Эффективная работа с файлами для Java-разработчиков

Dmitriy Govorukhin
dmitriy.govorukhin@gmail.com

GridGain

План

- Устройства хранения

План

- Устройства хранения
- Оптимизация на уровне ОС

План

- Устройства хранения
- Оптимизация на уровне ОС
- Java disk API
  - FileChannel под капотом
  - что можно ускорить на стороне приложения
  - типичные ошибки при работе с FileChannel

Типы нагрузки

- sequential reading
  - копирование
  - восстановление из бэкапа

Типы нагрузки

- sequential reading
  - копирование
  - восстановление из бэкапа
- append-only write
  - журналирование
  - создание бэкапа, архивирование

Типы нагрузки

- sequential reading
  - копирование
  - восстановление из бэкапа
- append-only write
  - журналирование
  - создание бэкапа, архивирование
- random read
  - раздача нескольких файлов в несколько источников

Типы нагрузки

- sequential reading
  - копирование
  - восстановление из бэкапа
- append-only write
  - журналирование
  - создание бэкапа, архивирование
- random read
  - раздача нескольких файлов в несколько источников
- random write
  - прием нескольких файлов из нескольких источников

Типы нагрузки

- sequential reading
  - копирование
  - восстановление из бэкапа
- append-only write
  - журналирование
  - создание бэкапа, архивирование
- random read
  - раздача нескольких файлов в несколько источников
- random write
  - прием нескольких файлов из нескольких источников
- random read/write

Типы нагрузки

- sequential reading
  - копирование
  - восстановление из бэкапа
- append-only write
  - журналирование
  - создание бэкапа, архивирование
- random read
  - раздача нескольких файлов в несколько источников
- random write
  - прием нескольких файлов из нескольких источников
- random read/write
  - работа БД в смешаном режиме работы (запросы чтение/запись/модификация)

benchmark

	1GB
seqRead	0.536
randomRead	0.624
appendWrite	1.243
randomWrite	1.691

Скорость обработки файла (сек) в зависимости от размера

benchmark

	1GB	2GB
seqRead	0.536	1.133
randomRead	0.624	1.346
appendWrite	1.243	2.848
randomWrite	1.691	4.493

Скорость обработки файла (сек) в зависимости от размера

benchmark

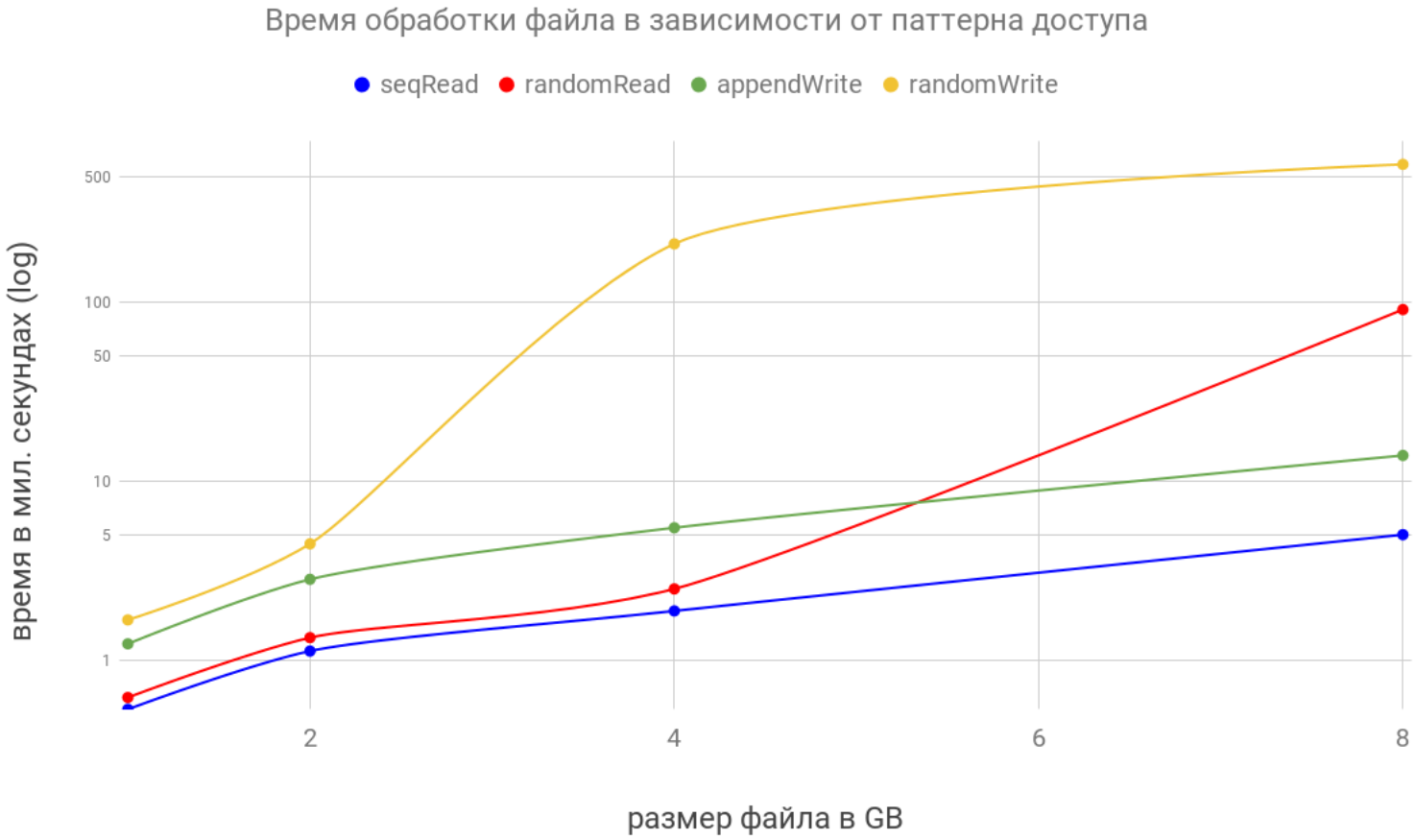
	1GB	2GB	4GB
seqRead	0.536	1.133	1.897
randomRead	0.624	1.346	2.518
appendWrite	1.243	2.848	5.524
randomWrite	1.691	4.493	211.99

Скорость обработки файла (сек) в зависимости от размера

benchmark

	1GB	2GB	4GB	8GB
seqRead	0.536	1.133	1.897	5.05
randomRead	0.624	1.346	2.518	91.161
appendWrite	1.243	2.848	5.524	13.987
randomWrite	1.691	4.493	211.99	588.857

Скорость обработки файла (сек) в зависимости от размера


Время обработки файла в зависимости от паттерна доступа
seqRead
randomRead
appendWrite
randomWrite
время в мил. секундах (log)
500
100
50
10
5
1
2
4
6
8
размер файла в GB

GridGain

Устройства хранения

HDD

+ Seq read
+ Append write
- Random read
- Random write

Устройства хранения

HDD

+ Seq read
+ Append write
- Random read
- Random write

Page - единица чтения/записи block device (4/8/16 kb)

Устройства хранения

HDD

+ Seq read
+ Append write
- Random read
- Random write

SSD

+ Seq read
+ Append write
+ Random read
- Random write

Page - единица чтения/записи block device (4/8/16 kb)

Устройства хранения

HDD

- \+ Seq read
- \+ Append write
- \- Random read
- \- Random write

SSD

- \+ Seq read
- \+ Append write
- \+ Random read
- \- Random write

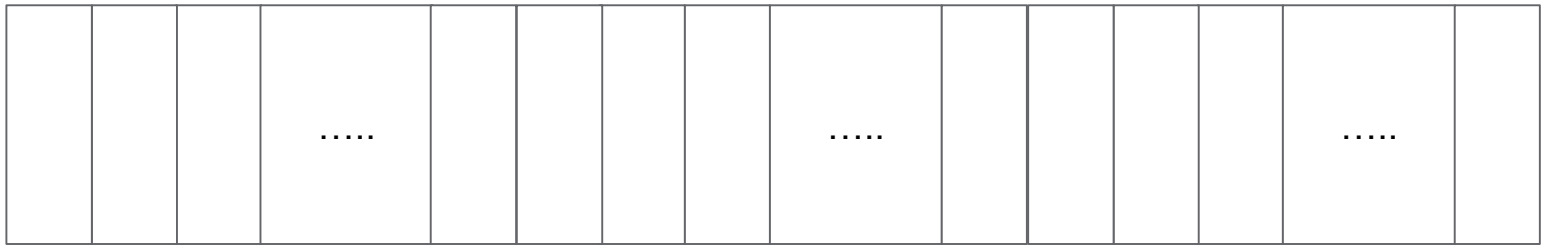
(NVMe, Intel Optane etc)

- \+ Seq read
- \+ Append write
- \+ Random read
- \+ Random write ???

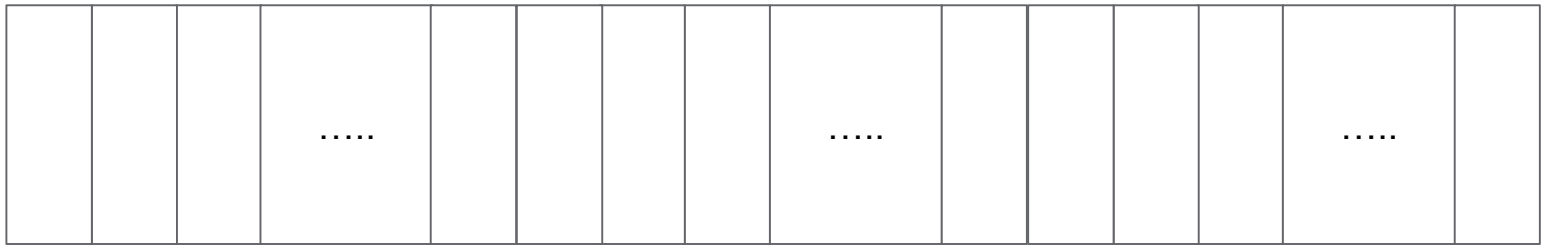
Page - единица чтения/записи block device (4/8/16 kb)

NAND flash: Блочная структура

NAND flash: Блочная структура

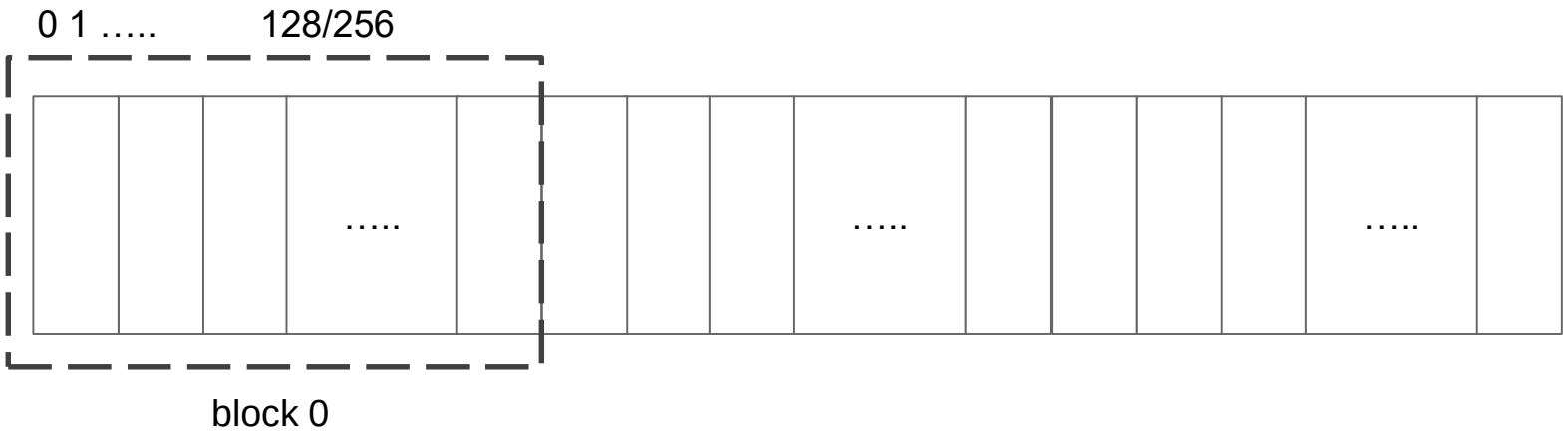


NAND flash: Блочная структура



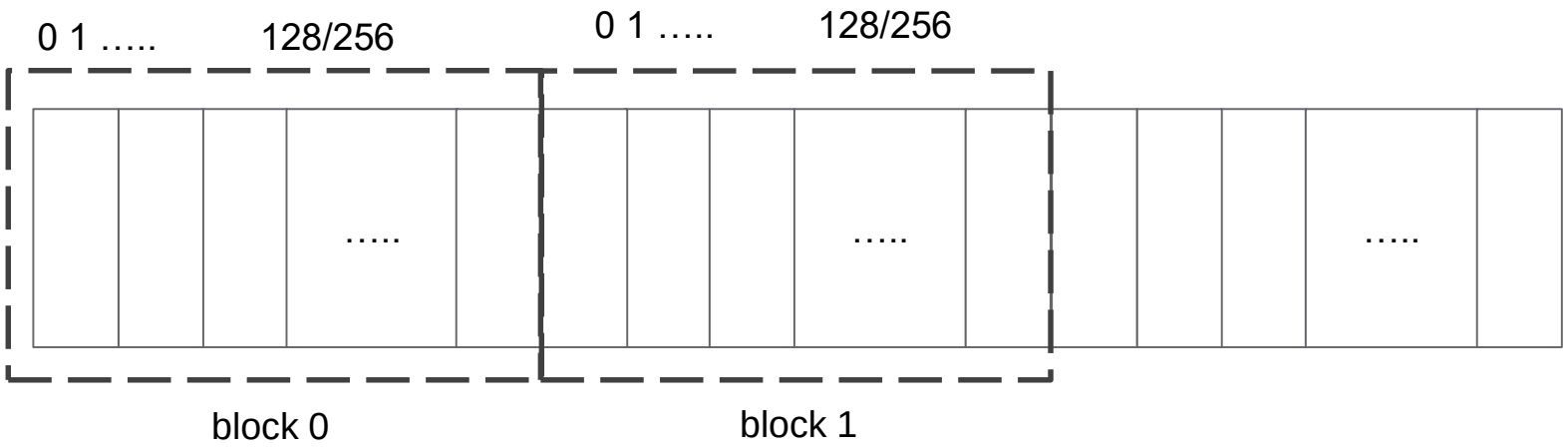
- Page - единица хранения данных (4k/8k/16k)

NAND flash: Блочная структура



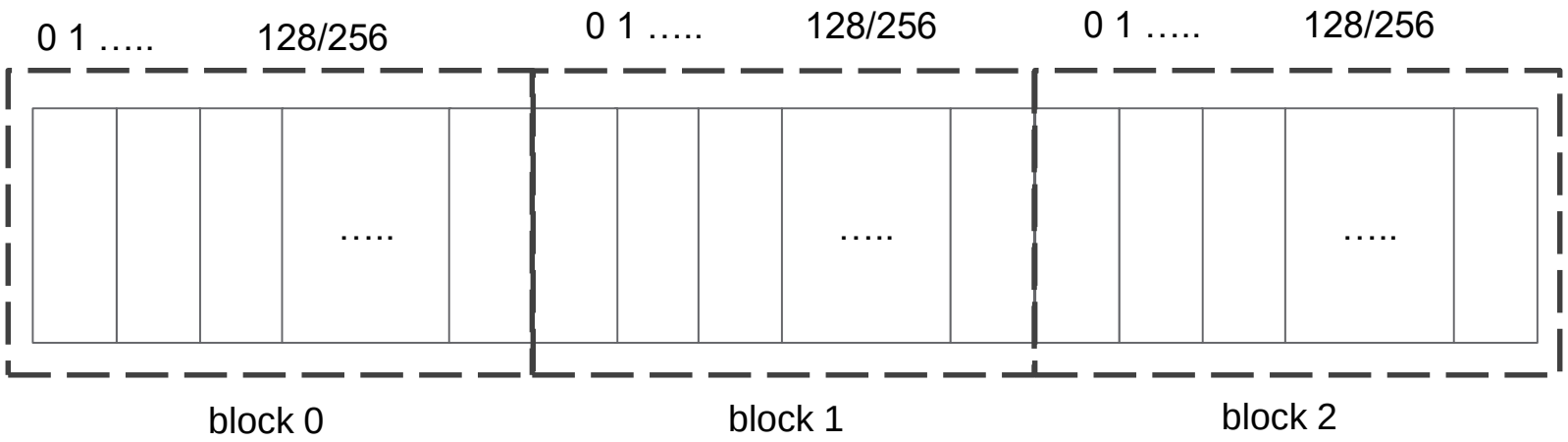
- Page - единица хранения данных (4k/8k/16k)
- Block - группа страницы из (128/256)

NAND flash: Блочная структура



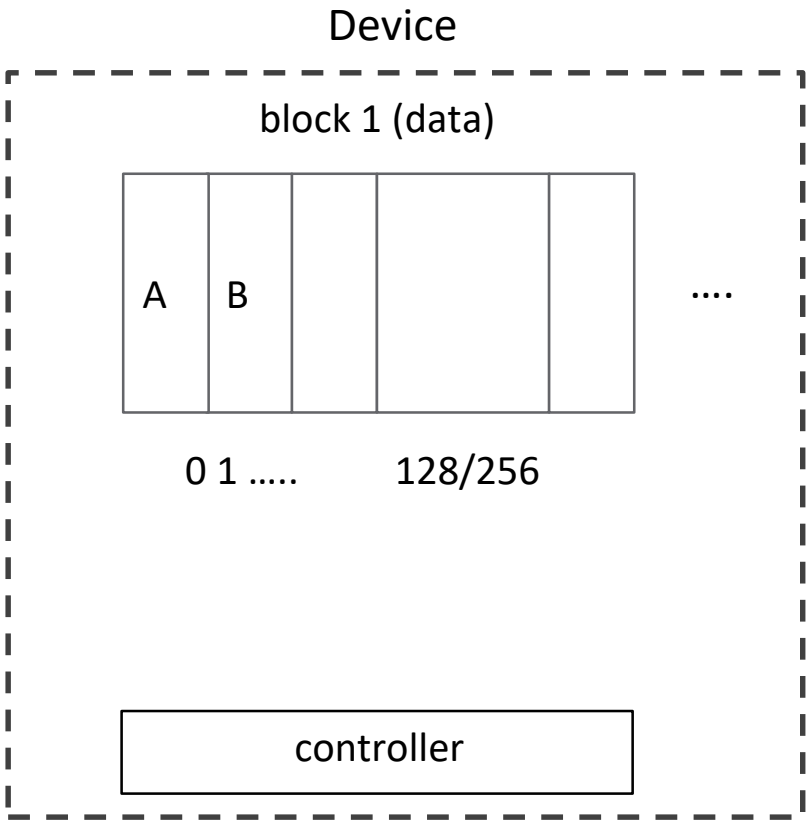
- Page - единица хранения данных (4k/8k/16k)
- Block - группа страницы из (128/256)

NAND flash: Блочная структура

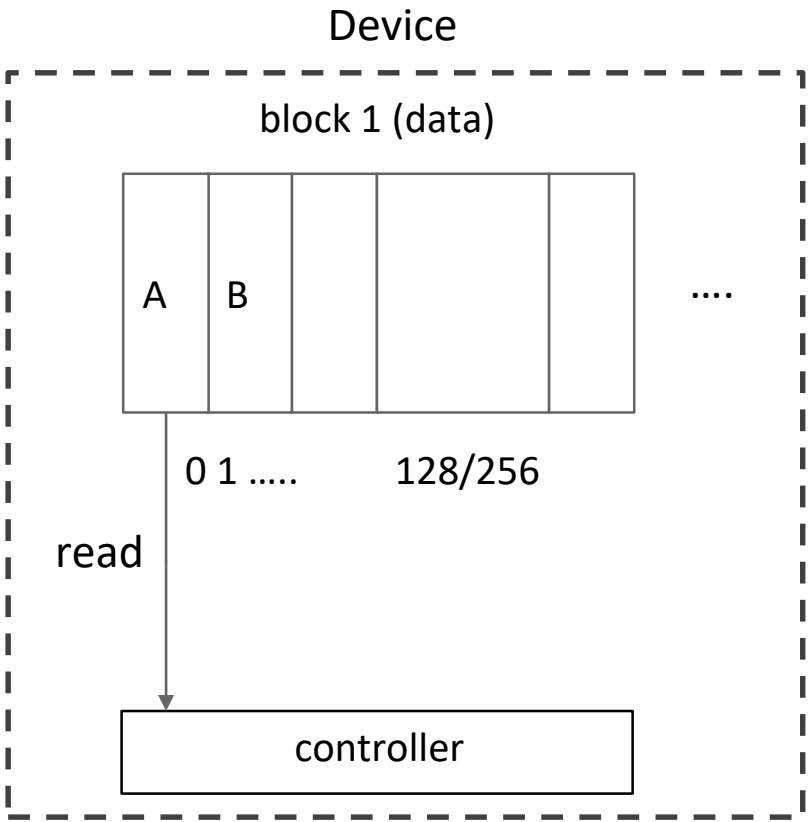


- Page - единица хранения данных (4k/8k/16k)
- Block - группа страницы из (128/256)

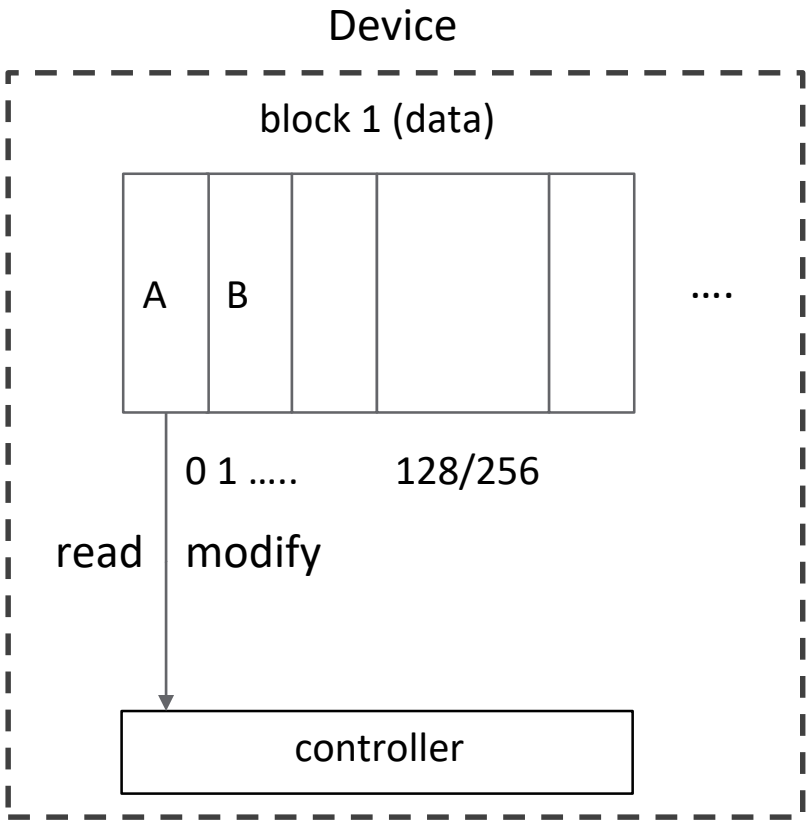
NAND flash: Запись



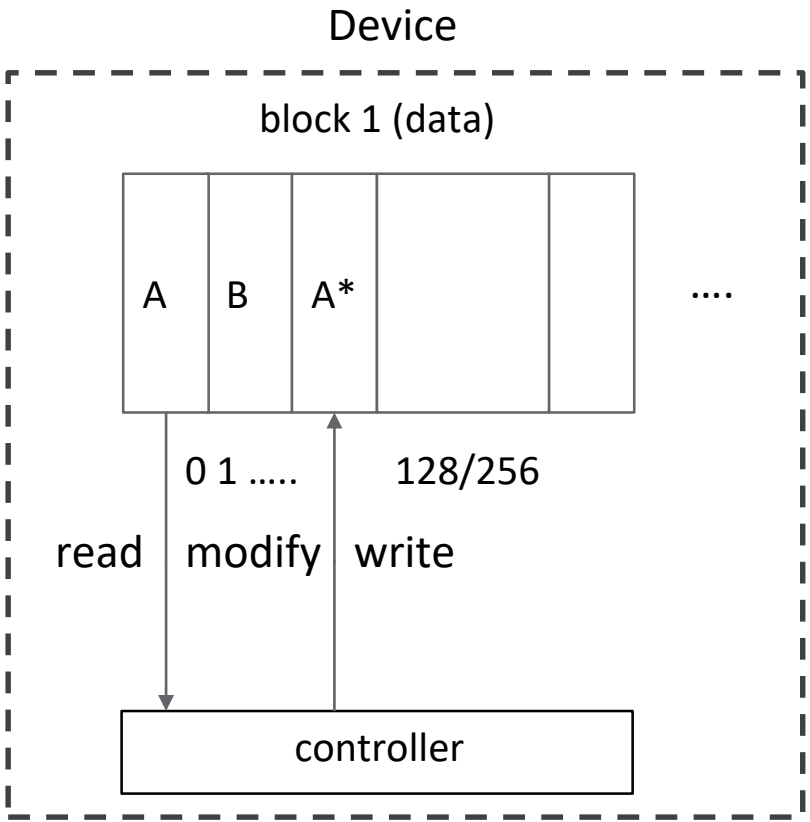
NAND flash: Запись



NAND flash: Запись

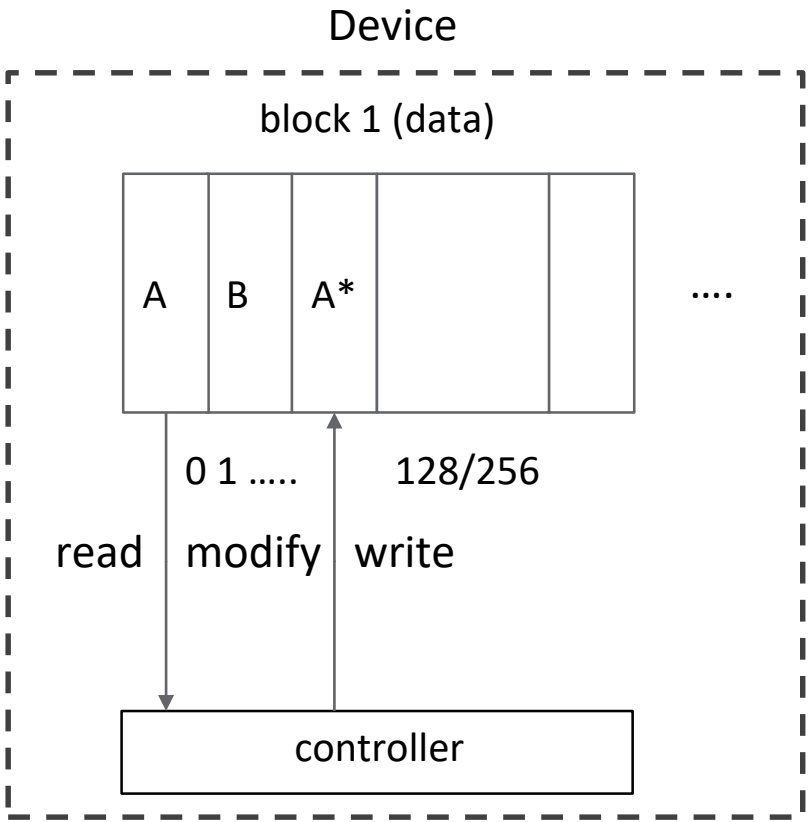


NAND flash: Запись



GridGain

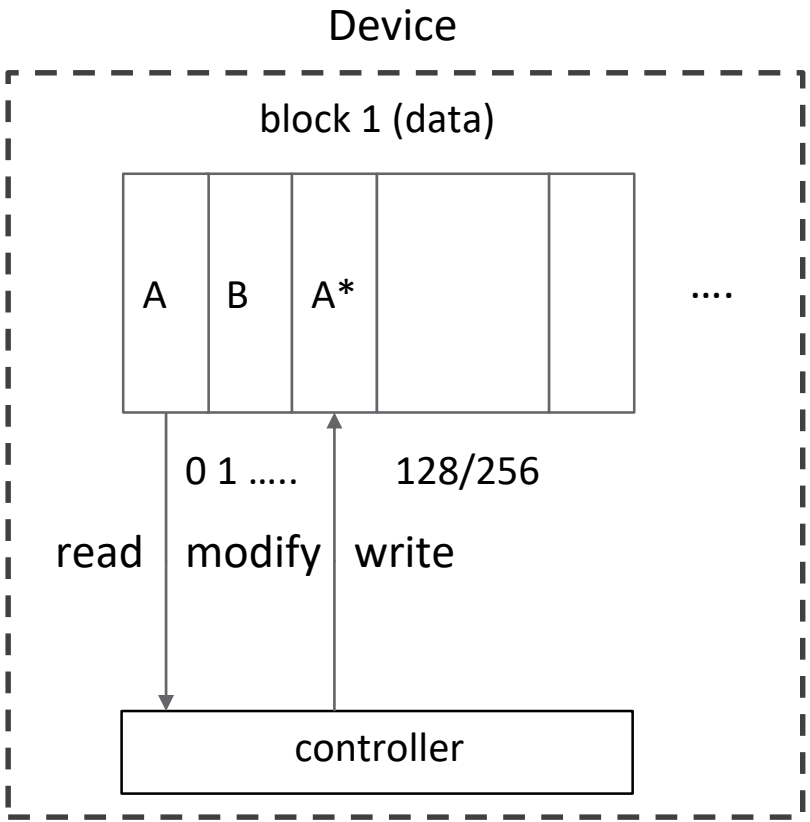
NAND flash: Запись



- Запись возможна только страницей целиком (изменения 1 байта приведет к перезаписи целой страницы, read-modify-write,write amplification)

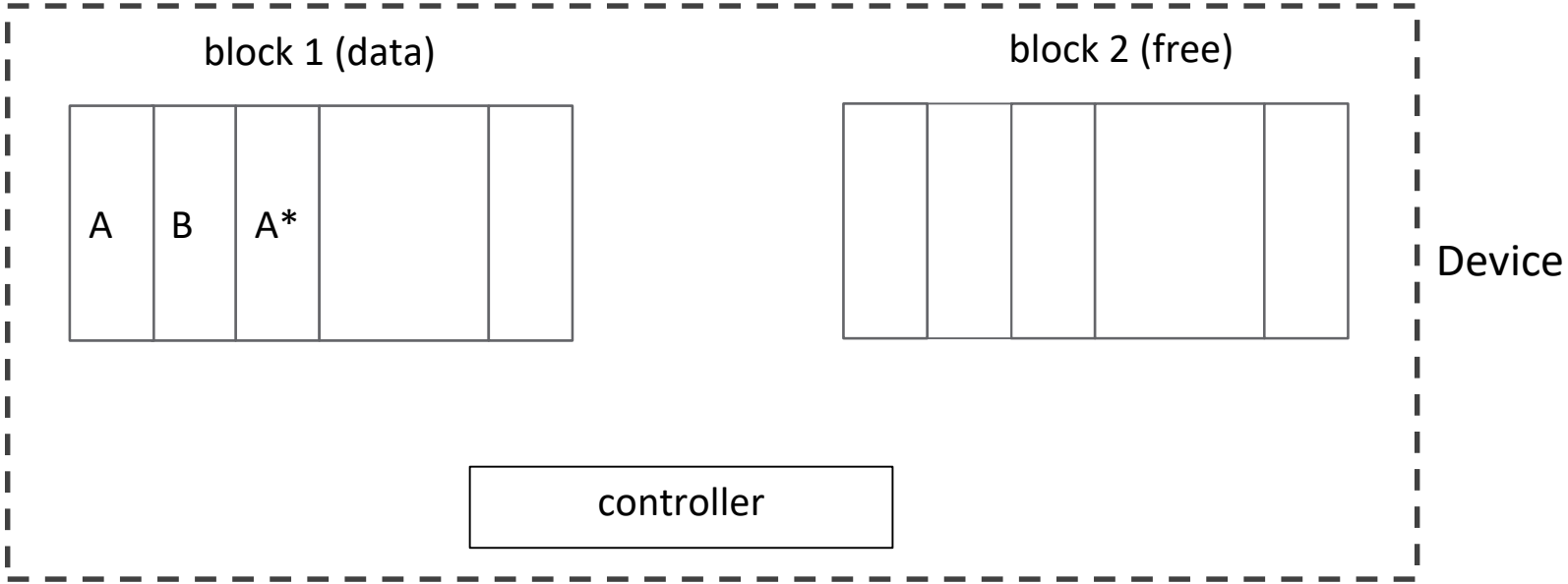
GridGain

NAND flash: Запись

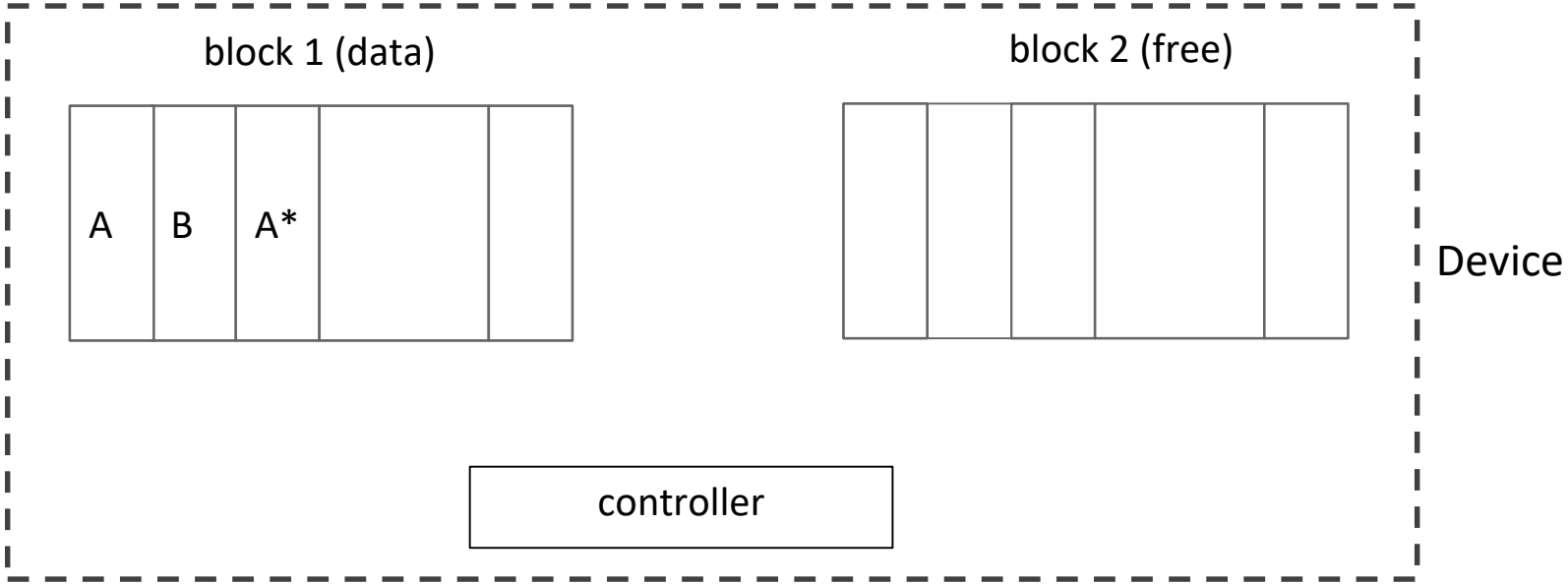


- Запись возможна только страницей целиком (изменения 1 байта приведет к перезаписи целой страницы, read-modify-write,write amplification)
- Запись возможна только в чистые страницы

