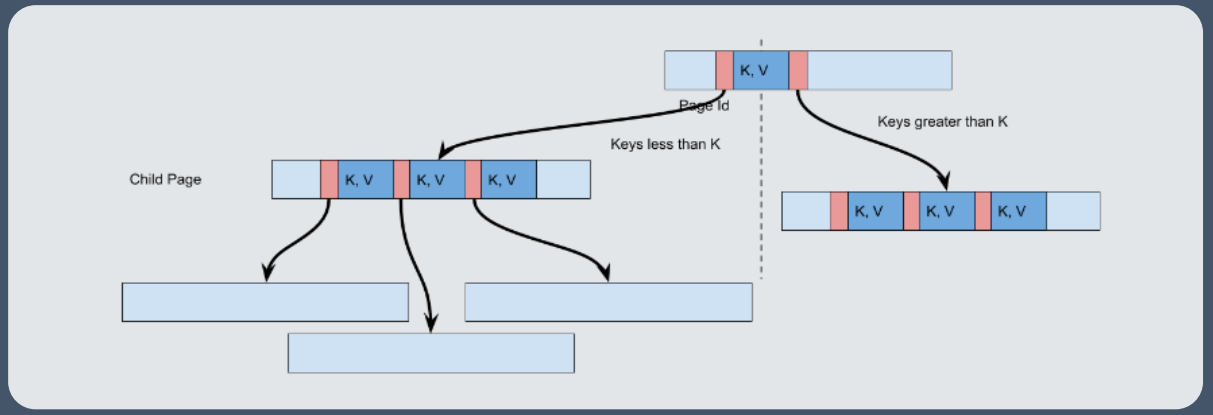
**Андрей Паньгин - Аварийный дамп — «черный ящик» упавшей JVM**
https://youtu.be/6qpQjEQ547o

# Процесс прервался: быстрая проверка

Ctrl+F : Ignite

## A fatal error has been detected by the Java Runtime Environment
.....
Stack slot to memory mapping:stack at sp + 0 slots: 0x0000001000001018 is an unknown valuestack at sp + 1 slots: 0x00007e9937330aa8 points into unknown readable memory: 0x0000000000010001 | 01 00 01 00 00 00 00 00stack at sp + 2 slots: 0x000000100f83d380 is an oop: org.apache.ignite.internal.processors.cache.persistence.tree.io.DataPageIO {0x000000100f83d380} - klass: 'org/apache/ignite/internal/processors/cache/persistence/tree/io/DataPageIO'

В Unsafe – что-то строили, строили … и не построили



# В креш дампе ничего про продукт нет

```
Current thread (0x00007f8cfe76b800):  JavaThread "C2 CompilerThread0" daemon [_thread_in_vm, id=442381,
stack(0x00007f8c12ef3000,0x00007f8c12ff4000)]


Current CompileTask:

C2:73265187 21710   !   4       sun.security.ssl.SSLEngineInputRecord::decodeInputRecord (812 bytes)

Stack: [0x00007f8c12ef3000,0x00007f8c12ff4000],  sp=0x00007f8c12fee340,  free space=1004k

Native frames: (J=compiled Java code, A=aot compiled Java code, j=interpreted, Vv=VM code, C=native code)

V  [libjvm.so+0xf077e2]  VMError::report_and_die(int, char const*, char const*, __va_list_tag*, Thread*,
unsigned char*, void*, void*, char const*, int, unsigned long)+0x1c2

V  [libjvm.so+0xf08793]  VMError::report_and_die(Thread*, char const*, int, unsigned long, VMErrorType,
char const*, __va_list_tag*)+0x43

V  [libjvm.so+0x6fc210]  report_vm_out_of_memory(char const*, int, unsigned long, VMErrorType, char
const*, ...)+0x110
```

Вероятно, JVM баг

https://bugs.openjdk.org/browse/JDK-8291665

C2: assert compiling SSLEngineInputRecord::decodeInputRecord

# Куда смотреть? #6

- В лог самого процесса: ignite.log
- Мониторинг
- Настройки Linux
- Настройки JVM
- Heapdump (если есть)
- Логи GC
- **Crashdump (если есть)**

# Обрыв логов без крешдампа

# Обрыв логов без крешдампа

- проверка логов ядра

  dmesg -T

(- T - human readable date)

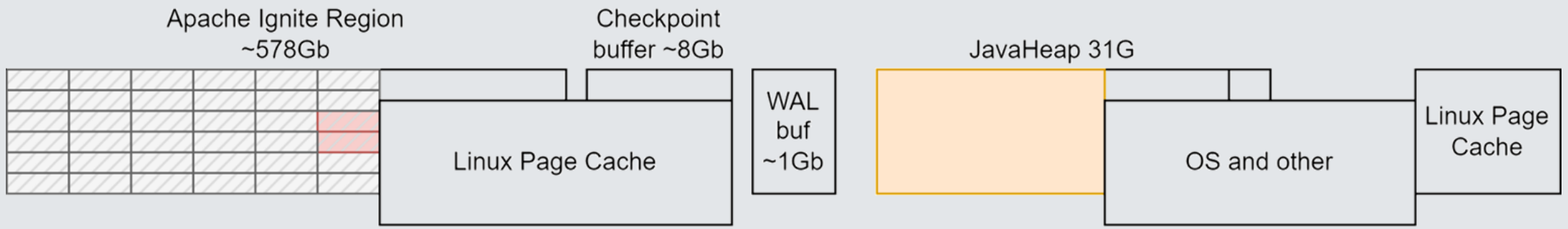
- ищем Out of memory или kill process

Out of memory: Kill process 191841 (java) score 982 or sacrifice child

# Куда смотреть? #7

- В лог самого процесса: ignite.log
- Мониторинг
- Настройки Linux
- Настройки JVM
- Heapdump (если есть)
- Логи GC
- Crashdump (если есть)
- **Логи линукса: /var/log/messages; /var/log/dmesg**

# Отключаем оптимизации с памятью




**Вывод 1**

`-Xms = -Xmx`

\+ ставим `-XX:+AlwaysPreTouch`

**Вывод 2**

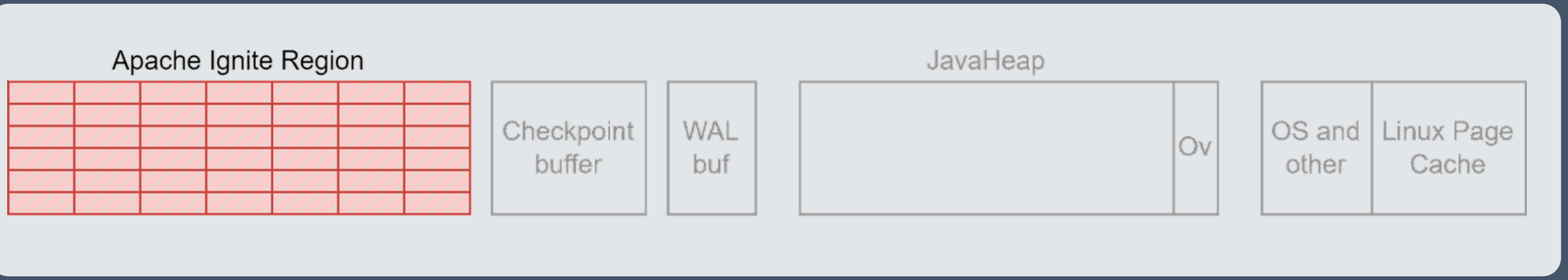
```
vm.overcommit_ratio = 100
vm.overcommit_memory = 2
```

# Куда смотреть? #7A

- В лог самого процесса: ignite.log
- Мониторинг
- Настройки Linux: `vm.overcommit_ratio=100; vm.overcommit_memory=2`
- Настройки JVM: `-Xms = -Xmx; -XX:+AlwaysPreTouch`
- Heapdump (если есть)
- Логи GC
- Crashdump (если есть)
- **Логи линукса**

# Исчерпание региона

IgniteOutOfMemoryException: Out of memory in data region




Обычно не проблема, т.к. обычно есть одно из трех:

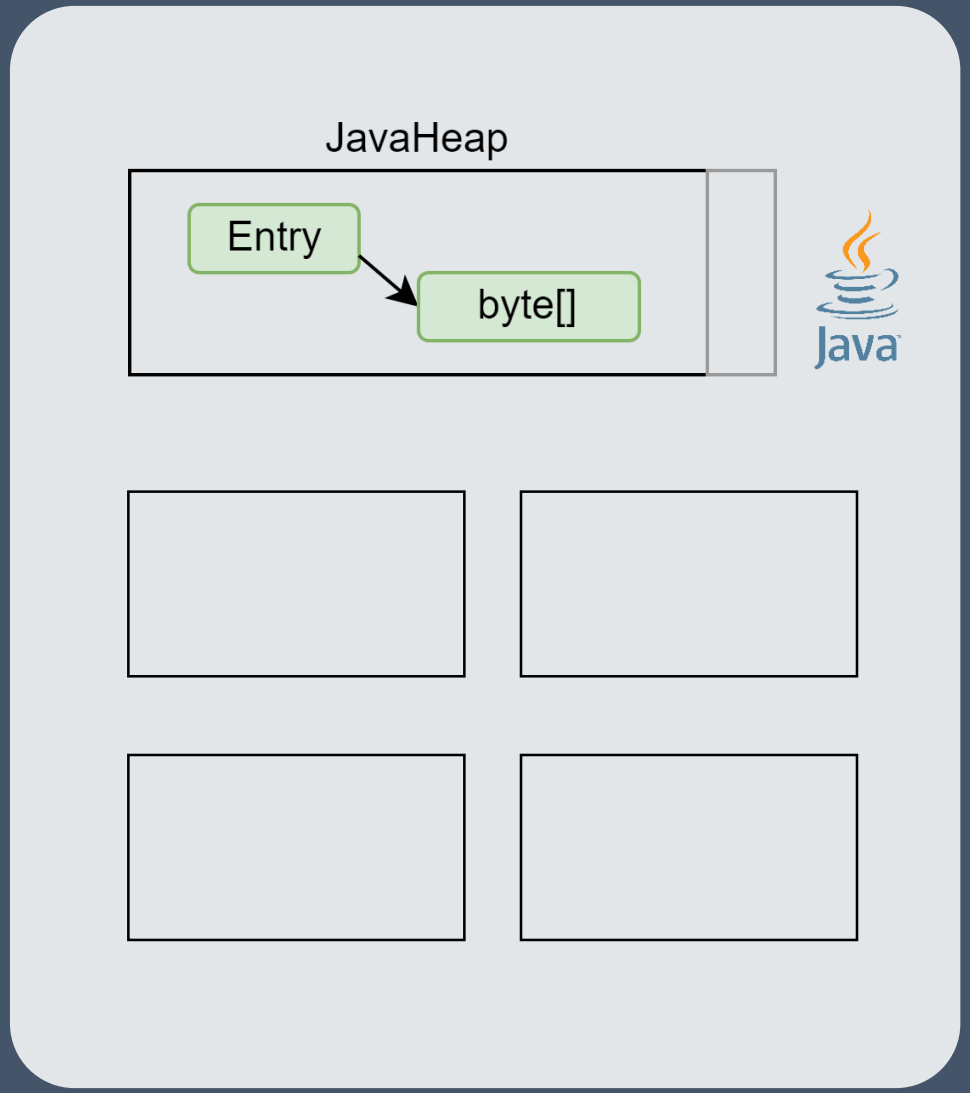
- Eviction (не повезло этой записи)
- Expiration (time to live)
- Persistence (будет ротация с диском)

# VM Capacity planning

Мы тут все упростили

- Overhead GC (зависит от коллектора)
- Direct память (буферы для IO)
- Классы
- Скомпилированный код

**Андрей Паньгин - Память Java процесса по полочкам**
**https://youtu.be/kKigibHrV5I**



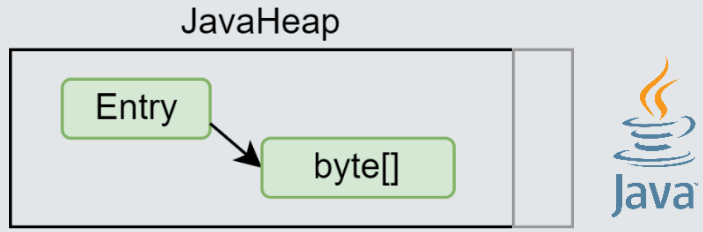
# Проблемы с диском

- Ваш Ignite тормозит
- Не укладываемся в SLA проекта (200ms)
- Провалы в пропускной способности

