

# Основы построения файловых систем

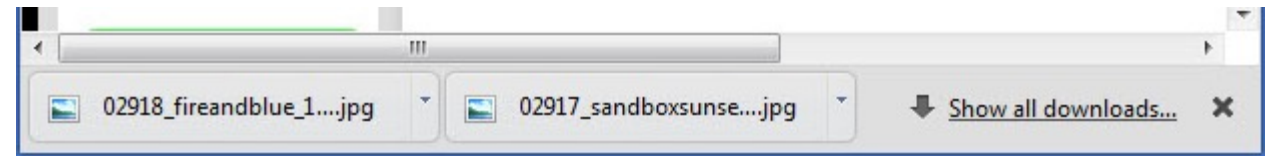


## Что мы хотим от ФС:

- сохранять данные

## Что мы хотим от ФС:

- сохранять данные



Если файлов тысячи, то  
список становится неудобным

# Что мы хотим от ФС:

- сохранять данные,
- упорядочивать их



## Что мы хотим от ФС:

- сохранять данные,
- упорядочивать их,
- предоставлять к ним доступ

Сегодня мы будем говорить только о локальных ФС, т.е. тех, которые хранят данные на том компьютере, где исполняется ОС.

## Какой интерфейс хочется иметь:

```
f = open("./pstorage-fes/src/fes.c");  
read(f, buffer, size);  
.....  
write(f, buffer, size);  
.....  
close(f);
```

## Какой интерфейс хочется иметь:

```
f = open("./pstorage-fes/src/fes.c");  
read(f, buffer, size);  
.....  
write(f, buffer, size);  
.....  
close(f);
```

## Какой интерфейс есть у жёсткого диска:

- \* прочесть сектор\* номер N,
- \* записать сектор номер M.

*\* сектор – блок длиной 512 или 4096 байт*



**Какой интерфейс хочется иметь:**

```
f = open("./pstorage-fes/src/fes.c");  
read(f, buffer, size);  
.....  
write(f, buffer, size);  
.....  
close(f);
```

**Какой интерфейс есть у жёсткого диска:**

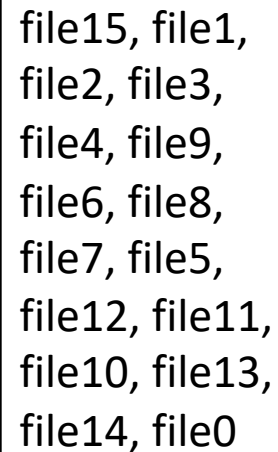
\* пр  
\*  
Это уже не всегда верно: технически стало можно в собрать в форм-факторе PCIe-карты компьютер с 16 ядрами ARM, 512Gb RAM, 2x100Gb ethernet-портами и 16 портами для подключения PCIe-устройств хранения и 24 SATA-портами. Например, Mellanox BlueField. Такое устройство может предоставлять куда более богатый интерфейс.

Какой интерфейс хочется иметь:	Какой интерфейс есть у жёсткого диска:
<pre>f = open("./pstorage-fes/src/fes.c"); read(f, buffer, size); ..... write(f, buffer, size); ..... close(f);</pre>	<ul style="list-style-type: none"><li>* прочесть сектор* номер N,</li><li>* записать сектор номер M.</li></ul> <p><i>* сектор – блок длиной 512 или 4096 байт</i></p>

Задача ФС:	
Используя только интерфейс жёсткого диска, предоставить пользователю возможность	
<ul style="list-style-type: none"><li>• создавать каталоги и файлы,</li><li>• отыскивать каталоги и файлы по имени,</li><li>• писать данные в файлы (в произвольные позиции) и читать их,</li><li>• делать вышеперечисленное надёжно и эффективно.</li></ul>	

## Пример задачи, которая возникает у авторов ФС: как организовать список файлов\*?

Линейный список, где файлы идут в порядке создания



file15, file1,  
file2, file3,  
file4, file9,  
file6, file8,  
file7, file5,  
file12, file11,  
file10, file13,  
file14, file0

*\* для простоты пусть каждый прямоугольник на рисунке будет непрерывным блоком на диске*

## Пример задачи, которая возникает у авторов ФС: как организовать список файлов\*?

Линейный список, где файлы идут в порядке создания

```
file15, file1,  
file2, file3,  
file4, file9,  
file6, file8,  
file7, file5,  
file12, file11,  
file10, file13,  
file14, file0
```