NAND flash: Очистка

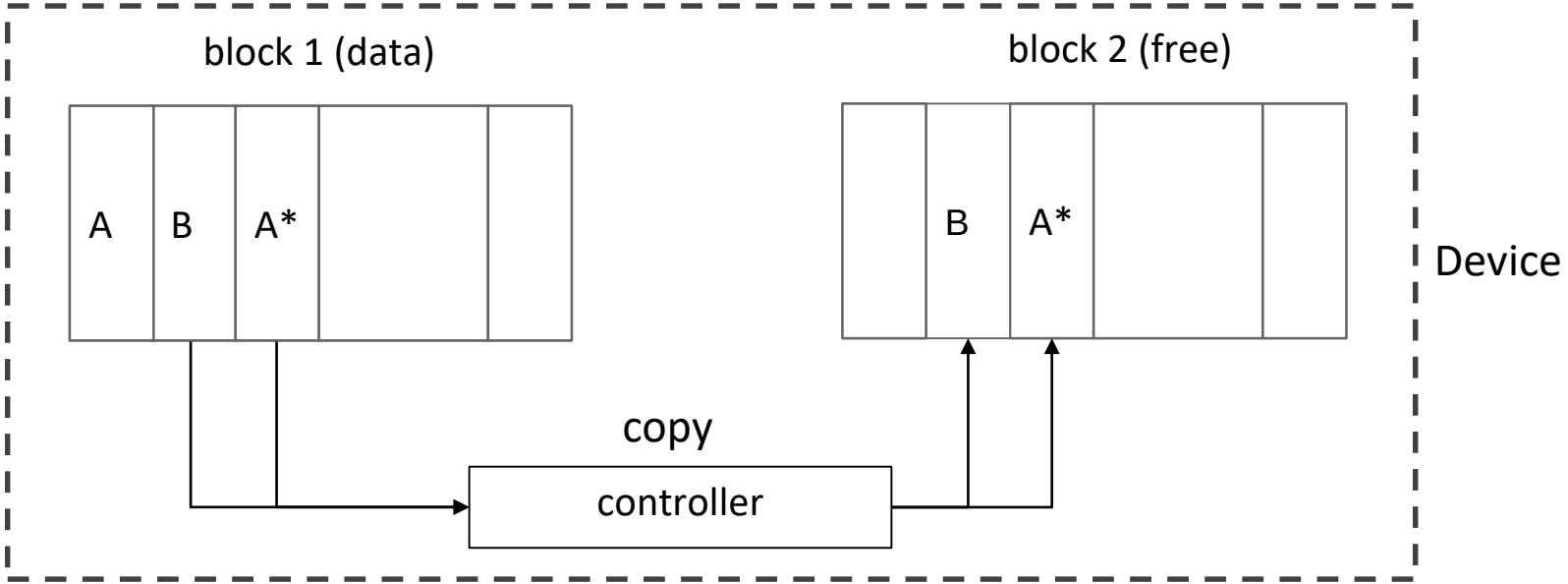


NAND flash: Очистка



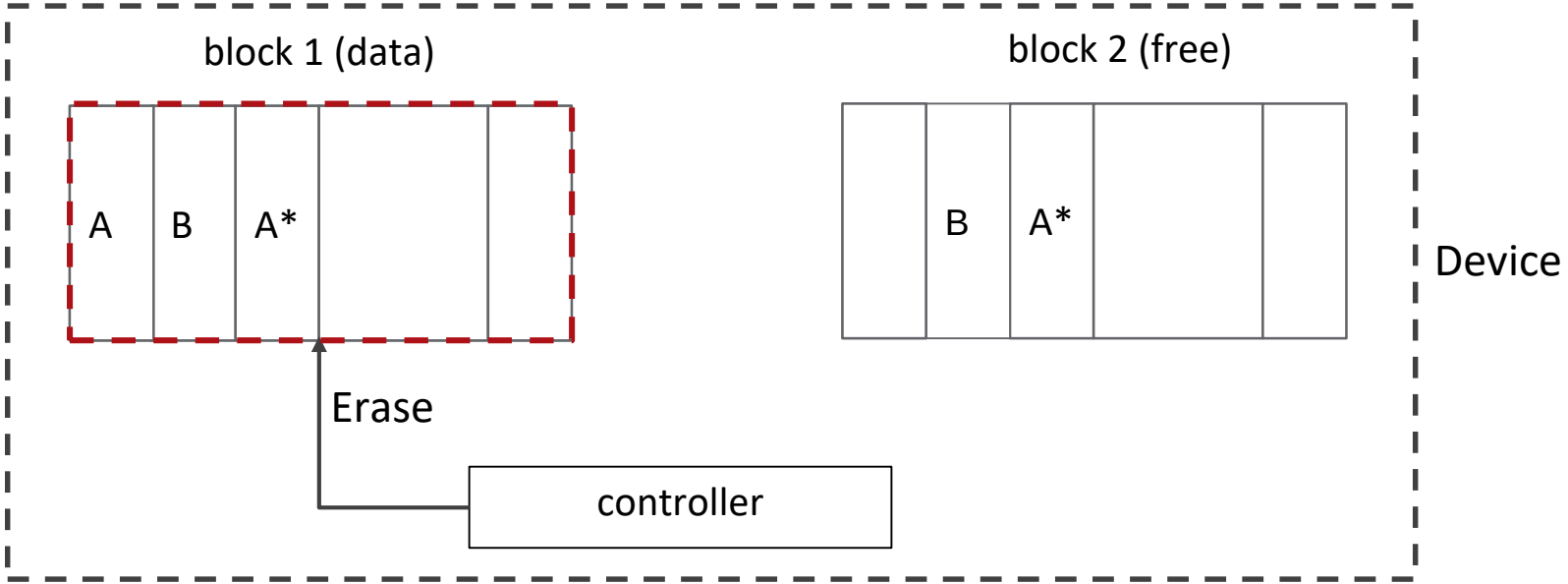
- Очистка возможна только блоками
- GC - это удаление старых версий страниц через копирование и очистку блоков

NAND flash: Очистка



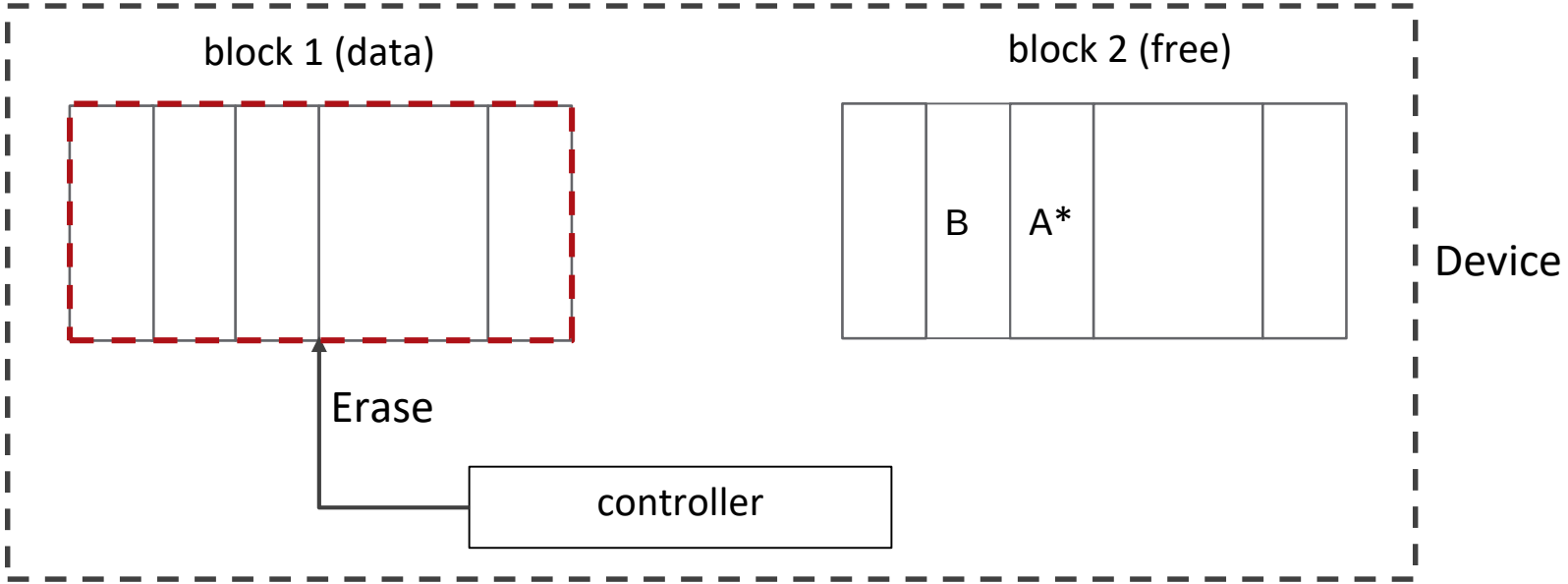
- Очистка возможна только блоками
- GC - это удаление старых версий страниц через копирование и очистку блоков

NAND flash: Очистка



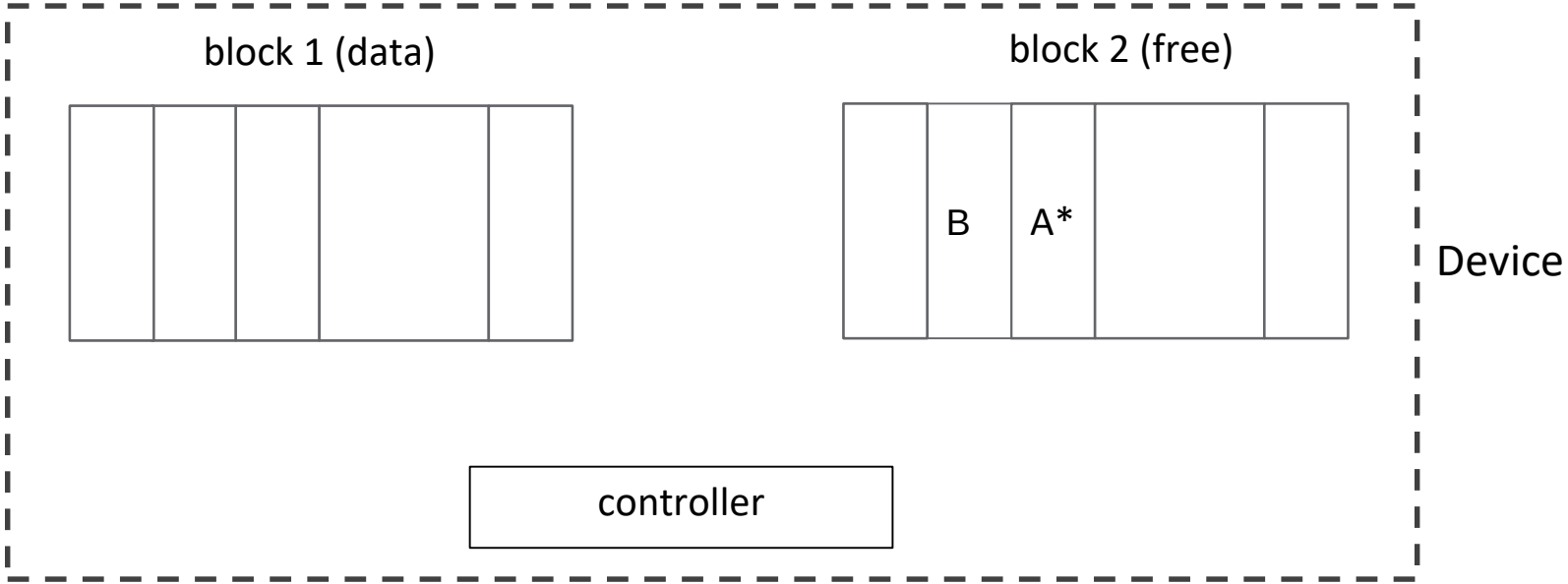
- Очистка возможна только блоками
- GC - это удаление старых версий страниц через копирование и очистку блоков

NAND flash: Очистка



- Очистка возможна только блоками
- GC - это удаление старых версий страниц через копирование и очистку блоков

NAND flash: Очистка



- Очистка возможна только блоками
- GC - это удаление старых версий страниц через копирование и очистку блоков

Что важно помнить

- Диски работают с страницами (пишите всегда кратно размеру страницы, это максимально эфективный способ работы)

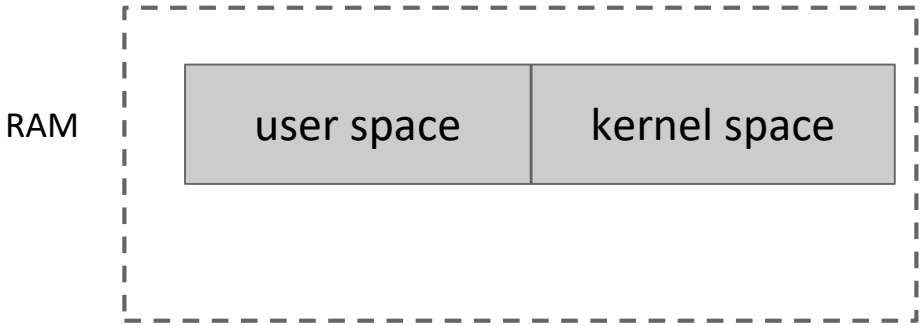
Что важно помнить

- Диски работают с страницами (пишите всегда кратно размеру страницы, это максимально эфективный способ работы)
- Если вы пишете меньше размера страницы, вы повышаете износ NAND flash (устройство перезаписывает страницу целиком)

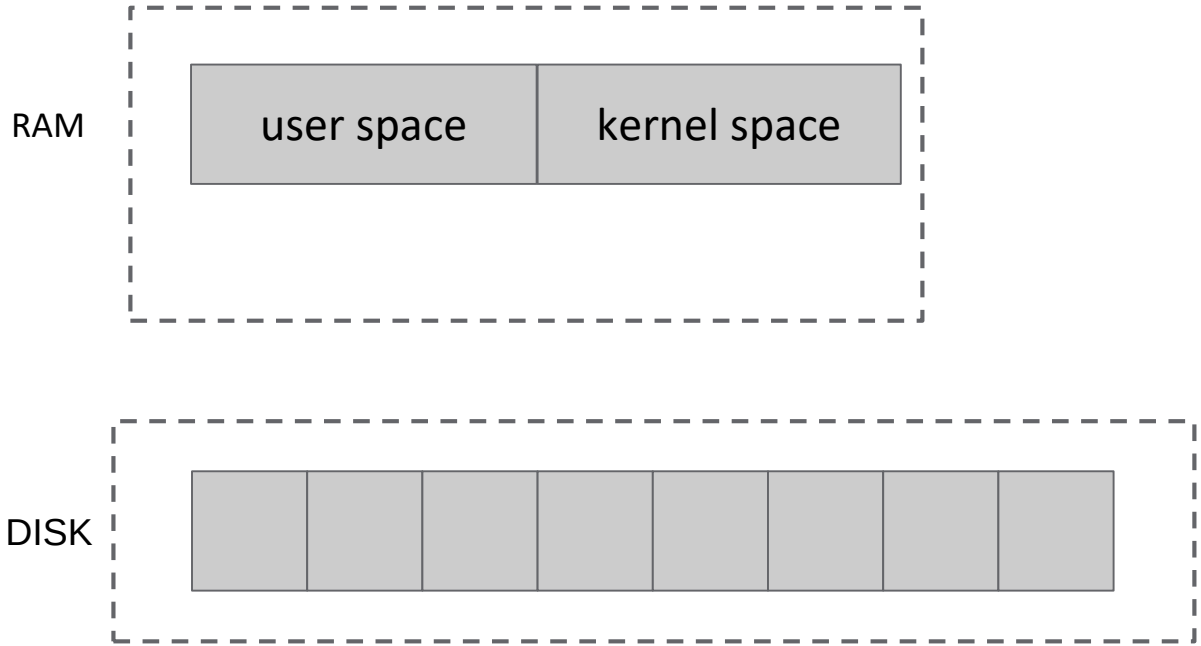
Что важно помнить

- Диски работают с страницами (пишите всегда кратно размеру страницы, это максимально эфективный способ работы)
- Если вы пишете меньше размера страницы, вы повышаете износ NAND flash (устройство перезаписывает страницу целиком)
- Иногда NAND flash controller запускает процедуру освобождения места (GC), что может повлиять на latency операций

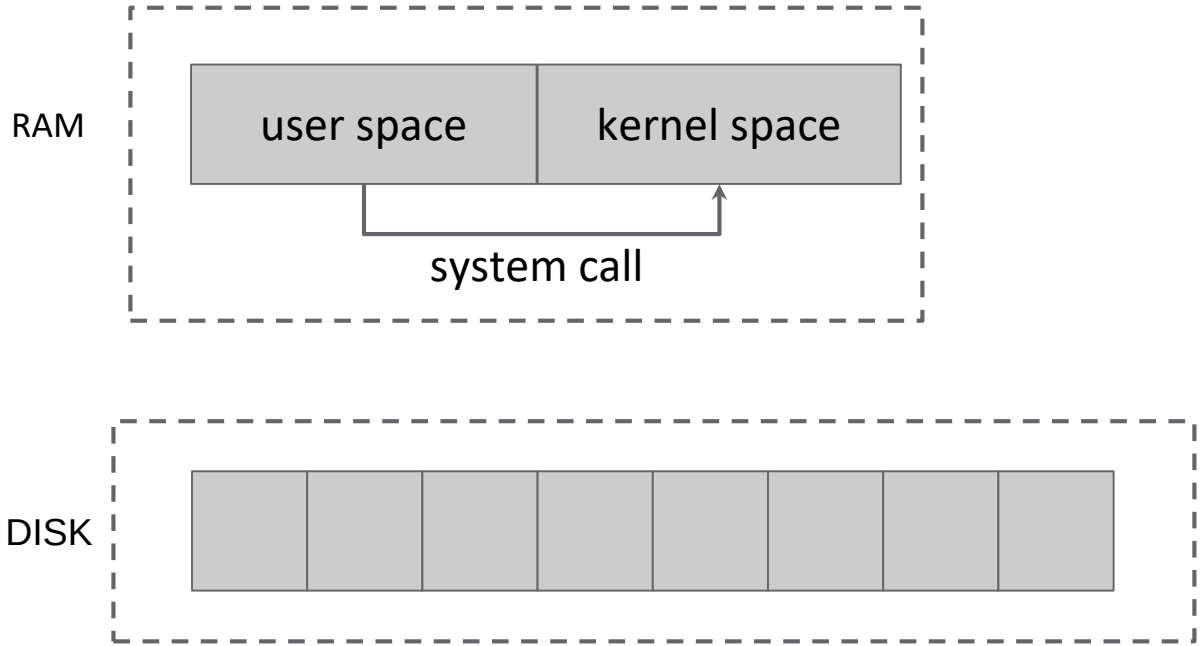
Чтение и запись



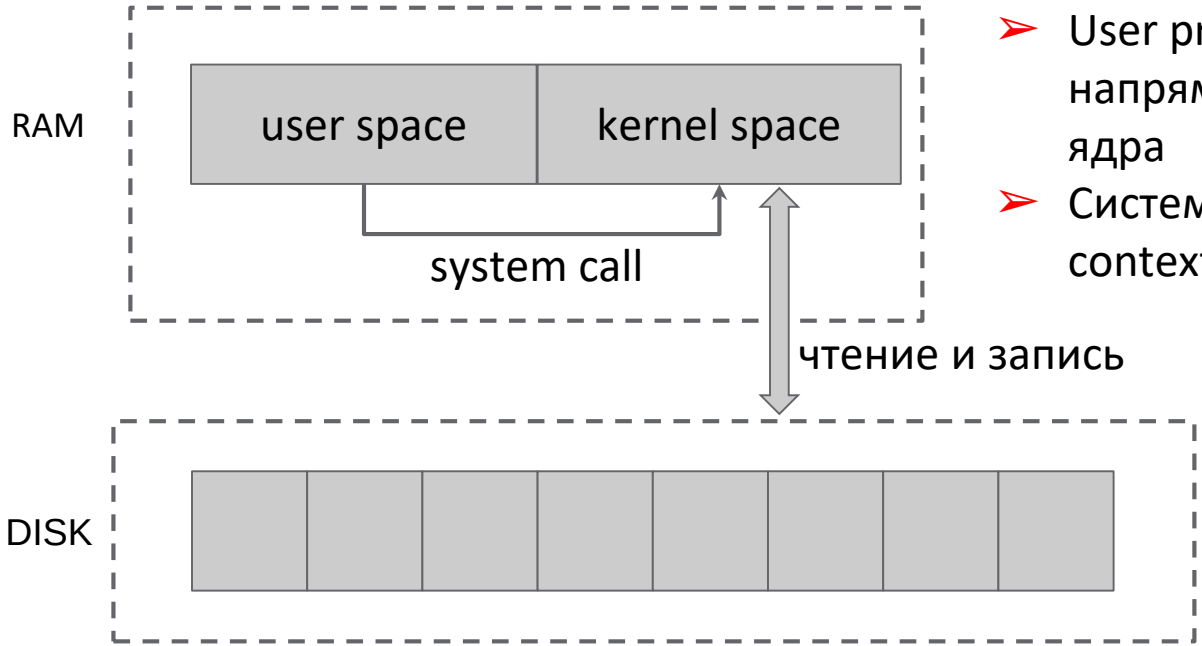
Чтение и запись



Чтение и запись

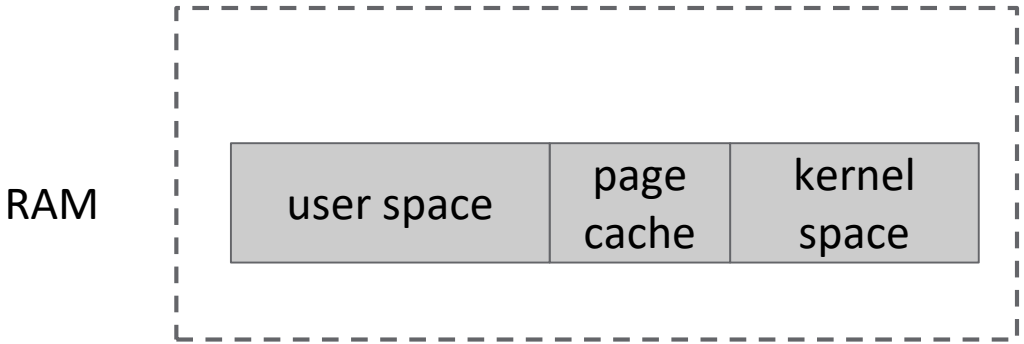


Чтение и запись

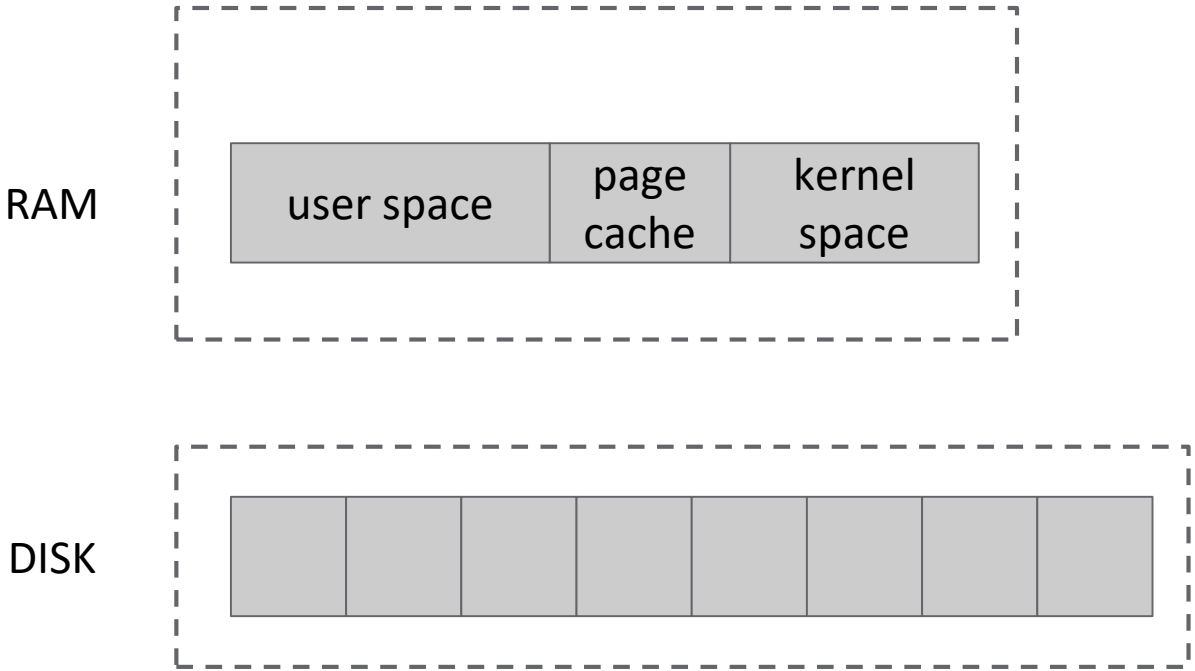


- User process не работает с диском напрямую только через вызовы ядра
- Системный вызов приводит к context switch потока

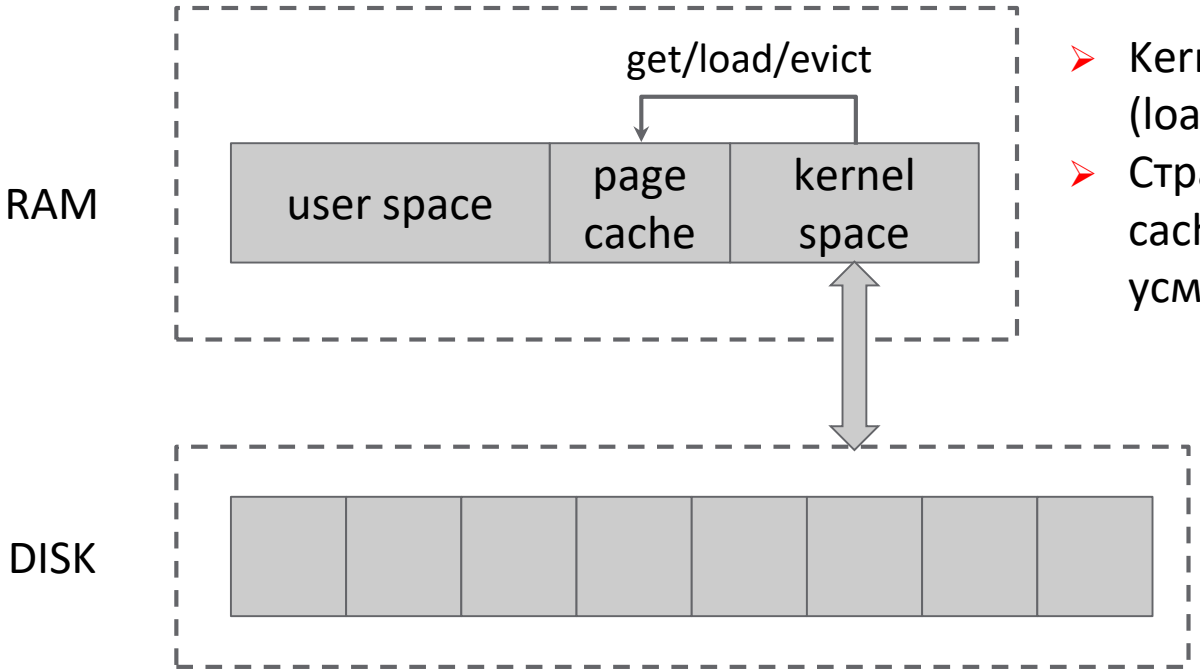
Page Cache



Page Cache



Page Cache



- Kernel управляет page cache (load/evict page)
- Страницы могут вытесняться из cache в любой момент на усмотрение kernel

FileInputStream/FileOutputStream

```
 1 byte[] buf = new byte[...];
 2
 3 try (FileInputStream is = new FileInputStream(new File("pathFile1"))) {
 4     is.read(buf);
 5 }
 6
 7 try (FileOutputStream out = new FileOutputStream(new File("pathFile2"))) {
 8     out.write(buf);
 9
10     out.flush();
11 }
```

FileChannel

```
1 class FileChannel {
2     static FileChannel open(Path path, OpenOption... options);
3
4     void close();
5
6     long position();
7
8     FileChannel position(long newPosition);
9
10    int read(ByteBuffer dst);
11
12    int write(ByteBuffer src);
13
14    void force(boolean metaData);
15 }
```

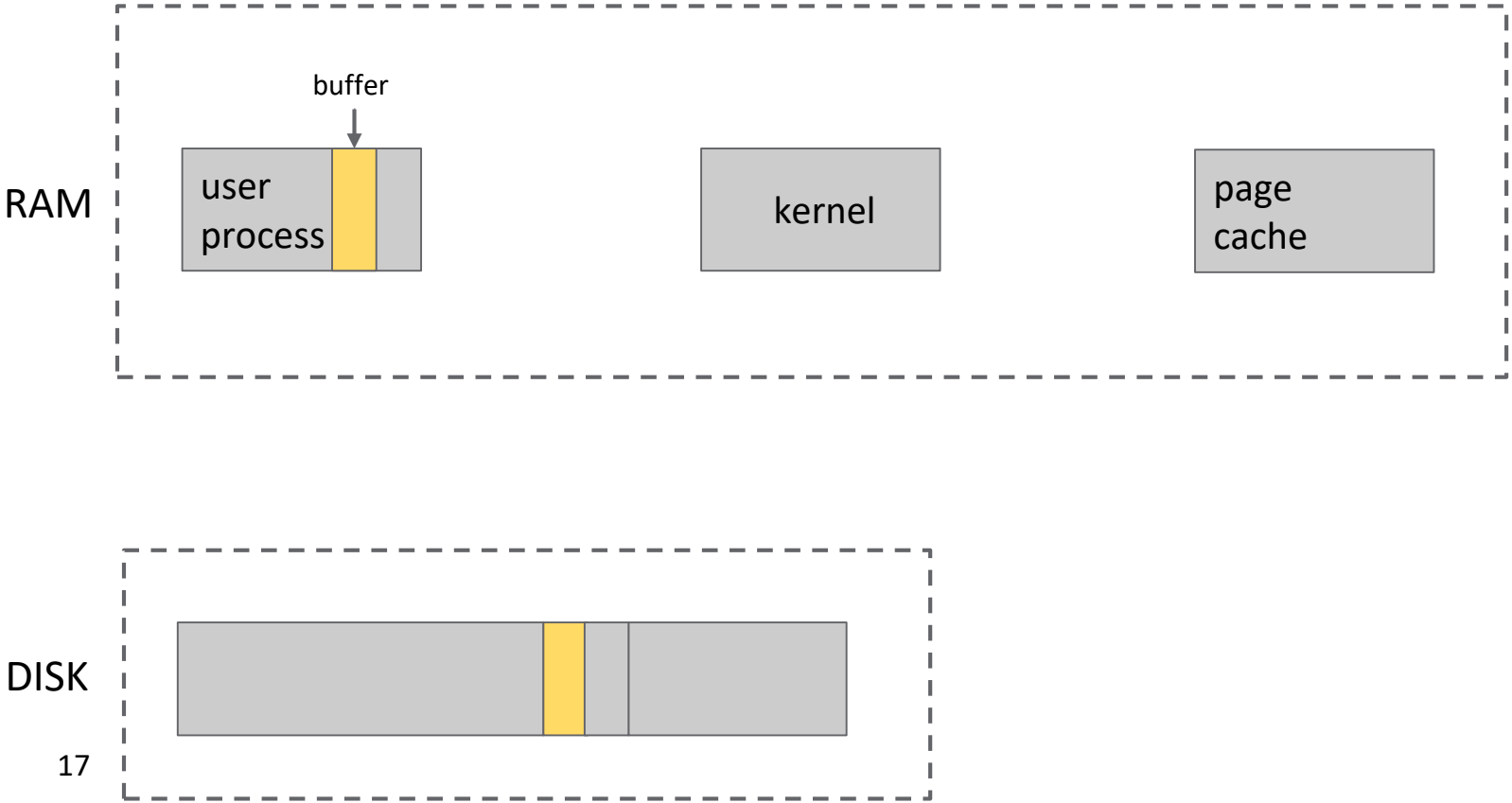
FileChannel Read/Write

```
 1 ByteBuffer buf = ByteBuffer.allocate(...);
 2
 3 try (FileChannel ch = FileChannel.open(Paths.get("pathFile1"), READ)) {
 4     ch.read(buf);
 5 }
 6
 7 buf.flip();
 8
 9 try (FileChannel ch = FileChannel.open(Paths.get("pathFile2"), WRITE)) {
10     ch.write(buf);
11
12     ch.force(true);
13 }
```