# Наша Entry уже в пути

Это же просто

1. Преобразоваться в байты
2. Записать на диск

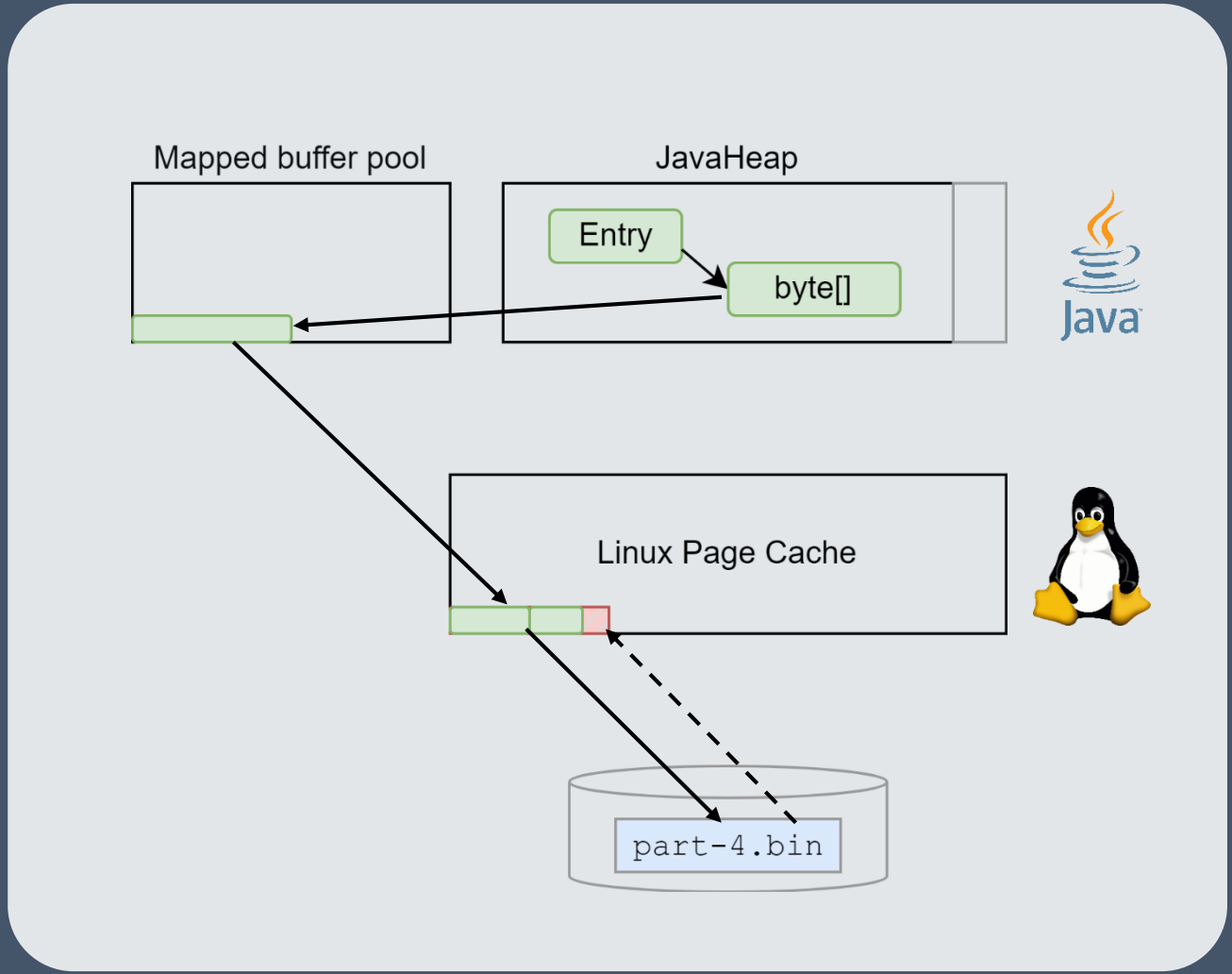




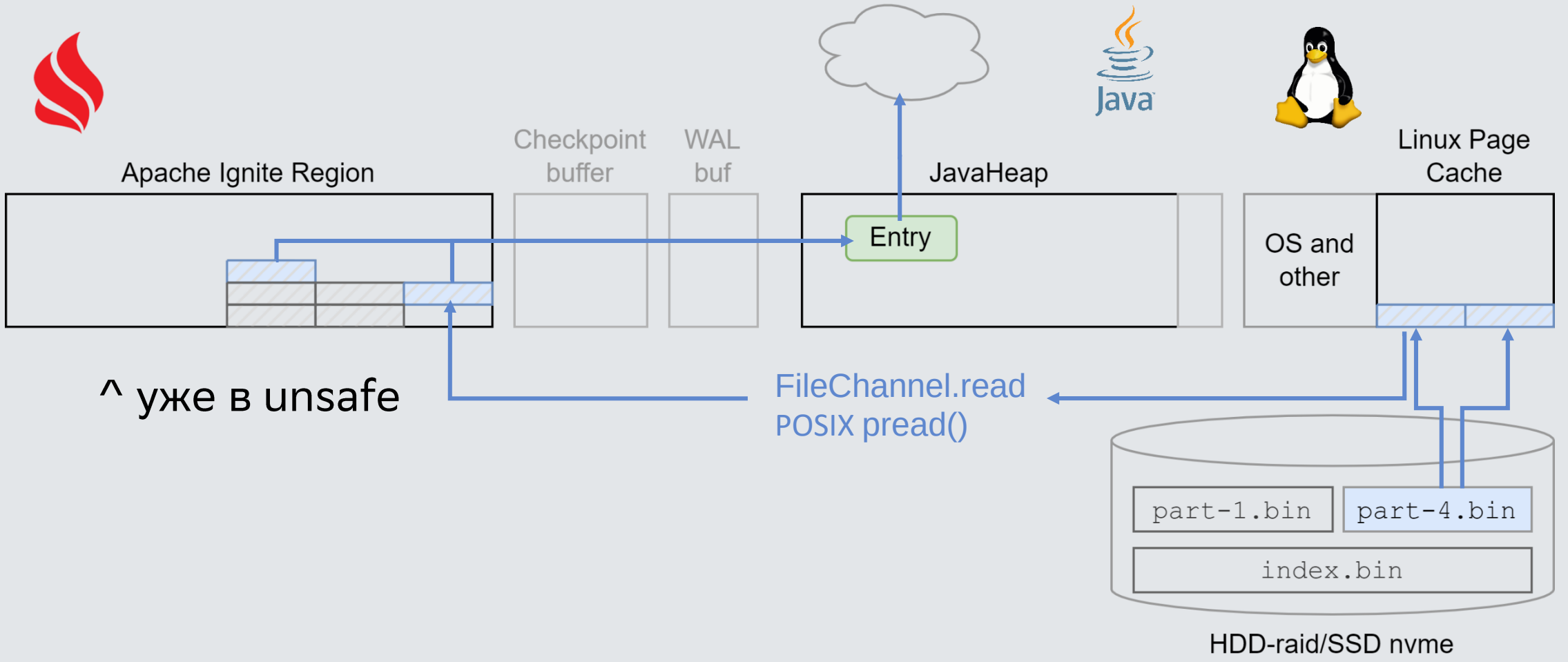
**Дмитрий Говорухин — Эффективная работа с файлами для Java-разработчиков**
https://youtu.be/7GlMS630dt8

# Дальше еще много уровней

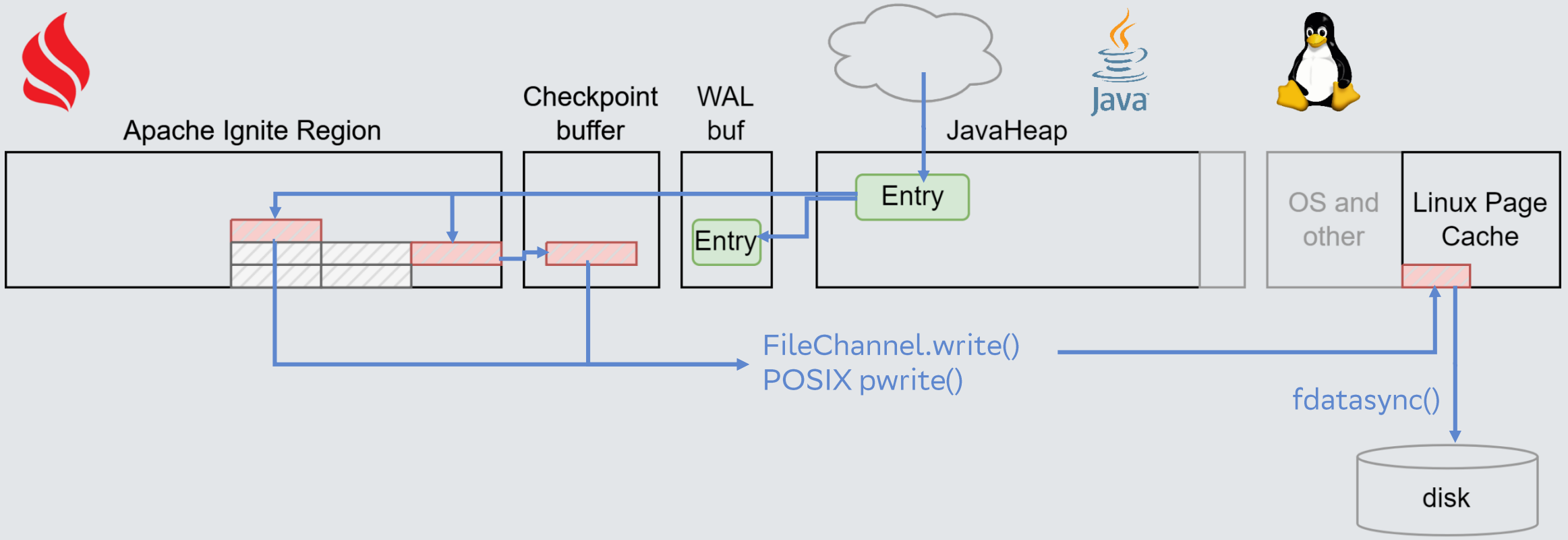
- Используем/выделим нативный буфер
- Он побьётся на страницы и попадет в кеш страниц линукса
- Еще файловая система
- Еще оборудование



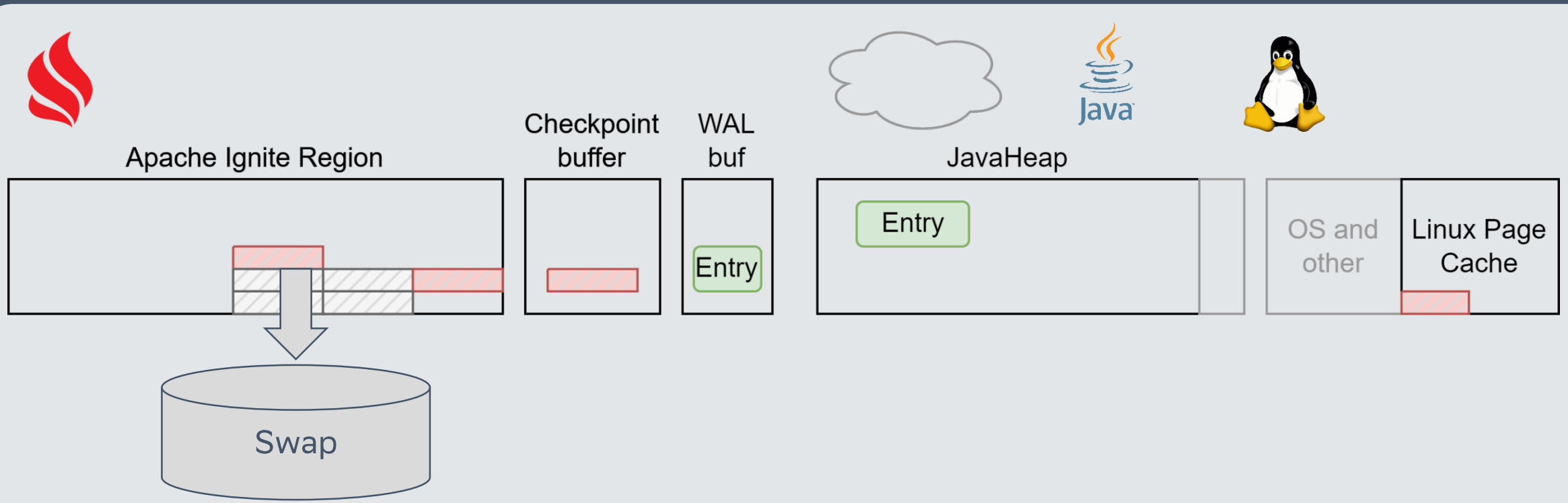
# Ignite: Чтение данных



# Ignite: Запись на диск



# А если мы ошиблись с Capacity planning?




Отключать swap совсем
или sysctl: `vm.swappiness = 0`

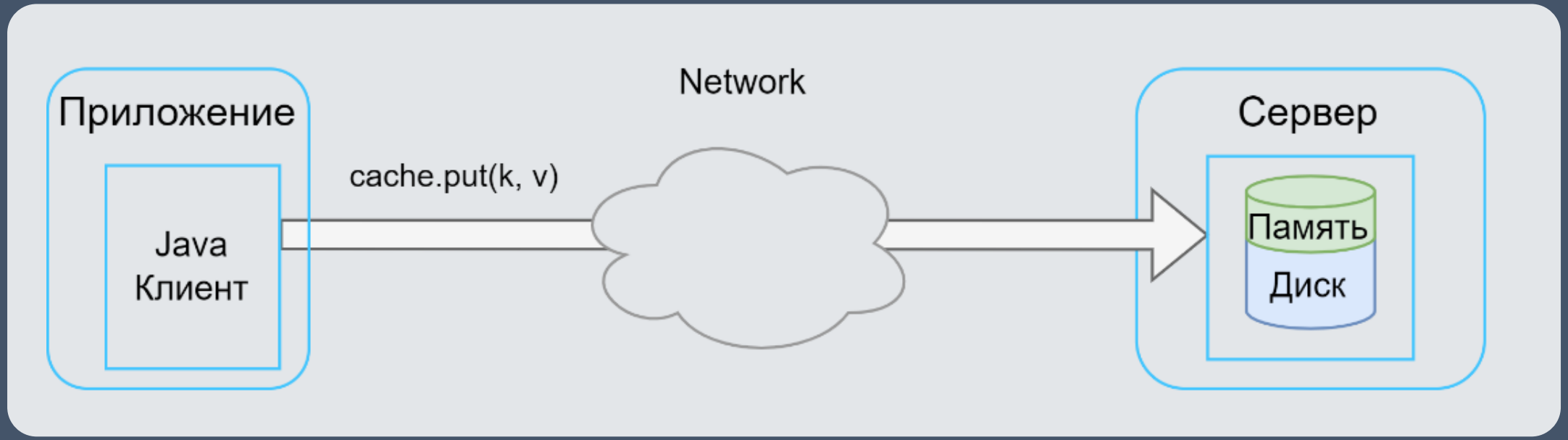
# Куда смотреть? #8

- В лог самого процесса: ignite.log
- Мониторинг
- Настройки Linux: `vm.swappiness` **и нет включенного swap**
- Настройки JVM
- Heapdump (если есть)
- Логи GC
- Crashdump (если есть)
- Логи линукса

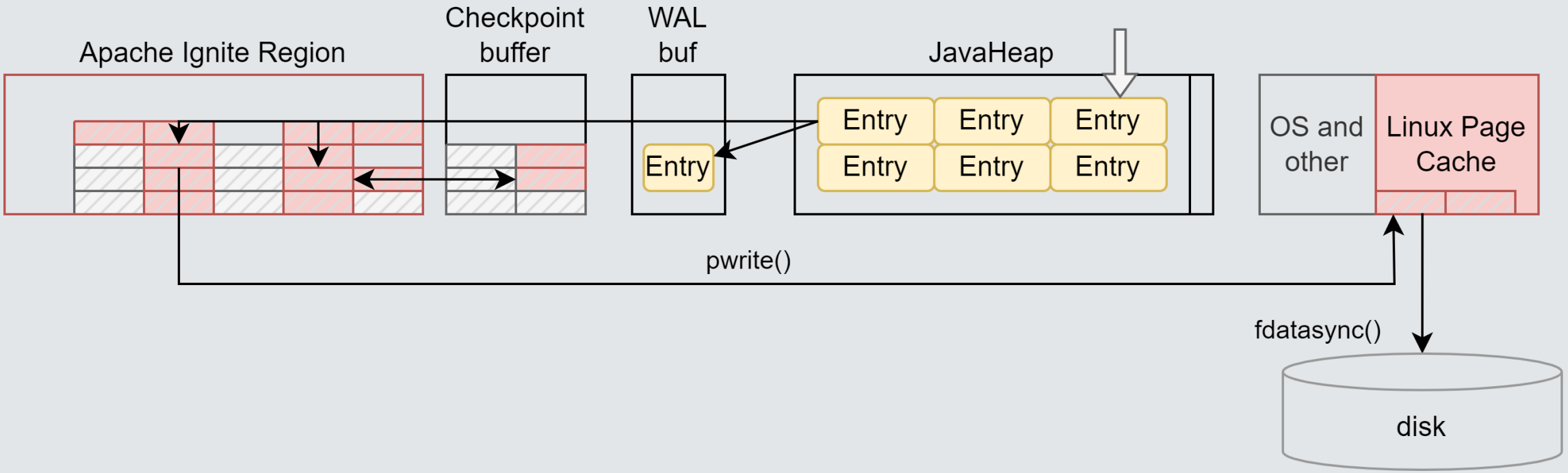
# Но я обещал провал?