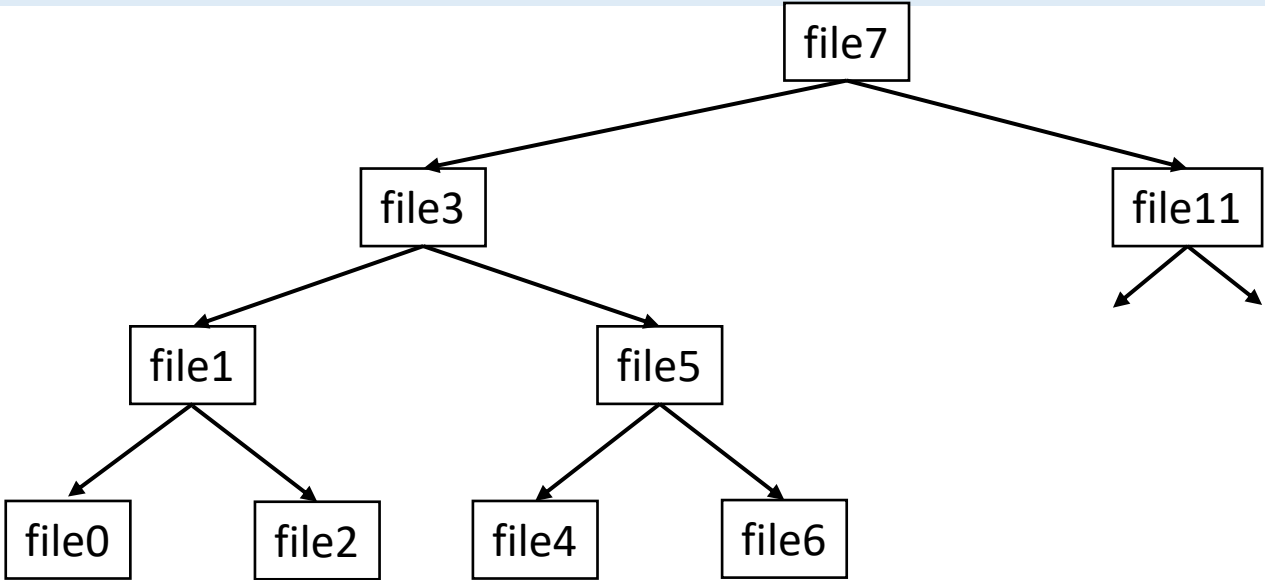
Чтобы найти элемент, надо 16 сравнений

*\* для простоты пусть каждый прямоугольник на рисунке будет непрерывным блоком на диске*

# Пример задачи, которая возникает у авторов ФС: как организовать список файлов\*?

Линейный список, где файлы идут в порядке создания      Дерево поиска

file15, file1,  
file2, file3,  
file4, file9,  
file6, file8,  
file7, file5,  
file12, file11,  
file10, file13,  
file14, file0



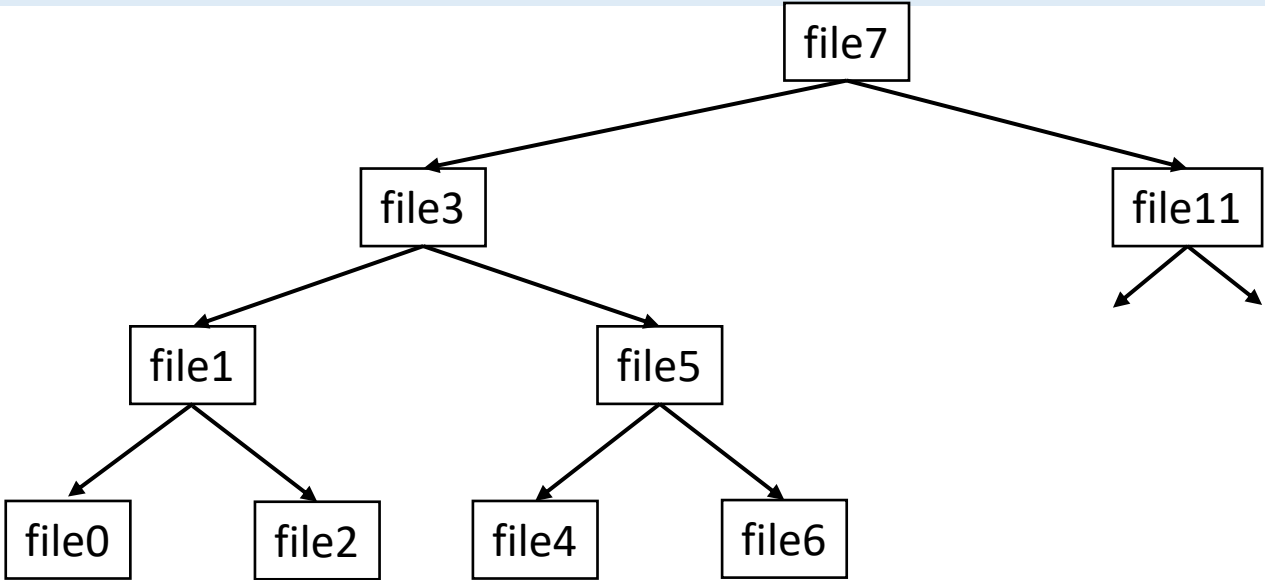
Чтобы найти элемент, надо 16 сравнений      Чтобы найти элемент, надо 4 сравнения. Win?

*\* для простоты пусть каждый прямоугольник на рисунке будет непрерывным блоком на диске*

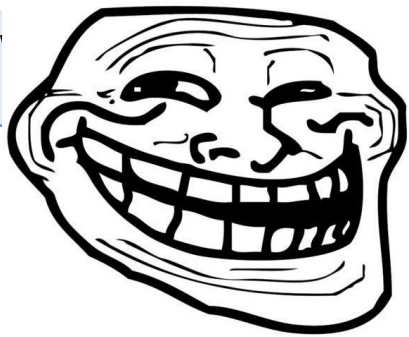
# Пример задачи, которая возникает у авторов ФС: как организовать список файлов\*?

