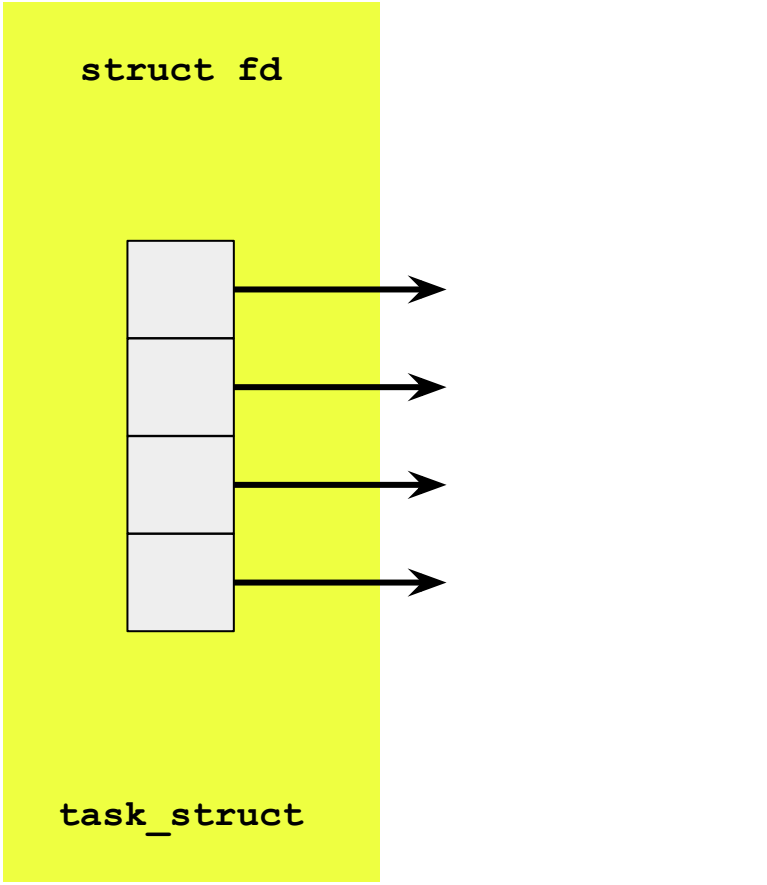
Filesystems vol. 2

Theory

Practice

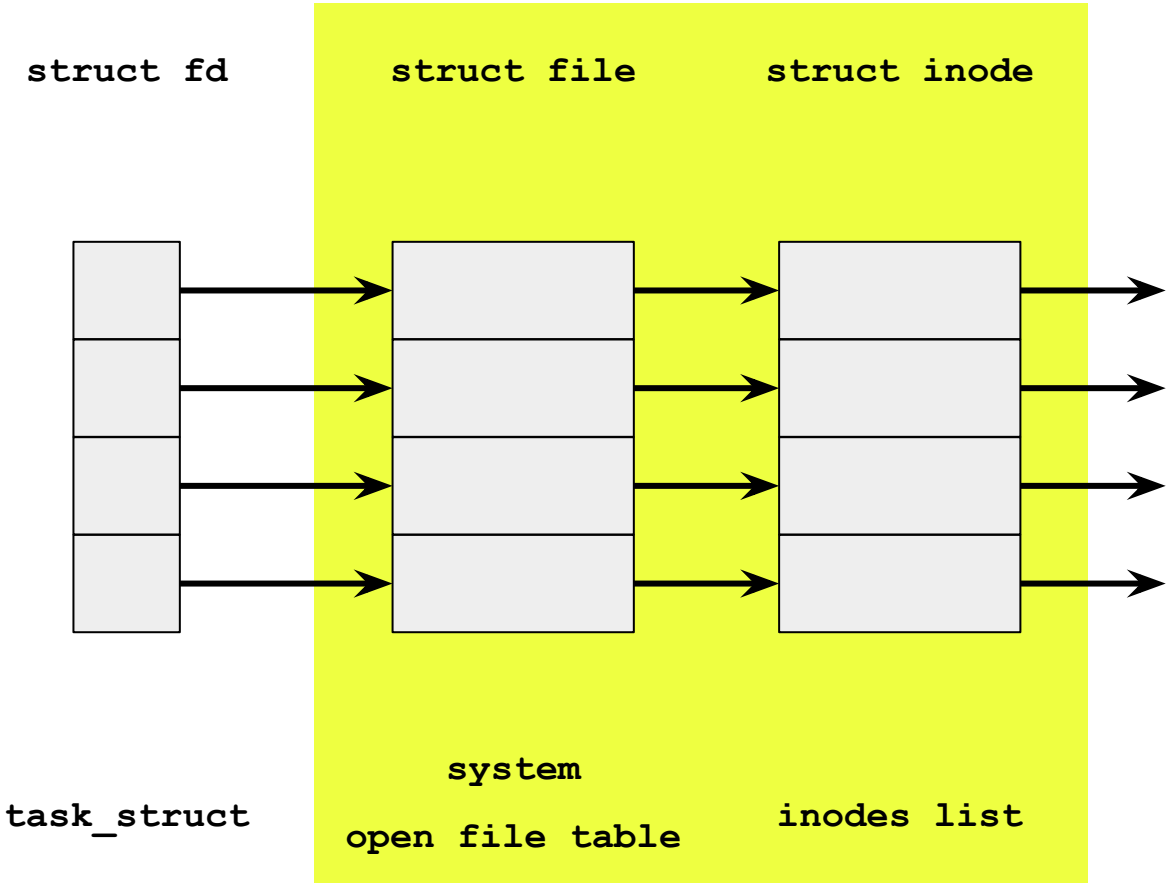
File descriptors

File descriptors are