Активно кладем что-то в кеш: put, putAll

Уходит по сети на сервер



# Entries->Region->Linux->Disk

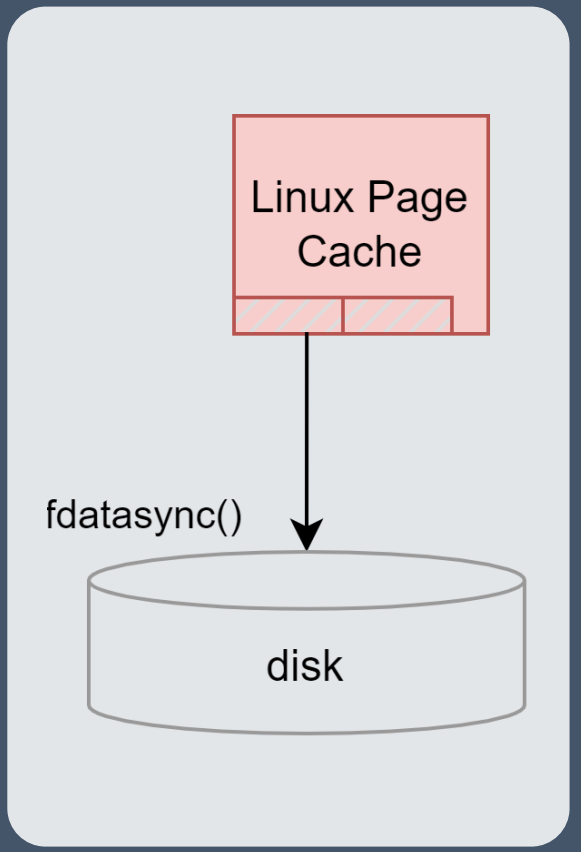



- Страница ждет write
- Write ждет сброса

# Получили паузу в Ignite

VM работает – условно хорошо

Но вся большая цепочка ждет большого сброса




Проценты от **свободной** памяти для начала сброса

```
vm.dirty_background_ratio = 1
```

- начнем сброс при 1% утилизации – в фоне

```
vm.dirty_ratio = 20
```

- 20% утилизации - для безусловного, синхронно

# Получили паузу в Ignite

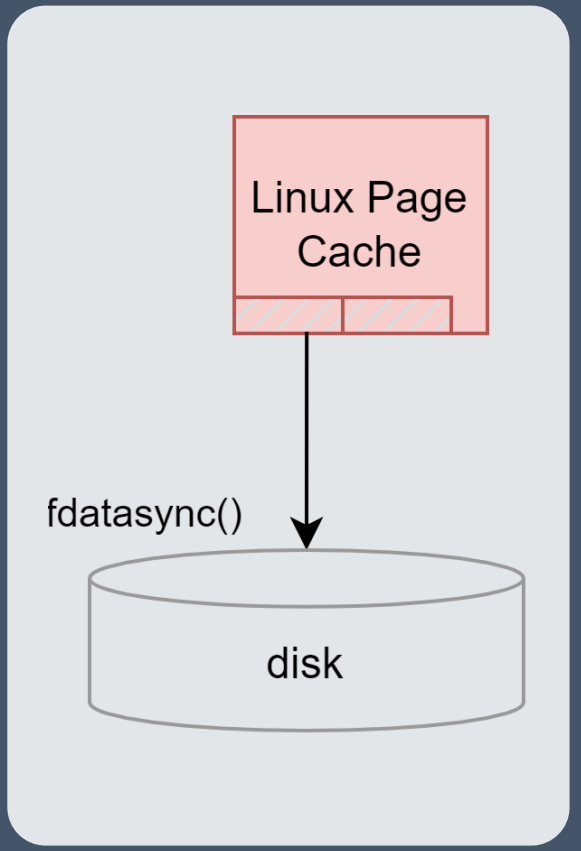
Еще опции по сбросу по времени:

```
vm.dirty_expire_centisecs = 500
```

- Через пол секунды – прости, но ты слишком стара для кеша. На диск, дорогая, на диск

```
vm.dirty_writeback_centisecs = 100
```

- как часто надо это проверять – 10 раз в секунду

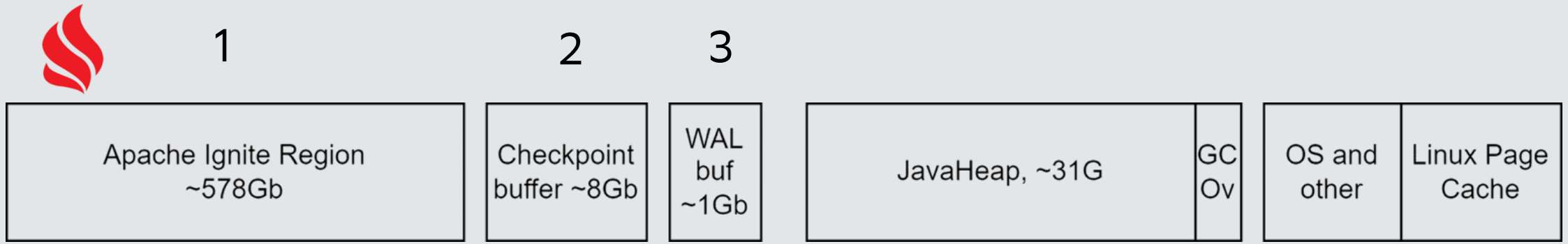


# Куда смотреть? #8а

- В лог самого процесса: ignite.log
- Мониторинг
- Настройки Linux: `vm.dirty_background_ratio`, `vm.dirty_ratio`, `vm.dirty_expire_centisecs`, `vm.dirty_writeback_centisecs`
- Настройки JVM
- Heapdump (если есть)
- Логи GC
- Crashdump (если есть)
- Логи линукса

# Еще кейсы

1. Регион полный + page rotation
2. Checkpoint buffer полный
3. WAL buf не успевает списаться




Direct IO – предсказуемые паузы очень большой ценой – not recommended
Чаще сброс на диск – в настройках Ignite

# Чудес не бывает, нам не превзойти

Суммарные возможности всех дисковых подсистем кластера

- для page store – IOPS
- для WAL – пропускная способность

Компрессия

- Apache Ignite – Storage
- PlatformV DataGrid – Storage + RAM

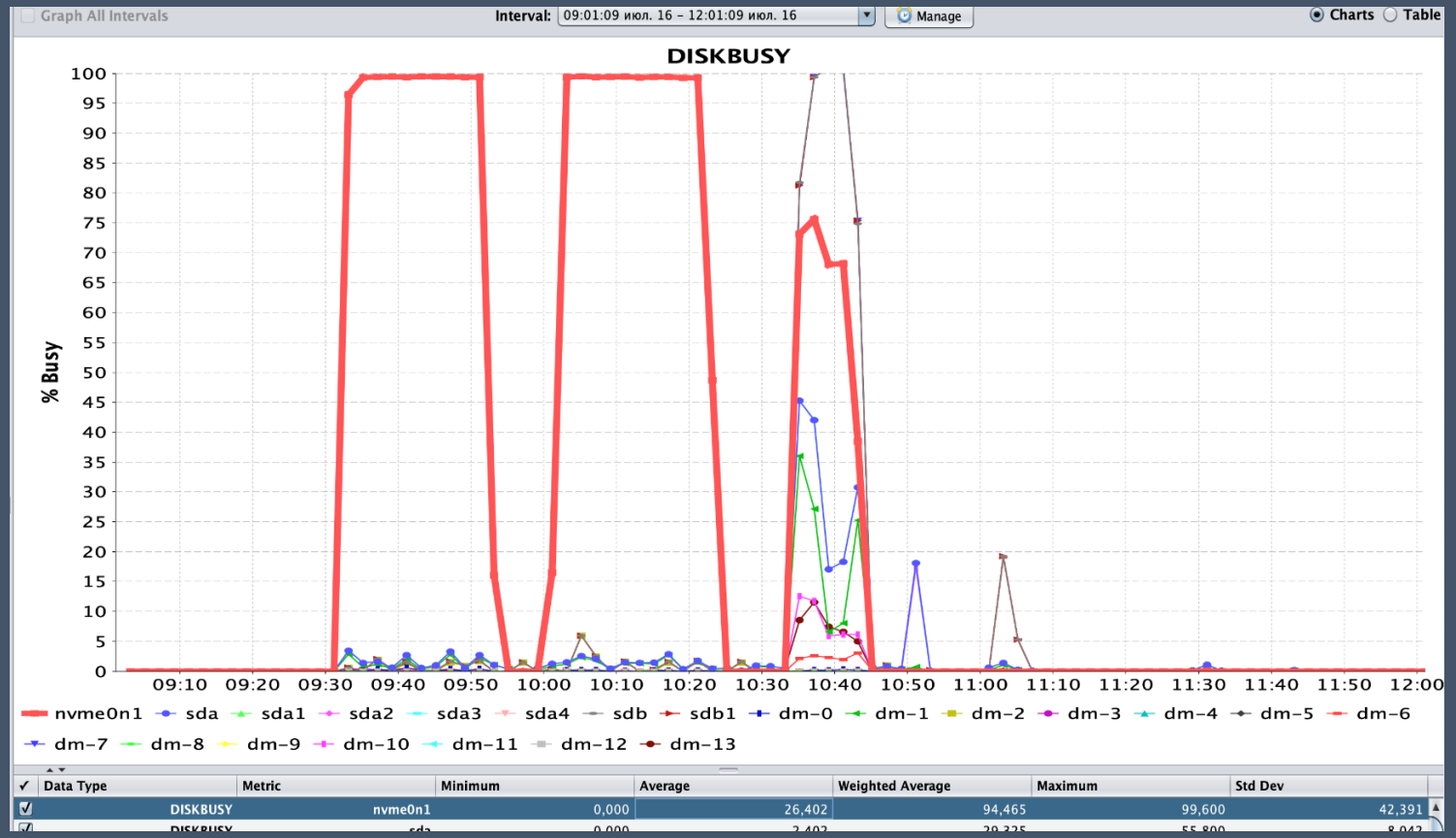

Надо бы посмотреть на состояние дисков

# nmon – что было с оборудованием

- https://nmon.sourceforge.io/
- Обычно: Настраивается в cron, раз в секунду или более
- Пишет /var/log/nmon/ хорошо жмется

```
DISKBUSY,T0025,23.4,0.0,0.0,101.00,0.0,0.0,101.00,0.0,0.0,0.0,0.0,0.0,0.0,0.5,0.0,1.0,0.0,0.0,101.00
DISKREAD,T0025,121307.9,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
DISKWRITE,T0025,97.4,21.0,21.0,123201.3,0.0,0.0,123201.3,0.0,0.0,0.0,0.0,21.0,0.0,1.6,0.0,5.2,0.0,0.0,1
23211.8
DISKXFER,T0025,1119.5,0.3,0.3,483.5,0.0,0.0,483.5,0.0,0.0,0.0,0.0,0.3,0.0,0.2,0.0,0.7,0.0,0.0,483.0
DISKBSIZE,T0025,108.4,65.3,65.3,254.8,0.0,0.0,254.8,0.0,0.0,0.0,0.0,65.3,0.0,10.7,0.0,7.7,0.0,0.0,255.1
```