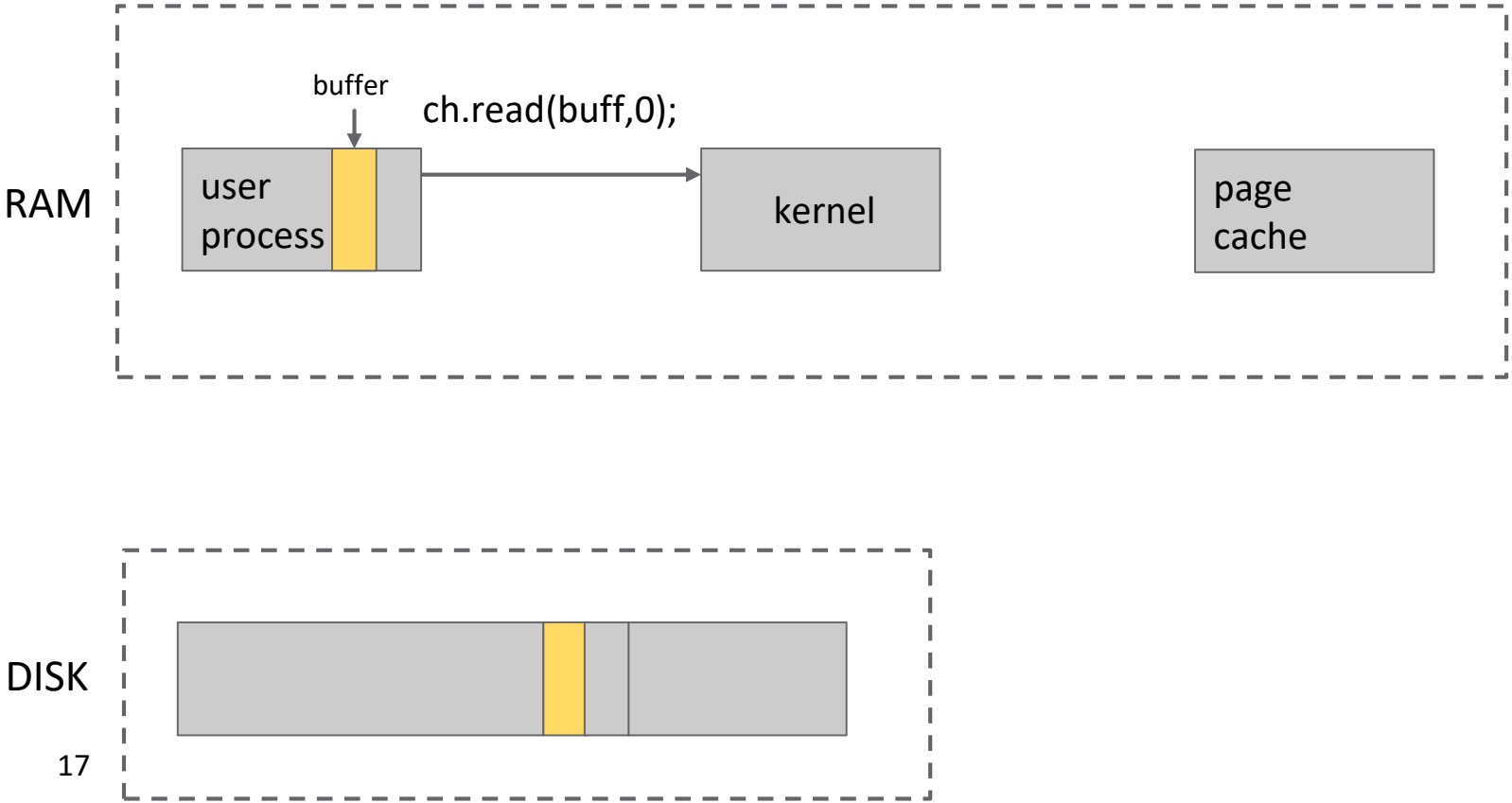
Read (aligned)

```
 1 ByteBuffer buf = ByteBuffer.allocate(...);
 2
 3 try (FileChannel ch = FileChannel.open(Paths.get("pathFile"), READ)) {
 4     long size = ch.size();
 5     long position = 0;
 6
 7     while (position < size) {
 8         int read = ch.read(buf, position);
 9         if (read <= 0)
10             break;
11
12         position += buf.position();
13         ......
14     }
15 }
```

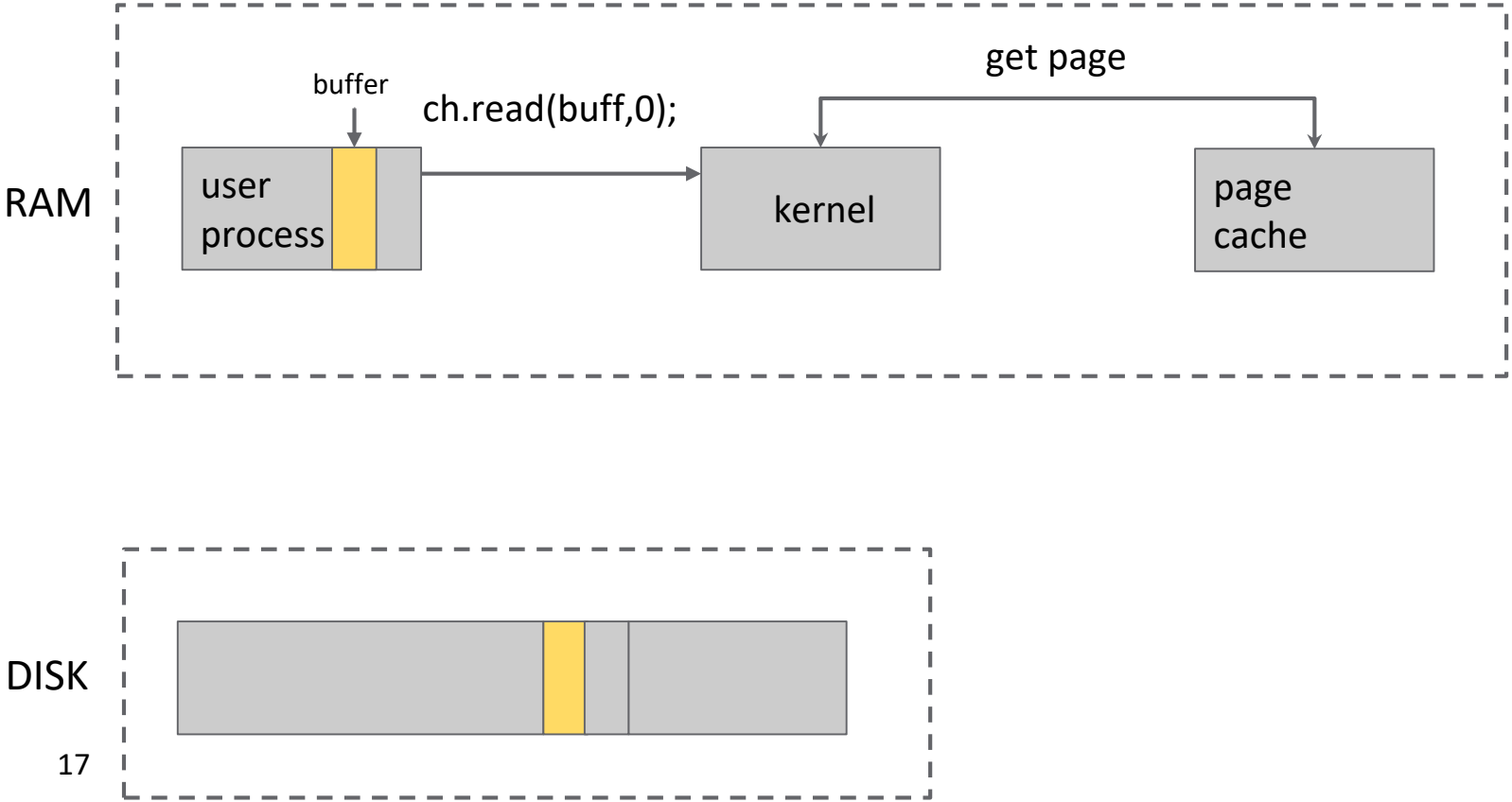
Page Cache Load Flow (aligned)



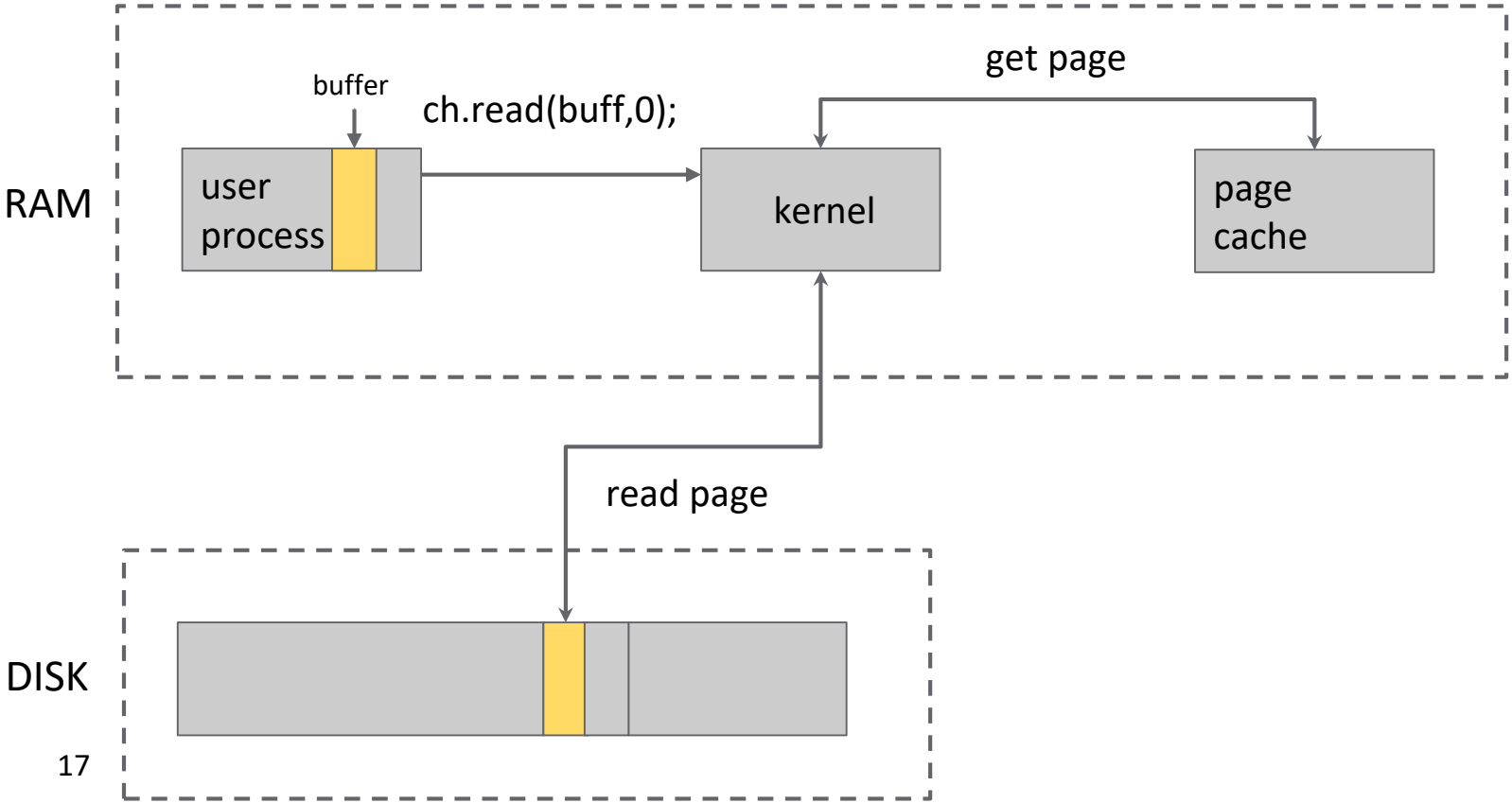
Page Cache Load Flow (aligned)



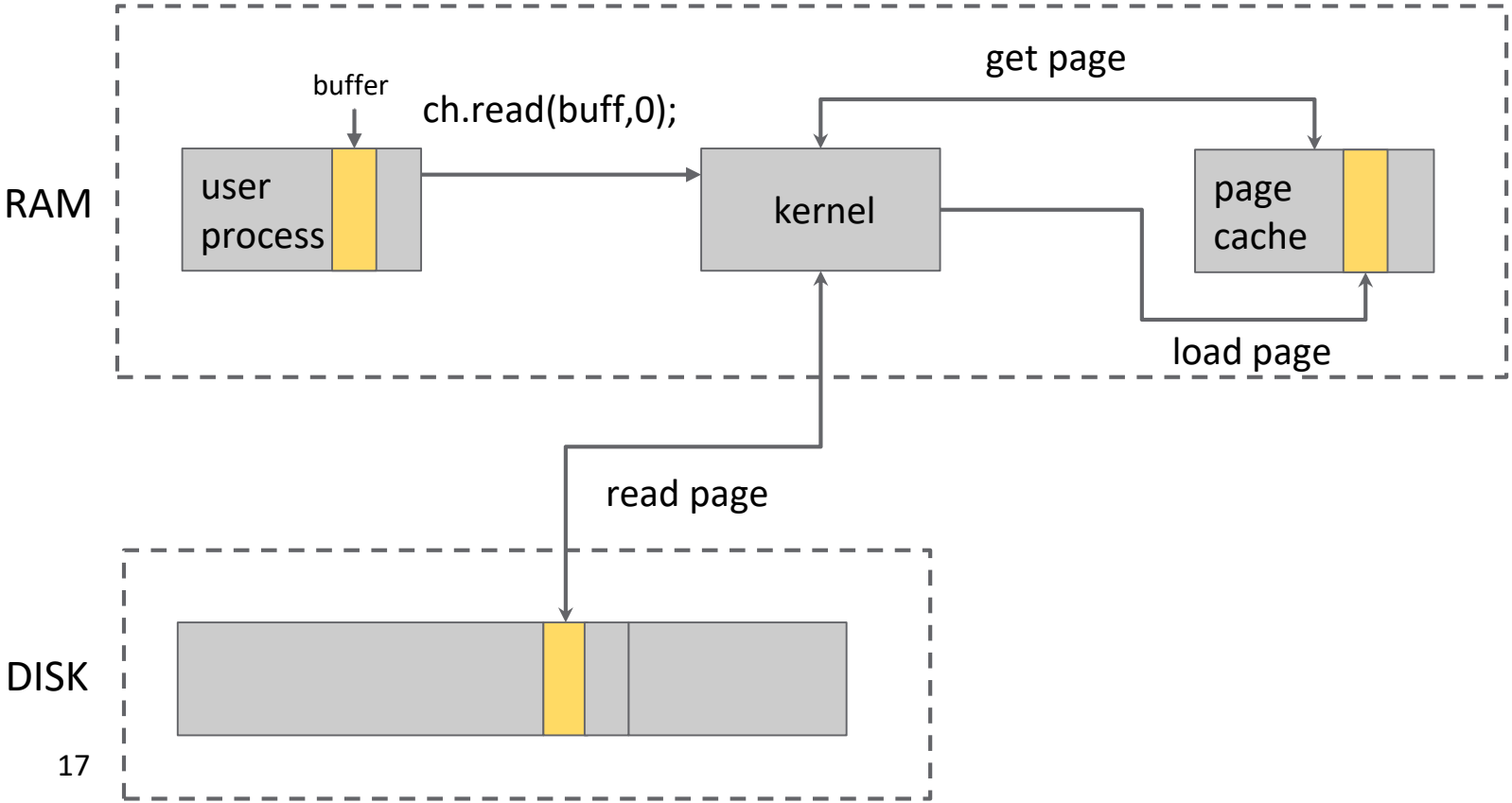
Page Cache Load Flow (aligned)



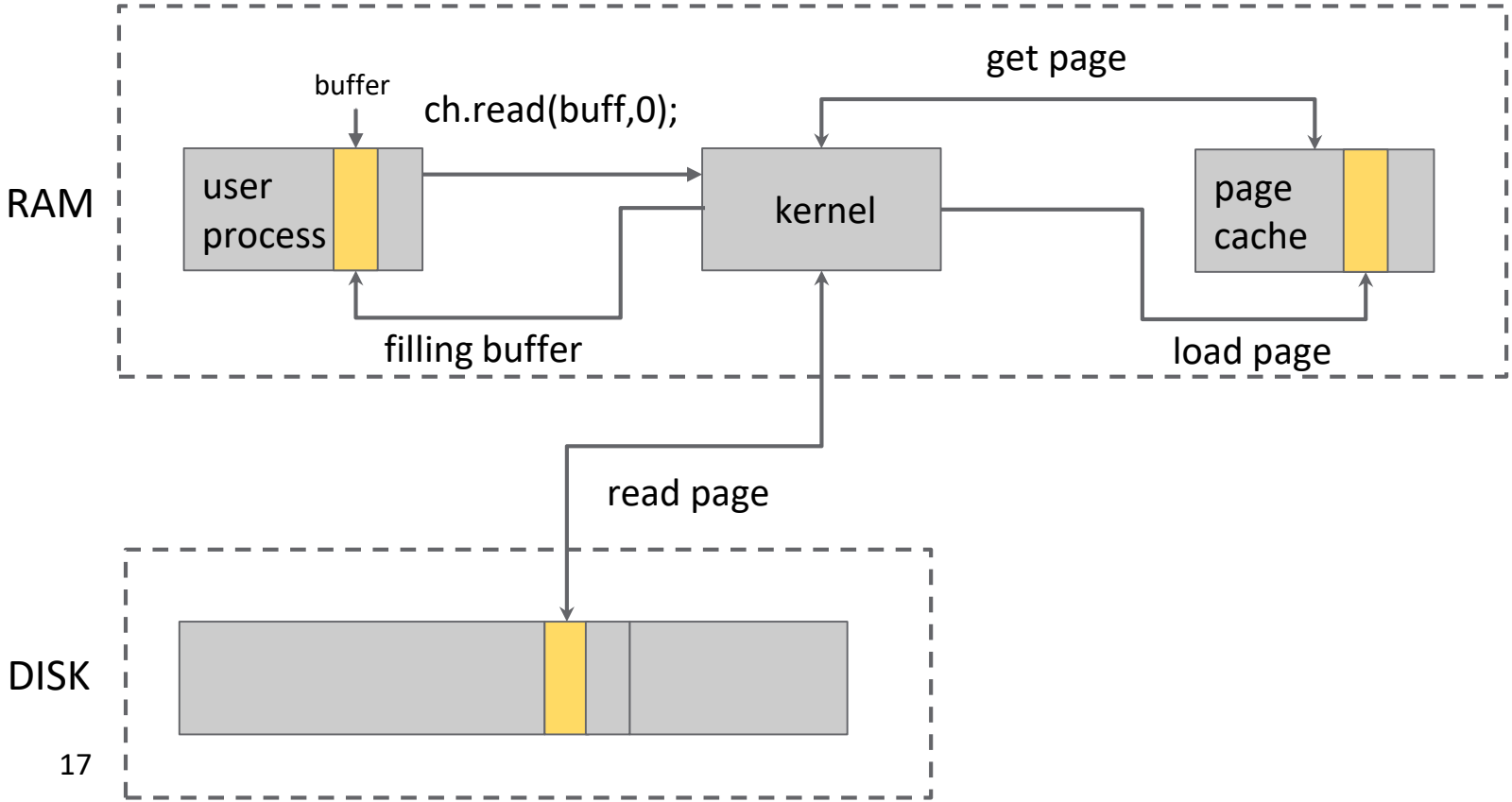
Page Cache Load Flow (aligned)



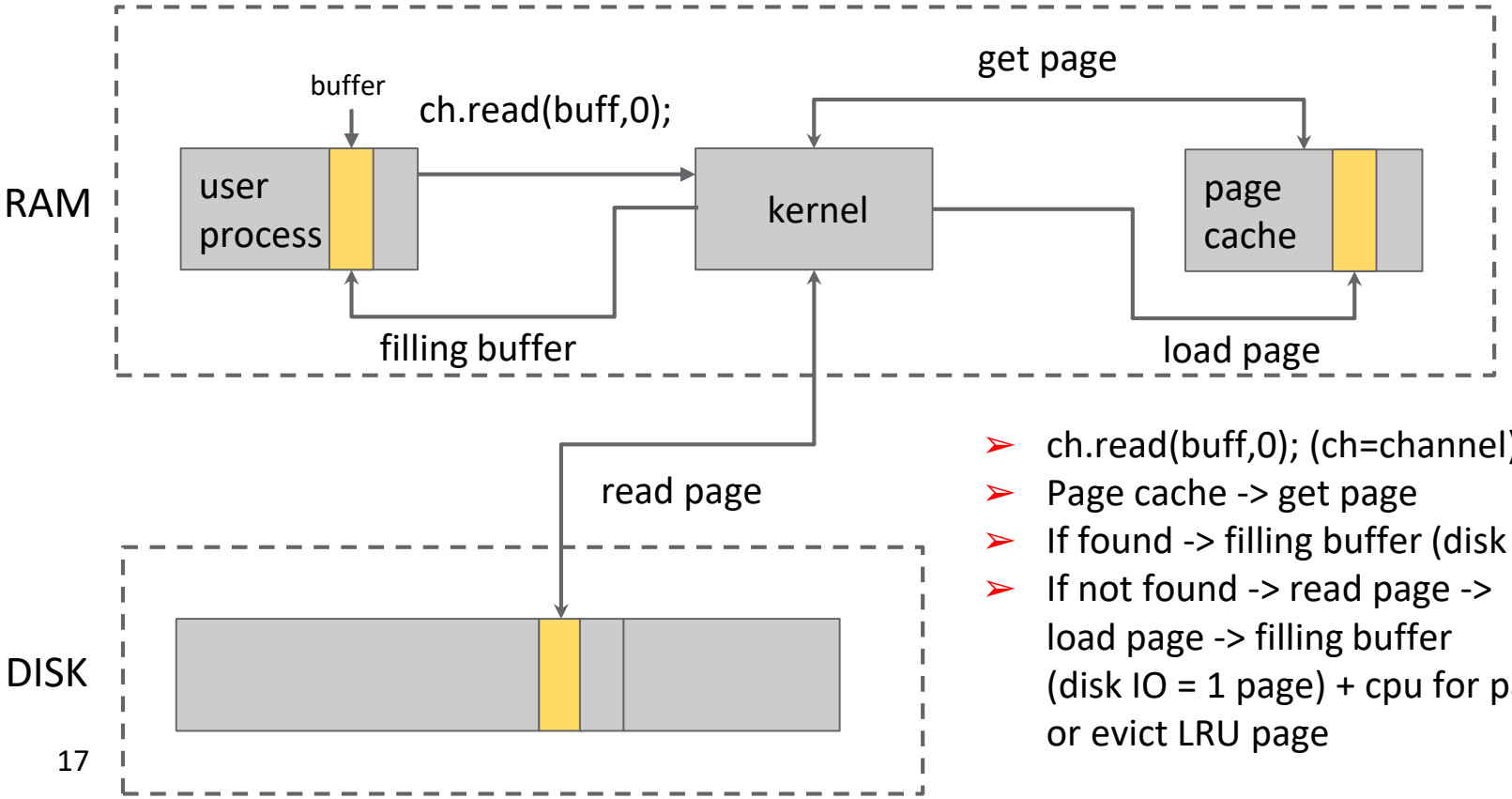
Page Cache Load Flow (aligned)



Page Cache Load Flow (aligned)



Page Cache Load Flow (aligned)



- ch.read(buff,0); (ch=channel)
- Page cache -> get page
- If found -> filling buffer (disk IO = 0)
- If not found -> read page -> load page -> filling buffer (disk IO = 1 page) + cpu for page cache load or evict LRU page

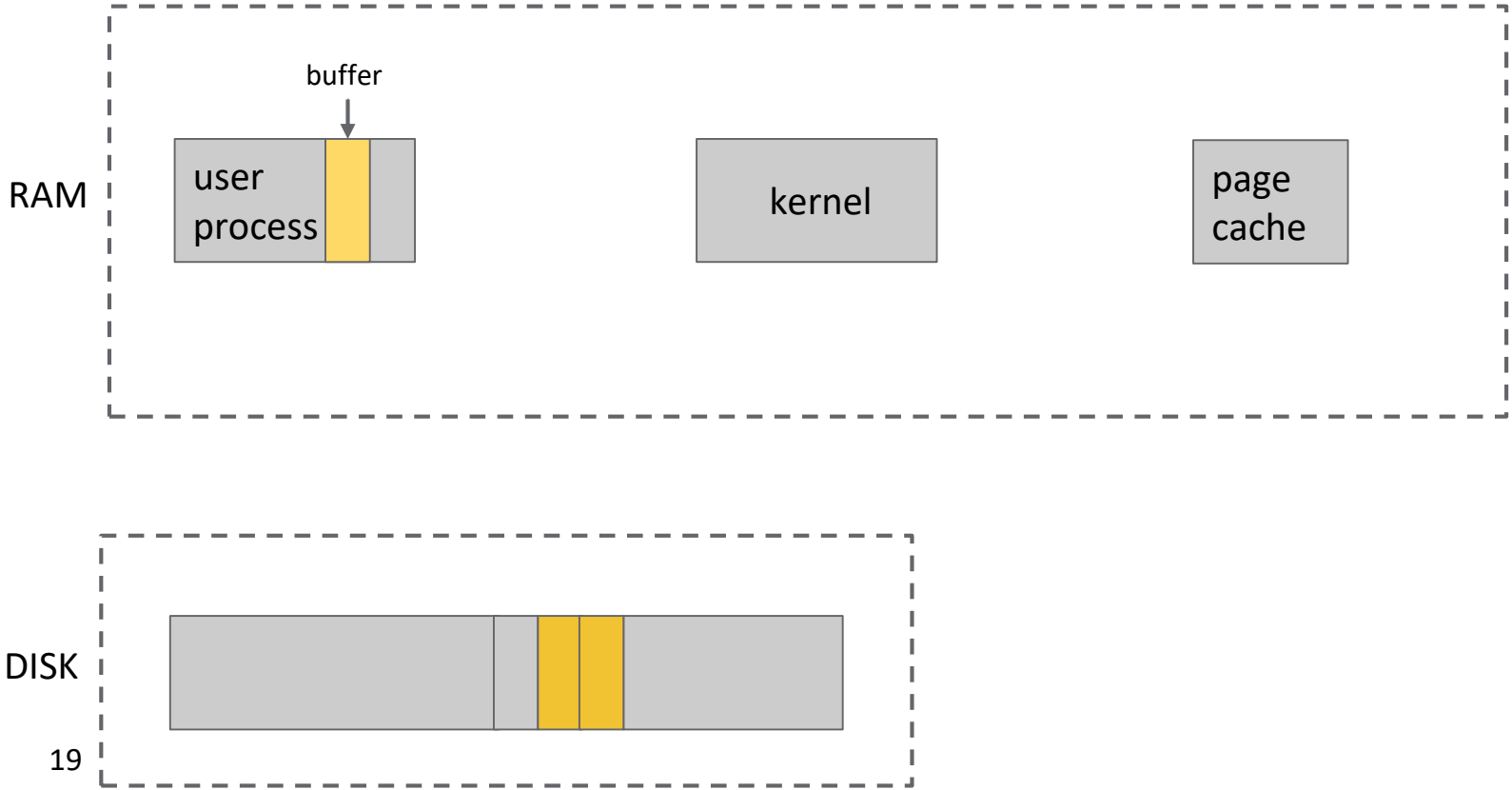
Read (no aligned)

```
1 try (FileChannel ch = FileChannel.open(Paths.get("pathFile"), READ)) {
2     long size = ch.size();
3     long position = 2048;
4
5     while (position < size) {
6         int read = ch.read(buf, position);
7         if (read <= 0)
8             break;
9
10        position += buf.position();
11        ......
12    }
13 }
```

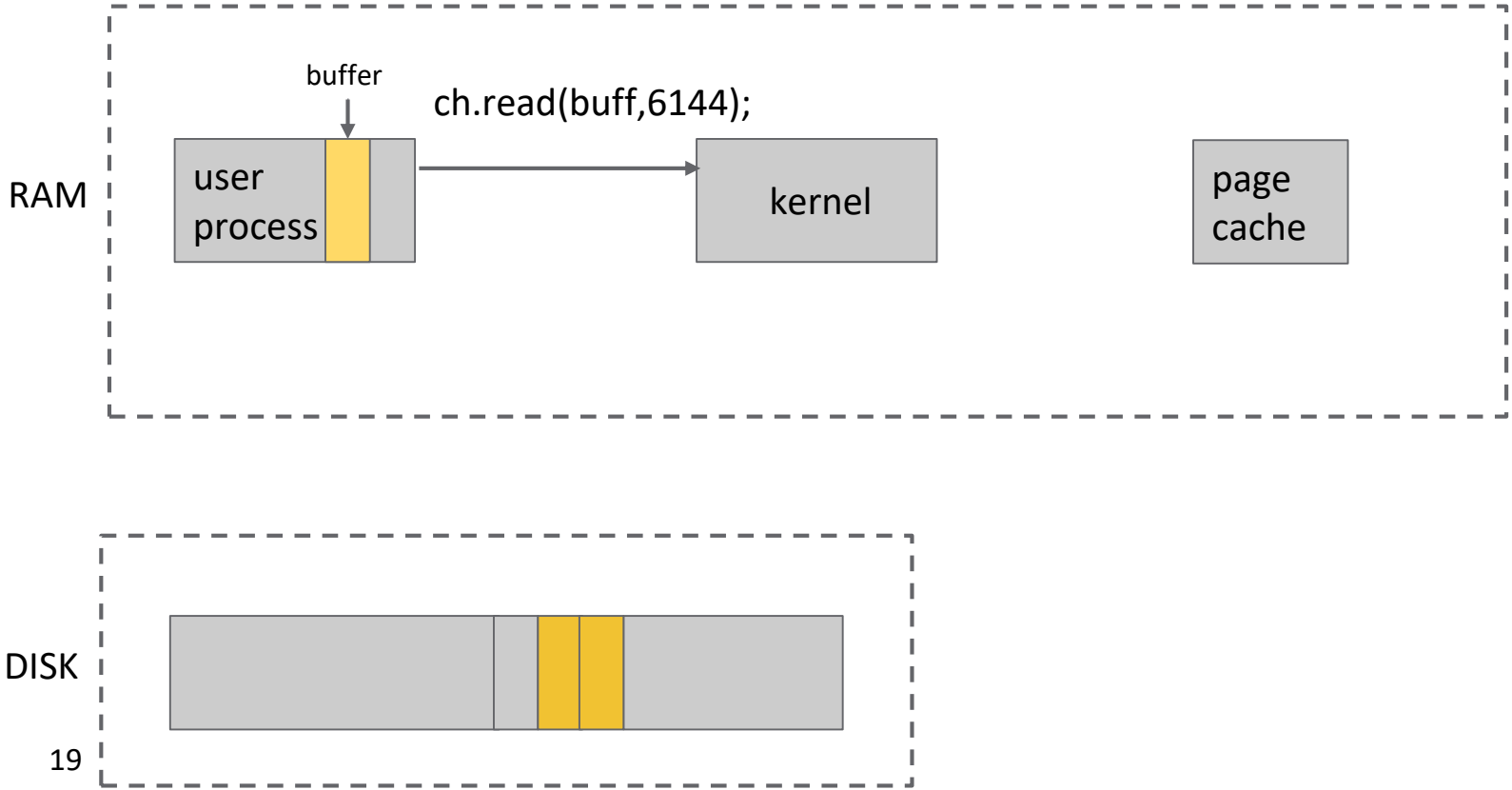
Read (no aligned)

```
1 try (FileChannel ch = FileChannel.open(Paths.get("pathFile"), READ)) {
2     long size = ch.size();
3     long position = 2048;
4
5     while (position < size) {
6         int read = ch.read(buf, position);
7         if (read <= 0)
8             break;
9
10        position += buf.position();
11        ......
12    }
13 }
```