userspace references to kernel objects.



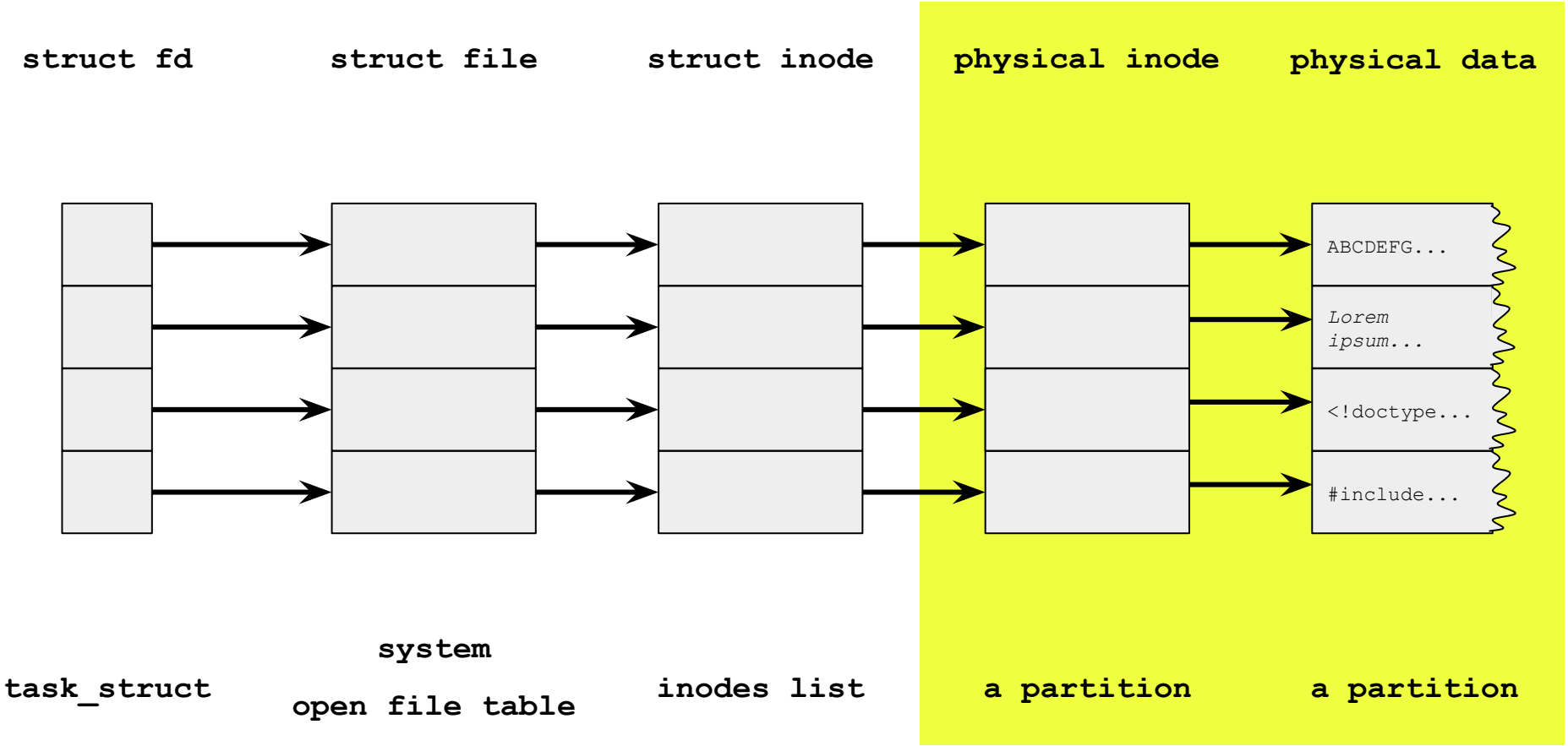
File descriptors

File descriptors are userspace references to kernel objects.