# Nmon хорошо визуализируется




https://nmonvisualizer.github.io/nmonvisualizer/

java -jar NMONVisualizer_2021-04-04.jar

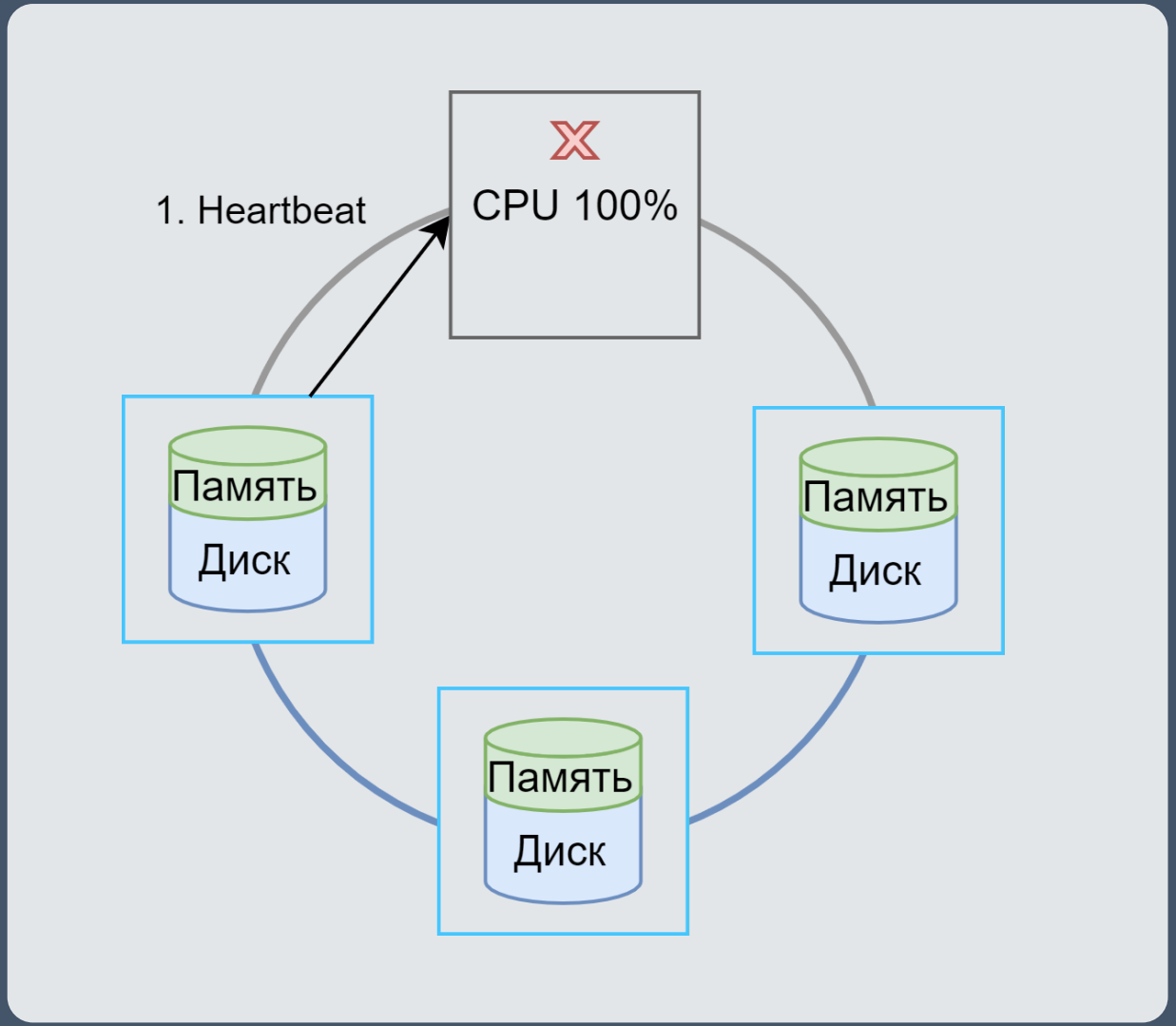
# Куда смотреть? #9

- В лог самого процесса: ignite.log
- Мониторинг
- Настройки Linux
- Настройки JVM
- Heapdump (если есть)
- Логи GC
- Crashdump (если есть)
- Логи линукса <- + **NMON + /var/log/nmon/ и** NMONVisualizer

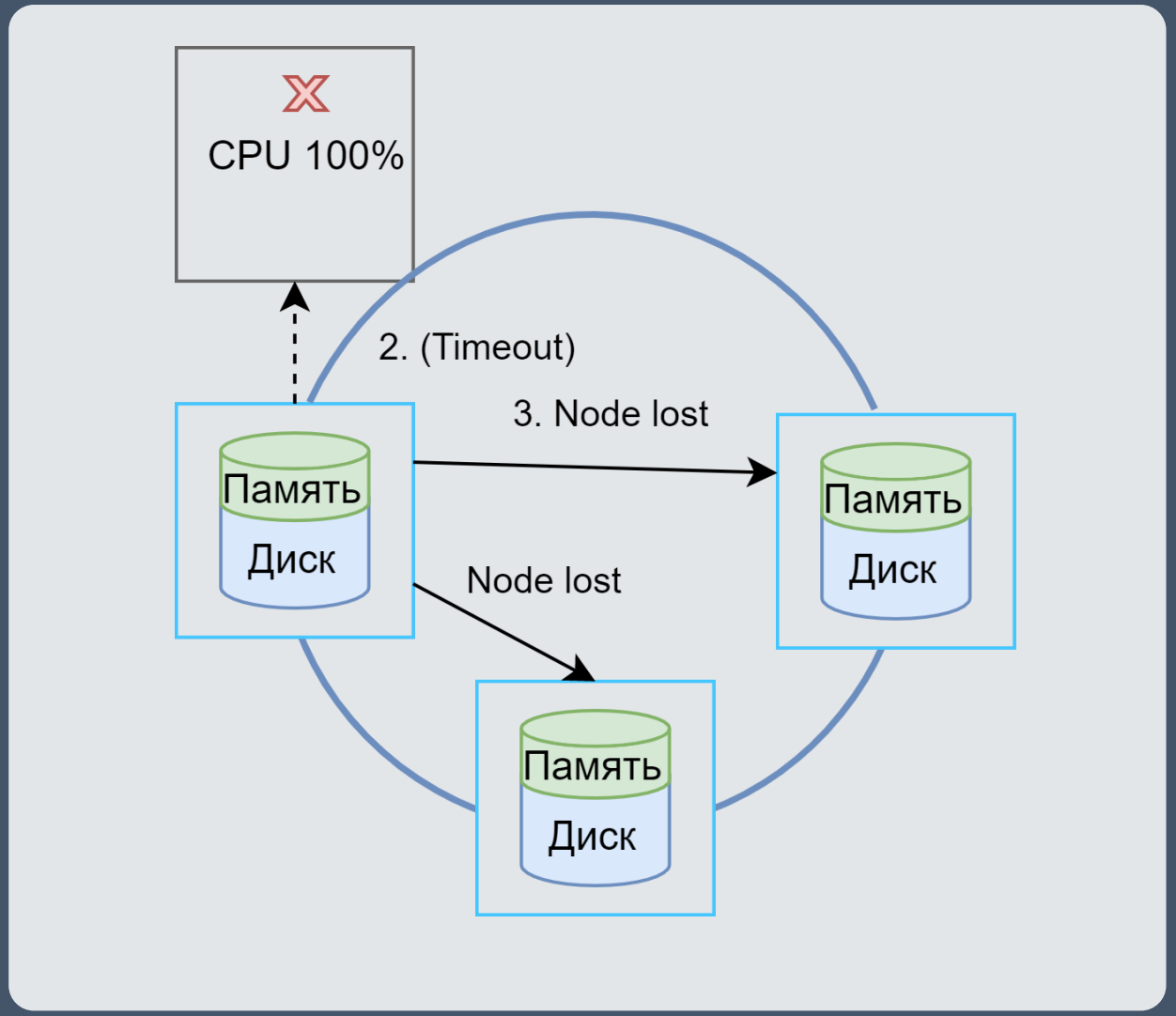
# Производительность

- Когда вообще все плохо

# Последствия очень длинной паузы



# Последствия очень длинной паузы



# Обнаружение пауз в VM / hiccup

```
long lastWakeUpTime = System.nanoTime();
Thread.sleep(PRECISION);
long pause = NANOSECONDS.toMillis(
                        System.nanoTime() - lastWakeUpTime);


if (pause >= THRESHOLD + PRECISION) {
      !ALERT!
}
```

Вывод: как первый этап для анализа работает

Лучше - jHiccup:
https://github.com/giltene/jHiccup

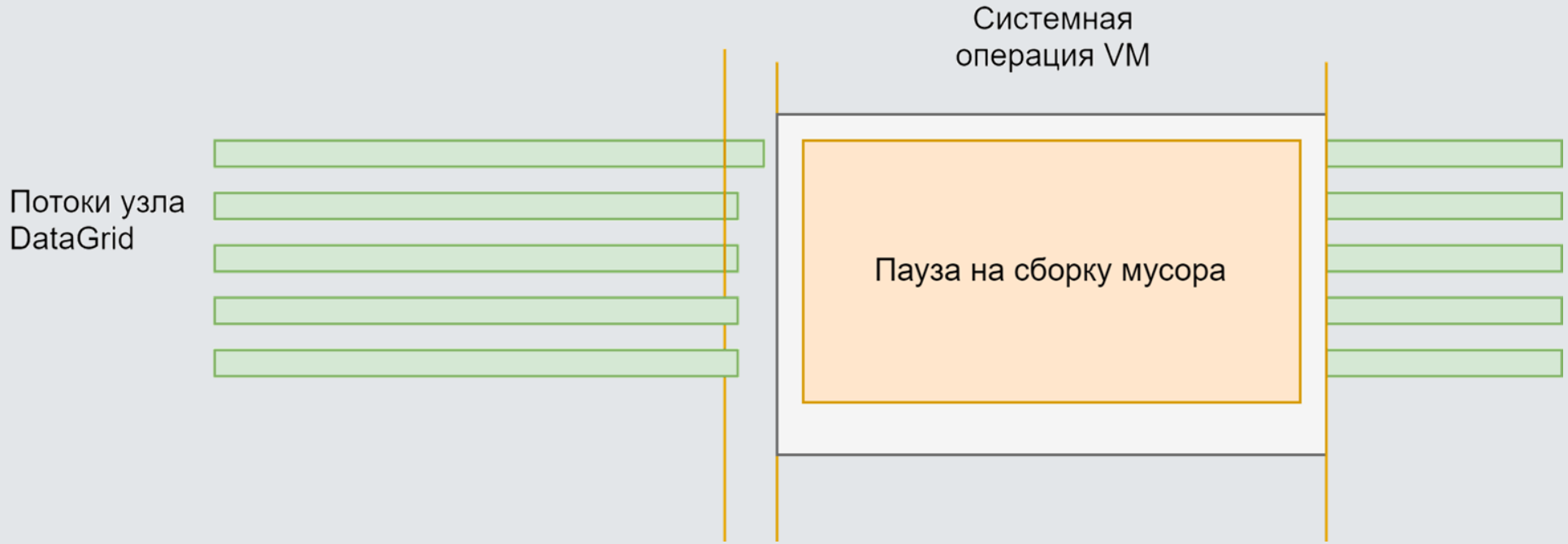
# Куда смотреть? #10

- В лог самого процесса: ignite.log **<- +предупреждения о паузах**
- Мониторинг: **<- + состояние пауз/hiccup**
- Настройки Linux
- Настройки JVM
- Heapdump (если есть)
- Логи GC
- Crashdump (если есть)
- Логи линукса

# Если пауза в GC

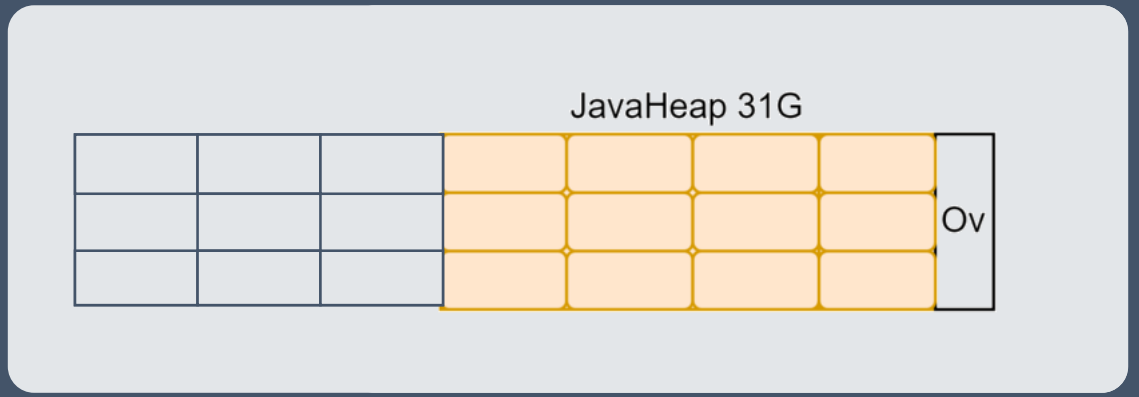
Пока Java11 + коллектор G1

(-XX:+UseG1GC)



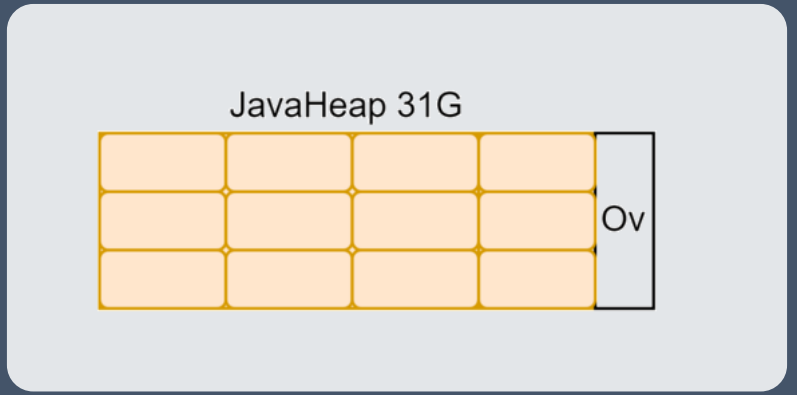
# А что делать с паузами?