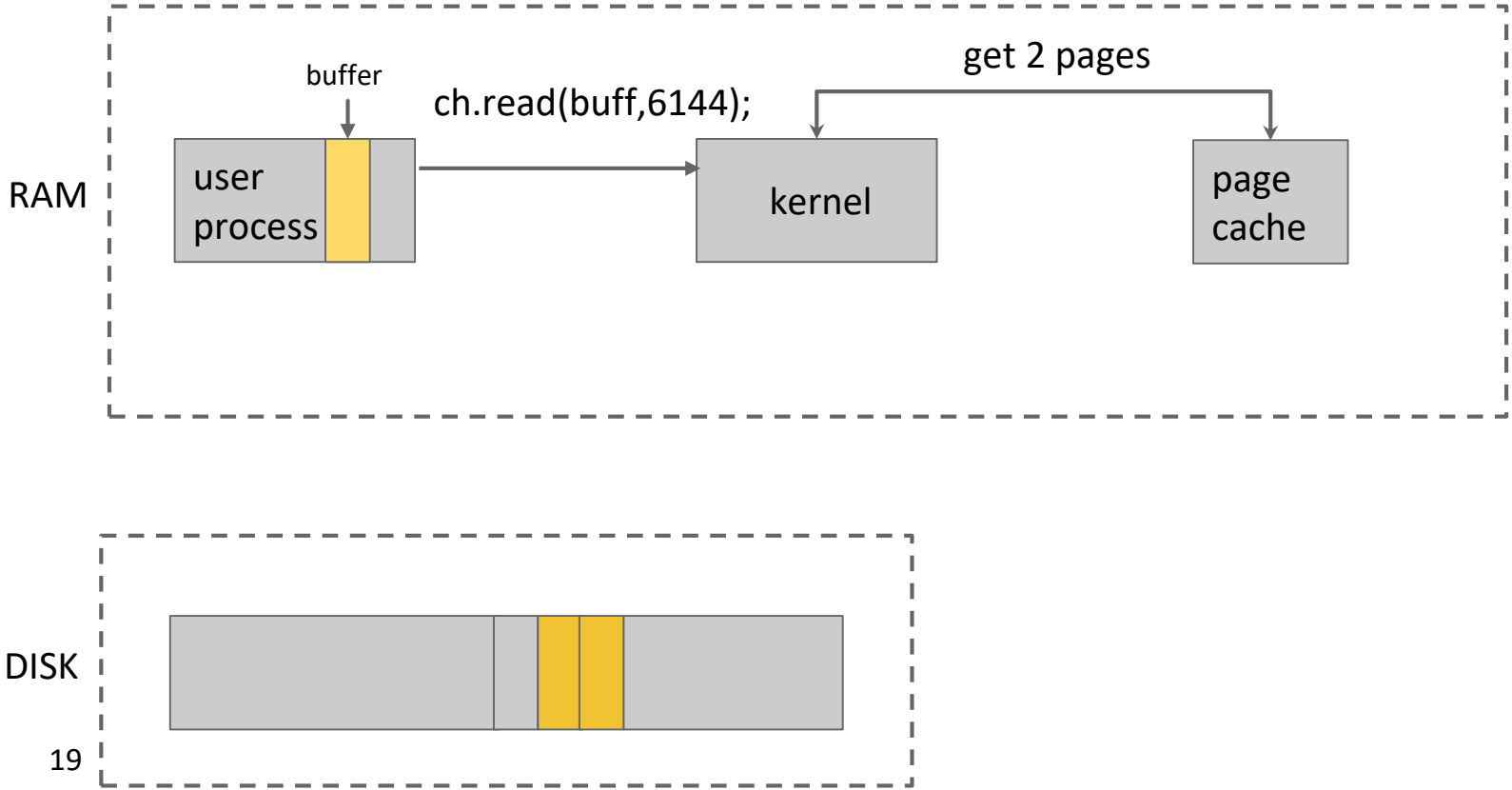
Page Cache Load Flow (no aligned)



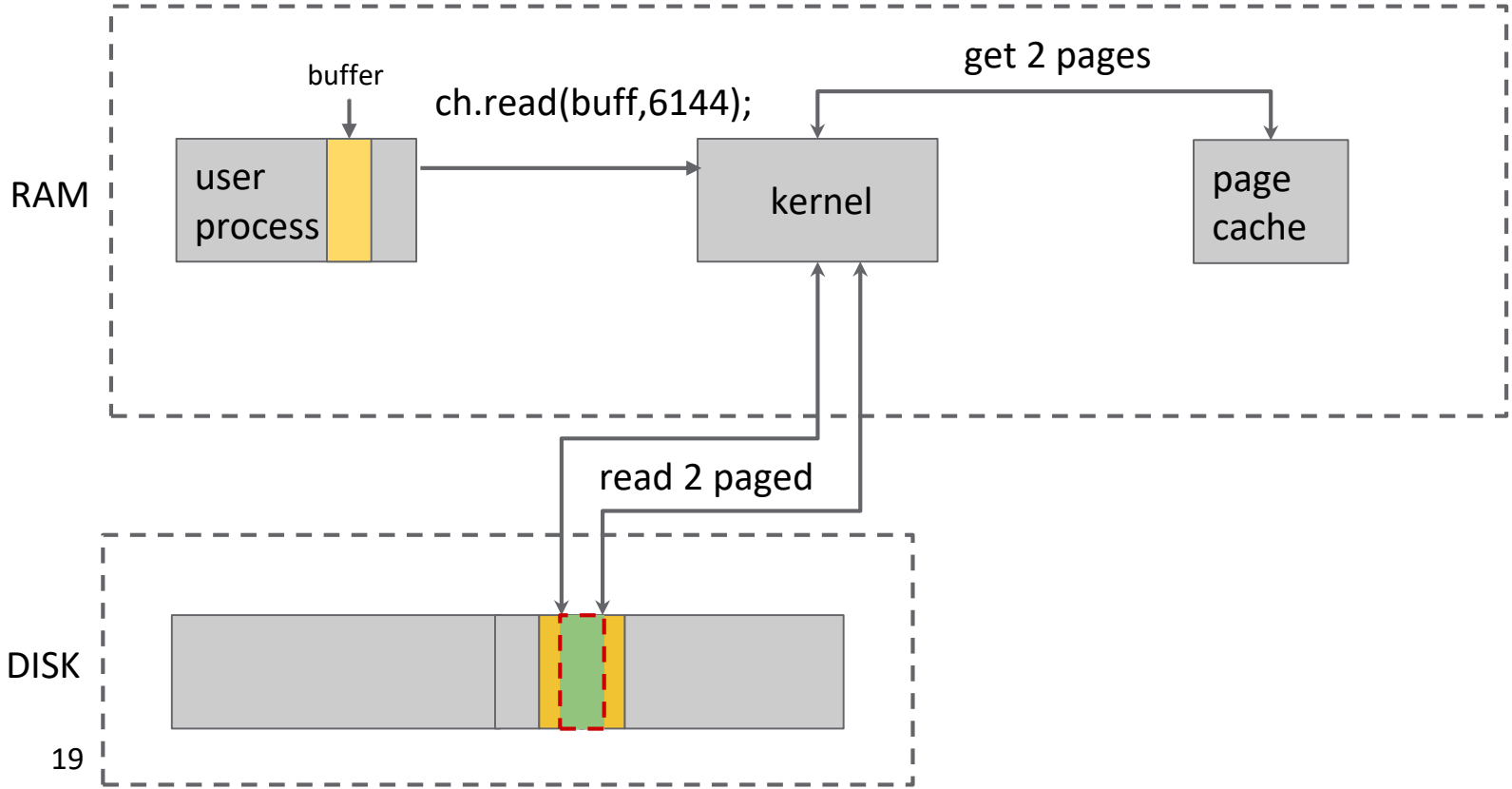
Page Cache Load Flow (no aligned)



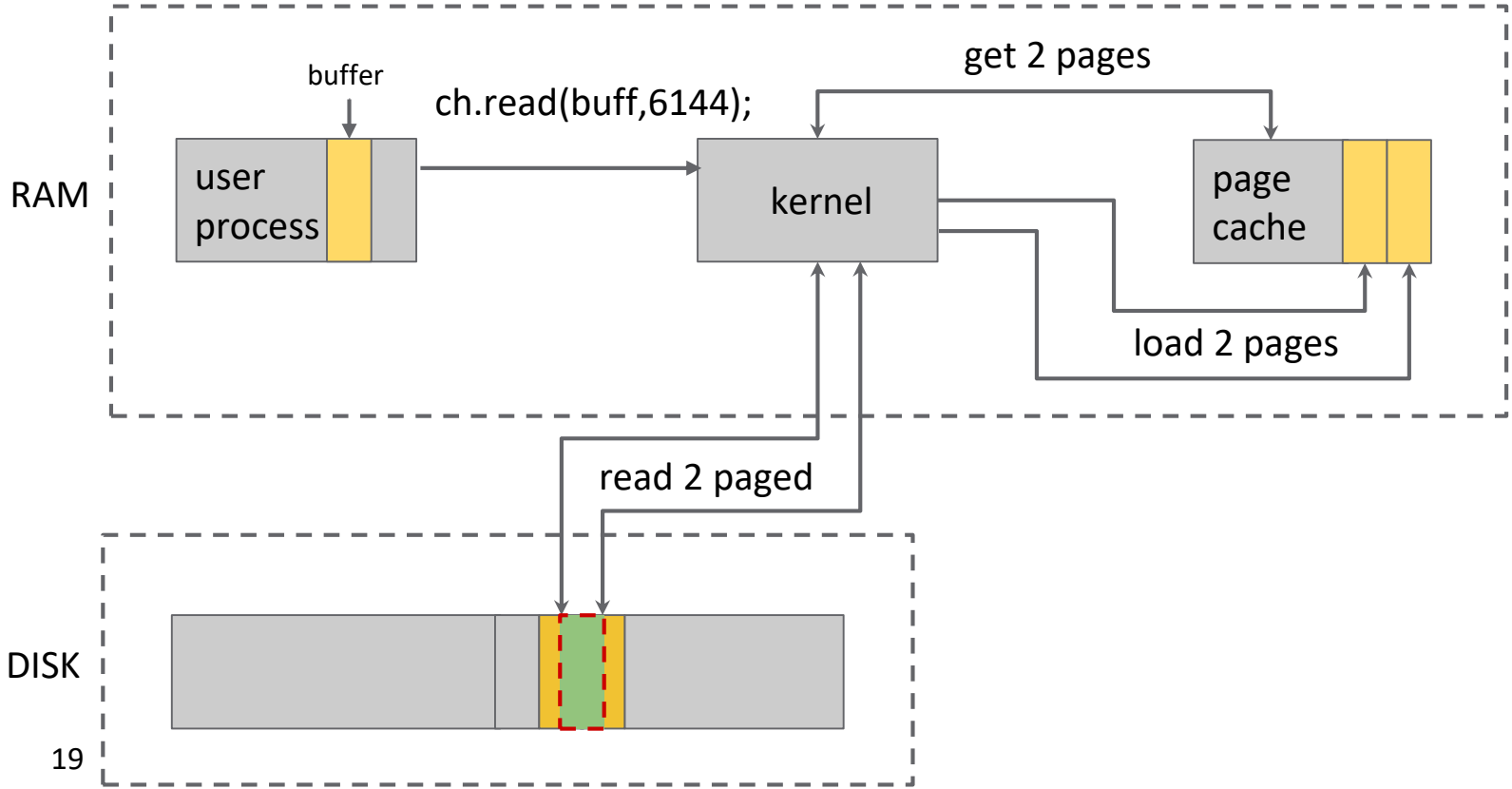
Page Cache Load Flow (no aligned)



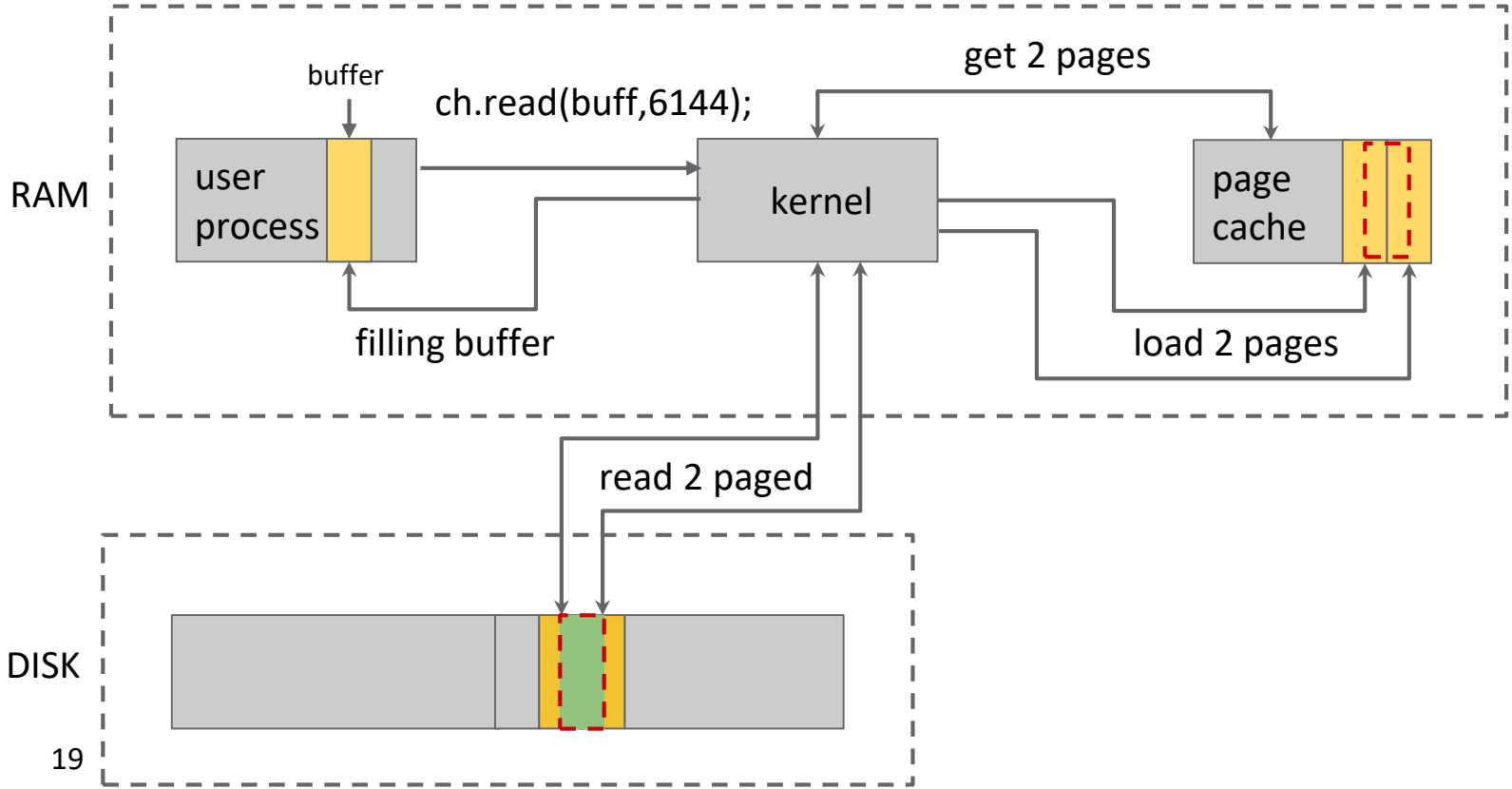
Page Cache Load Flow (no aligned)



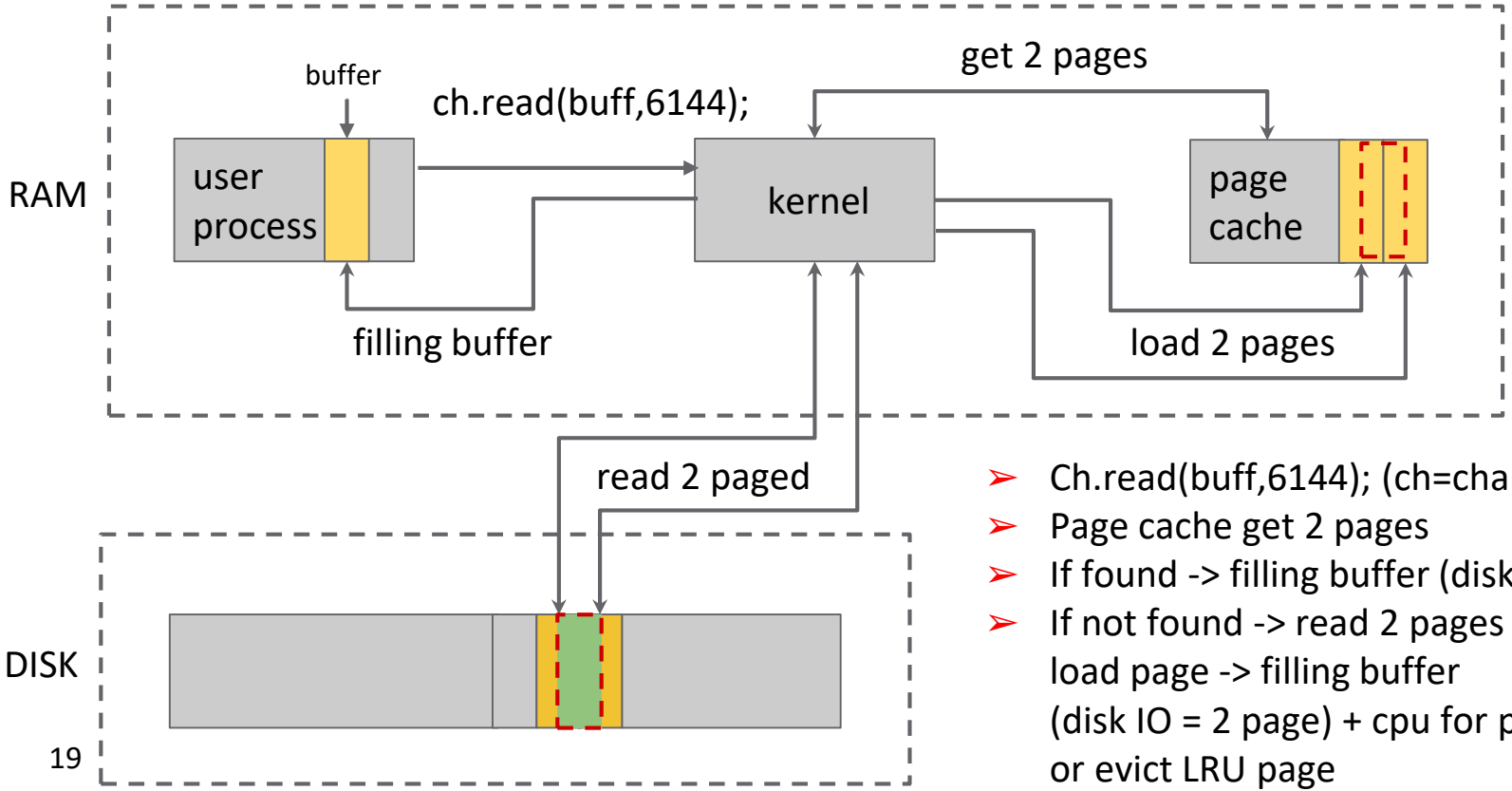
Page Cache Load Flow (no aligned)



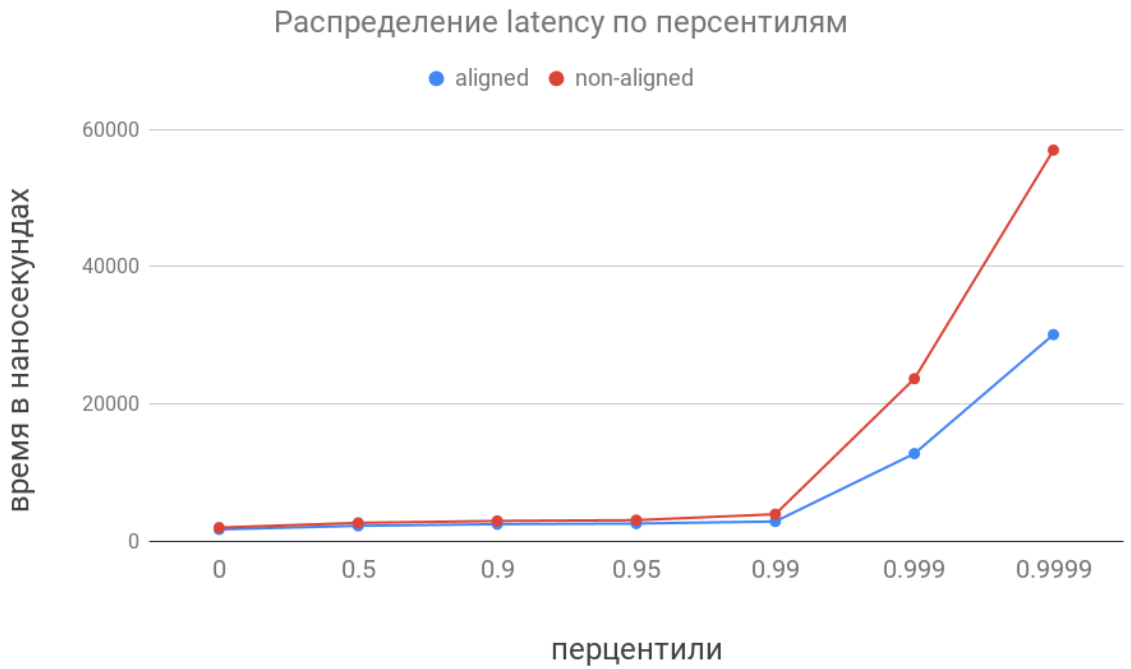
Page Cache Load Flow (no aligned)



Page Cache Load Flow (no aligned)

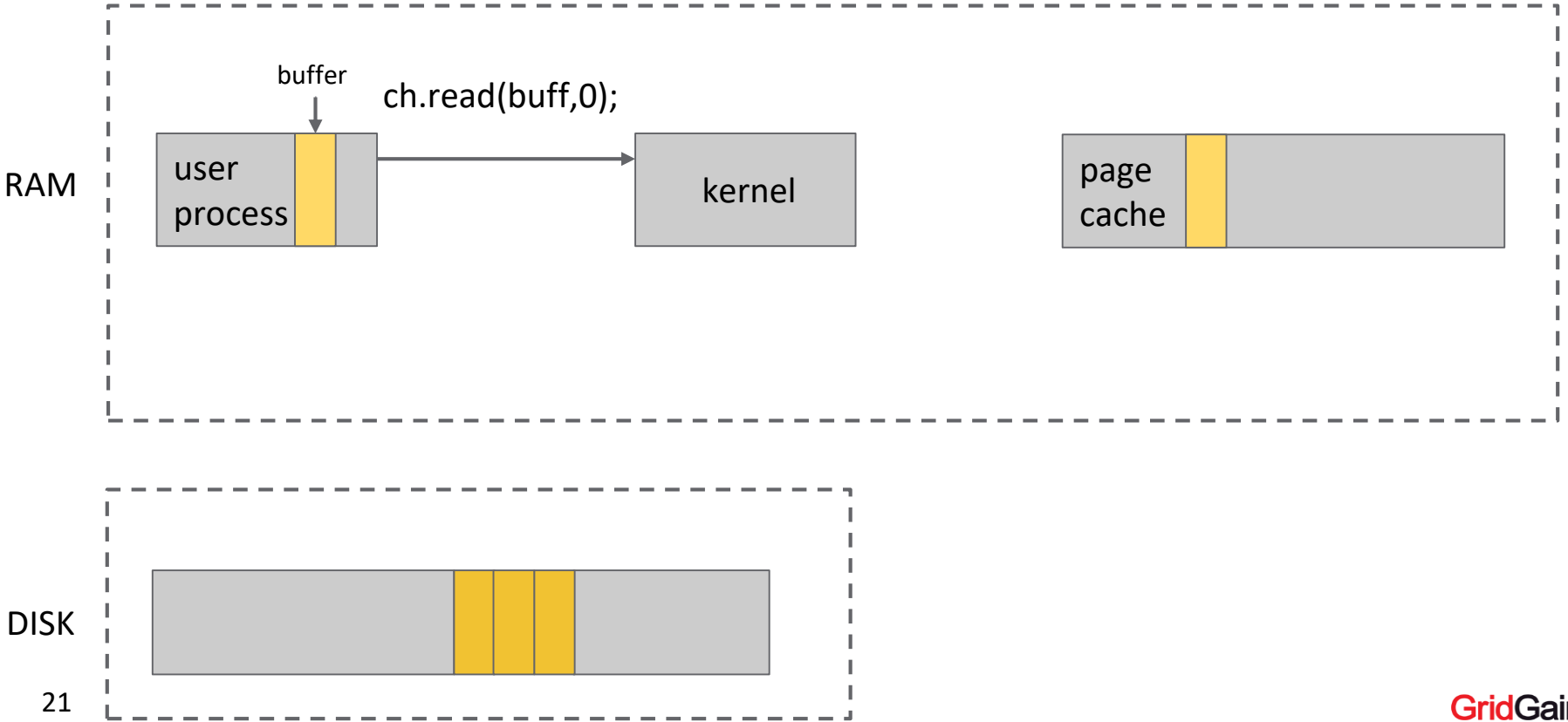


- Ch.read(buff,6144); (ch=channel)
- Page cache get 2 pages
- If found -> filling buffer (disk IO = 0)
- If not found -> read 2 pages -> load page -> filling buffer (disk IO = 2 page) + cpu for page cache load or evict LRU page

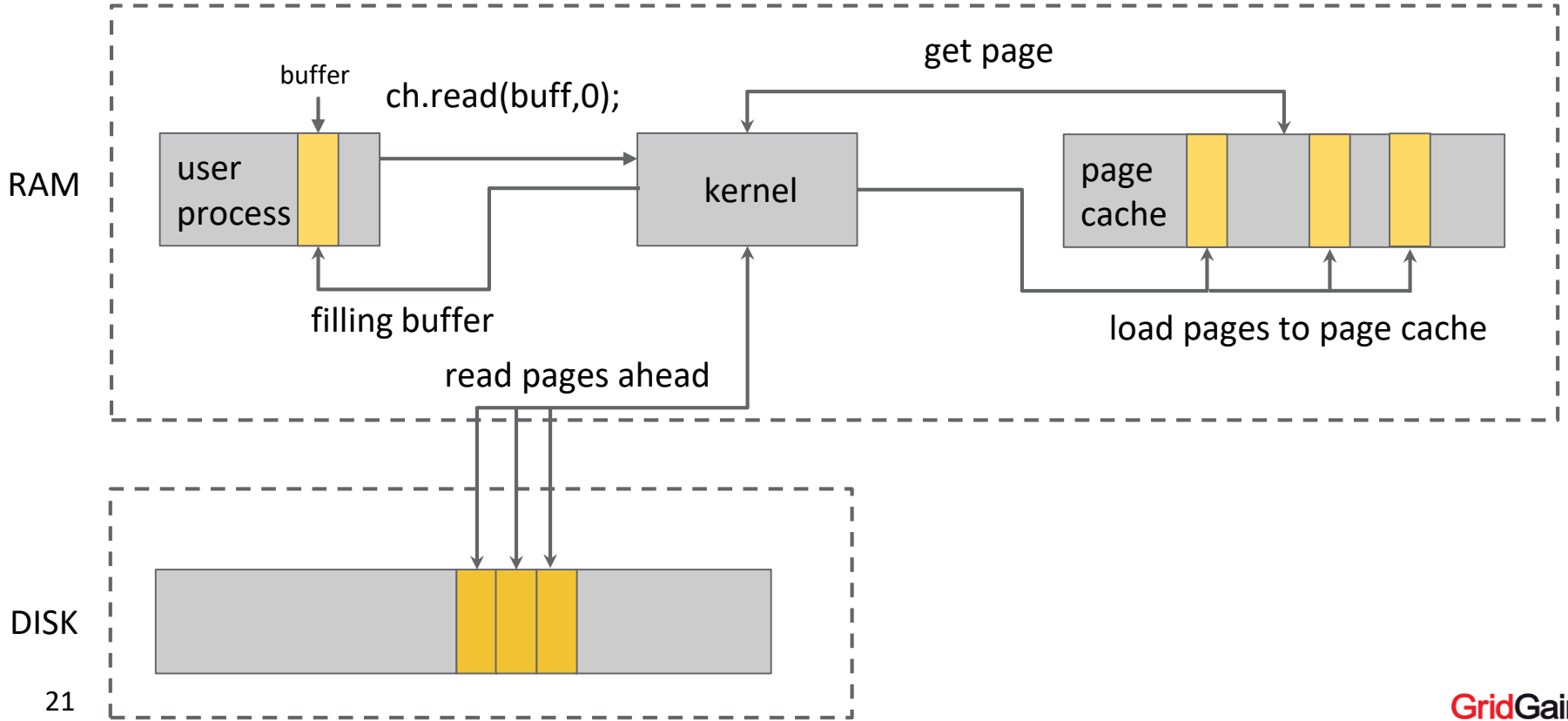


p	0	0.5	0.9	0.95	0.99	0.999	0.9999
Aligned	1692	2208	2456	2544	2844	12728.960	30077.594
Non-aligned	1952	2640	2924	3036	3900	23646.816	56962.618
%	15.37	19.57	19.06	19.34	37.13	85.77	89.39

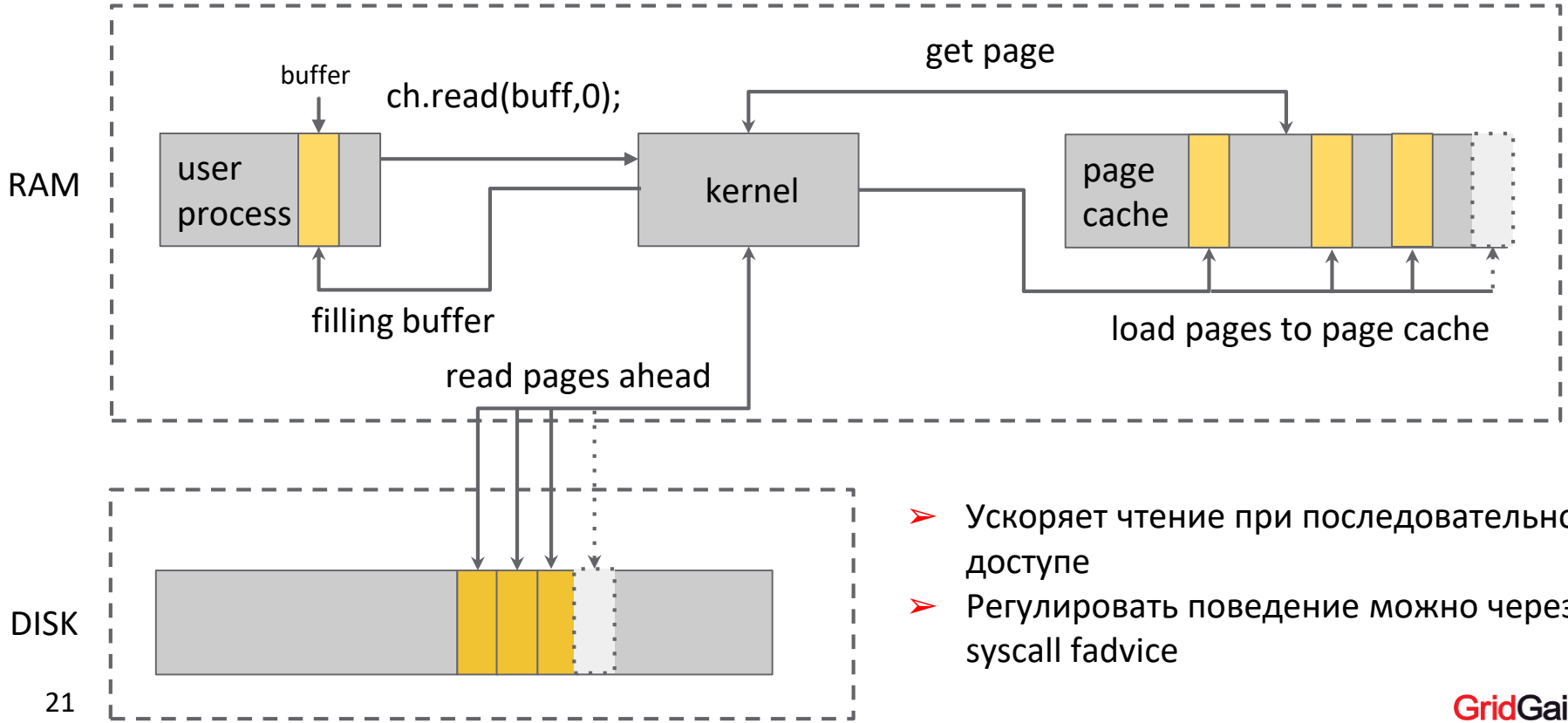
Page Cache Read-ahead



Page Cache Read-ahead



Page Cache Read-ahead



- Ускоряет чтение при последовательном доступе
- Регулировать поведение можно через syscall fadvice

fadvice

```
1 try (FileChannel ch = FileChannel.open(Paths.get("pathFile"), READ)) {
2     Field fdField = JavaInternals.getField(FileChannelImpl.class, "fd");
3     int fd = fdField.getInt(ch);
4     int fadivceFlag = Mem.FADV_RANDOM;
5
6     Mem.posix_fadvise(fd, 0, ch.size(), fadivceFlag);
7
8     // Perfome random read and write operation.
9 }
```

https://github.com/odnoklassniki/one-nio

http://man7.org/linux/man-pages/man2/posix_fadvise.2.html

fadvice

```
1 try (FileChannel ch = FileChannel.open(Paths.get("pathFile"), READ)) {
2     Field fdField = JavaInternals.getField(FileChannelImpl.class, "fd");
3     int fd = fdField.getInt(ch);
4     int fadivceFlag = Mem.FADV_RANDOM;
5
6     Mem.posix_fadvise(fd, 0, ch.size(), fadivceFlag);
7
8     // Perfome random read and write operation.
9 }
```