Линейный список, где файлы идут в порядке создания      Дерево поиска

file15, file1,  
file2, file3,  
file4, file9,  
file6, file8,  
file7, file5,  
file12, file11,  
file10, file13,  
file14, file0



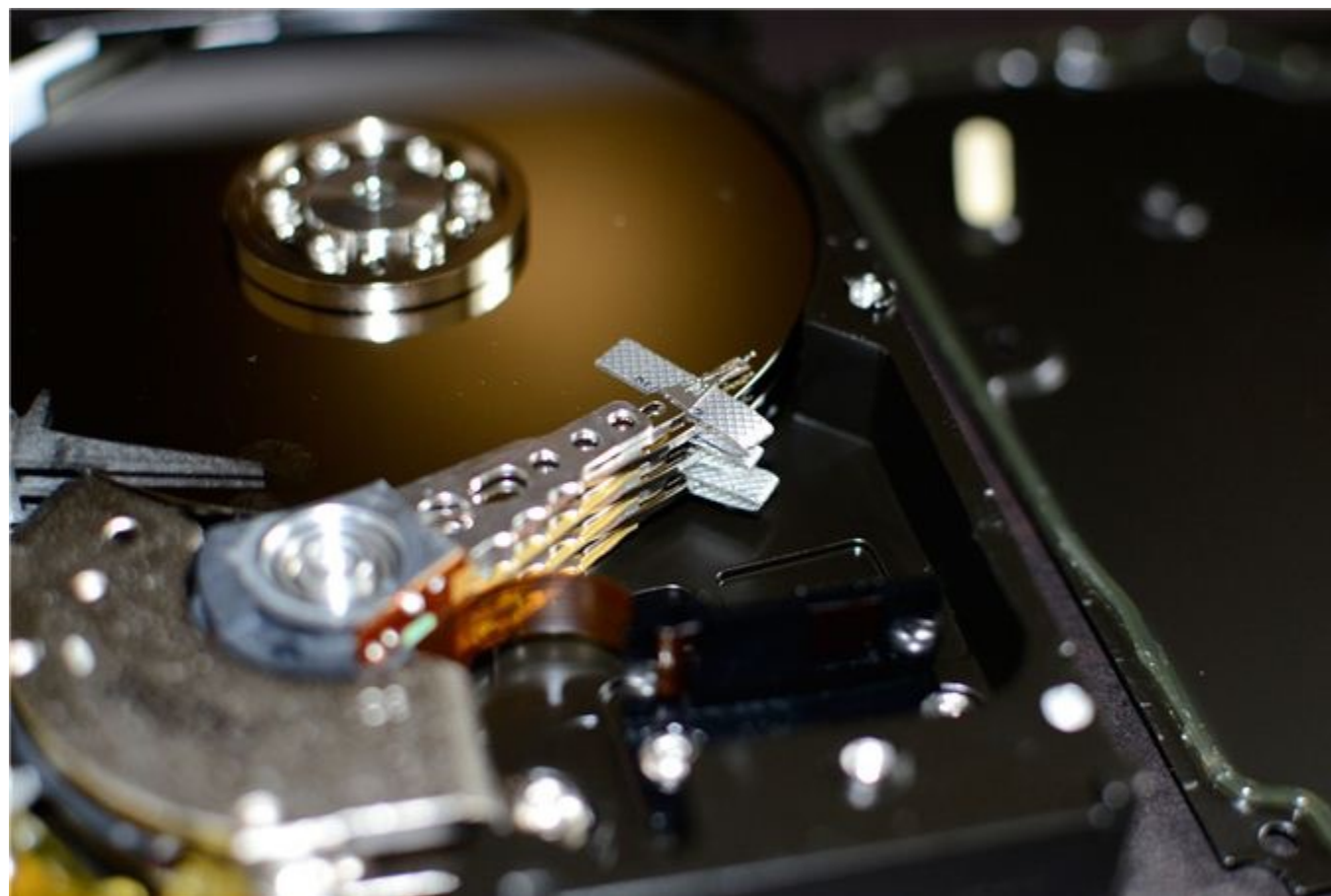
Чтобы найти элемент, надо 16 сравнений      Чтобы найти элемент, надо 4 сравнения.



*\* для простоты пусть каждый прямоугольник на рисунке будет непрерывным блоком на диске*

Читающая головка диска движается медленно, случайные чтения у диска получаются плохо.  
Для сравнения:

- Скорость линейного чтения  $\approx 100 \text{ MB/sec}$ , т.е.  $\approx 10 \text{ ms}$  на  $1 \text{ MB}$ ,
- Время позиционирования головки  $\approx 10 \text{ ms}$ .