-Xmx увеличить: **4, 8, 16,**



# Почему 42?

-Xmx увеличить: **4, 8, 16, 31**




Обычно **31**

Чтобы работала оптимизация Compressed Ordinary Object Pointers (oops):

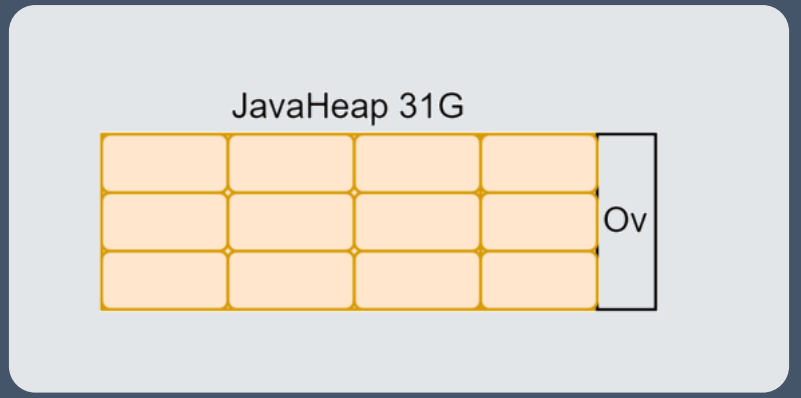
- выравнивание до 8 => 3 последних бита адреса можем не хранить
- 2^32*8= 32g

https://bugs.openjdk.org/browse/JDK-8320710
Adjust heap size when close to compressed oops limit

# Попросим G1: сделай нам все хорошо

- G1: -XX:MaxGCPauseMillis = 50

Цель по паузам – 50 миллисекунд




- Вывод – не сработало

- G1: -XX:MaxGCPauseMillis = 100
- как вариант допускаем, но не требуем у проектов

# Придется смотреть в логи

```
-Xlog:gc*=debug,gc+classhisto=trace:file=ignite_se_jvm_gc_log:
time,uptime,tags:filecount=50,filesize=50M
```

```
-Xlog:safepoint:file=ignite_se_jvm_safepoint.log:
time,uptime,tags:filecount=50,filesize=50M
```

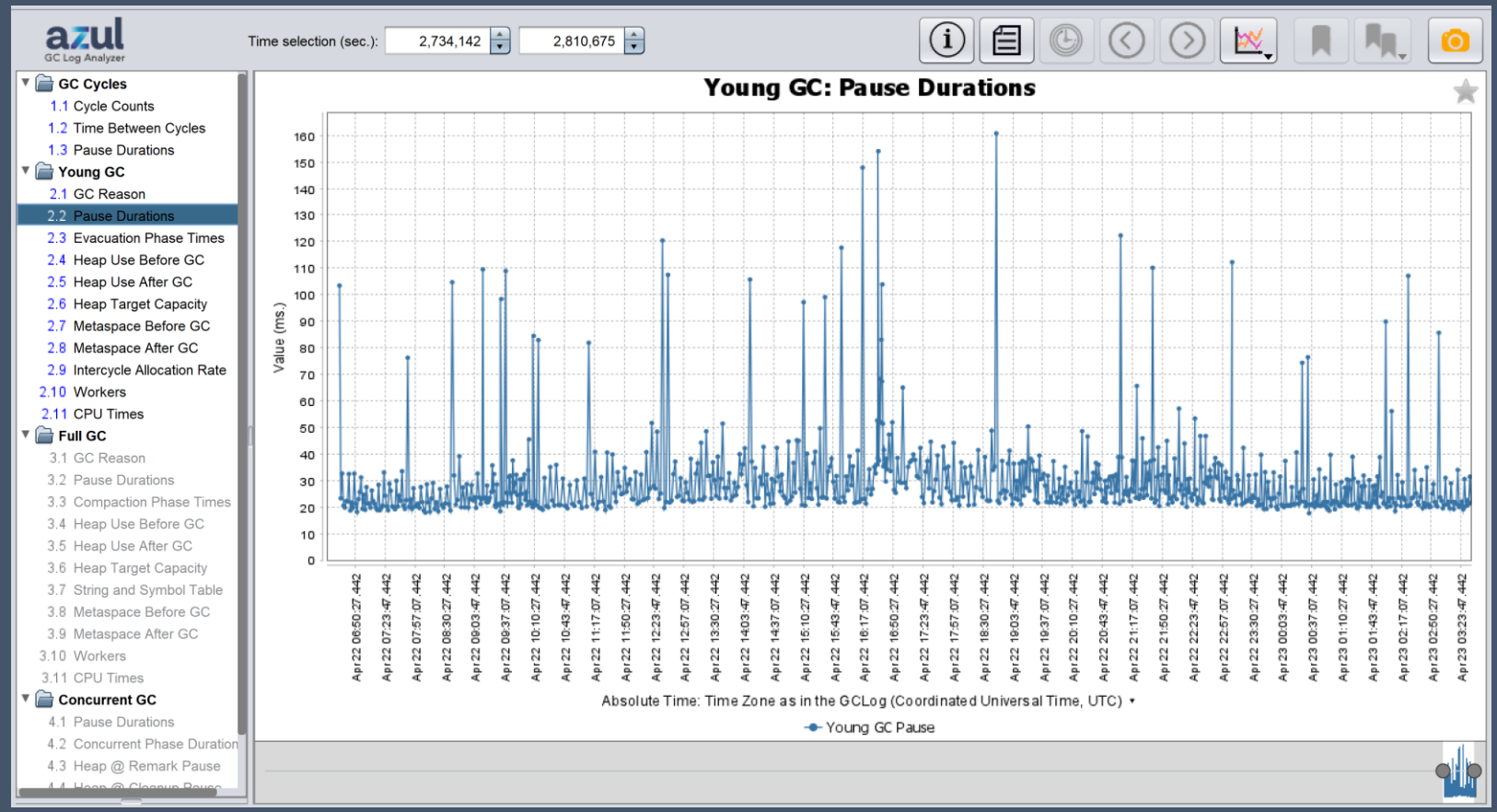
- Удобно оказалось разнести, разная интенсивность
- Храним много (возможно слишком)

# Еще один jar нам в помощь

Корреляции

Allocation рейт

Пример за сутки:

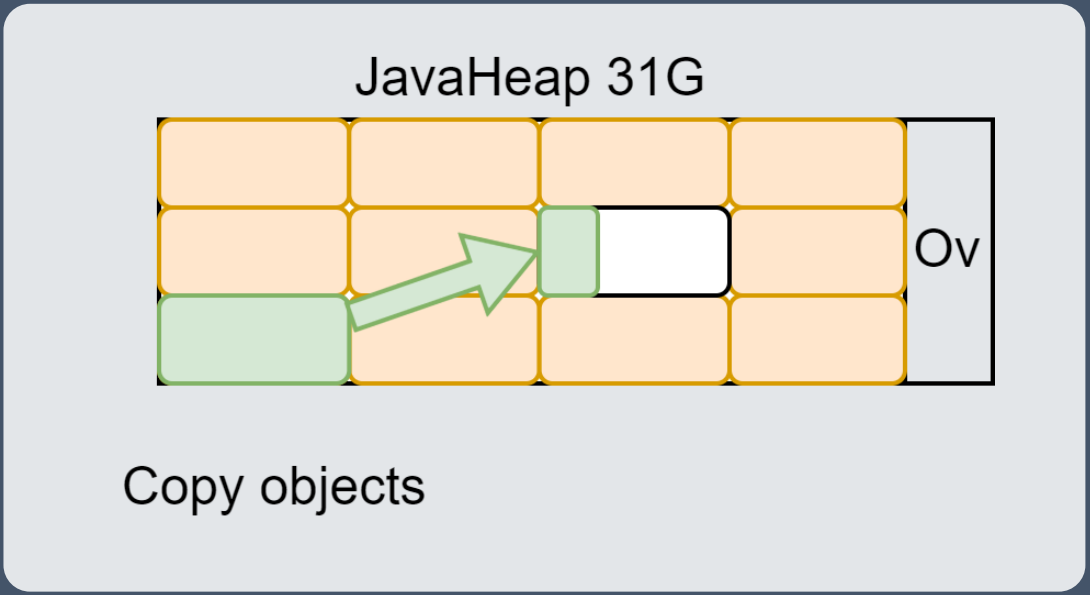



https://docs.azul.com/prime/GC-Log-Analyzer

# Долго выполняется «копирование объектов» (пауза эвакуации)

```
GC(7)      Object Copy (ms):
Min:  9.3, Avg: 43.1, Max: 193.5, Diff: 184.2, Sum: 4866.5,
...
```

Перенос объектов
из одного региона в другой
(синхронная в G1)



# Меньше young-меньше эвакуировать

- Можно увеличить young, чтобы дождаться недостижимости

- У нас не всегда сработает
- Не все объекты в young вовремя станут недостижимыми:
  - putAll – пока всю пачку не положим в page store
  - transaction – ждать от до begin prepare
  - SQL fetch – пока не соберем result set (а если sorted)

```
--XX:G1MaxNewSizePercent = 5
```

# Большое количество потоков

- CPUs=56 или выше - 5/7  - 40 потоков

```
GC(25)     Object Copy (ms):          Min: 175.7, Avg: 194.0, Max:
207.5, Diff: 31.8, Sum: 21920.2, Workers: 113

GC(25)     Termination (ms):          Min: 1008.6, Avg: 1021.1,
Max: 1039.5, Diff: 30.9, Sum: 115388.8, Workers: 113
```

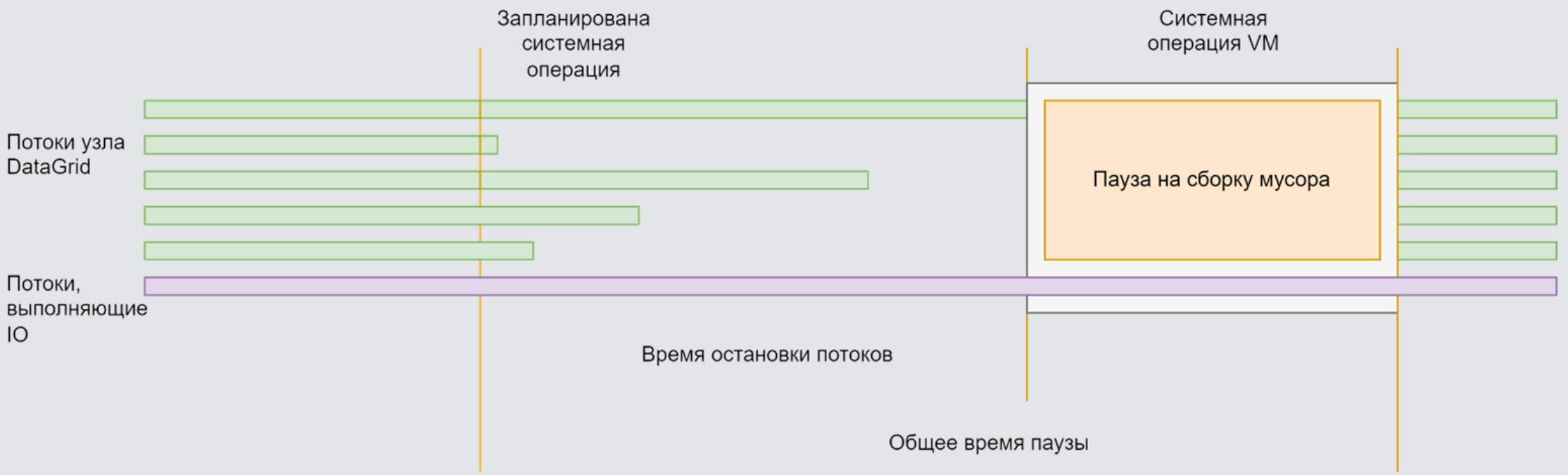
```
-XX:ParallelGCThreads=CPUs/4
```

( 14 потоков )

# Куда смотреть? #11

- В лог самого процесса: ignite.log
- Мониторинг
- Настройки Linux
- Настройки JVM <- **Опции GC: max newsize, par threads**
- Heapdump (если есть)
- Логи GC <- **+object copy, termination, аномалии + GC Log Analyzer**
- Crashdump (если есть)
- Логи линукса

# А все ли паузы в GC?




Остановка произойдет только в определенных точках

# На сцену выходит -Xlog:safepoint

```
Total time for which application threads were stopped:
0.2019948 seconds, Stopping threads took: 0.0004585 seconds
```

```
Total time for which application threads were stopped:
2.6714757 seconds, Stopping threads took: 2.6679943 seconds
```

- Не обычная пауза, время остановки потоков сильно больше vmop

# Как отлаживать

- -XX:+SafepointTimeout
- -XX:SafepointTimeoutDelay=500

```
# SafepointSynchronize::begin: Timeout detected:
# SafepointSynchronize::begin: Timed out while spinning to reach a safepoint.
# SafepointSynchronize::begin: Threads which did not reach the safepoint:
# "kafka-producer-network-thread | producer-reliability-main-sender" #61
daemon prio=5 os_prio=0 cpu=131.89ms elapsed=1568.90s
tid=0x00007f8aa4f3f180 nid=0x1ee9e1 runnable  [0x0000000000000000]
   java.lang.Thread.State: RUNNABLE

# SafepointSynchronize::begin: (End of list)
```

# Как еще отлаживать

- Встроенными средствами подробнее не получится, нужен нативный
- Async Profiler версии старше 2.0 с опциями
  `--begin SafepointSynchronize::begin --end RuntimeService::record_safepoint_synchronized`
- Async Profiler идет в составе Platform V DataGrid

https://github.com/async-profiler/async-profiler/releases

# Для проектов с low latency

- Отключить -XX:-UseBiasedLocking (с 15-й и так удалили)
  - как одна из рекомендаций, у нас не помогла
- Обновлять JVM
- Не увлекаться большими тред пулами

**Андрей Паньгин — Safepoint — и пусть весь мир подождёт**

https://youtu.be/rthWVvU9gWo

# Если в логике проблем нет

```
Total time for which application threads were stopped:
6.2605075 seconds, Stopping threads took: 4.0258420 seconds
```

```
GC Worker Other (ms):     Min: 1120.0, Avg: 28525.1, Max:
38447.5, Diff: 37327.5, Sum: 1083955.5, Workers: 38
```

- что с другими приложениями в этот момент?
- Sleepydog
- NMON – что с диском
  - если его долго ждем, потоки VM могут выгрузить
  - Если хотим написать gc log, тоже будем ждать

# Паузы – hiccup всего сервера

- Не перегружать диск
  - Снимать бекапы вне нагрузки
  - Ограничитель скорости для снятия бекапа
  - Чаше скидывать состояние памяти (checkpoint)


- SSD или несколько дисковых устройств SSD/ HDD (backups)
  - Убрать GC логи с нагруженного диска