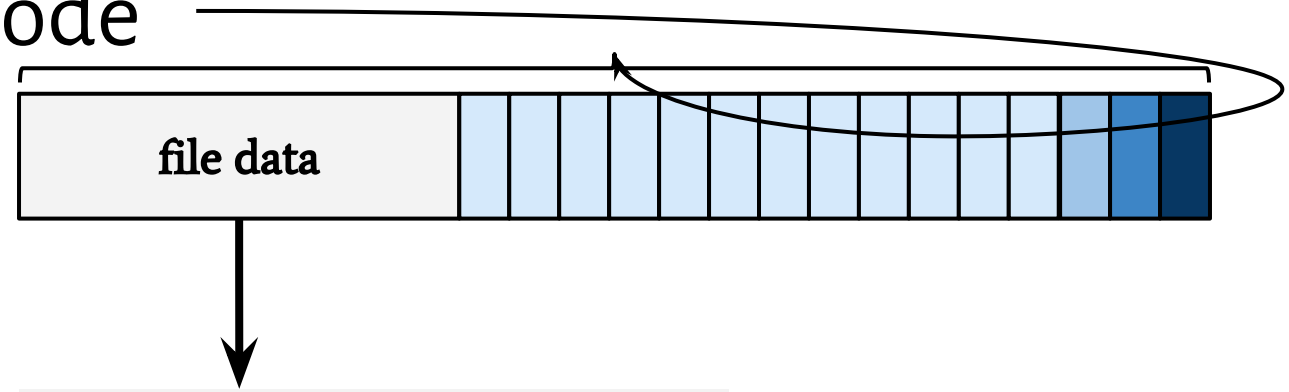
File descriptors

File descriptors are userspace references to kernel objects.



Inode

Inode



size in bytes

IDs of device, user, group

file mode

file type | rwxrw---

flags of protection

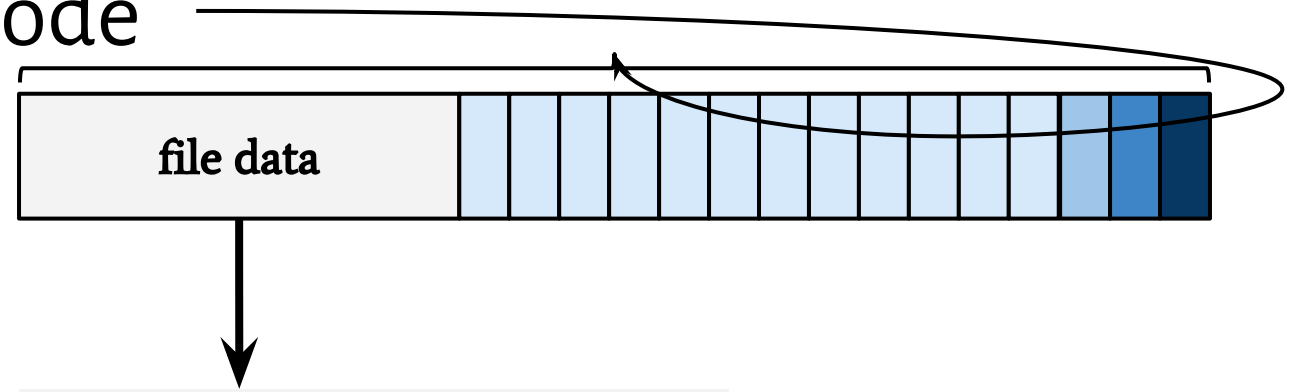
timestamps

last modified

last accessed

hard link count

Inode



size in bytes

IDs of device, user, group

file mode

file type | rwxrw---

flags of protection

timestamps

last modified

last accessed

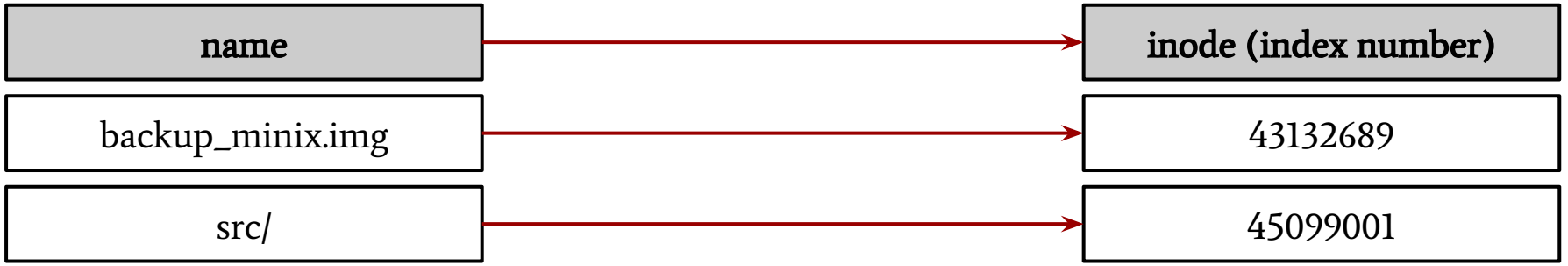
hard link count

No filename?