FADV_NORMAL
FADV_RANDOM
FADV_SEQUENTIAL
FADV_WILLNEED
FADV_DONTNEED
FADV_NOREUSE

https://github.com/odnoklassniki/one-nio

http://man7.org/linux/man-pages/man2/posix_fadvise.2.html

Что важно помнить

- Читать и писать лучше выровненными страницами

Что важно помнить

- Читать и писать лучше выровненными страницами
- Писать лучше полностью заполненные страницы

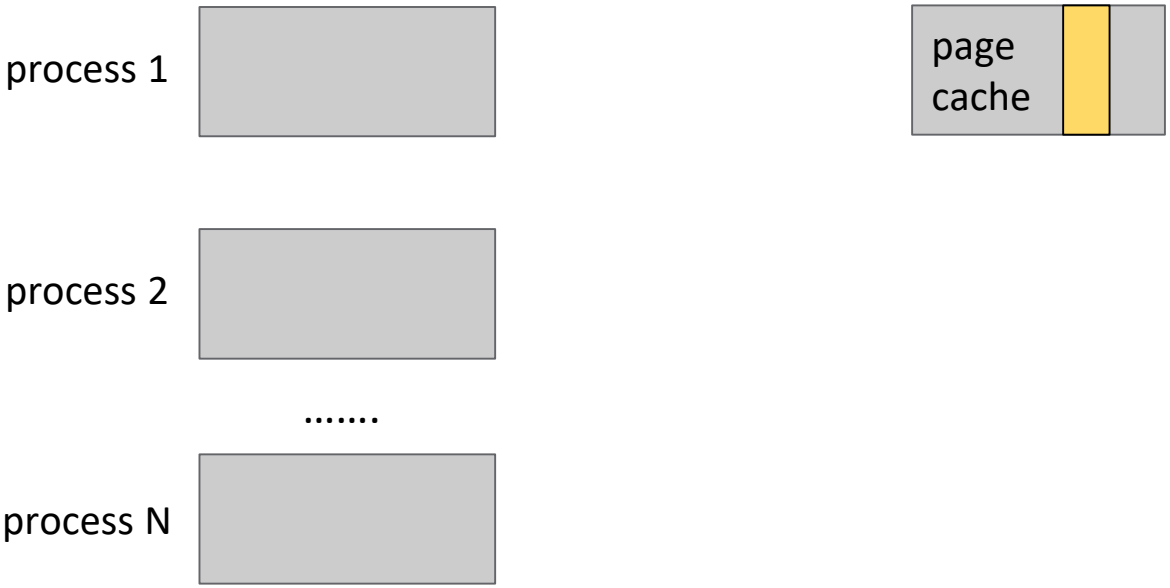
Что важно помнить

- Читать и писать лучше выровненными страницами
- Писать лучше полностью заполненные страницы
- Page cache - часть памяти под управлением kernel, которая кэширует данные с диска при чтении и буферизирует при записи

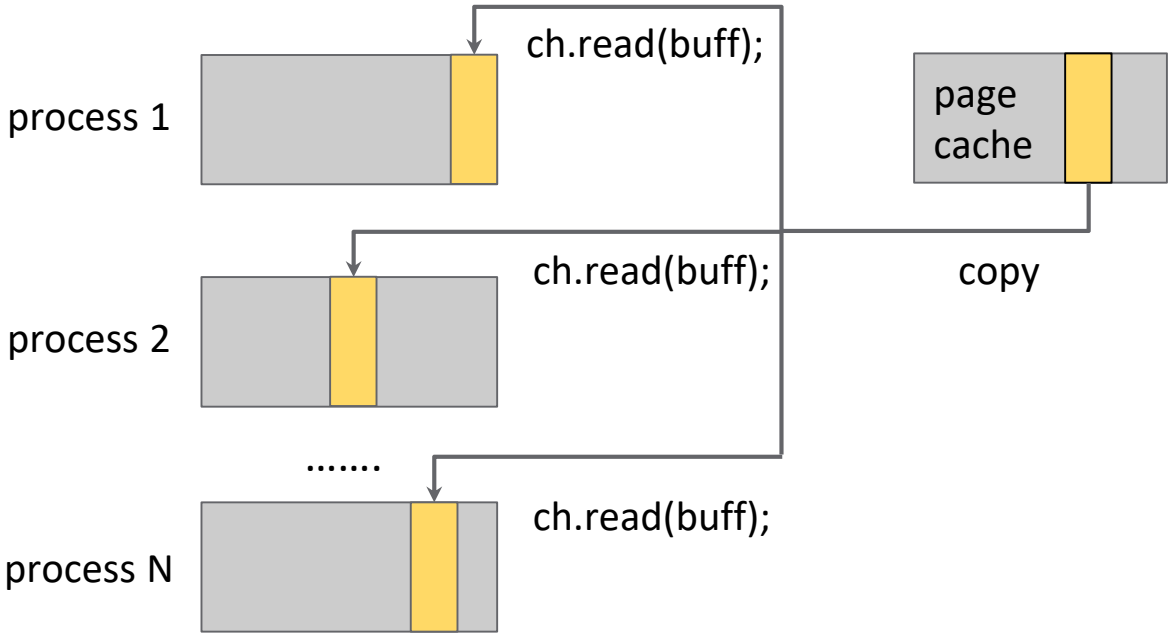
Что важно помнить

- Читать и писать лучше выровненными страницами
- Писать лучше полностью заполненные страницы
- Page cache - часть памяти под управлением kernel, которая кэширует данные с диска при чтении и буферизирует при записи
- Readahead - возможность ядра самому принять решить и прочитать данные из файла вперед относительно запрашиваемых (чтение файлов ОС на старте, чтение мультимедиа файлов etc.)

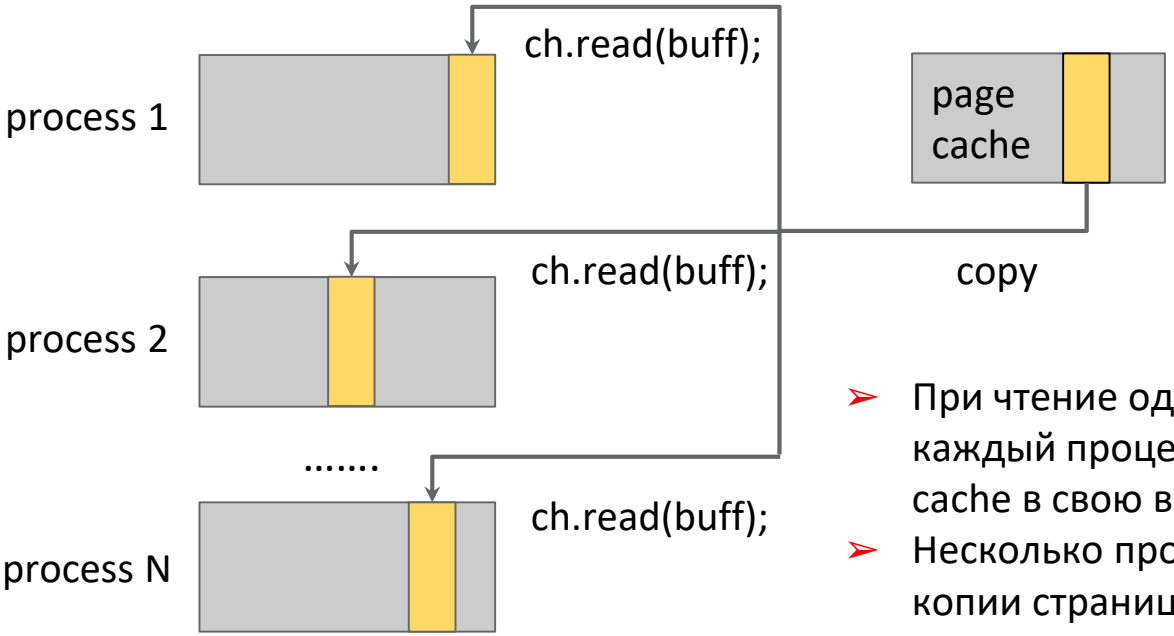
Reading redundancy



Reading redundancy

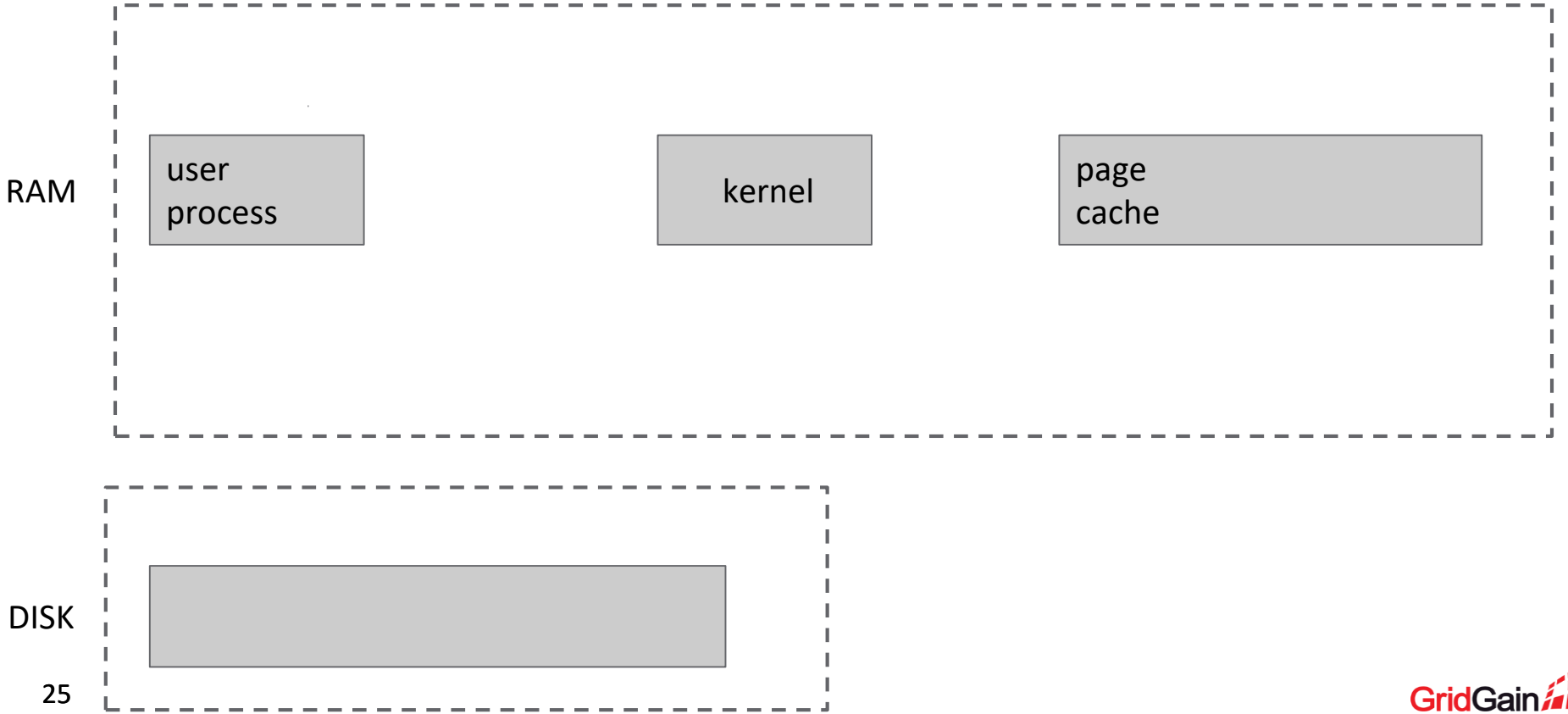


Reading redundancy

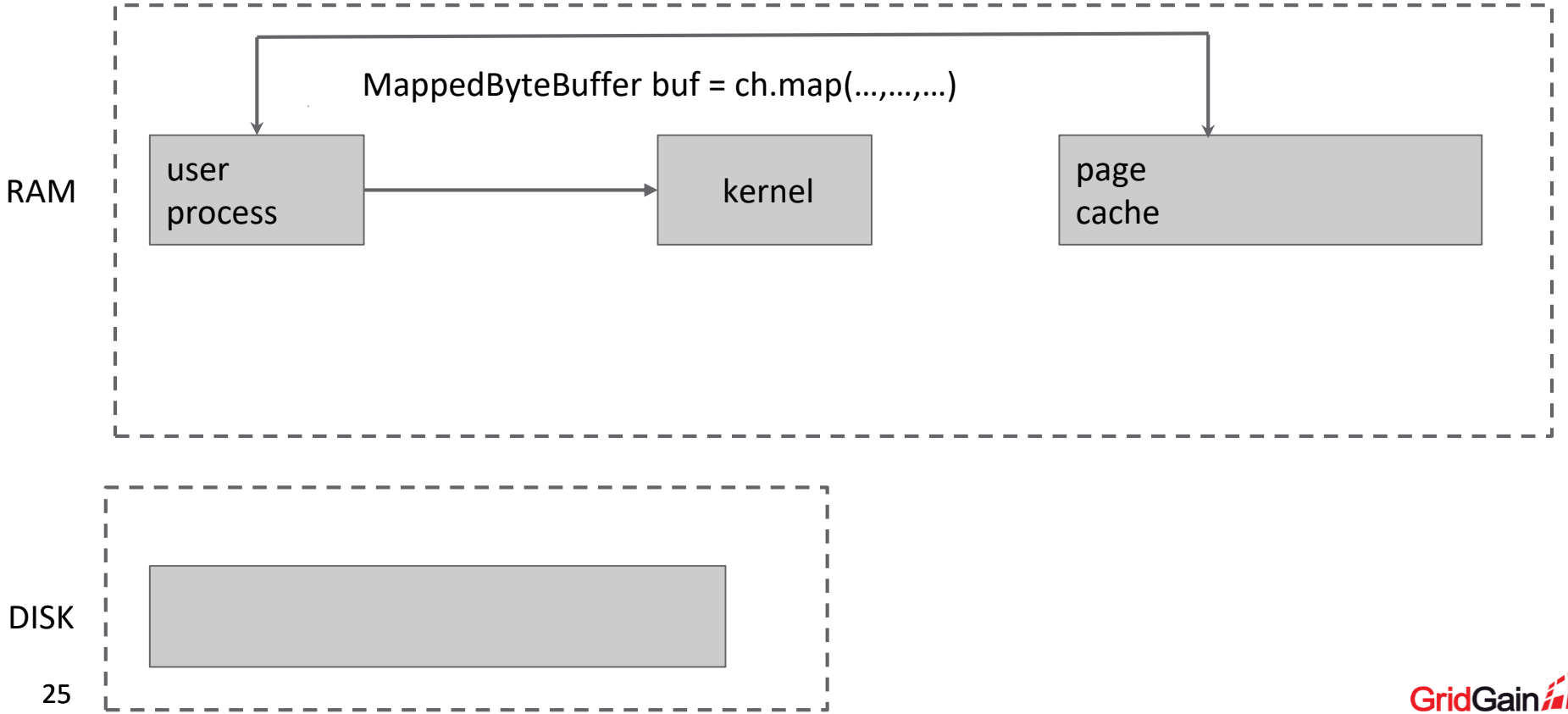


- При чтение одних и тех же страниц файла, каждый процесс копирует страницы page cache в свою виртуальную память
- Несколько процессов имеют идентичные копии страницы (неэффективное расходование vm)

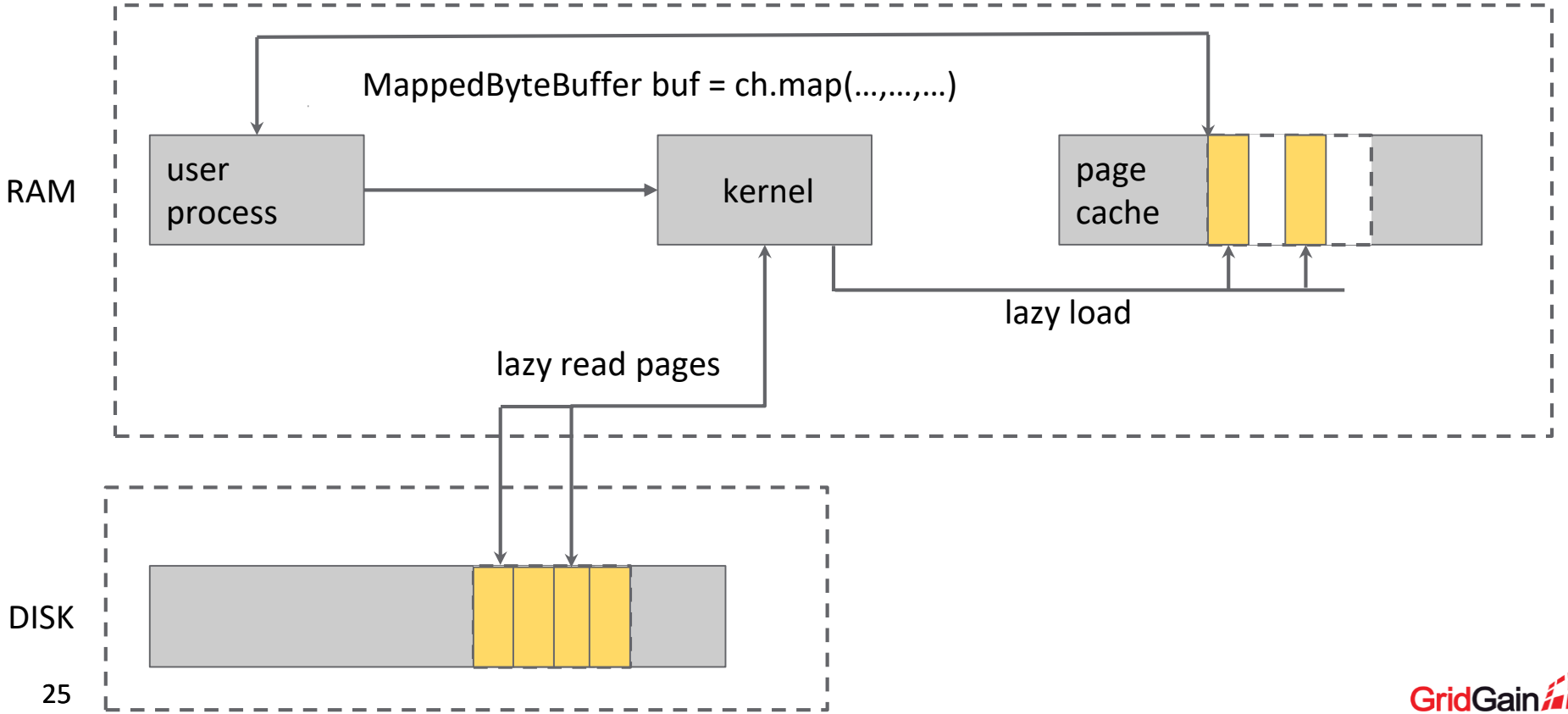
mmap



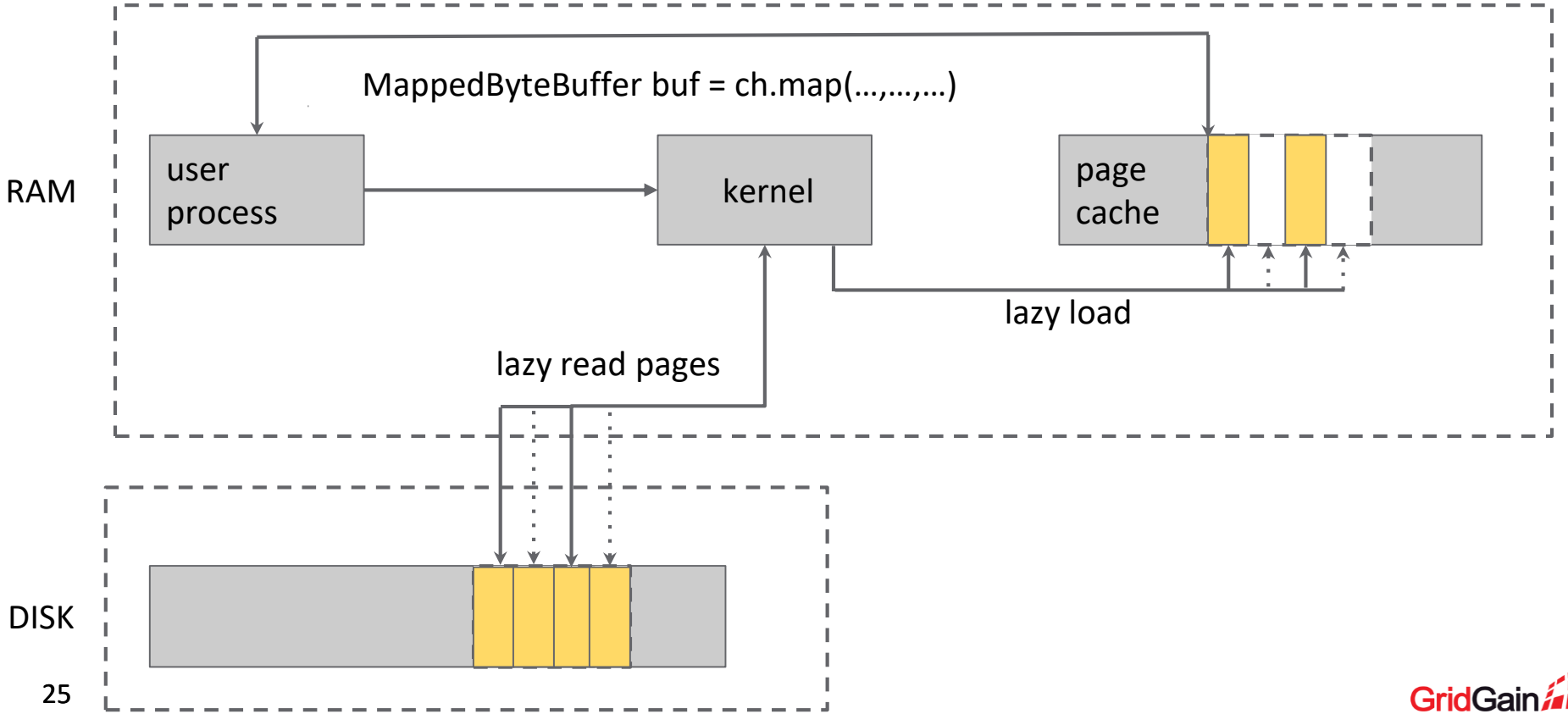
mmap



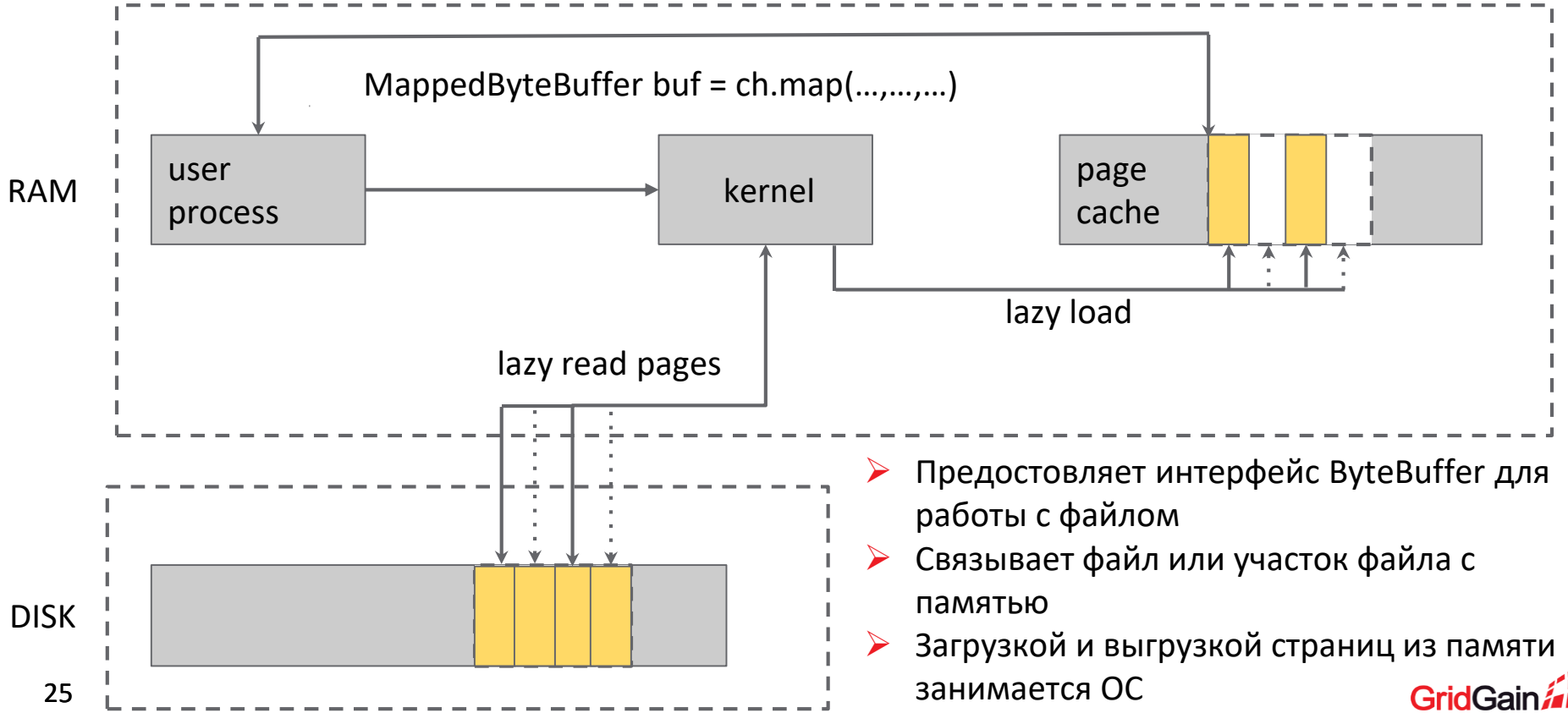
mmap



mmap



mmap



- Предостовляет интерфейс ByteBuffer для работы с файлом
- Связывает файл или участок файла с памятью
- Загрузкой и выгрузкой страниц из памяти занимается ОС

mmap

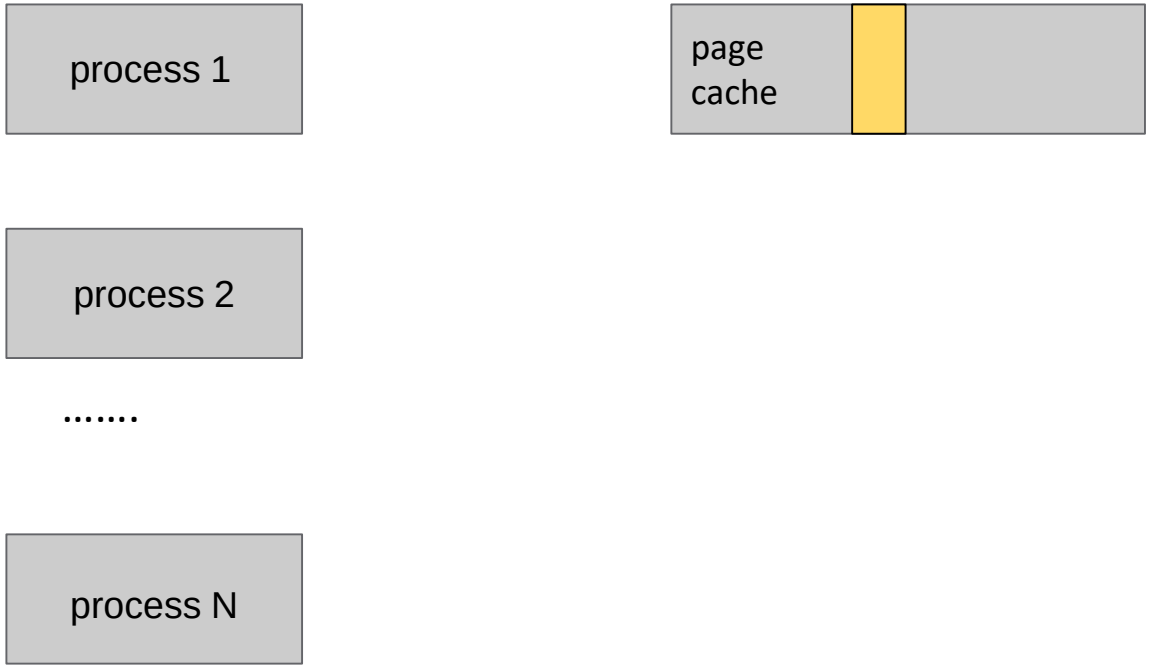
```
1 try (FileChannel ch = FileChannel.open(Paths.get("pathFile"), READ)) {
2     MappedByteBuffer mappedByteBuffer = ch.map(MapMode.READ_WRITE, 0, ch.size());
3
4     // Load data in memory if needed.
5     mappedByteBuffer.load();
6
7     // Get or put operations with file like a buffer.
8     ......
9
10    // Sync data in memory with disk.
11    map.force();
12 }
```

mmap

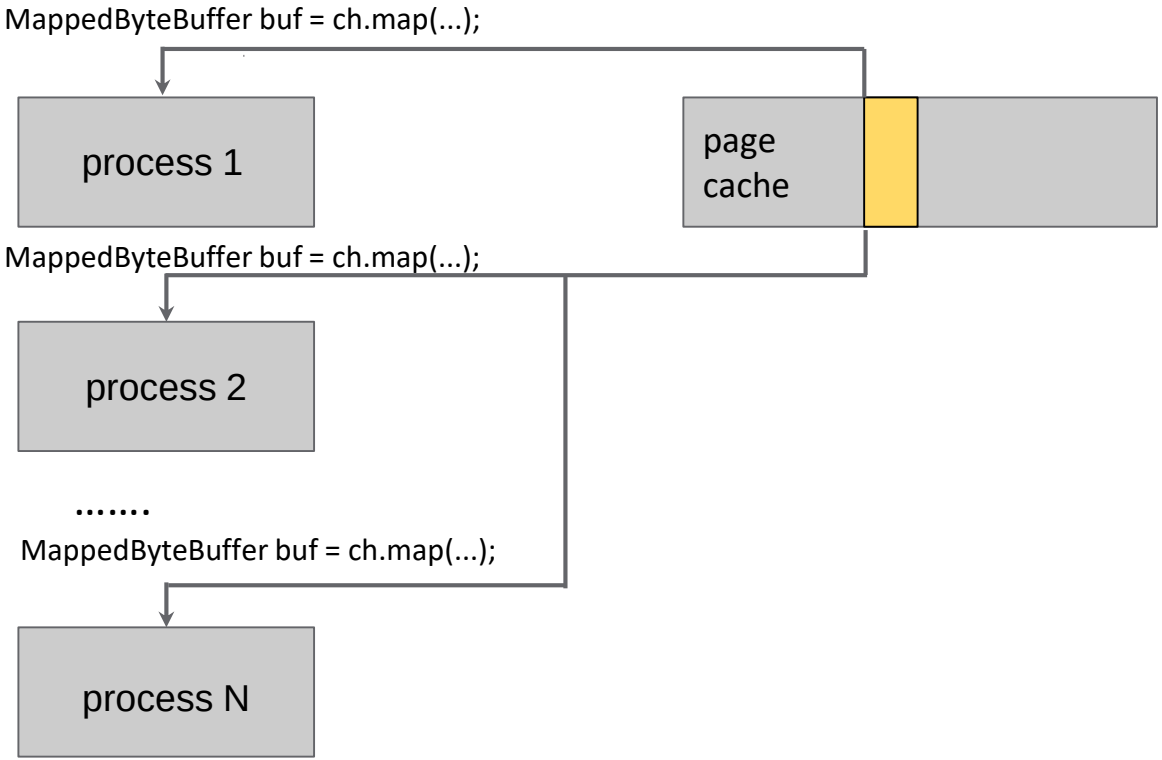
```
1 try (FileChannel ch = FileChannel.open(Paths.get("pathFile"), READ)) {
2     MappedByteBuffer mappedByteBuffer = ch.map(MapMode.READ_WRITE, 0, ch.size());
3
4     // Load data in memory if needed.
5     mappedByteBuffer.load();
6
7     // Get or put operations with file like a buffer.
8     ......
9
10    // Sync data in memory with disk.
11    map.force();
12 }
```