# Куда смотреть? #12

- В лог самого процесса: ignite.log
- Мониторинг <- **Аномалии и корреляции с паузами**
- Настройки Linux
- Настройки JVM <- **Опции GC**: BiasedLocking, SafepointTimeout
- Heapdump (если есть)
- Логи GC <- **+ Safepoint.log**
- Crashdump (если есть)
- Логи линукса + **NMON, проблемы с оборудованием?**
- **console.out/console.err – перенаправлять потоки**

# Еще 5 слайдов

- Которые никуда не попали
- Но обязательно стоит упомянуть

# Java Flight Recorder

- Что делала система перед просадкой производительности

-XX:StartFlightRecording= disk=true,settings=<cfg>/profiling-openjdk11.jfc,maxage=120h,maxsize=10000m,dumponexit=true,

-XX:FlightRecorderOptions= repository=/opt/ignite/logs/diag,maxchunksize=30M

- Обычно много места, в последнюю очередь выгружается

# Черный ящик JVM

что было с сервером в каждый момент

собираем с точностью 30 сек

JMC для анализа https://jdk.java.net/jmc/8/

**Алексей Рагозин –**
**Мастер-класс по Java Mission Control**

**https://youtu.be/wm2JNlaJJ5k**

# Куда смотреть? #13

- В лог самого процесса: ignite.log
- Мониторинг
- Настройки Linux
- Настройки JVM
- Heapdump (если есть)
- Логи GC <- + Safepoint.log
- Crashdump (если есть)
- Логи линукса
- console.out/console.err – перенаправлять потоки
- JFR – открываем в JMC

# Еще несколько опций

которые всегда стоят на серверах

| | |
|---|---|
| -Dfile.encoding=UTF-8 | Всегда предсказуемая кодировка |
| -Djava.net.preferIPv4Stack=true | IPv6 нам не надо |
| -Dnetworkaddress.cache.ttl=-1 | Не удалять записи из DNS |
| -XX:-OmitStackTraceInFastThrow | Чтобы не терять трейсы |
| -XX:+CrashOnStackOverflowError | Из-за баги JDK-8319090 |

# Практика

Параметров много – проектов тоже, начинали

- смотреть глазами
- и менять письмами

Но 1314 кластеров...
и глазами не получится, нужен автоматический контроль

check-parameters-plugin – часть Platform V DataGrid

- Проверяет sysctl + Java options (учитывая версию Java)
- Добавлен в мониторинг

# Выводы

- Здоровый и крепкий сон возможен, если

- Собирать все логи и хранить достаточное время в прошлом
  - Приложение
  - GC + safepoint + classhisto
  - Linux (messages+dmesg)
  - Nmon
- Писать и хранить heapdump, crashdump, JFR
- Планировать capacity через
  - Мониторинг
  - Нагрузочные тесты
- Собирать и контролировать, автоматически тиражировать
  - опции VM
  - опции Linux - sysctl
- Завести свой чек-лист для production
- Показать этот и все упомянутые доклады коллегам-джунам

# Спасибо

Задать вопросы про Apache Ignite
user@ignite.apache.org
https://t.me/RU_Ignite

Дмитрий Павлов
dpavlov@apache.org
https://t.me/dspavlov

Про устройство БД https://www.youtube.com/@db.podcast

Обучающие материалы по БД https://t.me/db_links

Список параметров JVM, sysctl и ссылок
https://gitverse.ru/dpavlov/ignite_checklist?tab=readme

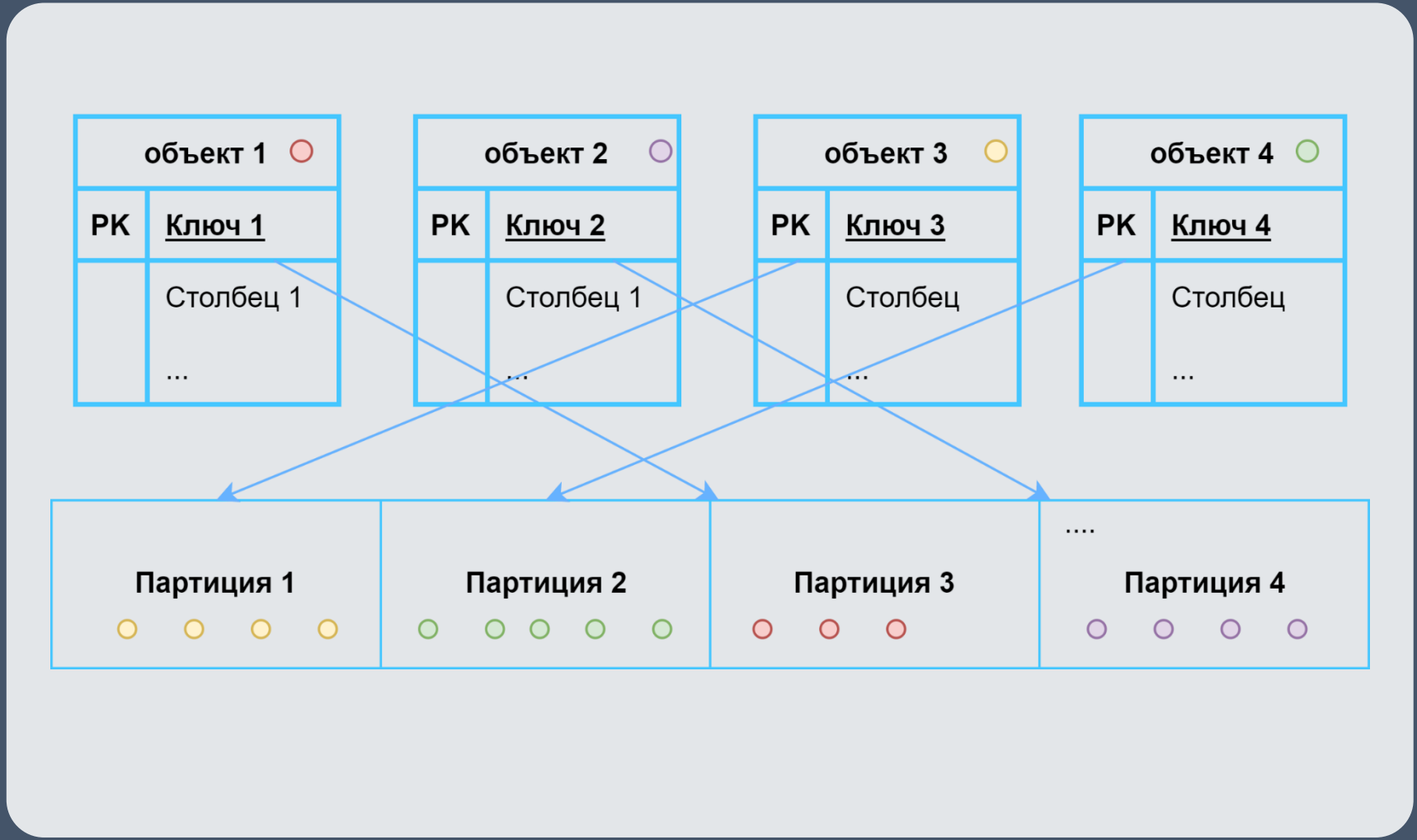
# Appendix A. Надежность в продукте

## Бекапы и резервы

- Данных внутри кластера
- Кластеров
- Задействуем HDD + снимки данных

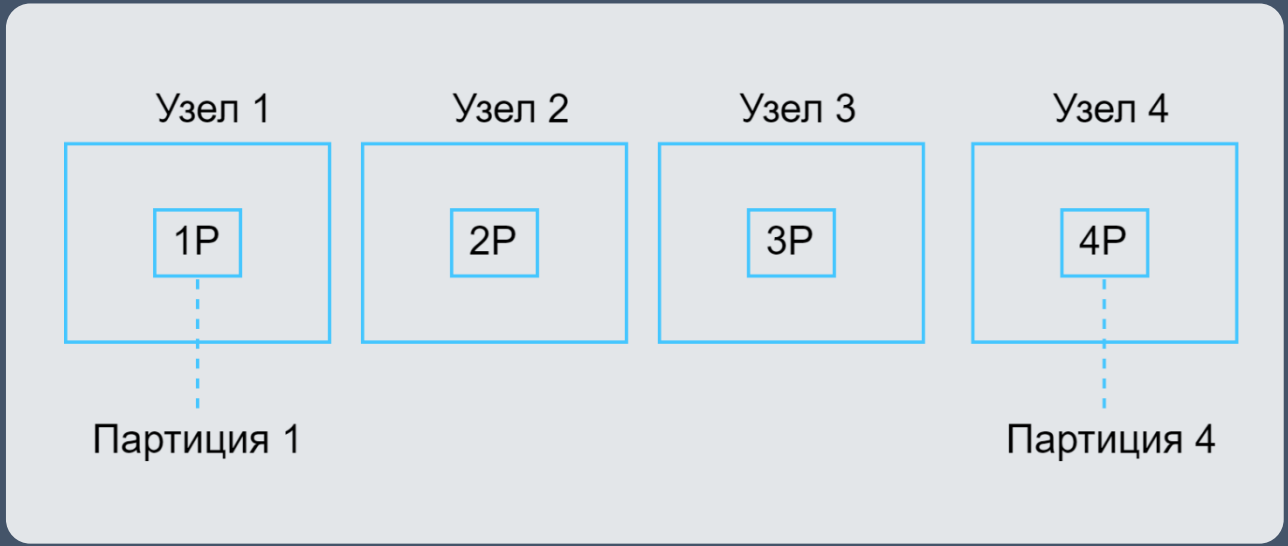
# Хранение данных

- DataGrid распределяет каждый ключ в свою партицию
- Партиции получаются примерно равными частями общего набора данных

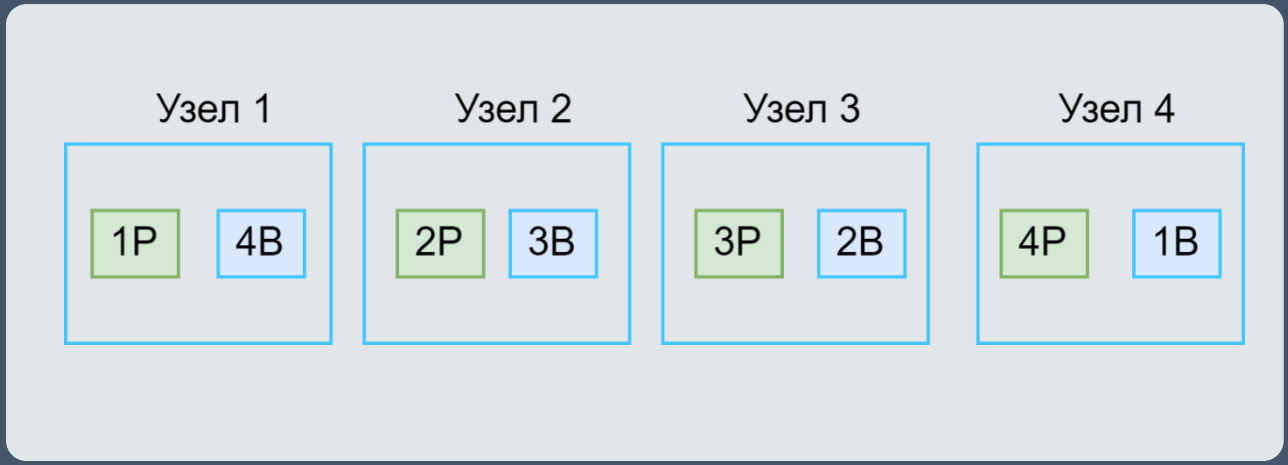


# Резервирование

- Патрициями распределяются между узлами
- Перемещается независимо друг от друга
- В примере 4 узла и 4 партиции (в реальности – 1024- default)