Filenames

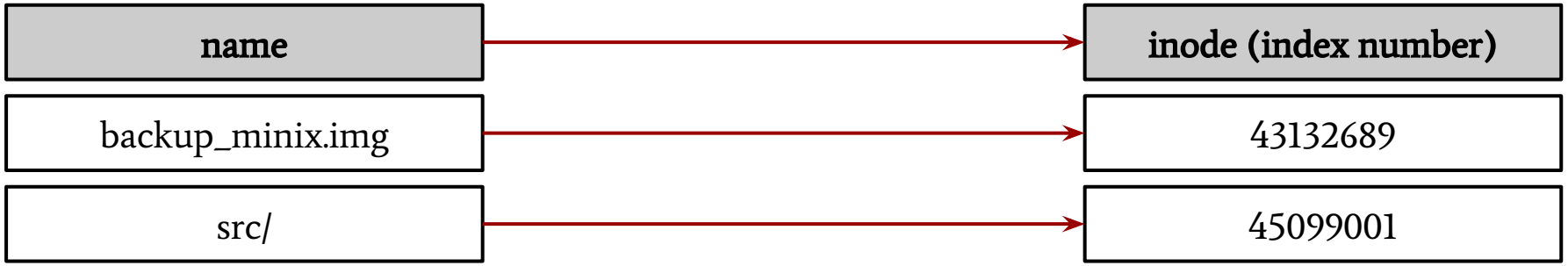
Filenames

Unix directories are just files which keep a list of filenames associated with inodes:



Filenames

Unix directories are just files which keep a list of filenames associated with inodes:

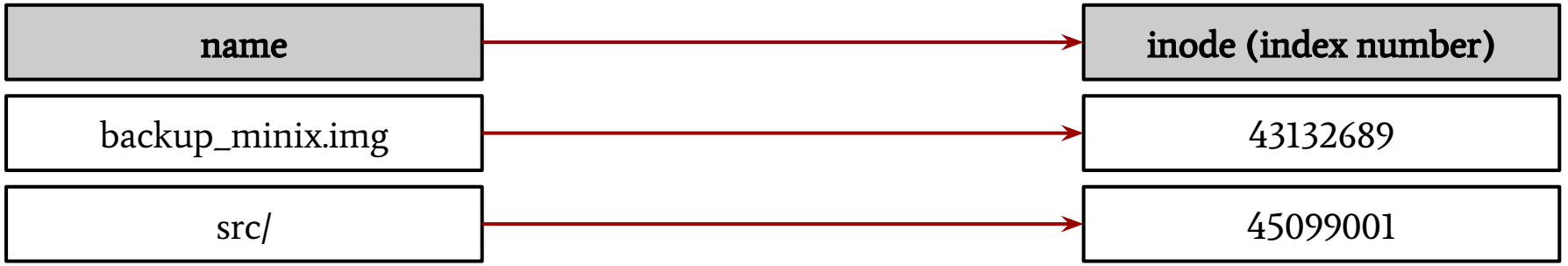


See the content of a directory:

```
$ ls -ai .
```

Filenames

Unix directories are just files which keep a list of filenames associated with inodes:



See the content of a directory:

```
$ ls -ai .
```

See the 'file info' of a directory:

```
$ stat test
```

```
  File: 'test'
  Size: 4096          Blocks: 8          IO Block: 4096   directory
Device: 802h/2050d    Inode: 43010572    Links: 2
Access: (0775/drwxrwxr-x)  Uid: ( 1000/ inga)   Gid: ( 1000/     inga)
Access: 2017-05-22 17:34:50.819948060 +0200
Modify: 2017-05-22 17:34:50.819948060 +0200
Change: 2017-05-22 17:34:50.819948060 +0200
 Birth: -
```

Btw: hard links vs. symbolic links

- a **hard link** is a directory entry that associates a name with a file on a file system

Create a hard link to a file:

```
$ ln <original_filename> <link_filename>
```

Btw: hard links vs. symbolic links

★ a **hard link** is a directory entry that associates a name with a file on a file system

Create a hard link to a file:

```
$ ln <original_filename> <link_filename>
```

★ a **symbolic link** is a file that contains another filename

Create a symbolic link to a file:

```
$ ln -s <original_filename> <link_filename>
```

Resolve a symbolic link:

```
$ readlink -f <symbolic_link>
```

Btw: hard links vs. symbolic links

★ a **hard link** is a directory entry that associates a name with a file on a file system

Create a hard link to a file:

```
$ ln <original_filename> <link_filename>
```

★ a **symbolic link** is a file that contains another filename

Create a symbolic link to a file:

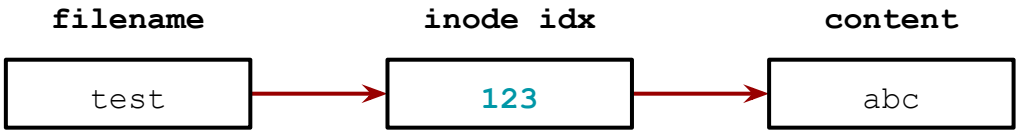
```
$ ln -s <original_filename> <link_filename>
```

Resolve a symbolic link:

```
$ readlink -f <symbolic_link>
```

Observe directory . entries:

```
$ echo "abc" > test
```



Btw: hard links vs. symbolic links

★ a **hard link** is a directory entry that associates a name with a file on a file system

Create a hard link to a file:

```
$ ln <original_filename> <link_filename>
```

★ a **symbolic link** is a file that contains another filename

Create a symbolic link to a file:

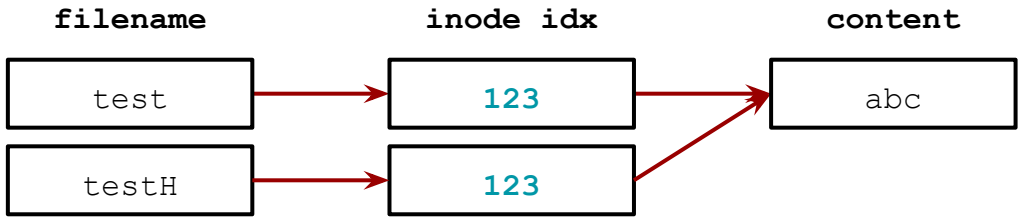
```
$ ln -s <original_filename> <link_filename>
```

Resolve a symbolic link:

```
$ readlink -f <symbolic_link>
```

Observe directory . entries:

```
$ echo "abc" > test
$ ln test testH
```



Btw: hard links vs. symbolic links

★ a **hard link** is a directory entry that associates a name with a file on a file system

Create a hard link to a file:

```
$ ln <original_filename> <link_filename>
```

★ a **symbolic link** is a file that contains another filename

Create a symbolic link to a file:

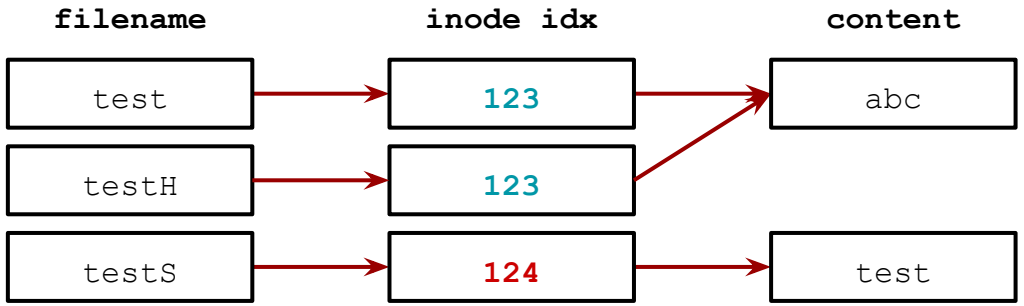
```
$ ln -s <original_filename> <link_filename>
```

Resolve a symbolic link:

```
$ readlink -f <symbolic_link>
```

Observe directory . entries:

```
$ echo "abc" > test
$ ln test testH
$ ln -s test testS
```



Btw: hard links vs. symbolic links

★ a **hard link** is a directory entry that associates a name with a file on a file system

Create a hard link to a file:

```
$ ln <original_filename> <link_filename>
```

★ a **symbolic link** is a file that contains another filename

Create a symbolic link to a file:

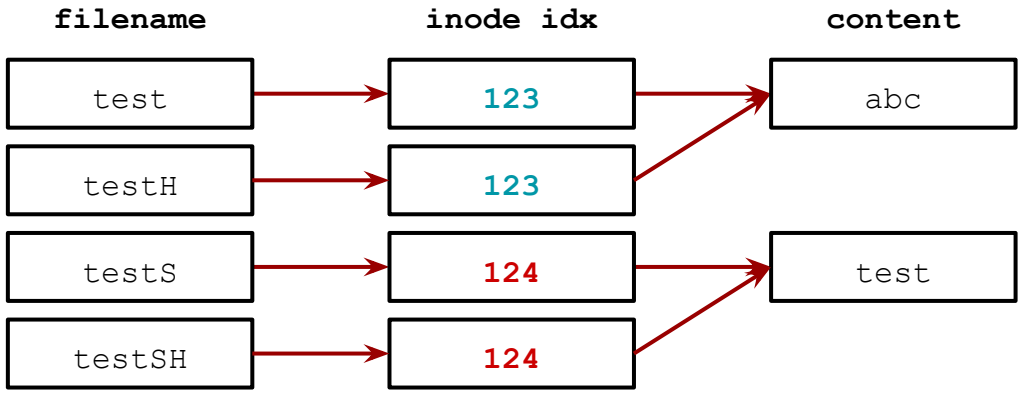
```
$ ln -s <original_filename> <link_filename>
```