- Нет возможности получить IO ошибку
- Нельзя замапить больше 2GB (int)
- Существует лимит mmap файлов на процесс (max_map_count)
- vm.max_map_count setting in /etc/sysctl.conf

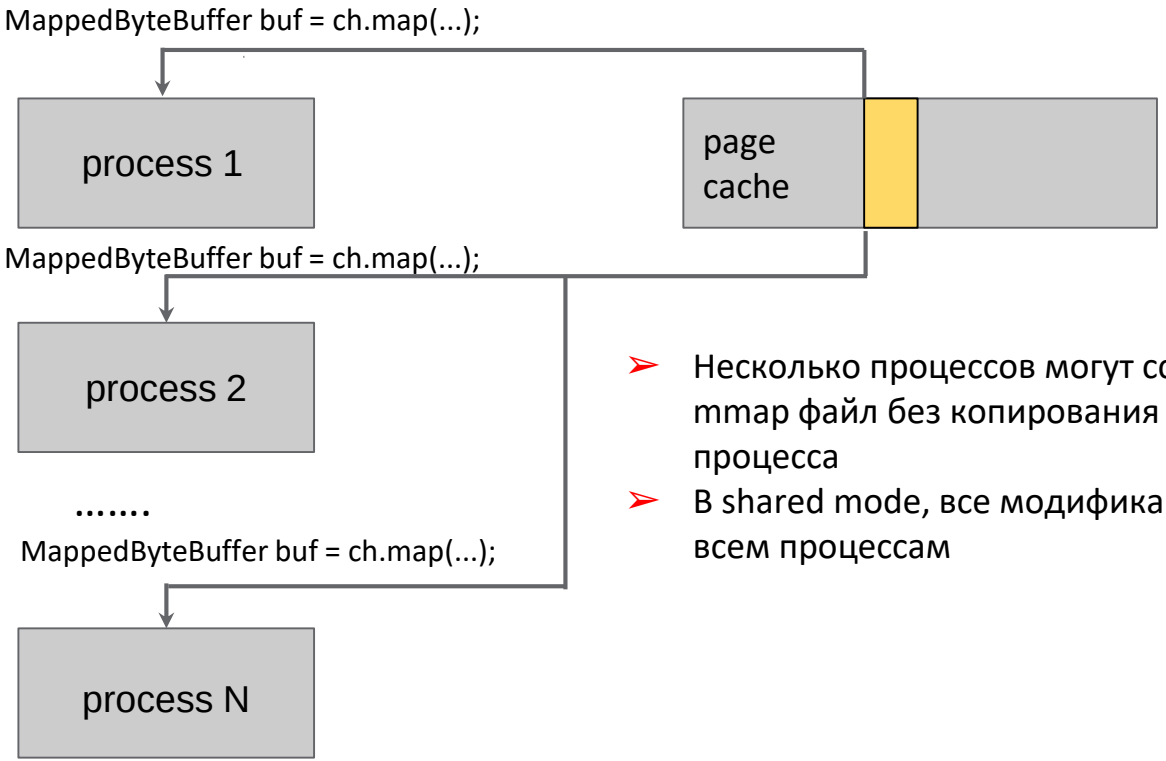
mmap (multiple process)



mmap (multiple process)

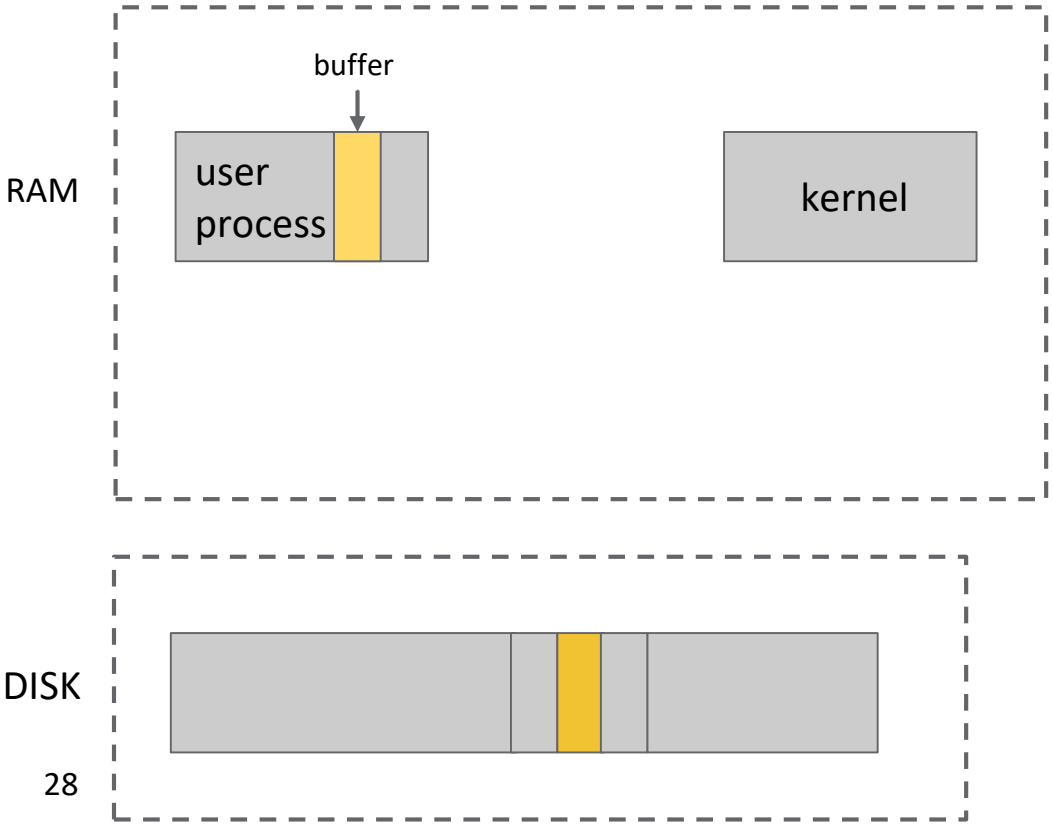


mmap (multiple process)

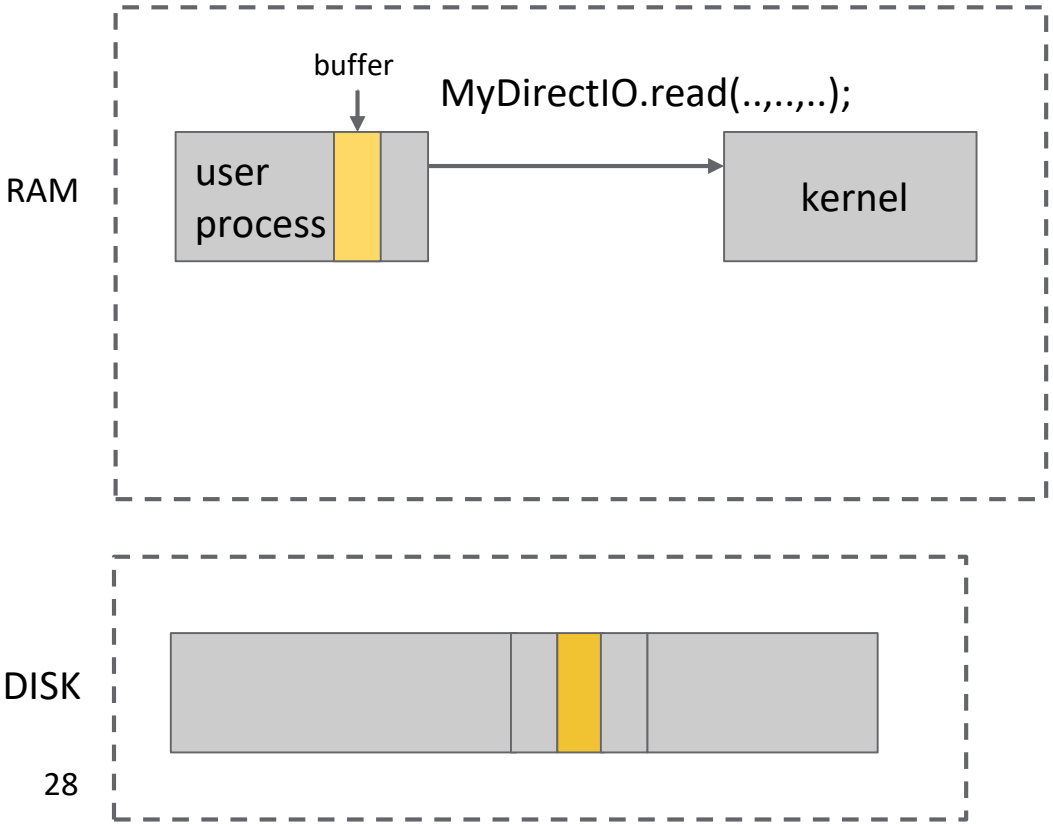


- Несколько процессов могут ссылаться на один mmap файл без копирования page cache в vm процесса
- В shared mode, все модификации файлов видны всем процессам

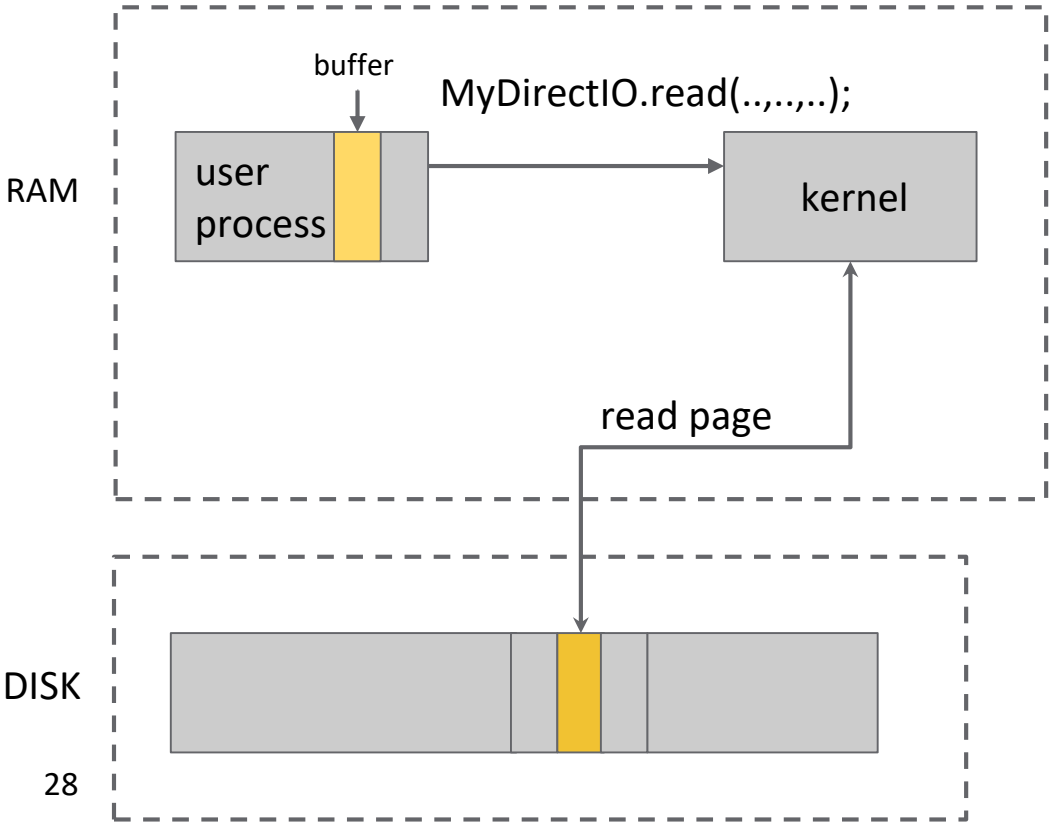
O_DIRECT



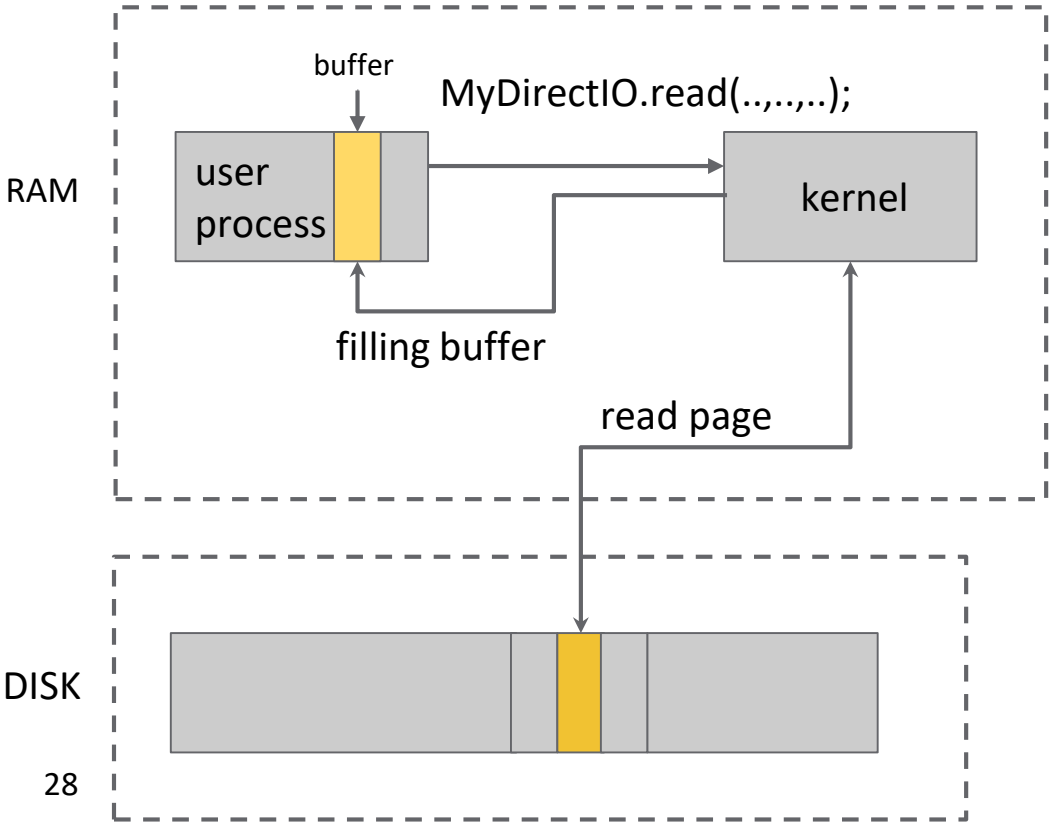
O_DIRECT



O_DIRECT

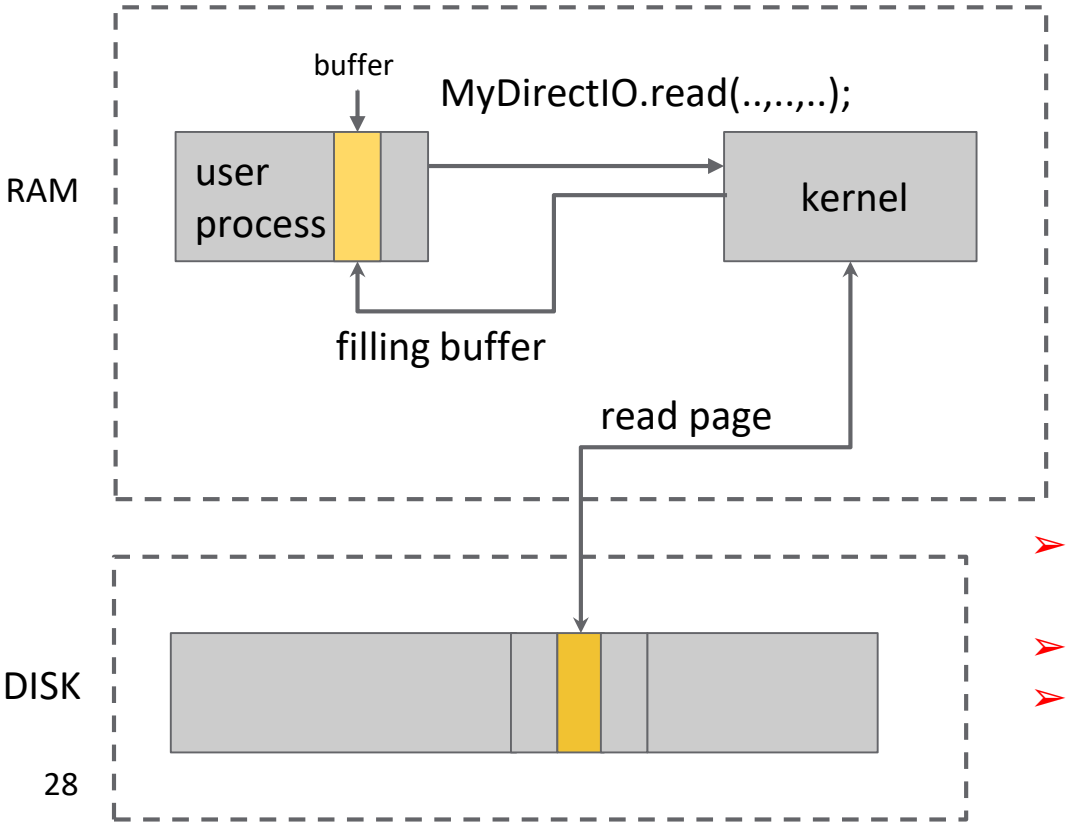


O_DIRECT



GridGain

O_DIRECT



- Читать/писать можно только выровненными блоками
- Ничего не кэшируется
- Нет предсказаний (readahead не работает)

Native IO

```
1 class MyNativeIO {
2     static {
3         Native.register(Platform.C_LIBRARY_NAME);
4     }
5     static native int open(String pathname, int flags, int mode);
6     static native int close(int fd);
7     static native int fsync(int fd);
8     static native NativeLong pwrite(int fd, Pointer buf,
9     NativeLong cnt, NativeLong off);
10    static native NativeLong write(int fd, Pointer buf, NativeLong cnt);
11    static native NativeLong pread(int fd, Pointer buf,
12    NativeLong cnt, NativeLong off);
13    static native NativeLong read(int fd, Pointer buf, NativeLong cnt);
14    static native int posix_memalign(PointerByReference memptr,
15    NativeLong alignment, NativeLong size);
16 }
```

Java Native Access (JNA) https://github.com/java-native-access/jna

O_DIRECT Example

```
 1 ByteBuffer buf = allocateDirectAlign(4096);
 2 // Filling buffer.
 3
 4 int flags =....|O_DIRECT | ....;
 5
 6 int fd = MyDirectIO.open(path, flags, mode);
 7
 8 Pointer ptr = new Pointer(bufferAddress(buf));
 9
10 int wt = MyDirectIO.write(fd, ptr, buf.capacity());
11
12 buf.clear();
13
14 int rd = MyDirectIO.read(fd, ptr, buf.capacity());
```

Java Native Access (JNA) https://github.com/java-native-access/jna

Что важно помнить

- mmap - позволяет избежать излишнего копирования в vm
  - лениво подгружает страницы в page cache
  - нет возможности получить IO exception
  - имеет лимиты на количество открытых файлов

Что важно помнить

- mmap - позволяет избежать излишнего копирования в vm
  - лениво подгружает страницы в page cache
  - нет возможности получить IO exception
  - имеет лимиты на количество открытых файлов
- o_direct flag - возможность читать/писать, минуя page cache

Что важно помнить

- mmap - позволяет избежать излишнего копирования в vm
  - лениво подгружает страницы в page cache
  - нет возможности получить IO exception
  - имеет лимиты на количество открытых файлов
- o_direct flag - возможность читать/писать, минуя page cache
  - читать можно только выровненные блоки

Что важно помнить

- mmap - позволяет избежать излишнего копирования в vm
  - лениво подгружает страницы в page cache
  - нет возможности получить IO exception
  - имеет лимиты на количество открытых файлов
- o_direct flag - возможность читать/писать, минуя page cache
  - читать можно только выровненные блоки
  - readahead не работает (readahead - часть работы page cache)

FileChannel Read/Write

```
 1 ByteBuffer buf = ByteBuffer.allocate(...);
 2
 3 try (FileChannel ch = FileChannel.open(Paths.get("pathFile1"), READ)) {
 4     ch.read(buf);
 5 }
 6
 7 buf.flip();
 8
 9 try (FileChannel ch = FileChannel.open(Paths.get("pathFile2"), WRITE)) {
10     ch.write(buf);
11
12     ch.force(true);
13 }
```

Native Call Read/Write

static native int read0(FileDescriptor fd, **long address**, int len) throws IOException;

Native Call Read/Write

static native int read0(FileDescriptor fd, **long address**, int len) throws IOException;

static native int write0(FileDescriptor fd, **long address**, int len) throws IOException;

Native Call Read/Write

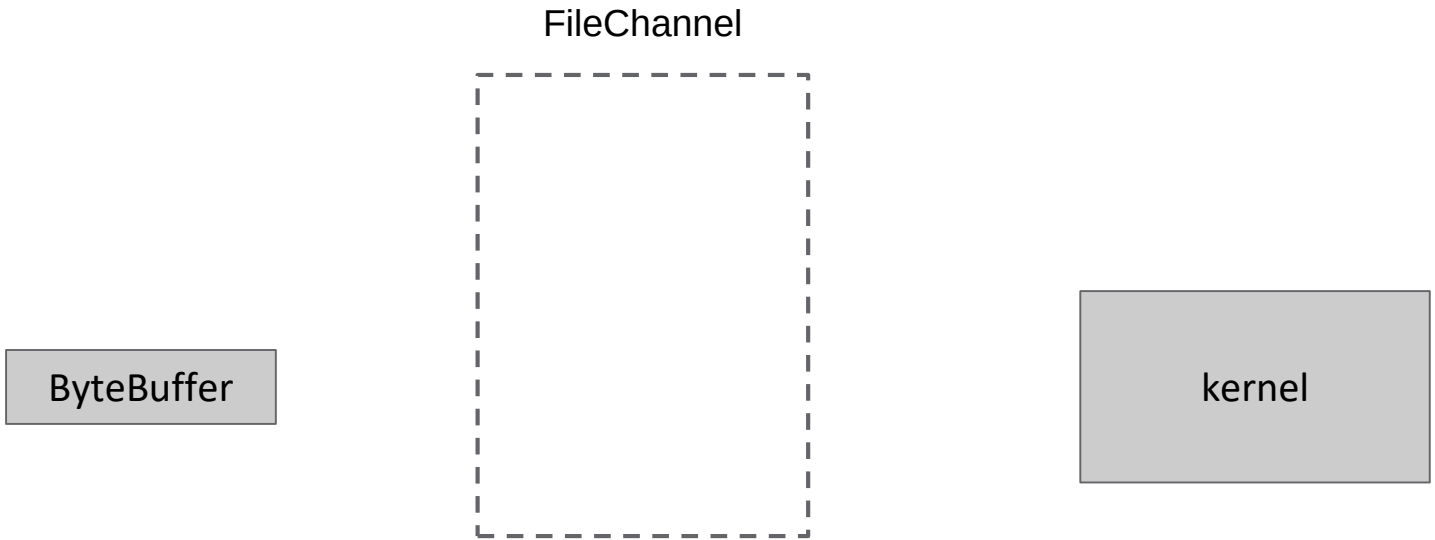
static native int read0(FileDescriptor fd, **long address**, int len) throws IOException;

static native int write0(FileDescriptor fd, **long address**, int len) throws IOException;

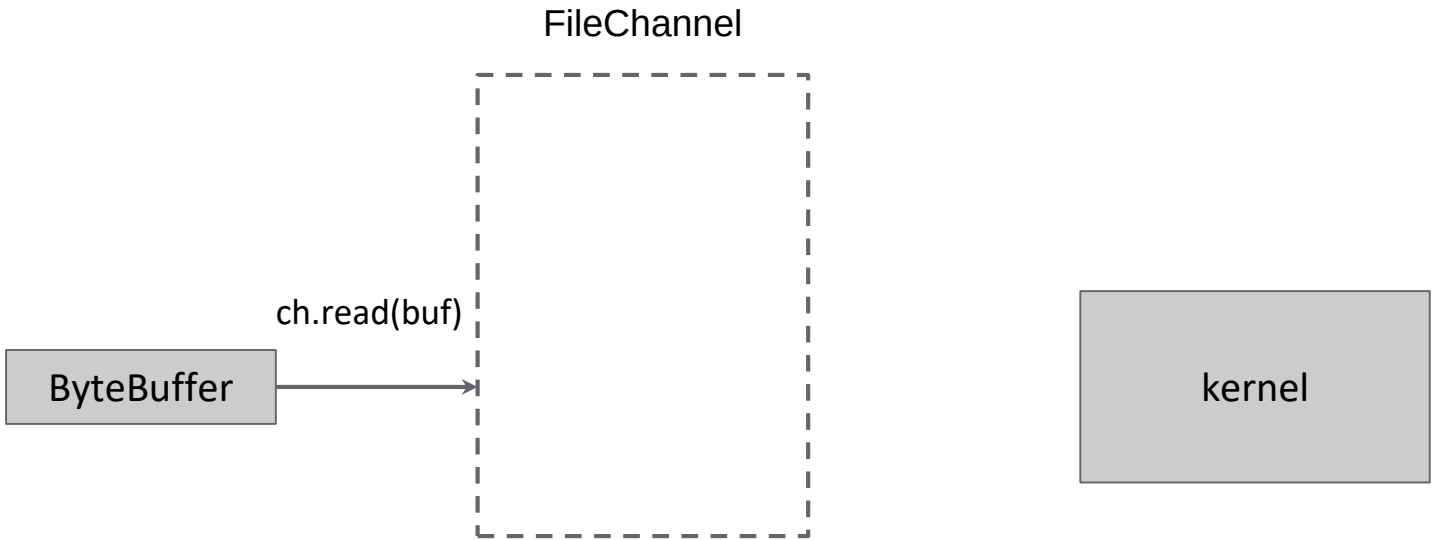
- FileDescriptor - идентификатор открытого файла в рамках процесса, выдаваемый kernel после открытия файла

  native **int** open0(long pathAddress, int flags, int mode)

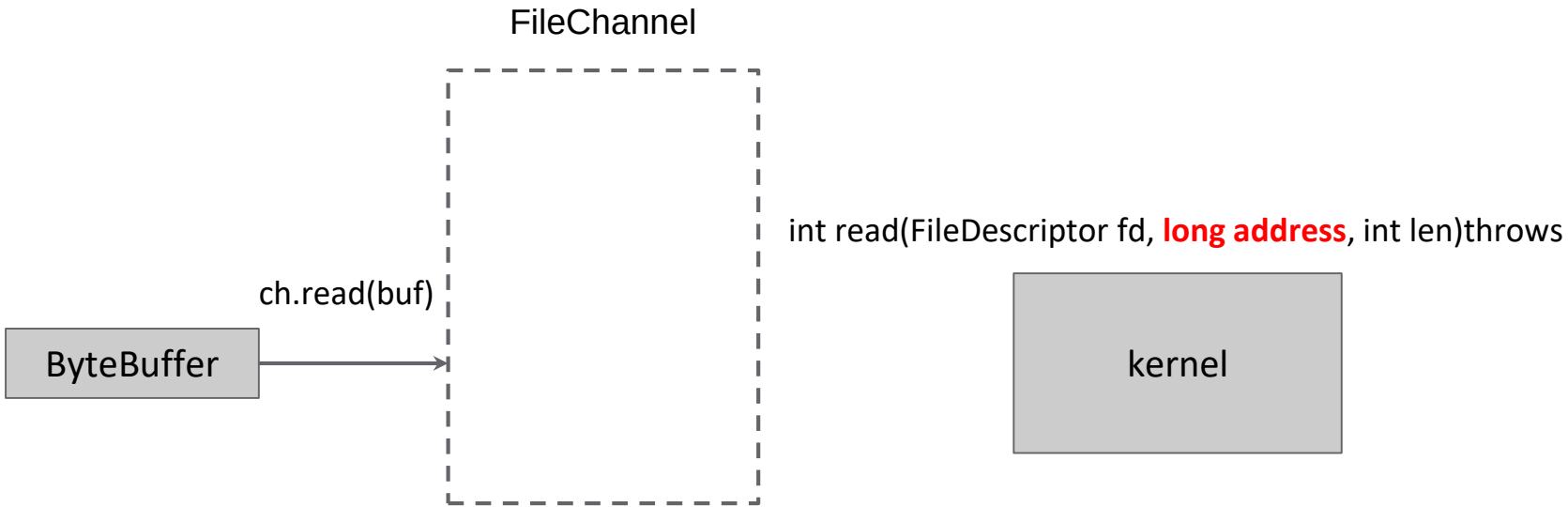
FileChannel Read/Write



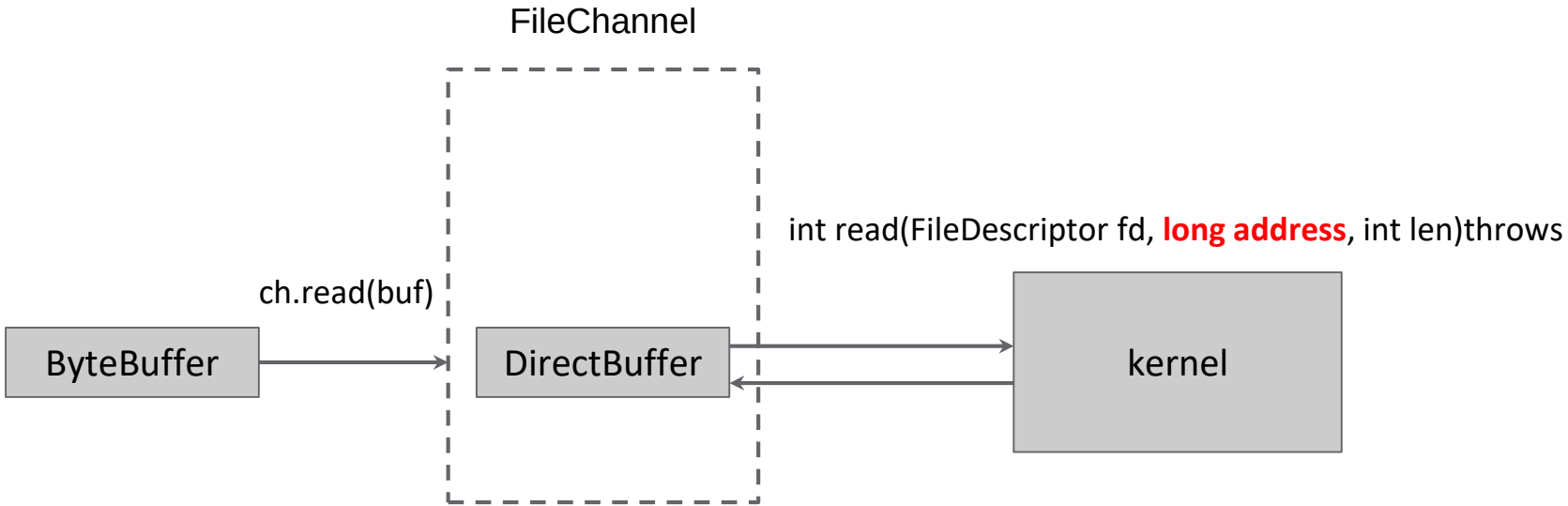
FileChannel Read/Write



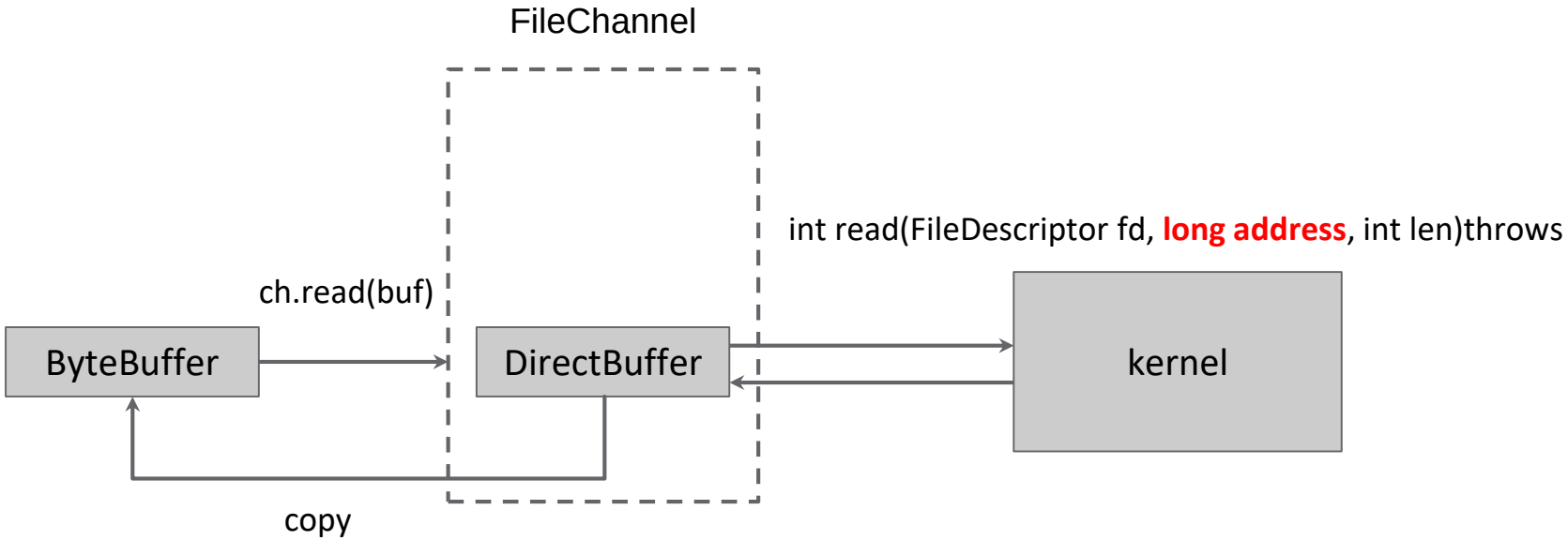
FileChannel Read/Write



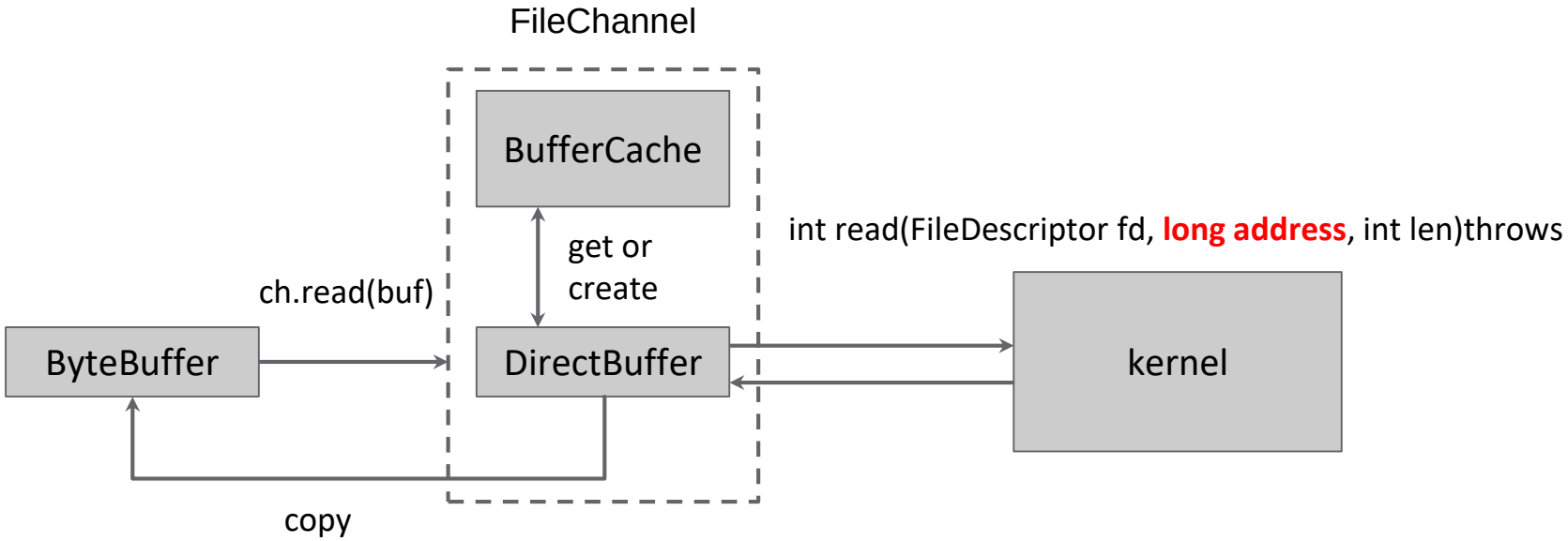
FileChannel Read/Write



FileChannel Read/Write



FileChannel Read/Write



FileChannel Read/Write

```
1 int read (ByteBuffer dst,long position){
2     ....
3
4     if (dst instanceof DirectBuffer)
5         return readIntoNativeBuffer(fd, dst, position, nd);
6
7     ....
8 }
```