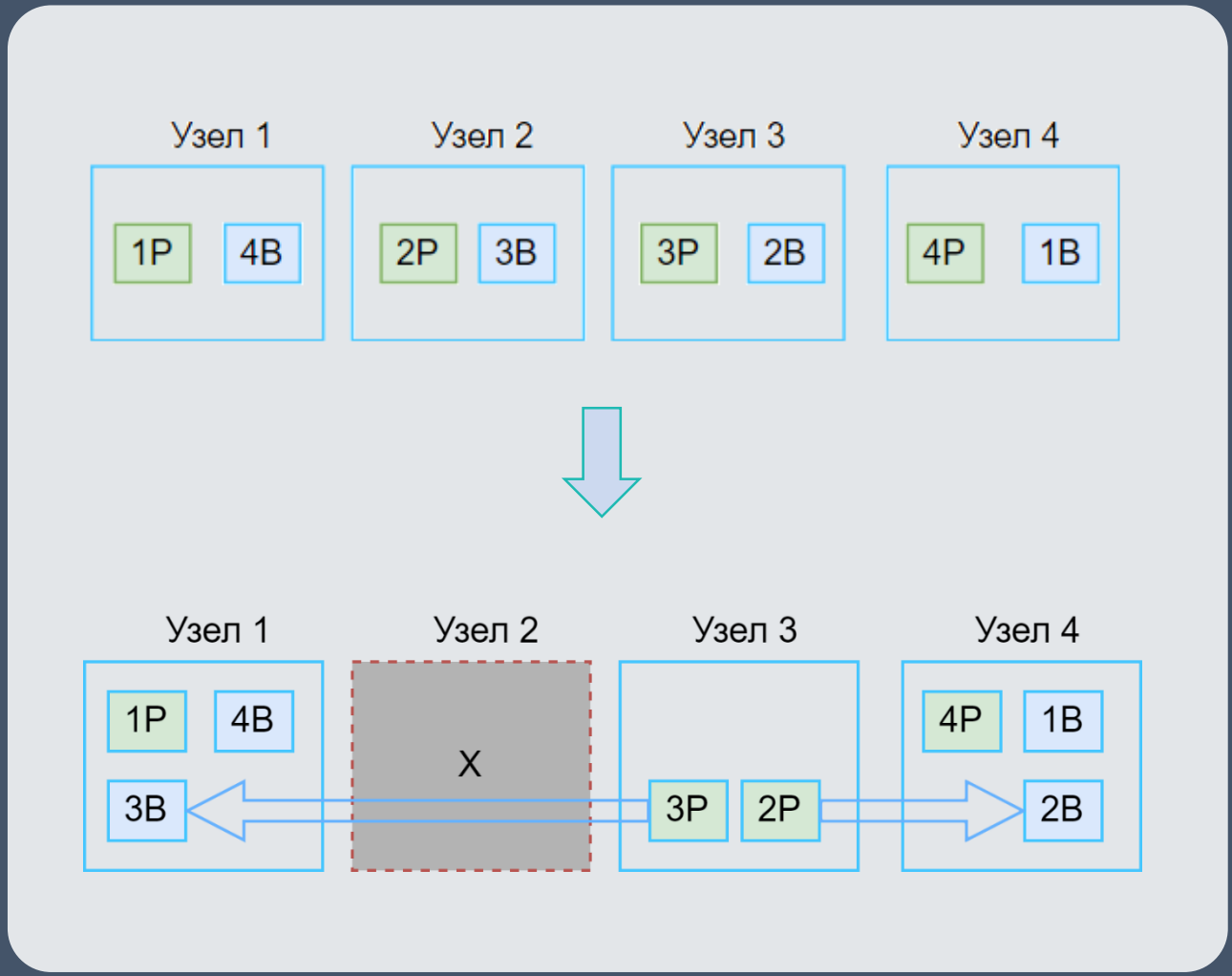



- Можно хранить данные в нескольких копиях
- Тогда есть (P)rimary (B)ackup



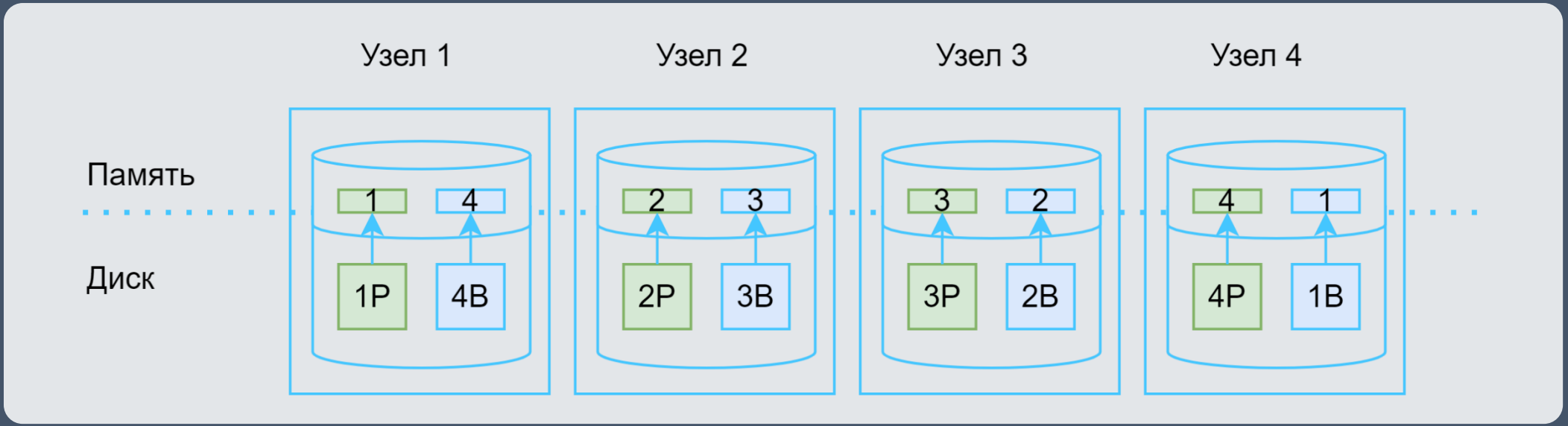
# Восстановление после сбоев

- В случае изменения топологии - минимизация перемещений
- В случае потери узла – восстановление нужного количества копий
- Теряем #replfactor-1 узел без остановки сервиса
- На практике обычно replfactor=2, редко 3



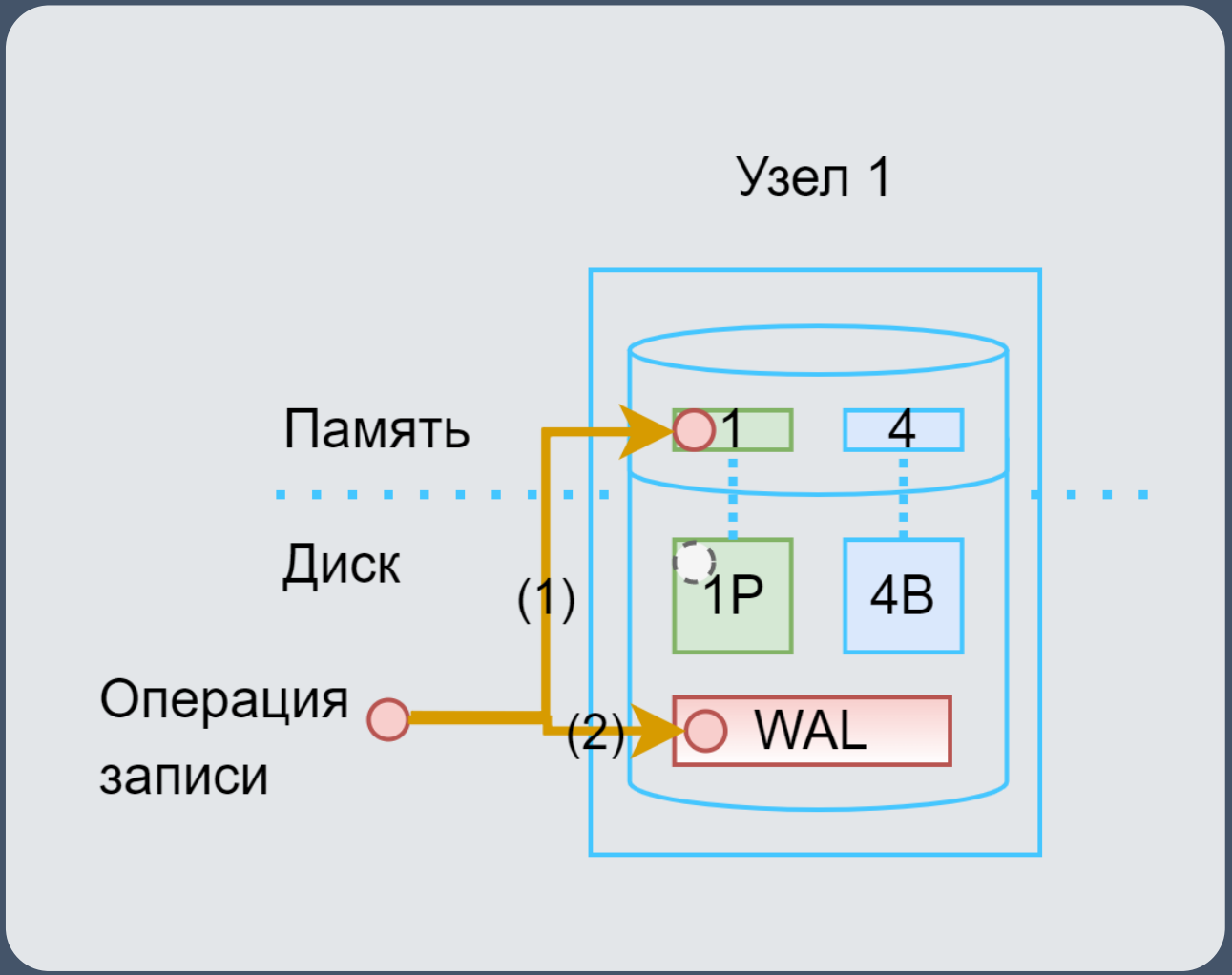
# Хранение на диске

- Persistence (встроенный)
- На практике обычно – включен в большинстве проектов
- Часть данных в RAM – полный набор на диске
- Если временно выводим узел - не надо его потом прогревать



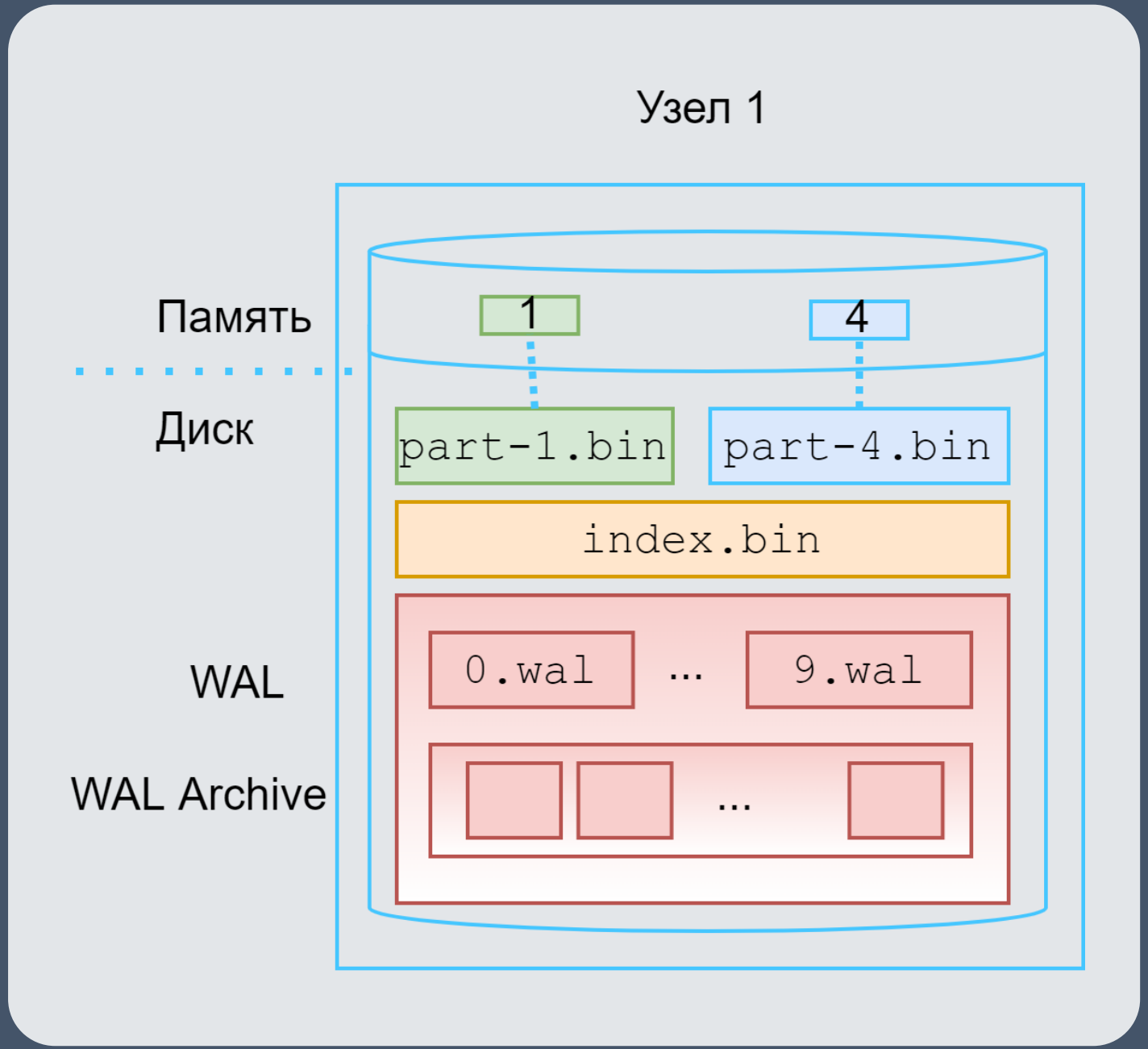
# Операции при включенном хранении данных

- Применяем изменения на узле в памяти
- Пишем в  WAL – Write Ahead Log – журнал транзакций
- Потом запишется в основное хранилище



# Используемые файлы

- File per partition
- Index - общий
  - индексы для тех партиций, которые есть на узле
- WAL:
  - Сегменты WAL – конечное число
  - Архив WAL – то что вытеснено из основного work, не нужно его удалять

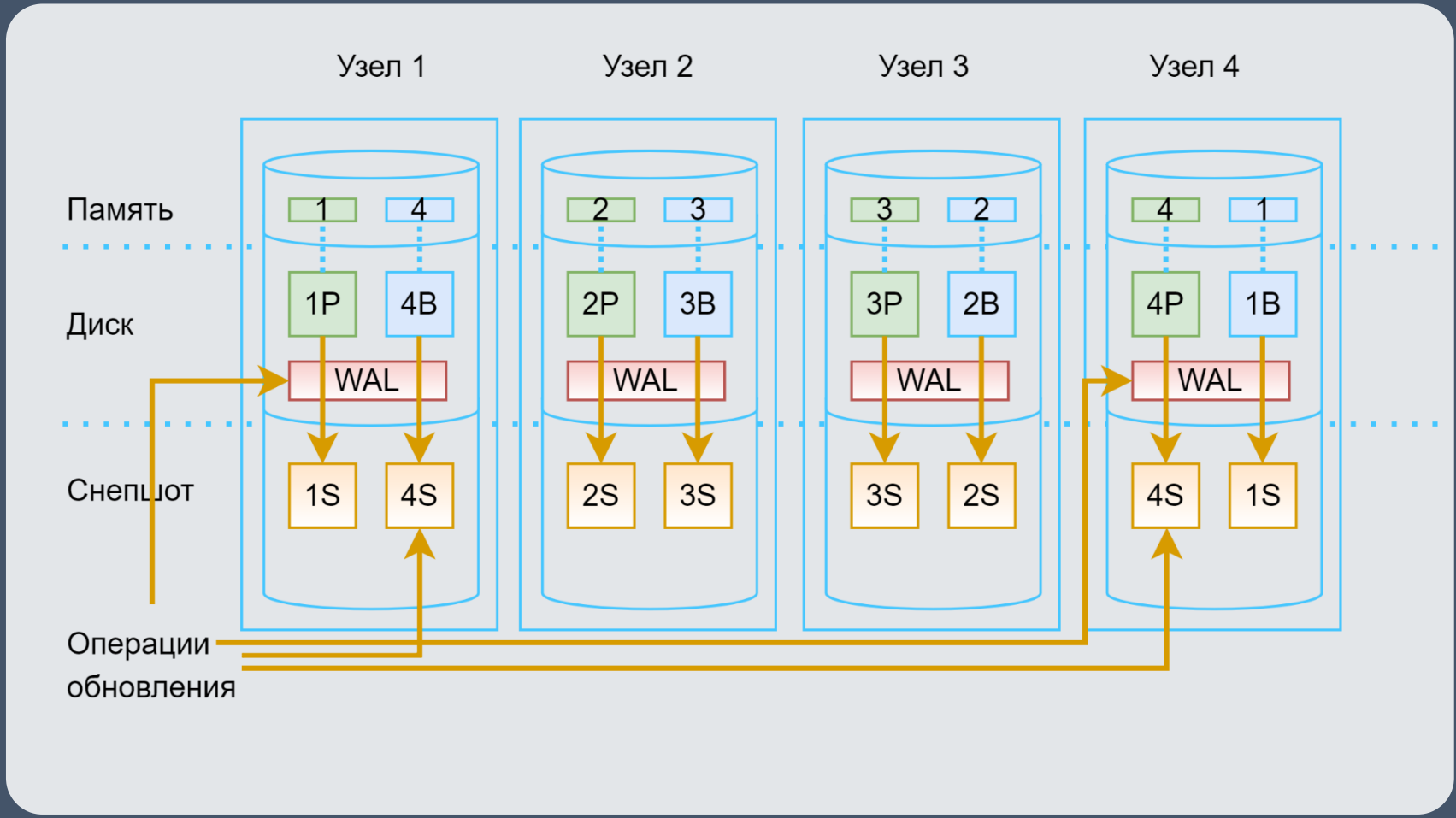


# A.1 Снимки данных

Максим Музафаров - Снятие и восстановление резервной копии в Apache Ignite
https://youtu.be/7jNliqwDJtQ