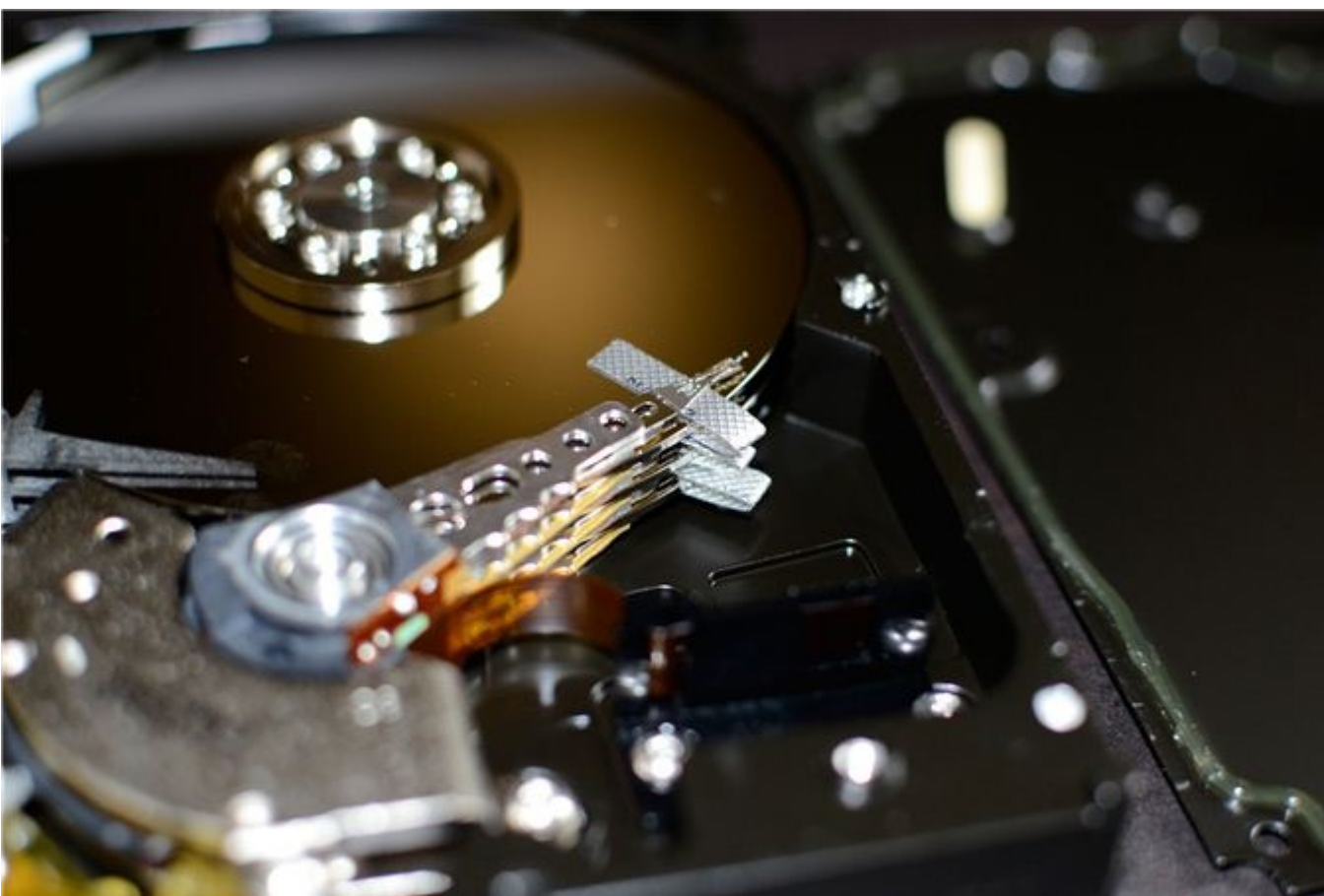


*Современные HDD работают быстрее. Например, Seagate Exos 16T имеет скорость линейного чтения до  $250 \text{ MB/sec}$ , а время случайного доступа  $6 \text{ ms}$ .*

*Но значения  $100 \text{ MB/sec}$  и  $10 \text{ ms}$  намного удобнее для быстрых оценок.*

Читающая головка диска движется медленно, случайные чтения у диска получаются плохо.  
Для сравнения:

- Скорость линейного чтения  $\approx 100$  MB/sec, т.е.  $\approx 10$  ms на 1 MB,
- Время позиционирования головки  $\approx 10$  ms.



Для большей ясности отмасштабируем величины до привычных нам:

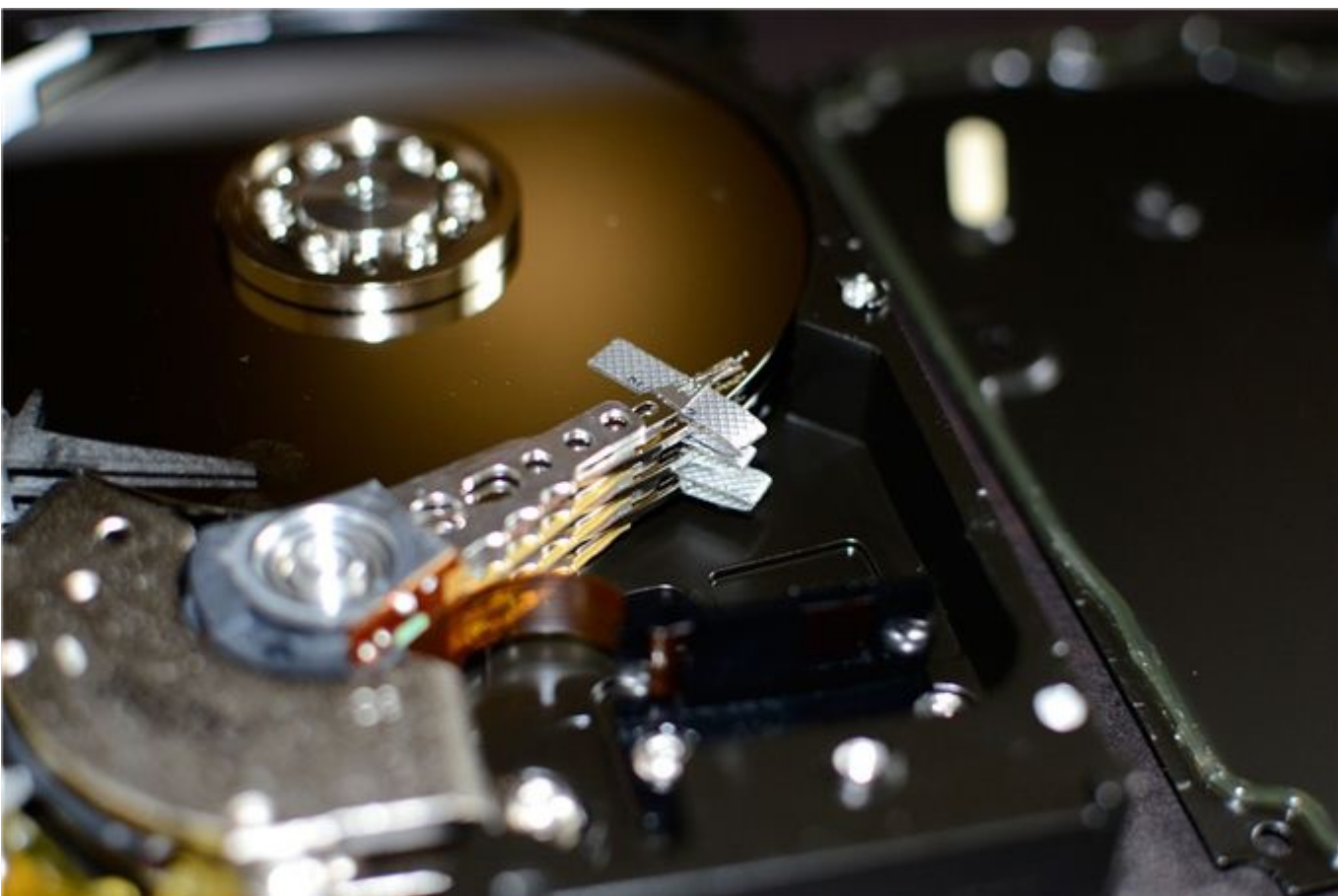
1ns ---> 1s

Чтение из L1 (Haswell, 4Ghz)	<b>1s</b>
Чтение из L2	<b>3s</b>
Чтение из L3	<b>9s</b>
Чтение из RAM	<b>9s + 57s</b>



Читающая головка диска движается медленно, случайные чтения у диска получаются плохо.  
Для сравнения:

- Скорость линейного чтения  $\approx 100\text{ MB/sec}$ , т.е.  $\approx 10\text{ ms}$  на  $1\text{ MB}$ ,
- Время позиционирования головки  $\approx 10\text{ ms}$ .

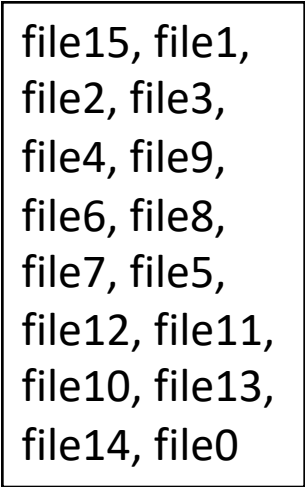


Для большей ясности отмасштабируем величины до привычных нам:  
 $1\text{ ns} \rightarrow 1\text{ s}$

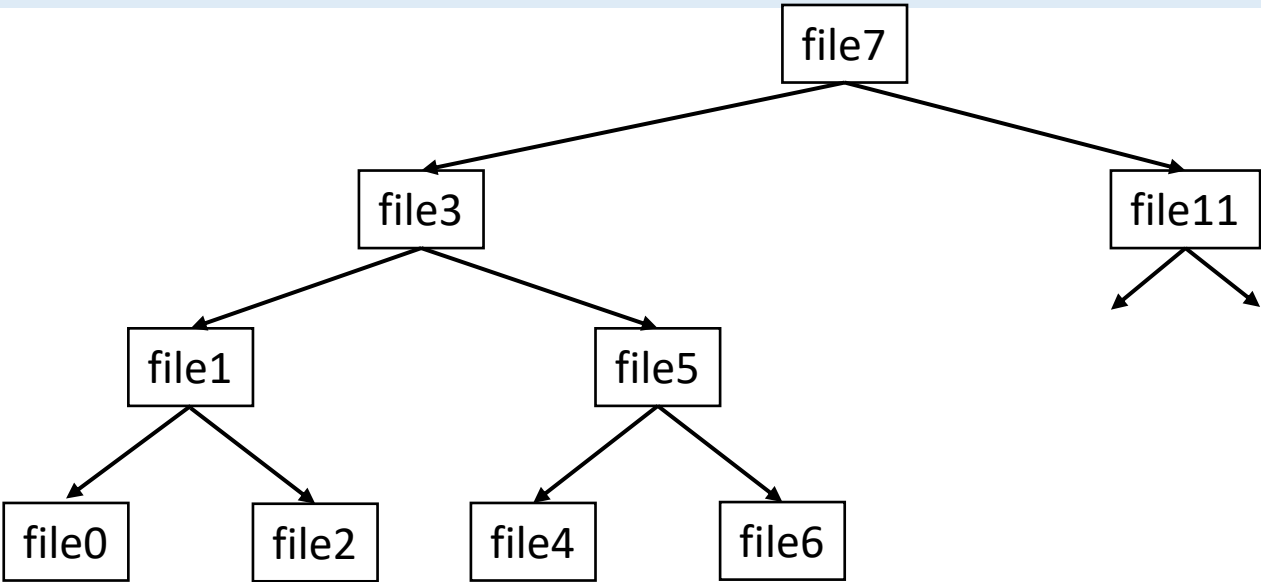
Чтение из L1 (Haswell, 4Ghz)	1s
Чтение из L2	3s
Чтение из L3	9s
Чтение из RAM	9s + 57s
Чтение с диска	116 дней только на позиционирование читающей головки