FileChannel Read/Write

```
1 int read (ByteBuffer dst,long position){
2     ....
3
4     if (dst instanceof DirectBuffer)
5         return readIntoNativeBuffer(fd, dst, position, nd);
6
7     ....
8 }
```

- Если directBuffer, читаем напрямую в него

FileChannel Read/Write

```
1  int read (ByteBuffer dst,long position){
2      ....
3      ByteBuffer bb = Util.getTemporaryDirectBuffer(dst.remaining());
4
5      try {
6          int n = readIntoNativeBuffer(fd, bb, position, nd);
7
8          bb.flip();
9          if (n > 0)
10             dst.put(bb);
11         return n;
12     }
13     finally {
14         Util.offerFirstTemporaryDirectBuffer(bb);
15     }
16     ....
17 }
```

FileChannel Read/Write

```
1  int read (ByteBuffer dst,long position){
2      ....
3      ByteBuffer bb = Util.getTemporaryDirectBuffer(dst.remaining());
4
5      try {
6          int n = readIntoNativeBuffer(fd, bb, position, nd);
7
8          bb.flip();
9          if (n > 0)
10             dst.put(bb);
11         return n;
12     }
13     finally {
14         Util.offerFirstTemporaryDirectBuffer(bb);
15     }
16     ....
17 }
```

- Берем directBuffer из bufferCache и читаем в него

FileChannel Read/Write

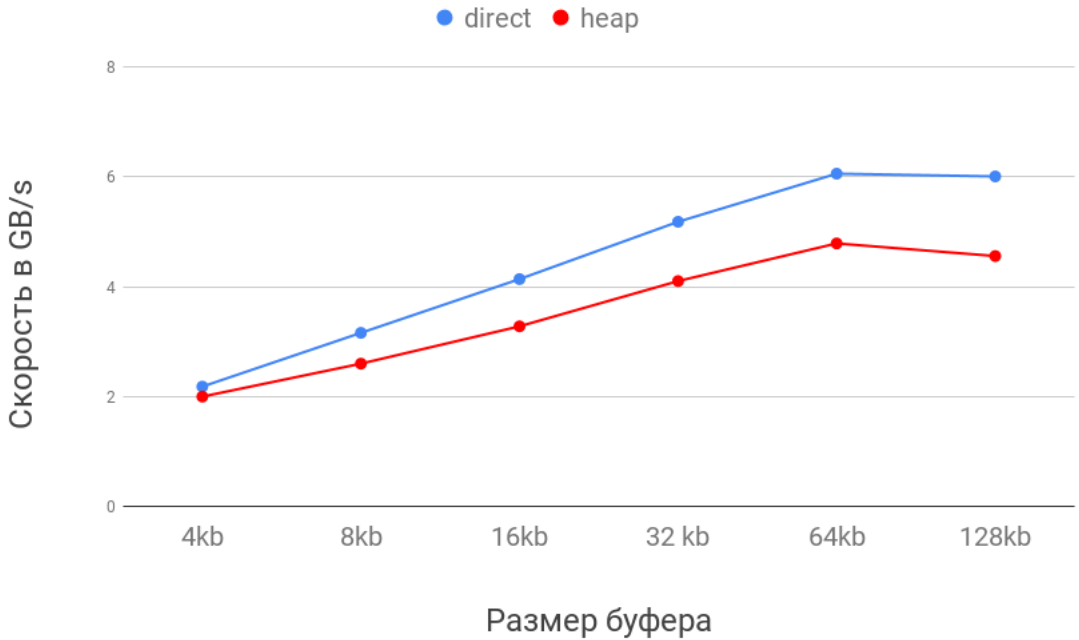
```
1 int read (ByteBuffer dst,long position){
2     ....
3     ByteBuffer bb = Util.getTemporaryDirectBuffer(dst.remaining());
4
5     try {
6         int n = readIntoNativeBuffer(fd, bb, position, nd);
7
8         bb.flip();
9         if (n > 0)
10            dst.put(bb);
11        return n;
12    }
13    finally {
14        Util.offerFirstTemporaryDirectBuffer(bb);
15    }
16    ....
17 }
```

- Берем directBuffer из bufferCache и читаем в него
- Копируем прочитанные байты в пользовательский buffer

FileChannel Read/Write

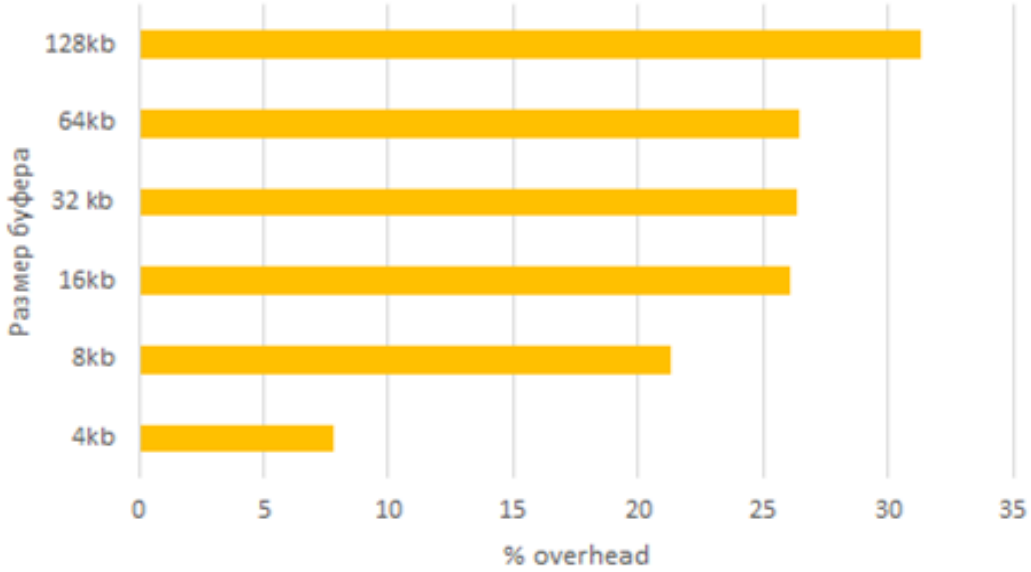
```
1 int read (ByteBuffer dst,long position){
2     ....
3     ByteBuffer bb = Util.getTemporaryDirectBuffer(dst.remaining());
4
5     try {
6         int n = readIntoNativeBuffer(fd, bb, position, nd);
7
8         bb.flip();
9         if (n > 0)
10            dst.put(bb);
11        return n;
12    }
13    finally {
14        Util.offerFirstTemporaryDirectBuffer(bb);
15    }
16    ....
17 }
```

- Берем directBuffer из bufferCache и читаем в него
- Копируем прочитанные байты в пользовательский buffer
- Возвращаем directBuffer в bufferCache



	4kb	8kb	16kb	32 kb	64kb	128kb
heap	530780.273	341550.363	215315.824	134497.58	78575.852	37437.151
direct	572147.154	414411.094	271425.203	169896.952	99333.937	49165.052

Больше буфер - больше overhead



GridGain

Direct buffer

```
ByteBuffer buf = ByteBuffer.allocateDirect(...);
```

Direct buffer

```
ByteBuffer buf = ByteBuffer.allocateDirect(...);
```

- Память не освобождается, если вызвать buf.clear();
- Память освобождается только когда GC удаляет объект buf
- Нет прямой возможности освободить память – только через reflection

BufferCache

- BufferCache создается для каждого потока (thread local)

BufferCache

- BufferCache создается для каждого потока (thread local)
- BufferCache capacity - TEMP_BUF_POOL_SIZE = IOUtil.IOV_MAX (1024);

BufferCache

- BufferCache создается для каждого потока (thread local)
- BufferCache capacity - TEMP_BUF_POOL_SIZE = IOUtil.IOV_MAX (1024);
- jdk.nio.maxCachedBufferSize - максимальный размер byteBuffer в BufferCache,

BufferCache

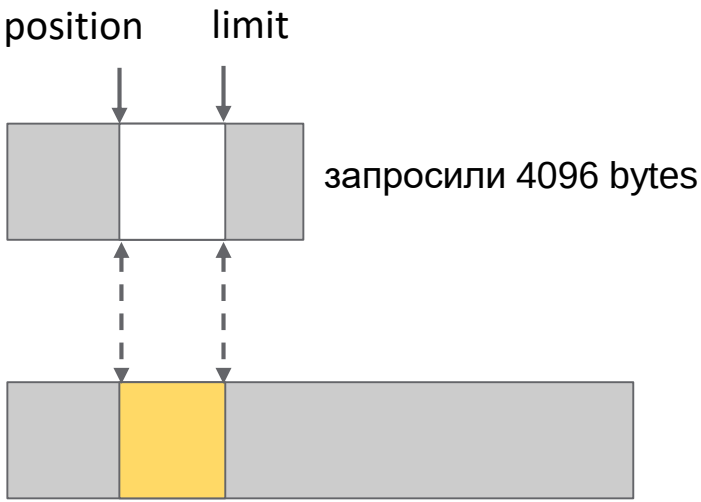
- BufferCache создается для каждого потока (thread local)
- BufferCache capacity - TEMP_BUF_POOL_SIZE = IOUtil.IOV_MAX (1024);
- jdk.nio.maxCachedBufferSize - максимальный размер byteBuffer в BufferCache, если буфер будет большего размера, то он не будет сохраняться в BufferCache

Неполное чтение

```
int read(FileDescriptor fd, long address, int len) throws IOException;
```

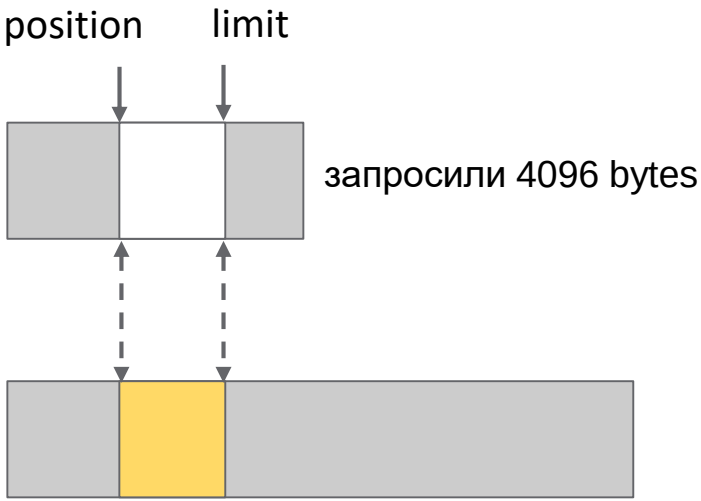
Неполное чтение

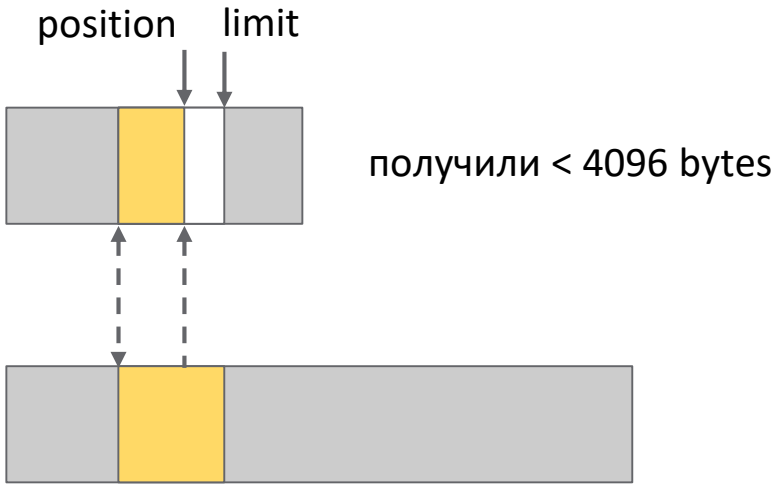
```
int read(FileDescriptor fd, long address, int len) throws IOException;
```



Неполное чтение

```
int read(FileDescriptor fd, long address, int len) throws IOException;
```



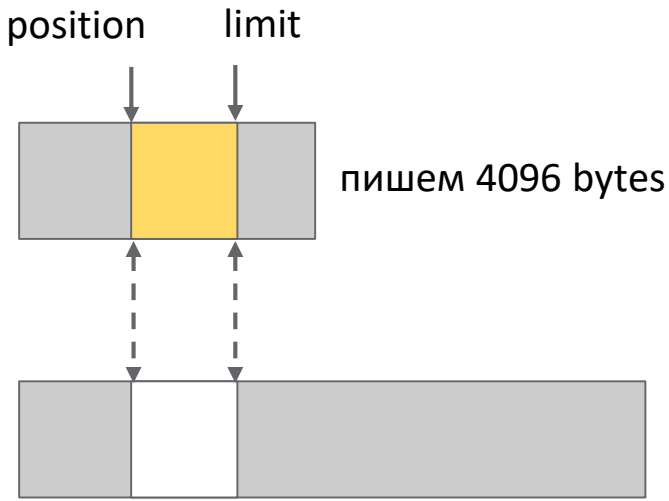


Неполная запись

```
int write(FileDescriptor fd, long address, int len) throws IOException;
```

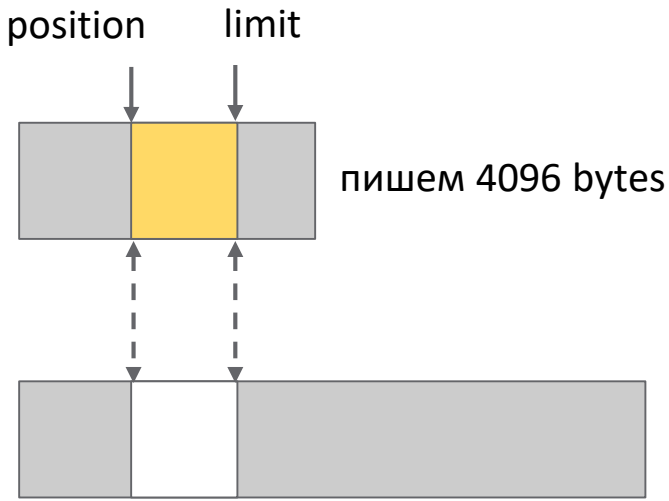
Неполная запись

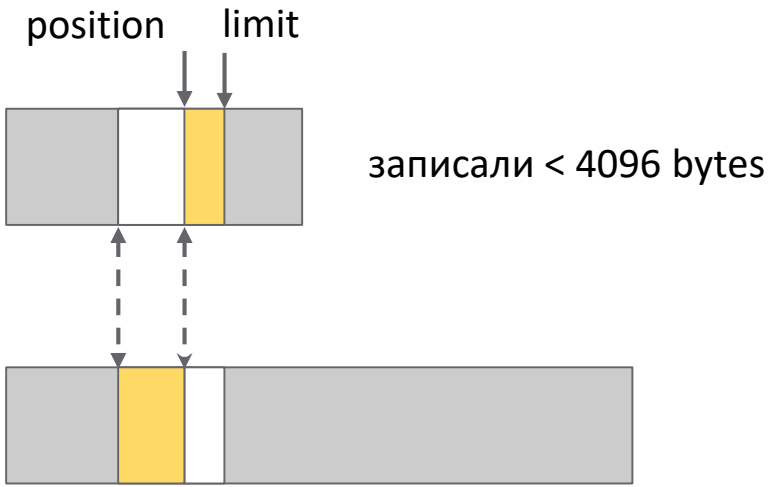
int write(FileDescriptor fd, long address, int len) throws IOException;



Неполная запись

int write(FileDescriptor fd, long address, int len) throws IOException;





Read

```
1 try (FileChannel ch = FileChannel.open(file, READ)) {
2     ch.read(buf);
3 }
```

Read

```
1 try (FileChannel ch = FileChannel.open(file, READ)) {
2     ch.read(buf);
3 }
```

Read

```
1 try (FileChannel ch = FileChannel.open(file, READ)) {
2     ch.read(buf);
3 }
```

```
1 try (FileChannel ch = FileChannel.open(file, READ)) {
2     do {
3         int bytes = ch.read(buf);
4         if (bytes <= 0)
5             break;
6     }
7     while (buf.position() < buf.limit());
8 }
```

Write

```
1 try (FileChannel ch = FileChannel.open(file, WRITE)) {
2     ch.write(buf);
3 }
```

Write

```
1 try (FileChannel ch = FileChannel.open(file, WRITE)) {
2     ch.write(buf);
3 }
```

Write

```
1 try (FileChannel ch = FileChannel.open(file, WRITE)) {
2     ch.write(buf);
3 }
```

```
1 try (FileChannel ch = FileChannel.open(file, WRITE)) {
2     while (buf.hasRemaining()) {
3         int bytes = ch.write(buf);
4         if (bytes <= 0)
5             break;
6     }
7     ch.force(true);
8 }
```

Что важно помнить

- Чтение и запись возможны только через direct буферы

Что важно помнить

- Чтение и запись возможны только через direct буферы
- Важно проверять, сколько байт прочитано/записано

Прерывание потоков (interrupt)

```
 1 FileChannel ch = FileChannel.open(file, READ);
 2
 3 Thread th1:{
 4    while (....){
 5        ch.read(buf1);
 6    }
 7 }
 8
 9 Thread th2 :{
10     while (....){
11        ch.read(buf2);
12    }
13 }
```

Прерывание потоков (interrupt)

```
 1 FileChannel ch = FileChannel.open(file, READ);
 2
 3 Thread th1:{
 4    while (....){
 5        ch.read(buf1);
 6    }
 7 }
 8
 9 Thread th2 :{
10     while (....){
11        ch.read(buf2);
12    }
13 }
```

th2.interrupt(); - что произойдет?

Прерывание потоков (interrupt)

```
 1 FileChannel ch = FileChannel.open(file, READ);
 2
 3 Thread th1:{
 4    while (....){
 5        ch.read(buf1);
 6    }
 7 }
 8
 9 Thread th2 :{
10     while (....){
11        ch.read(buf2);
12    }
13 }
```

th2.interrupt(); - что произойдет?

ch - будет закрыт

Прерывание потоков (interrupt)

```
1 FileChannel ch = FileChannel.open(file, READ);
2
3 Thread th1:{
4    while (....){
5        ch.read(buf1);
6    }
7 }
8
9 Thread th2 :{
10    while (....){
11       ch.read(buf2);
12   }
13 }
```

th2.interrupt(); - что произойдет?

ch - будет закрыт

- При многопоточном доступе к filechannel, прерывание любого потока закрывает filechannel
- Каждый поток при следующей операции получит ClosedByInterruptException
- FileChannel extends AbstractInterruptibleChannel

Прерывание потоков (interrupt)

```
 1 public int read (ByteBuffer dst) throws IOException {
 2     ...
 3     try {
 4         begin();
 5         ...
 6         IOUtil.read(...,dst,...);
 7         ...
 8     }
 9     finally {
10         ...
11         end(n > 0);
12         ...
13     }
14 }
```

Прерывание потоков (interrupt)

```
 1 protected final void begin () {
 2     if (interruptor == null) {
 3         interruptor = new Interruptible() {
 4             public void interrupt(Thread target) {
 5                 ...
 6                 AbstractInterruptibleChannel.this.implCloseChannel();
 7                 ...
 8             }
 9         };
10     }
11     blockedOn(interruptor);
12     ...
13 }
```

Открываем много файлов

```
1 int filesNumber = ????;
2
3 for (int i = 0; i < filesNumber; i++) {
4     FileChannel.open(path, CREATE, READ, WRITE);
```

Открываем много файлов

```
1 int filesNumber = ????;
2
3 for (int i = 0; i < filesNumber; i++) {
4     FileChannel.open(path, CREATE, READ, WRITE);
```

- Базы данных могут открывать сотни, тысячи файлов
- Зависимые jar библиотеки могут также работать с файлами

Открывается много файлов

java.nio.file.FileSystemException: /Users/dgovorukhin/workspace/projects/disk-io/file/file10211.bin: Too many open files

at sun.nio.fs.UnixException.translateToIOException(UnixException.java:91)

at sun.nio.fs.UnixException.rethrowAsIOException(UnixException.java:102)

at sun.nio.fs.UnixException.rethrowAsIOException(UnixException.java:107)

at sun.nio.fs.UnixFileSystemProvider.newFileChannel(UnixFileSystemProvider.java:177)

at java.nio.channels.FileChannel.open(FileChannel.java:287)

at java.nio.channels.FileChannel.open(FileChannel.java:335)

at io.dgovorukhin.diskio.AbstractExample.createChannel(AbstractExample.java:15)

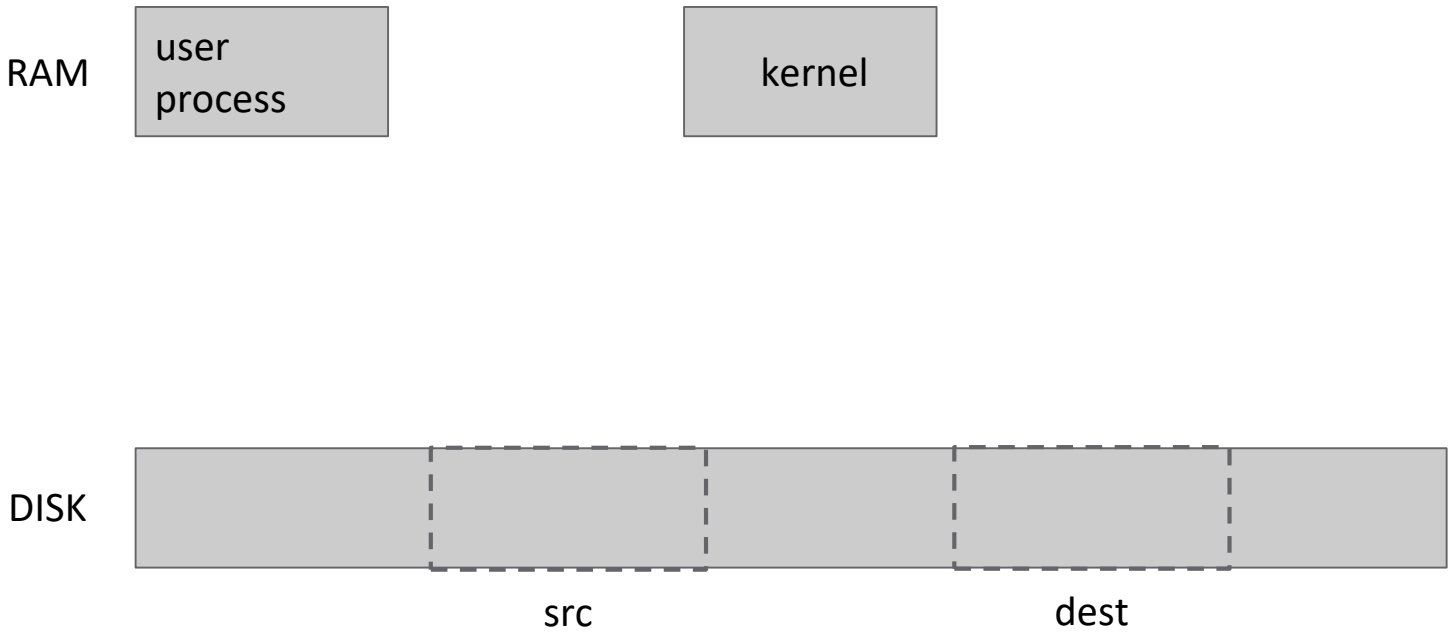
at io.dgovorukhin.diskio.ToManyOpenFilesExmaple.main(ToManyOpenFilesExmaple.java:16)

/etc/sysctl.conf добавляем fs.file-max=100000

Копирование файлов

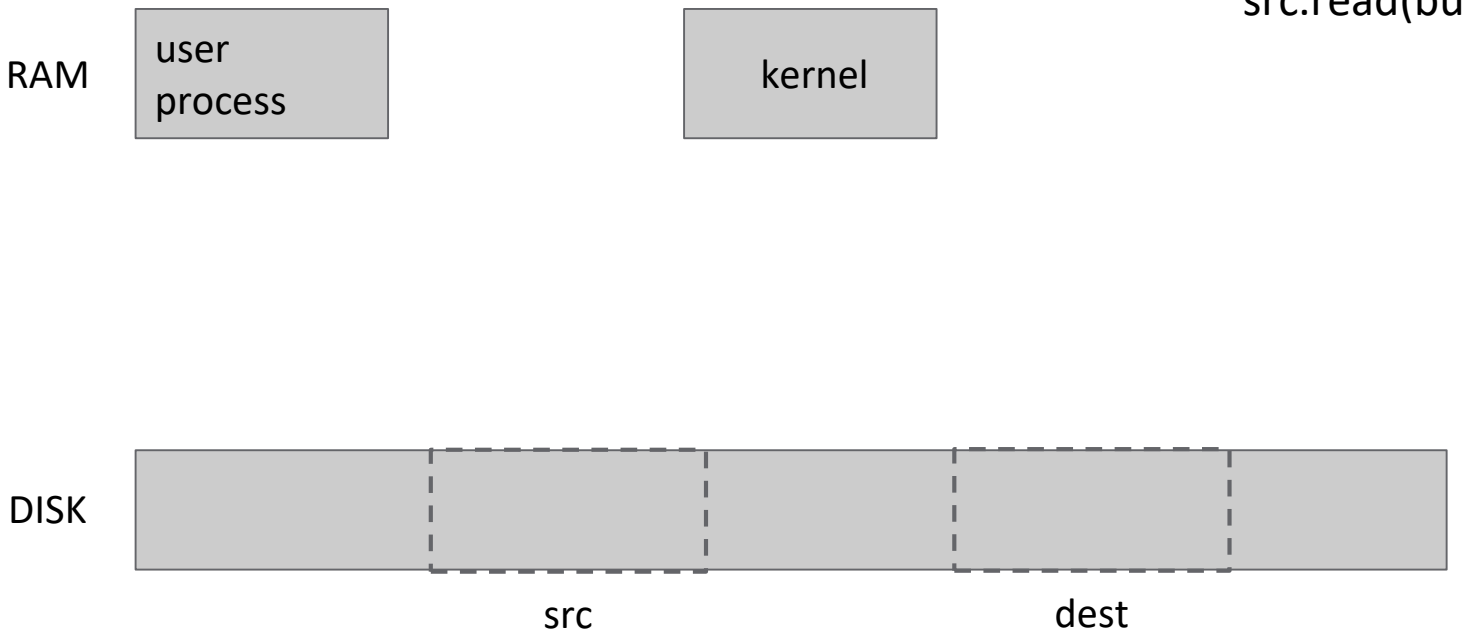
```
 1 try (FileChannel src = FileChannel.open(file1, READ);
 2       FileChannel dest = FileChannel.open(file2, WRITE)
 3 ) {
 4     while (....){
 5         src.read(buf);
 6         buf.flip();
 7         dest.write(buf);
 8         buf.clear();
 9     }
10 }
```