# Снимки (Ignite-snapshots, обычно-backups)

- Настроено снятие снимков раз в сутки - вне периода активности
- Операции во время снятия снимка попадают в WAL и в снимок напрямую

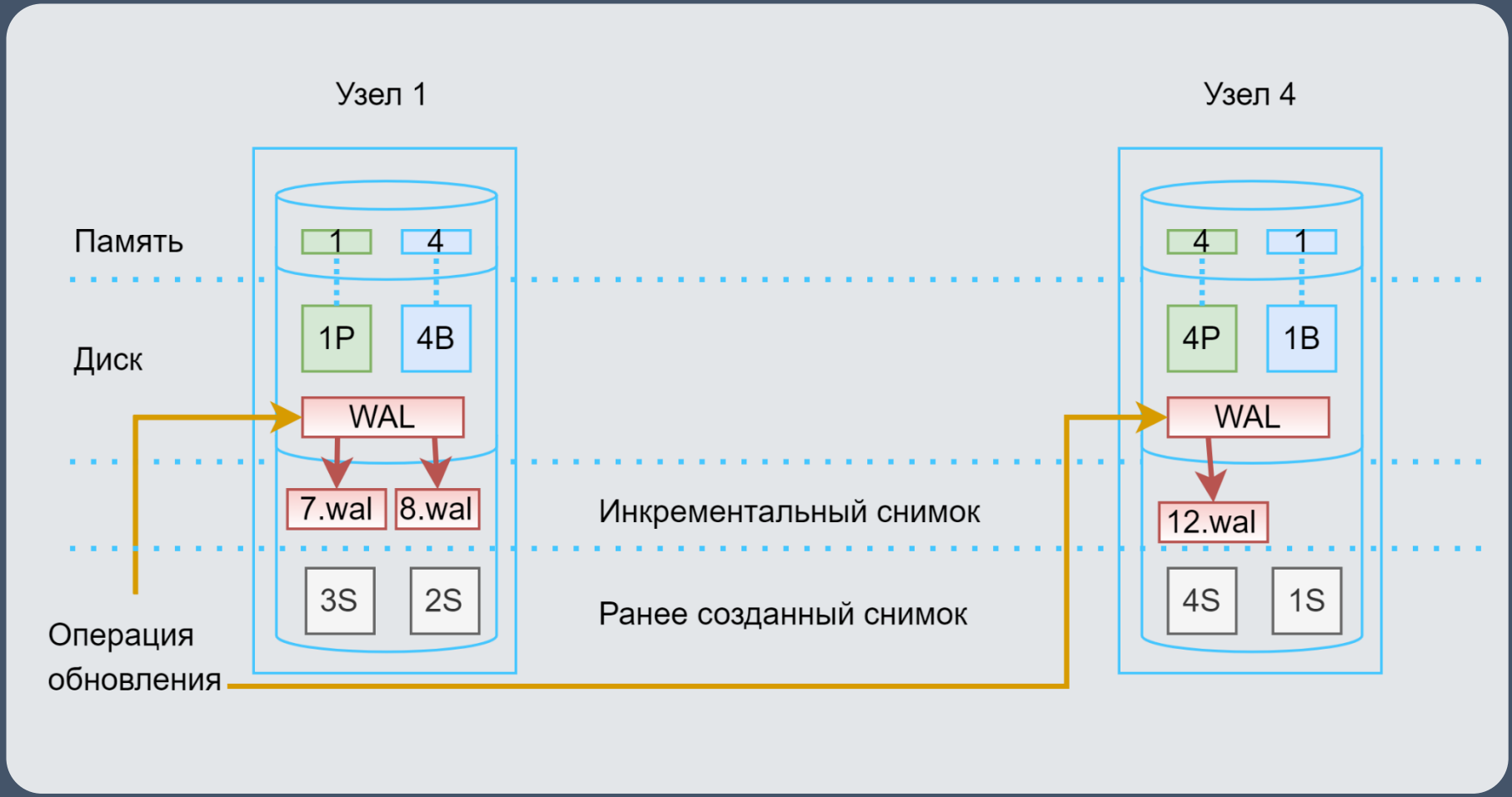


# A.2 Инкрементальные снимки

**Максим Тимонин - Алгоритм инкрементальных бэкапов в Apache Ignite**
https://youtu.be/EvzNBr6toWI

# Инкрементальные снимки

- Только дельты от последнего снимка
- Используем сегменты WAL, которые изменились за это время
- Пока на пути в прод, но все тесты прошли



# Appendix B. Межкластерная репликация

**Бекапы и резервы**

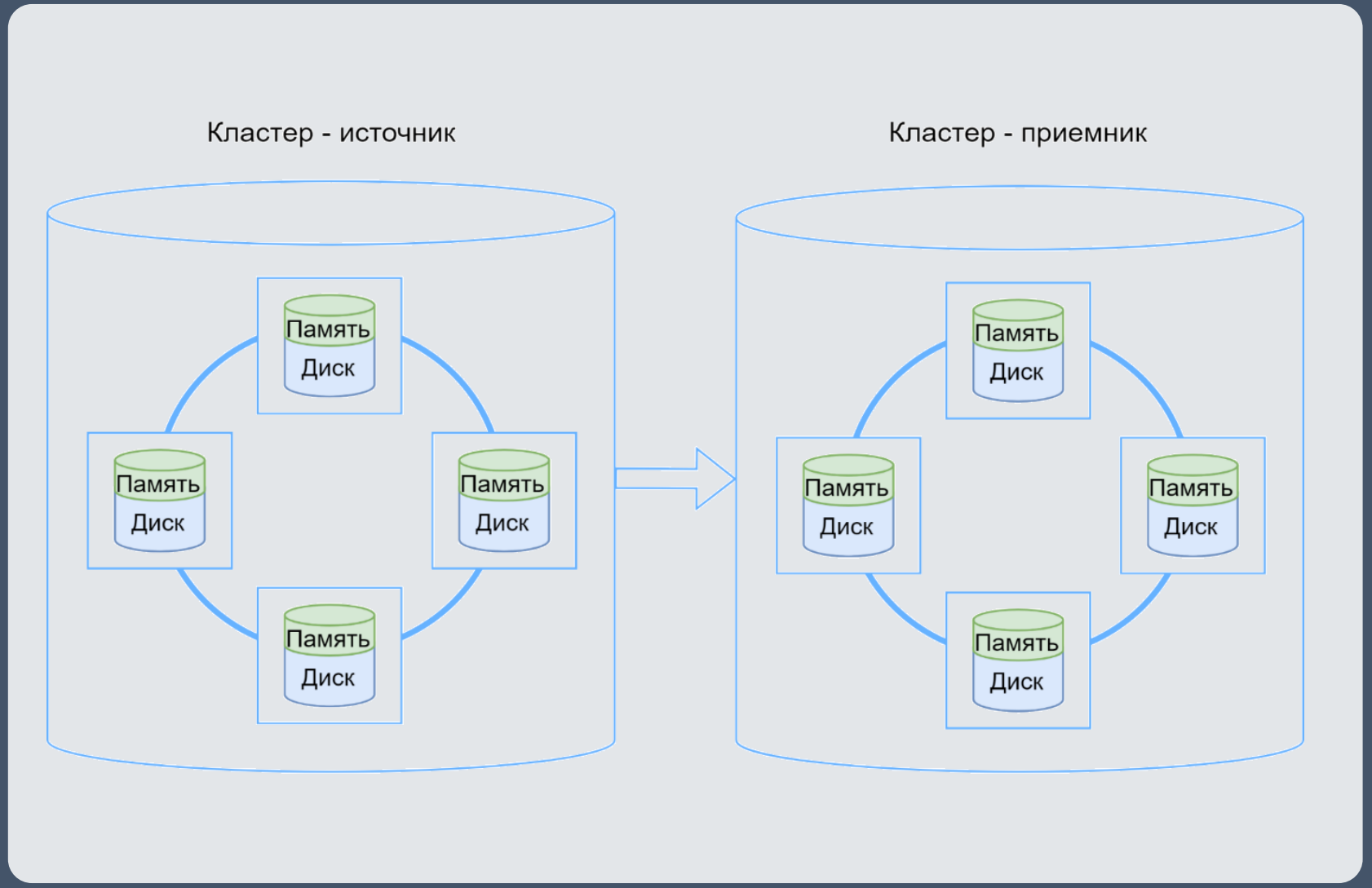
- Через Ignite
- Через Kafka/PlatformV Corax

Николай Ижиков –
Apache Ignite теперь с CDC!

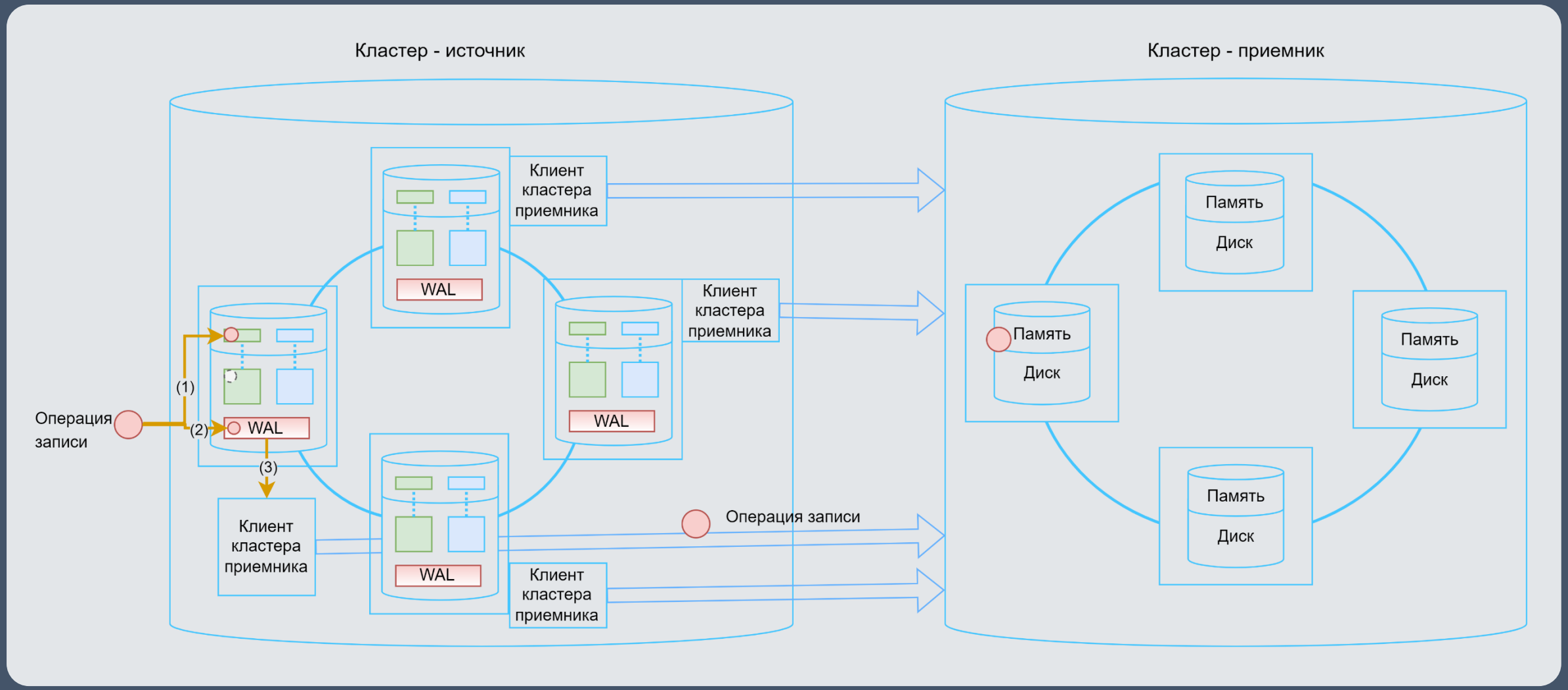
https://youtu.be/Xx8ErhM6U9Q

# Межкластерная репликация

- Обычно 2 кластера, реже – 3
- и Active-Acitive, и Active-Standby
- Встроено в Ignite - лучше последние версии: Ignite 2.17+ и PV DataGrid 15.0.2+



# Прямая: Ignite2Ignite



# Через брокера: Ignite2Kafka

- Независимые версии продукта на каждом кластере
- Kafka To Ignite – можно масштабировать количеством процессов приемников