Resolve a symbolic link:

```
$ readlink -f <symbolic_link>
```

Observe directory . entries:

```
$ echo "abc" > test
$ ln test testH
$ ln -s test testS
$ ln testS testSH
```



Btw: hard links vs. symbolic links

★ a **hard link** is a directory entry that associates a name with a file on a file system

Create a hard link to a file:

```
$ ln <original_filename> <link_filename>
```

★ a **symbolic link** is a file that contains another filename

Create a symbolic link to a file:

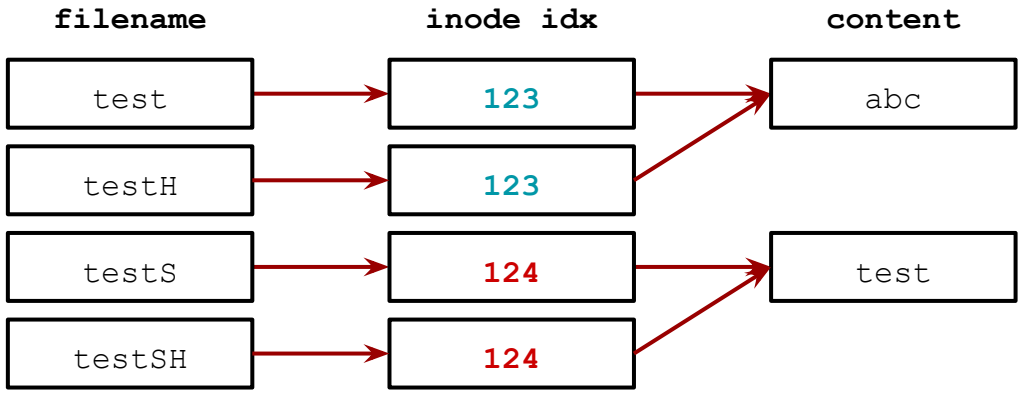
```
$ ln -s <original_filename> <link_filename>
```

Resolve a symbolic link:

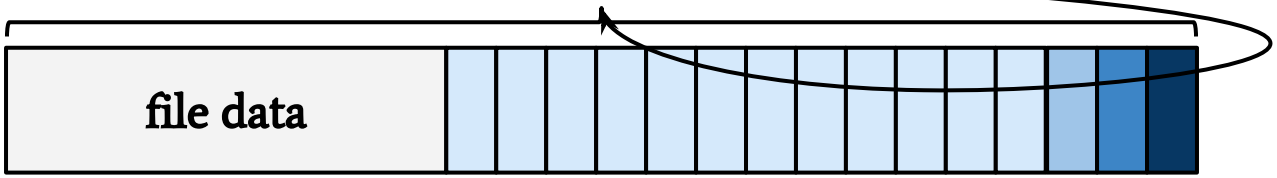
```
$ readlink -f <symbolic_link>
```

Observe directory . entries:

```
$ echo "abc" > test
$ ln test testH
$ ln -s test testS
$ ln testS testSH
$ ls -li
```