# Пример задачи, которая возникает у авторов ФС: как организовать список файлов\*?

Линейный список, где файлы идут в порядке создания



Дерево поиска



Переход на начало списка:	≈10msec	Тут нужны 4 позиционирования читающей головки, т.е. меньше, чем в 40msec мы не уложимся.
Чтение списка:	≈1msec (≈1MB)	
Поиск (список уместился в RAM):	<1msec	

*\* для простоты пусть каждый прямоугольник на рисунке будет непрерывным блоком на диске*

## Устройства хранения данных:

HDD (Hard Disk Drive, a.k.a Rotating drive,  
a.k.a Spinning rust)

+ достаточно быстрое линейное чтение ( $\approx 100$  MB/sec)  
- очень медленный случайный доступ ( $\approx 100$  IOPS)

*\* IOPS – Input/output Operations Per Second; Данные для Intel SSD DC S3700 и D7-P5600.*

# Устройства хранения данных:

HDD (Hard Disk Drive, a.k.a Rotating drive, a.k.a Spinning rust)	<ul style="list-style-type: none"><li>+ достаточно быстрое линейное чтение (<math>\approx 100</math> MB/sec),</li><li>- очень медленный случайный доступ (<math>\approx 100</math> IOPS).</li></ul>
Flash memory	<ul style="list-style-type: none"><li>+ быстрое линейное чтение,</li><li>+ нет времени «позиционирования головок»,</li><li>- перезаписывать можно только “rewrite block” целиком, а он несколько MB в размере,</li><li>- небольшое число циклов перезаписи.</li></ul>

*\* IOPS – Input/output Operations Per Second; Данные для Intel SSD DC S3700 и D7-P5600.*

# Устройства хранения данных:

HDD (Hard Disk Drive, a.k.a Rotating drive, a.k.a Spinning rust)	<ul style="list-style-type: none"><li>+ достаточно быстрое линейное чтение (<math>\approx 100</math> MB/sec),</li><li>- очень медленный случайный доступ (<math>\approx 100</math> IOPS).</li></ul>
Flash memory	<ul style="list-style-type: none"><li>+ быстрое линейное чтение,</li><li>+ нет времени «позиционирования головок»,</li><li>- перезаписывать можно только “rewrite block” целиком, а он несколько MB в размере,</li><li>- небольшое число циклов перезаписи.</li></ul>
SSD (Solid State Drive), SATA	<p>Flash + компьютер, который прячет сложность работы с “rewrite blocks”.</p> <ul style="list-style-type: none"><li>+ быстрый линейный доступ (<math>\approx 500</math> MB/sec sequential read*),</li><li>+ быстрый произвольный доступ (<math>\approx 75.000</math> IOPS),</li><li>- деградация производительности со временем,</li><li>- желательна специальная поддержки со стороны ОС, например, ATA TRIM.</li></ul>

*\* IOPS – Input/output Operations Per Second; Данные для Intel SSD DC S3700 и D7-P5600.*

# Устройства хранения данных:

HDD (Hard Disk Drive, a.k.a Rotating drive, a.k.a Spinning rust)	<ul style="list-style-type: none"><li>+ достаточно быстрое линейное чтение (<math>\approx 100</math> MB/sec),</li><li>- очень медленный случайный доступ (<math>\approx 100</math> IOPS).</li></ul>
Flash memory	<ul style="list-style-type: none"><li>+ быстрое линейное чтение,</li><li>+ нет времени «позиционирования головок»,</li><li>- перезаписывать можно только “rewrite block” целиком, а он несколько MB в размере,</li><li>- небольшое число циклов перезаписи.</li></ul>
SSD (Solid State Drive), SATA	<p>Flash + компьютер, который прячет сложность работы с “rewrite blocks”.</p> <ul style="list-style-type: none"><li>+ быстрый линейный доступ (<math>\approx 500</math> MB/sec sequential read*),</li><li>+ быстрый произвольный доступ (<math>\approx 75.000</math> IOPS),</li><li>- деградация производительности со временем,</li><li>- желательна специальная поддержки со стороны ОС, например, ATA TRIM.</li></ul>
SSD, NVMe	SSD с более быстрым интерфейсом: $\approx 5$ GB/sec sequential read, $\approx 1\text{M}$ IOPS*.