Копирование

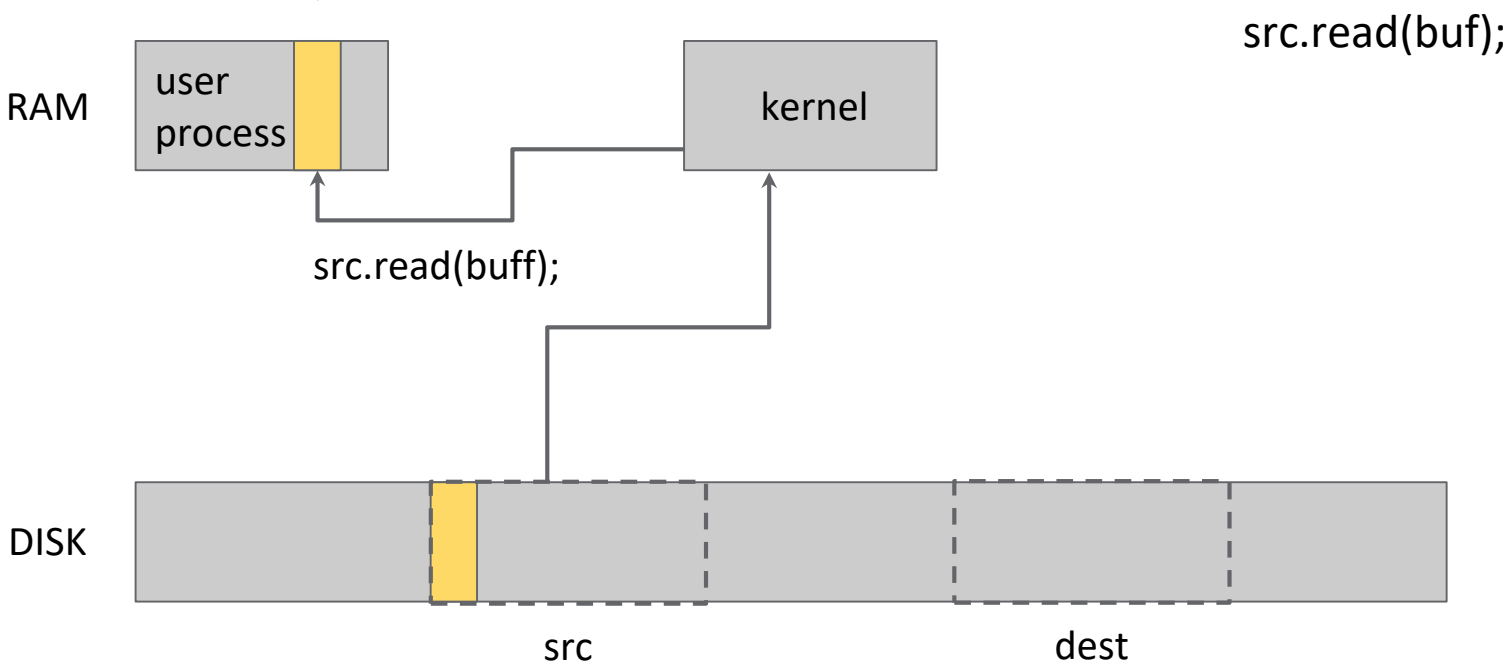


Копирование

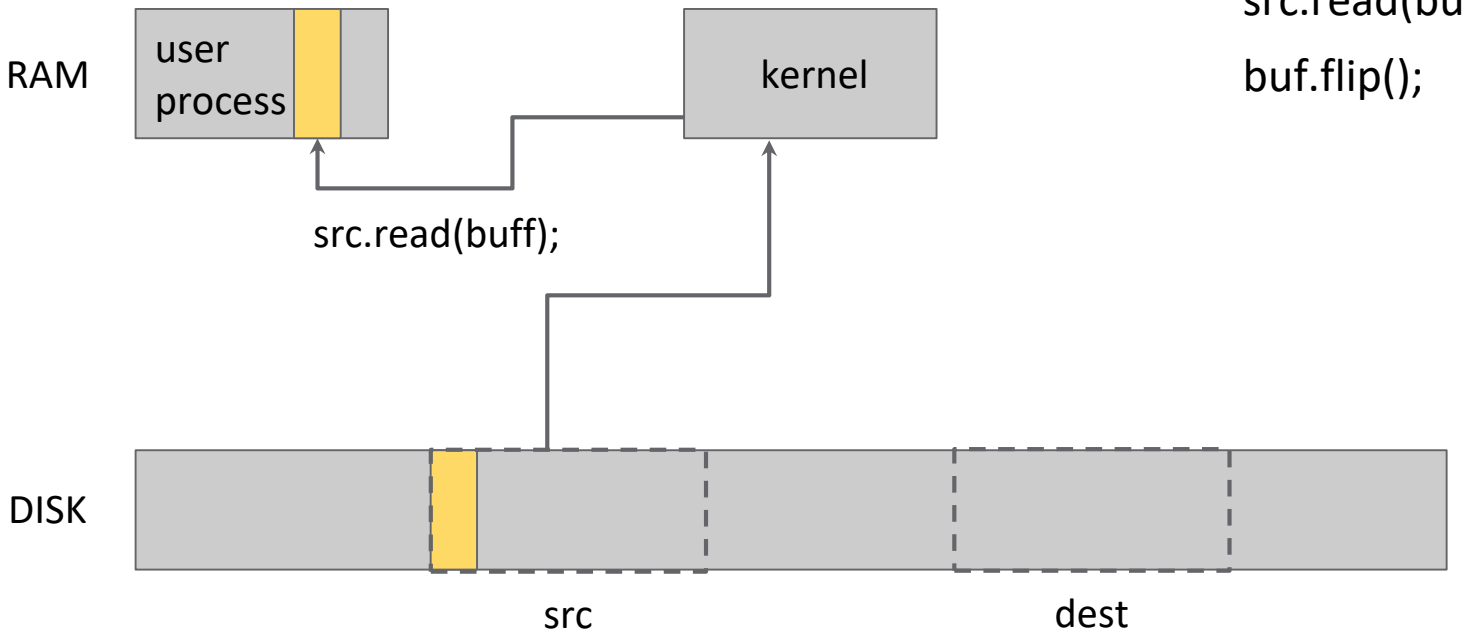
src.read(buf);



Копирование

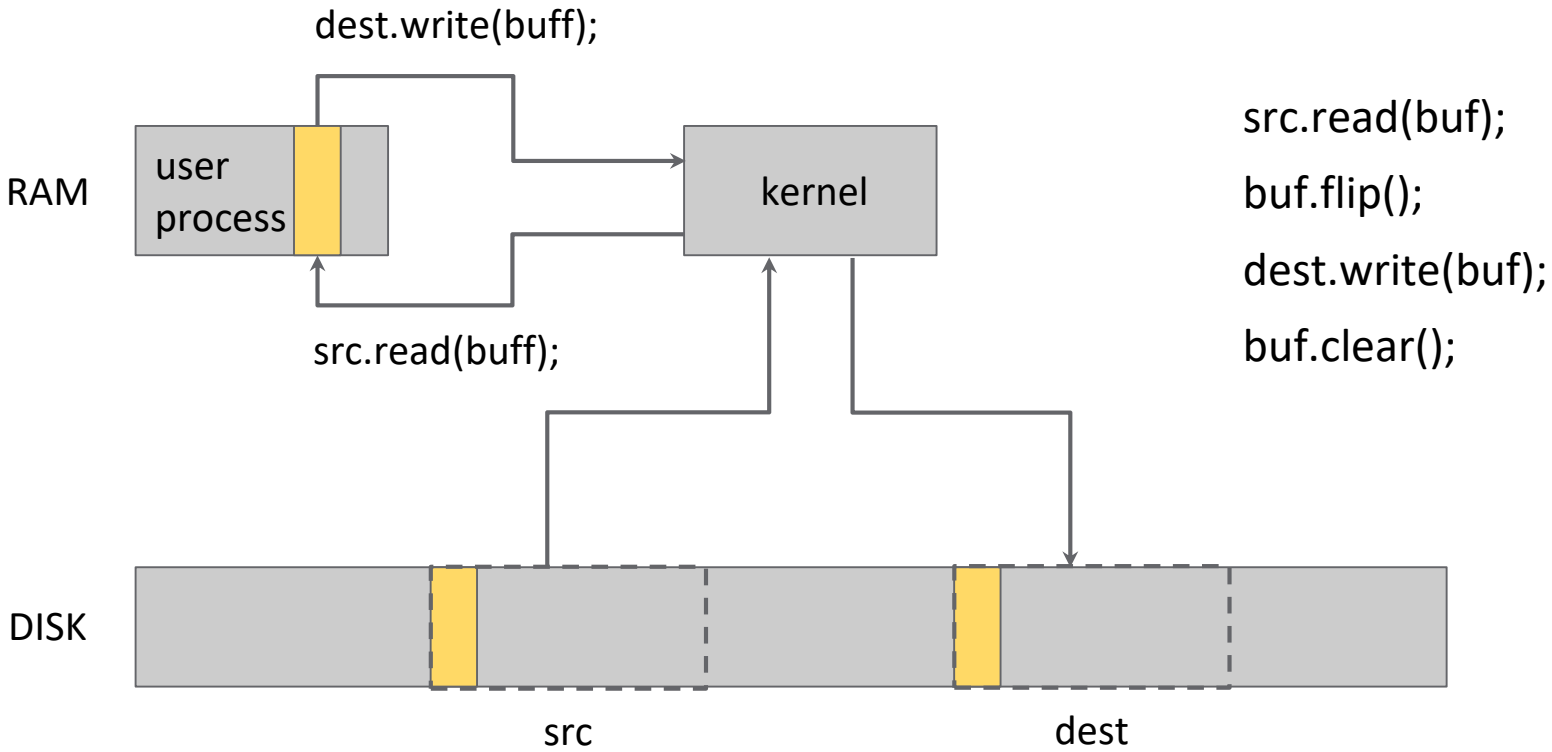


Копирование

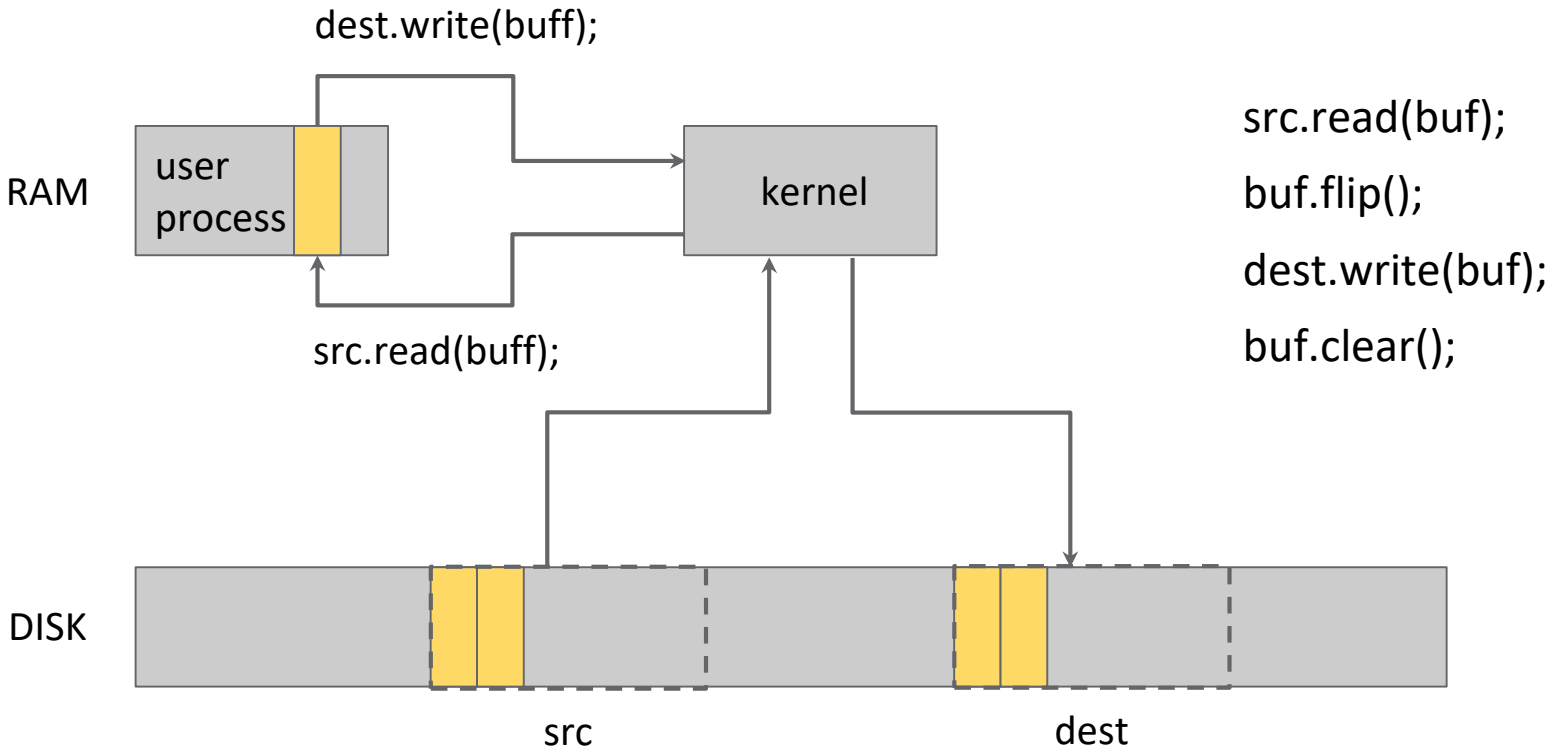



```
src.read(buf);
buf.flip();
```

Копирование

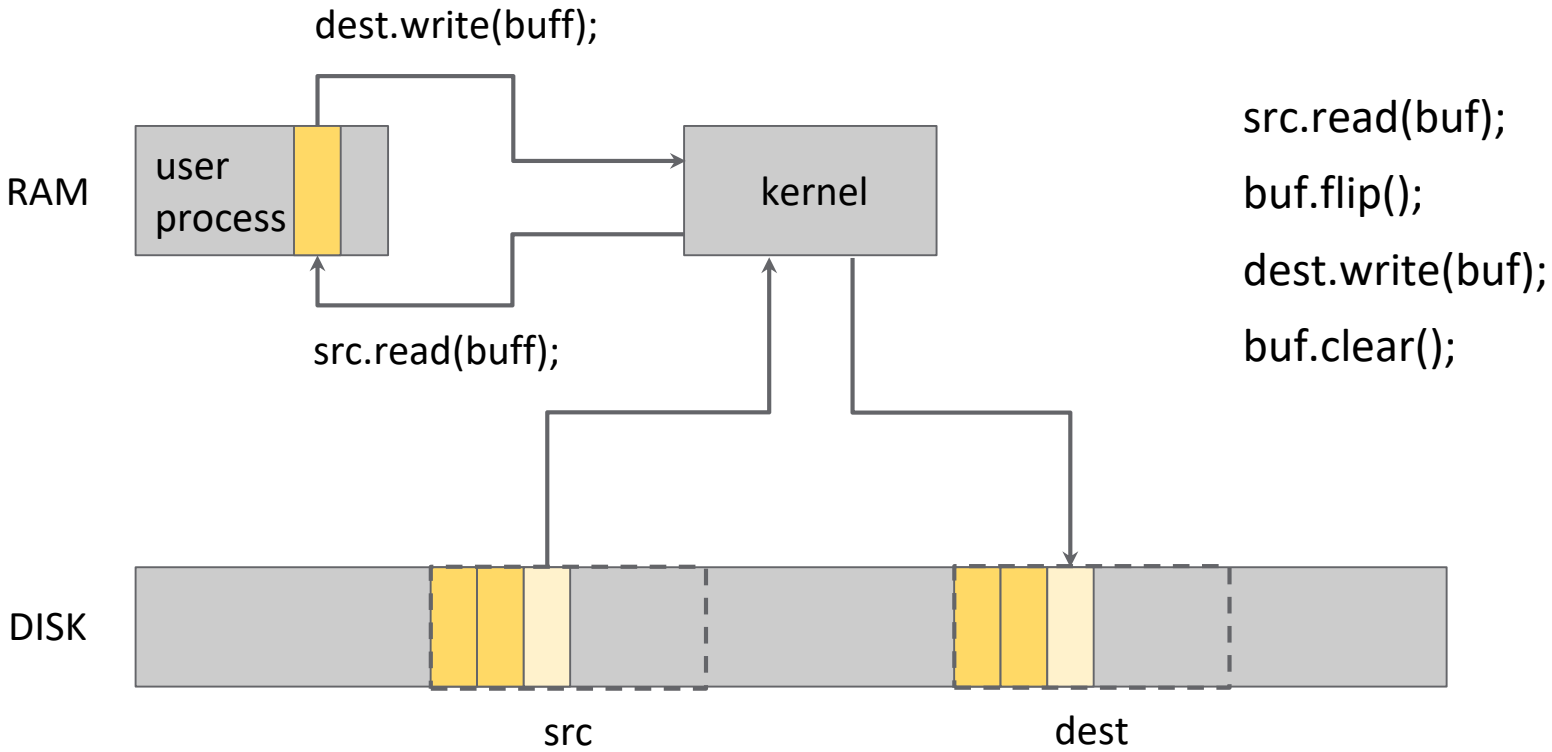


Копирование



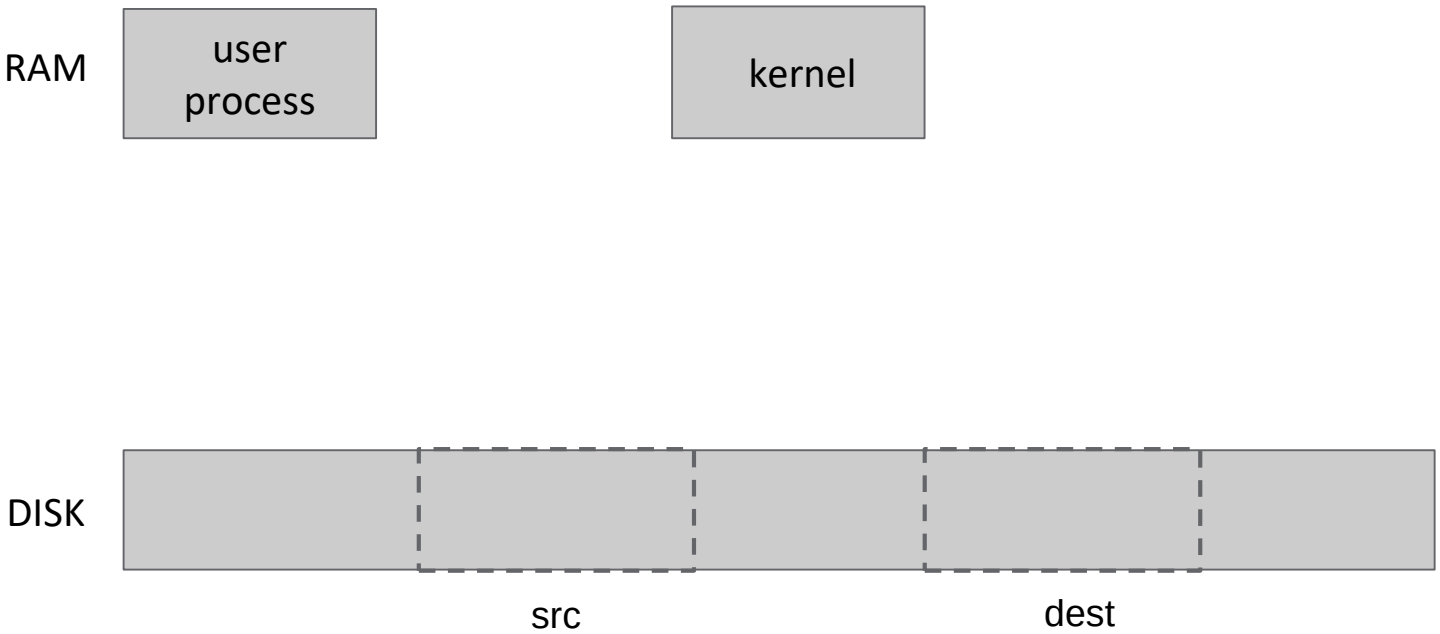

```
src.read(buf);
buf.flip();
dest.write(buf);
buf.clear();
```

Копирование

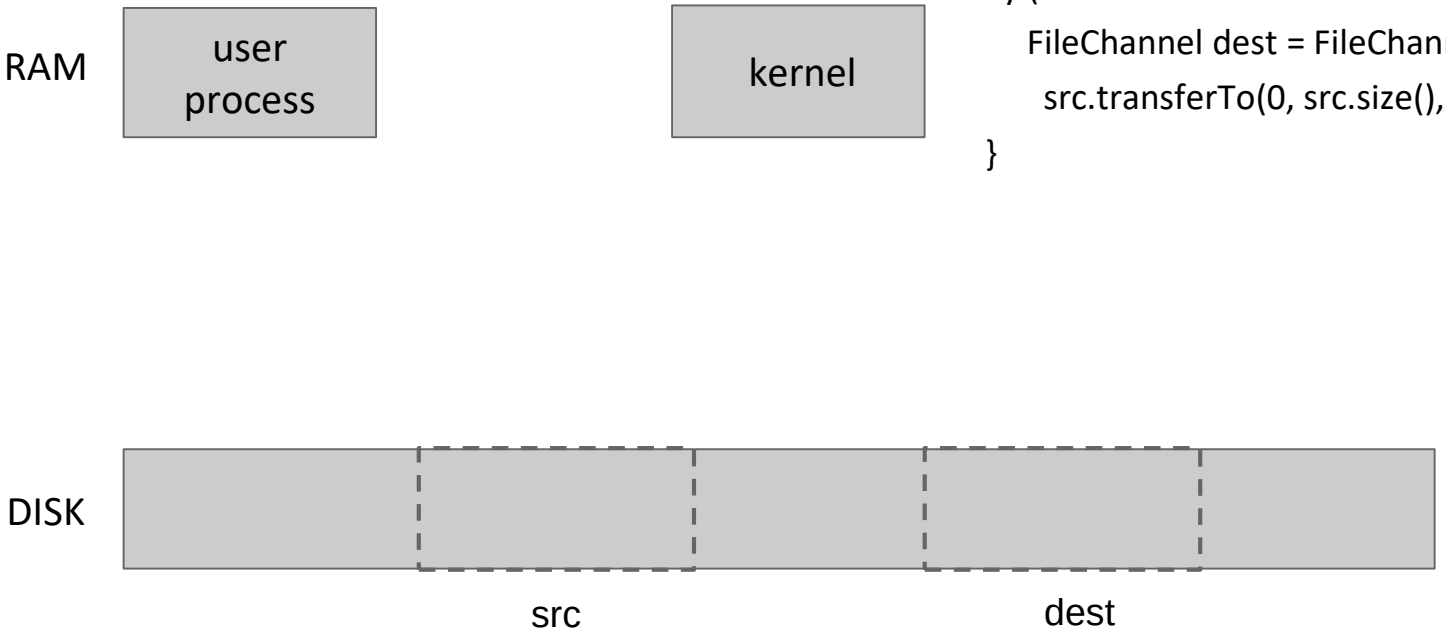



```
src.read(buf);
buf.flip();
dest.write(buf);
buf.clear();
```

Копирование



Копирование

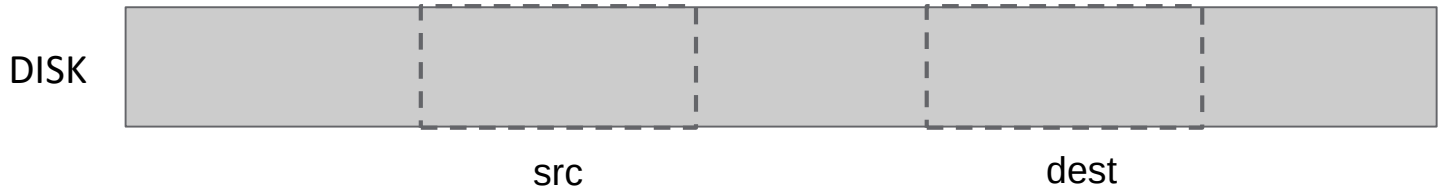



```
try (FileChannel src = FileChannel.open(file1, READ);
     FileChannel dest = FileChannel.open(file2, WRITE)) {
      src.transferTo(0, src.size(), dest);
}
```

Копирование

src.transferTo(0, src.size(), dest);

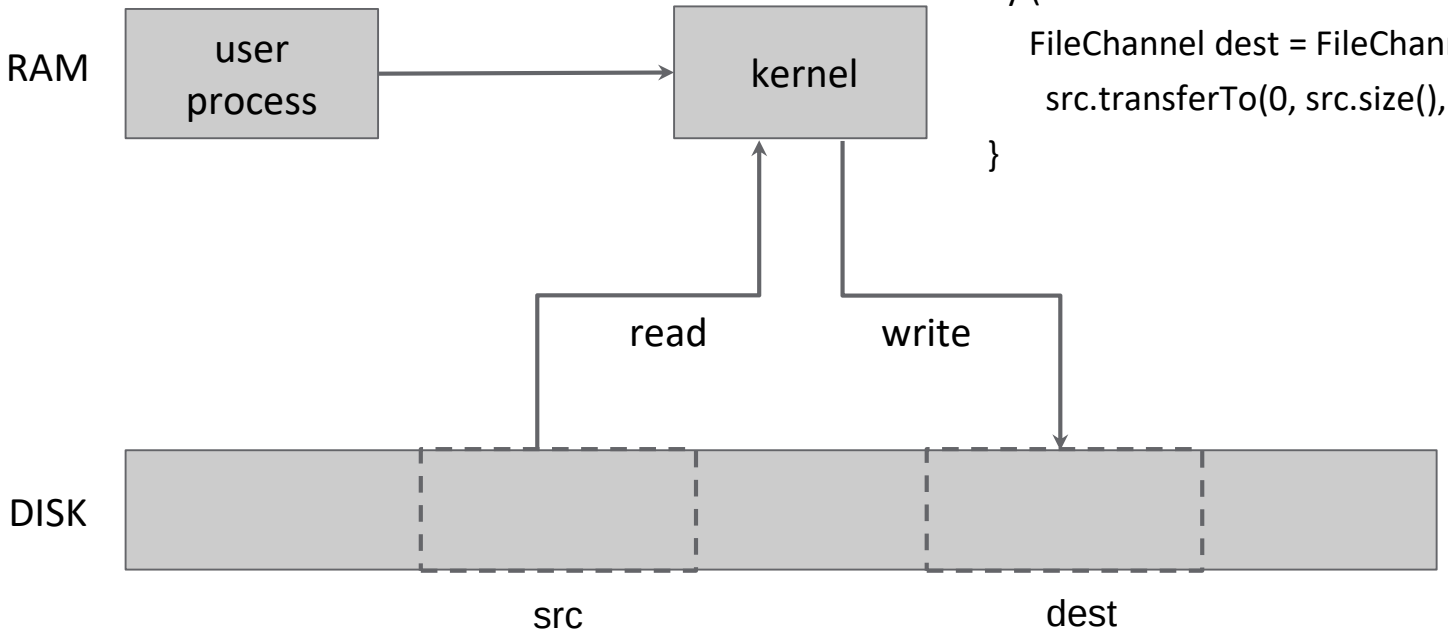

```
try (FileChannel src = FileChannel.open(file1, READ);
     FileChannel dest = FileChannel.open(file2, WRITE)) {
      src.transferTo(0, src.size(), dest);
}
```



GridGain

Копирование

src.transferTo(0, src.size(), dest);




```
try (FileChannel src = FileChannel.open(file1, READ);
     FileChannel dest = FileChannel.open(file2, WRITE)) {
      src.transferTo(0, src.size(), dest);
}
```

TransferFrom vs TransferTo

```
1 try (FileChannel src = createChannel(from);
2       FileChannel dest = createChannel(to)
3 ) {
4     dest.transferFrom(src, 0, src.size());
5 }
```

?

```
1 try (FileChannel src = createChannel(from);
2       FileChannel dest = createChannel(to)
3 ) {
4     src.transferTo(0, src.size(), dest);
5 }
```

Native Calls (TransferTo/TransferFrom)

```
native long transferTo0(FileDescriptor src, long position, long count, FileDescriptor dst);
```

Native Calls (TransferTo/TransferFrom)

native long transferTo0(FileDescriptor src, long position, long count, FileDescriptor dst);

native long transferFrom0(FileDescriptor src, long position, long count, FileDescriptor dst);

Native Calls (TransferTo/TransferFrom)

native long transferTo0(FileDescriptor src, long position, long count, FileDescriptor dst);

native long transferFrom0(FileDescriptor src, long position, long count, FileDescriptor dst);
(Нет такого)

TransferTo

```
 1 long transferTo( long position, long count, WritableByteChannel target) {
 2     ...
 3     if ((n = transferToDirectly(position, icount, target)) >= 0)
 4         return n;
 5     ...
 6     if ((n = transferToTrustedChannel(position, icount, target)) >= 0)
 7         return n;
 8
 9     ...
10     return transferToArbitraryChannel(position, icount, target);
11 }
```

TransferTo

```
1  long transferTo( long position, long count, WritableByteChannel target) {
2      ...
3      if ((n = transferToDirectly(position, icount, target)) >= 0)
4          return n;
5      ...
6      if ((n = transferToTrustedChannel(position, icount, target)) >= 0)
7          return n;
8
9      ...
10     return transferToArbitraryChannel(position, icount, target);
11 }
```

- Копируем через syscall,
  если kernel поддерживает
  такую возможность

TransferTo

```
1 long transferTo( long position, long count, WritableByteChannel target) {
2     ...
3     if ((n = transferToDirectly(position, icount, target)) >= 0)
4         return n;
5     ...
6     if ((n = transferToTrustedChannel(position, icount, target)) >= 0)
7         return n;
8
9     ...
10    return transferToArbitraryChannel(position, icount, target);
11 }
```

- Копируем через syscall, если kernel поддерживает такую возможность
- Копируем через mmap блоками по MAPPED_TRANSFER_SIZE =8L*1024L*1024L;

TransferTo

```
1  long transferTo( long position, long count, WritableByteChannel target) {
2      ...
3      if ((n = transferToDirectly(position, icount, target)) >= 0)
4          return n;
5      ...
6      if ((n = transferToTrustedChannel(position, icount, target)) >= 0)
7          return n;
8
9      ...
10     return transferToArbitraryChannel(position, icount, target);
11 }
```

- Копируем через syscall, если kernel поддерживает такую возможность
- Копируем через mmap блоками по MAPPED_TRANSFER_SIZE =8L*1024L*1024L;
- Копируем через directBuffer блоками по TRANSFER_SIZE = 8192;

TransferFrom

```
1 long transferFrom(ReadableByteChannel src, long position, long count) {
2     ...
3     if (src instanceof FileChannelImpl)
4         return transferFromFileChannel((FileChannelImpl)src, position, count);
5     ...
6     return transferFromArbitraryChannel(src, position, count);
7 }
```

TransferFrom

```
1 long transferFrom(ReadableByteChannel src, long position, long count) {
2     ...
3     if (src instanceof FileChannelImpl)
4         return transferFromFileChannel((FileChannelImpl)src, position, count);
5     ...
6     return transferFromArbitraryChannel(src, position, count);
7 }
```

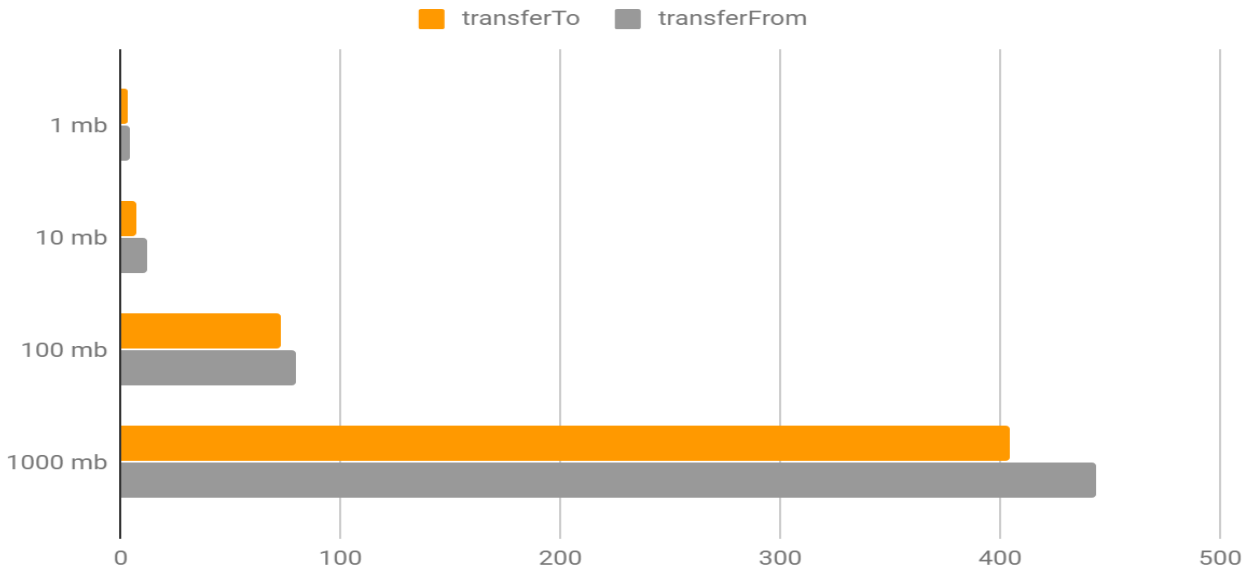
- Копируем через mmap блоками
  по MAPPED_TRANSFER_SIZE
  =8L*1024L*1024L;

TransferFrom

```
1 long transferFrom(ReadableByteChannel src, long position, long count) {
2     ...
3     if (src instanceof FileChannelImpl)
4         return transferFromFileChannel((FileChannelImpl)src, position, count);
5     ...
6     return transferFromArbitraryChannel(src, position, count);
7 }
```

- Копируем через mmap блоками по MAPPED_TRANSFER_SIZE =8L*1024L*1024L;
- Копируем через directBuffer блоками по TRANSFER_SIZE = 8192;

Benchmark



GridGain

Benchmark

	1 mb	10 mb	100 mb	1000 mb
transferTo				
DIRECT	3	4	73	404
MMAP	3	5	81	444
ARBITRARY	7	13	123	746
transferFrom				
MMAP	4	12	80	444
ARBITRARY	9	27	182	753

Скорость копирования (мсек)

Что важно помнить

- Используйте directBuffer там, где важна низкая latency

Что важно помнить

- Используйте directBuffer там, где важна низкая latency
- Проверяйте, сколько байт записали или прочитали

Что важно помнить

- Используйте directBuffer там, где важна низкая latency
- Проверяйте, сколько байт записали или прочитали
- Прерывать потоки нужно с умом (используйте AsyncFileChannel или RWLock для обычного FileChannel)

Выводы

- Операционная система всем силами пытается нам помочь (page cache, read-ahead …etc) а мы должны помочь ей (используйте fadvice, если знаете, как будете работать с файлом)

Выводы

- Операционная система всем силами пытается нам помочь (page cache, read-ahead …etc) а мы должны помочь ей (используйте fadvice, если знаете, как будете работать с файлом)
- Используйте FileChannel - он лучше чем File(I/O)Stream и RandomAccessFile

Выводы

- Операционная система всем силами пытается нам помочь (page cache, read-ahead …etc) а мы должны помочь ей (используйте fadvice, если знаете, как будете работать с файлом)
- Используйте FileChannel - он лучше чем File(I/O)Stream и RandomAccessFile
- Следите за лимитами (mmap limit, open files limit …etc)

Q&A

Dmitriy Govorukhin
dmitriy.govorukhin@gmail.com