*\* IOPS – Input/output Operations Per Second; Данные для Intel SSD DC S3700 и D7-P5600.*

# Устройства хранения данных:

HDD (Hard Disk Drive, a.k.a Rotating drive, a.k.a Spinning rust)	<ul style="list-style-type: none"><li>+ достаточно быстрое линейное чтение (<math>\approx 100</math> MB/sec),</li><li>- очень медленный случайный доступ (<math>\approx 100</math> IOPS).</li></ul>
Flash memory	<ul style="list-style-type: none"><li>+ быстрое линейное чтение,</li><li>+ нет времени «позиционирования головок»,</li><li>- перезаписывать можно только “rewrite block” целиком, а он несколько MB в размере,</li><li>- небольшое число циклов перезаписи.</li></ul>
SSD (Solid State Drive), SATA	<p>Flash + компьютер, который прячет сложность работы с “rewrite blocks”.</p> <ul style="list-style-type: none"><li>+ быстрый линейный доступ (<math>\approx 500</math> MB/sec sequential read*),</li><li>+ быстрый произвольный доступ (<math>\approx 75.000</math> IOPS),</li><li>- деградация производительности со временем,</li><li>- желательна специальная поддержки со стороны ОС, например, ATA TRIM.</li></ul>
SSD, NVMe	SSD с более быстрым интерфейсом: $\approx 5$ GB/sec sequential read, $\approx 1M$ IOPS*.
Storage-class memory (3D NAND, 3DXP, etc.)	<p>Память с произвольным доступом, которая не стирается при выключении питания.</p> <ul style="list-style-type: none"><li>+ по скорости сопоставима с DRAM (устанавливается на PCIe- или DDR-шины),</li><li>+ объём – единицы терабайт.</li></ul>

\* IOPS – Input/output Operations Per Second; Данные для Intel SSD DC S3700 и D7-P5600.

# API для чтения/записи файлов:

Надо скрыть от пользователя особенности оборудования и предоставить единообразный способ доступа к данным на разных устройствах.

- POSIX (Portable Operating System Interface),
- Windows,
- memory-mapped files.



## POSIX Filesystem API

Структура ФС древовидная:

```
/
├── bin
├── boot
├── dev
├── etc
├── home
│   └── artem
├── lib
├── lib32
├── lib64
├── libx32
├── lost+found
├── media
├── mnt
├── opt
└── proc
```

## POSIX Filesystem API

Структура ФС древовидная:

```
/
├── bin
├── boot
├── dev
├── etc
├── home
│   └── artem
├── lib
├── lib32
├── lib64
├── libx32
├── lost+found
├── media
├── mnt
├── opt
└── proc
```

## Windows Filesystem API

ФС представляет собой лес, корнями которого являются диски:

Drive		
C: fixed	931,17 G	775,76 G
D: fixed	0,91 T	482,52 G
X: network		
Y: network		
Z: network		

..	archive_io.h
dedup	archive_io_astor.h
Makefile	archive_io_local.h
TODO	archive_item.h
archive_api.c	archive_item_cache.h
archive_api_remote.c	archive_locking.h

POSIX Filesystem API

Структура ФС древовидная:



Filesystem Hierarchy Standard:

Linux:

[http://refspecs.linuxfoundation.org/FHS\\_2.3/fhs-2.3.pdf](http://refspecs.linuxfoundation.org/FHS_2.3/fhs-2.3.pdf)

FreeBSD:

<https://www.freebsd.org/doc/handbook/dirstructure.html>

Windows Filesystem API

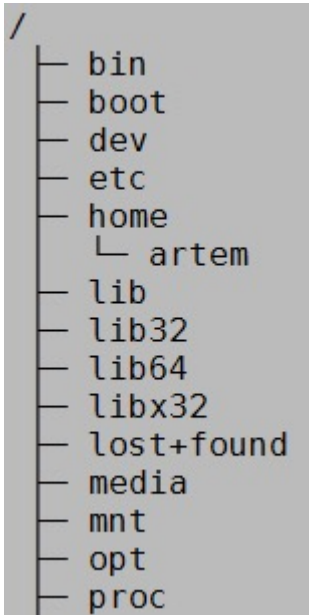
ФС представляет собой лес, корнями которого являются диски:

Drive		
C: fixed	931,17 G	775,76 G
D: fixed	0,91 T	482,52 G
X: network		
Y: network		
Z: network		

..	archive_io.h
dedup	archive_io_astor.h
Makefile	archive_io_local.h
TODO	archive_item.h
archive_api.c	archive_item_cache.h
archive_api_remote.c	archive_locking.h

POSIX Filesystem API

Структура ФС древовидная:



Filesystem Hierarchy Standard:

Linux:

[http://refspecs.linuxfoundation.org/FHS\\_2.3/fhs-2.3.pdf](http://refspecs.linuxfoundation.org/FHS_2.3/fhs-2.3.pdf)

FreeBSD:

<https://www.freebsd.org/doc/handbook/dirstructure.html>

Windows Filesystem API

ФС представляет собой лес, корнями которого являются диски:

Drive		
C: fixed	931,17 G	775,76 G
D: fixed	0,91 T	482,52 G
X: network		
Y: network		
Z: network		

..	archive_io.h
dedup	archive_io_astor.h
Makefile	archive_io_local.h
TODO	archive_item.h
archive_api.c	archive_item_cache.h
archive_api_remote.c	archive_locking.h

\Global??\C:\foo\bar.txt

## Проблема с терминологией

ФС – видимая пользователю иерархия каталогов и файлов

ФС – механизм расположения файлов на диске

```
/
├── bin
├── boot
├── dev
├── etc
├── home
│   └── artem
├── lib
├── lib32
├── lib64
├── libx32
├── lost+found
├── media
├── mnt
├── opt
└── proc
```

Drive		
C: fixed	931,17 G	775,76 G
D: fixed	0,91 T	482,52 G
X: network		
Y: network		
Z: network		

## Обзор-напоминание о POSIX Filesystem API

1. `open(path, flags, mode)` / `close(fd)`

## Обзор-напоминание о POSIX Filesystem API

### 1. open(path, flags, mode) / close(fd)

- O\_CREAT,
- O\_EXCL,
- O\_NOATIME,
- O\_CLOEXEC.

## Обзор-напоминание о POSIX Filesystem API

1. `open(path, flags, mode)` / `close(fd)`
2. `mkdir(path, flags)` / `rmdir(path)`



## Обзор-напоминание о POSIX Filesystem API

1. `open(path, flags, mode)` / `close(fd)`
2. `mkdir(path, flags)` / `rmdir(path)`
3. `chdir(path)`, `chroot(path)`

## Обзор-напоминание о POSIX Filesystem API

1. `open(path, flags, mode)` / `close(fd)`
2. `mkdir(path, flags)` / `rmdir(path)`
3. `chdir(path)`, `chroot(path)`
4. `openat(dirfd, path, flags)` / `mkdirat()` / `rmdirat()` / etc

Параметр `dirfd` играет роль рабочего каталога для данного вызова. Выгода:

- ???

## Обзор-напоминание о POSIX Filesystem API

1. `open(path, flags, mode) / close(fd)`
2. `mkdir(path, flags) / rmdir(path)`
3. `chdir(path), chroot(path)`
4. `openat(dirfd, path, flags) / mkdirat() / rmdirat() / etc`

Параметр `dirfd` играет роль рабочего каталога для данного вызова. Выгода:

- решает проблему гонок с `chdir()`,
- рабочие каталоги для потоков, а не всего процесса,
- меньше работы по обходу пути.

## Обзор-напоминание о POSIX Filesystem API

1. `open(path, flags, mode)` / `close(fd)`
2. `mkdir(path, flags)` / `rmdir(path)`
3. `chdir(path)`, `chroot(path)`
4. `openat(dirfd, path, flags)` / `mkdirat()` / `rmdirat()` / etc
5. `symlink()` / `readlink()`

## Обзор-напоминание о POSIX Filesystem API

1. `open(path, flags, mode)` / `close(fd)`
2. `mkdir(path, flags)` / `rmdir(path)`
3. `chdir(path)`, `chroot(path)`
4. `openat(dirfd, path, flags)` / `mkdirat()` / `rmdirat()` / etc
5. `symlink()` / `readlink()`
6. `link()` / `unlink()`

В POSIX файлы и имена существуют отдельно друг от друга. Возможны ситуации:

- файл имеет несколько имён,
- файл не имеет ни одного имени.

## Обзор-напоминание о POSIX Filesystem API

1. `open(path, flags, mode)` / `close(fd)`
2. `mkdir(path, flags)` / `rmdir(path)`
3. `chdir(path)`, `chroot(path)`
4. `openat(dirfd, path, flags)` / `mkdirat()` / `rmdirat()` / etc
5. `symlink()` / `readlink()`
6. `link()` / `unlink()`

В POSIX файлы и имена существуют отдельно друг от друга. Возможны ситуации:

- файл имеет несколько имён,
- файл не имеет ни одного имени.

`open(O_TMPFILE)` создаёт файл, у которого изначально нет имени.

## Обзор-напоминание о POSIX Filesystem API

1. `open(path, flags, mode)` / `close(fd)`
2. `mkdir(path, flags)` / `rmdir(path)`
3. `chdir(path)`, `chroot(path)`
4. `openat(dirfd, path, flags)` / `mkdirat()` / `rmdirat()` / etc
5. `symlink()` / `readlink()`
6. `link()` / `unlink()`
7. Специальные файлы:
  - `directory`,
  - `character devices`,
  - `block devices`,
  - `pipes`,
  - `unix domain sockets`.

## Обзор-напоминание о POSIX Filesystem API

1. `open(path, flags, mode)` / `close(fd)`
2. `mkdir(path, flags)` / `rmdir(path)`
3. `chdir(path)`, `chroot(path)`
4. `openat(dirfd, path, flags)` / `mkdirat()` / `rmdirat()` / etc
5. `symlink()` / `readlink()`
6. `link()` / `unlink()`
7. Специальные файлы.
8. `mmap()` / `munmap()`