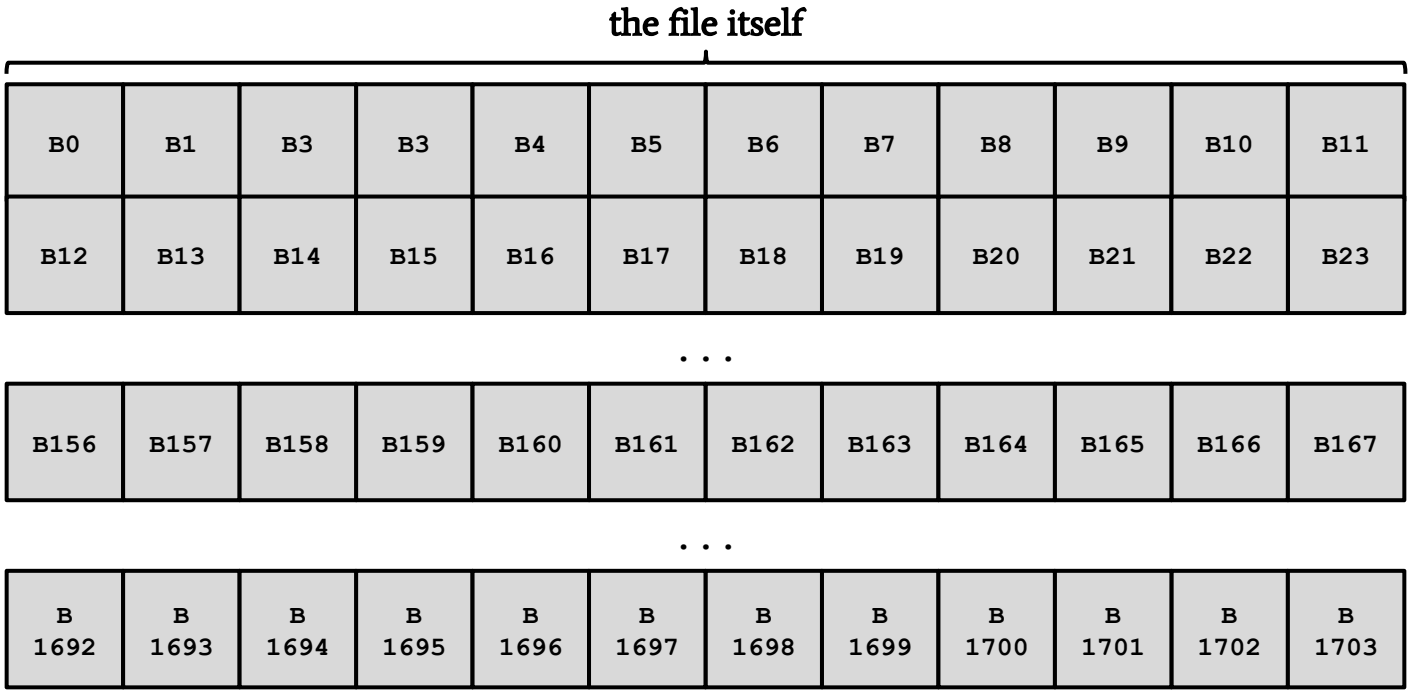


Inode

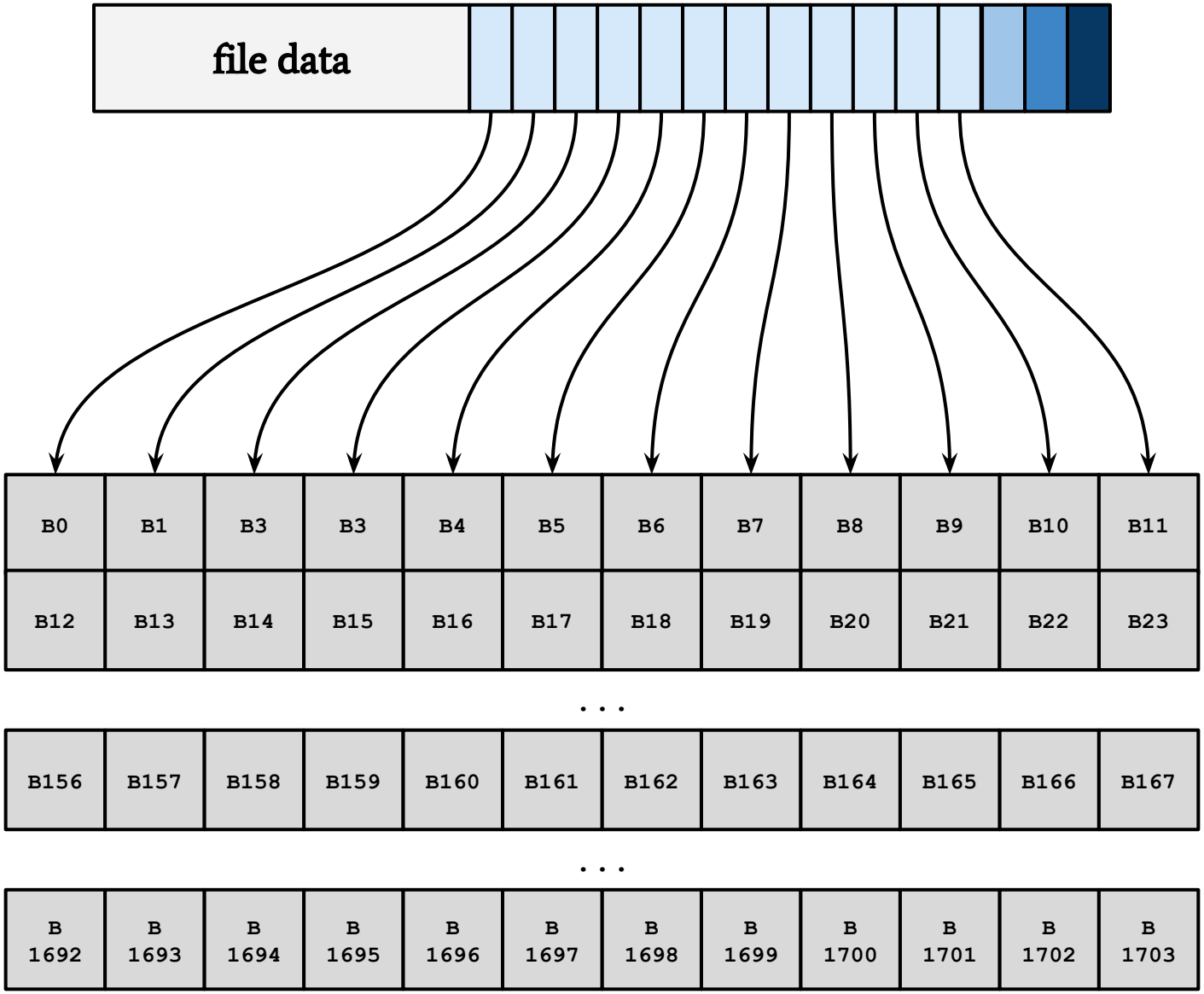


Inode

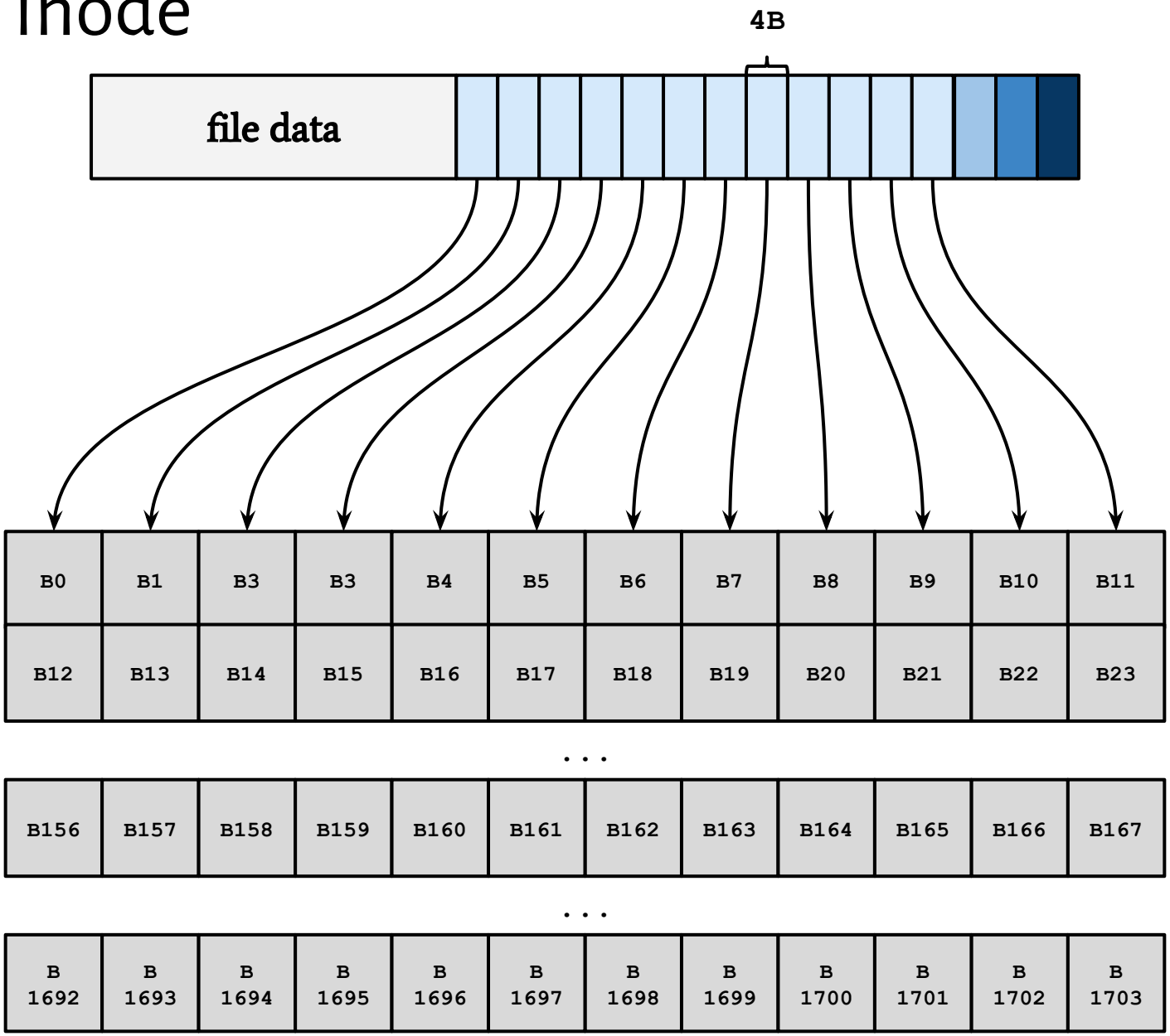

file data


the file itself
B0
B1
B3
B3
B4
B5
B6
B7
B8
B9
B10
B11
B12
B13
B14
B15
B16
B17
B18
B19
B20
B21
B22
B23
. . .
B156
B157
B158
B159
B160
B161
B162
B163
B164
B165
B166
B167
. . .
B 1692
B 1693
B 1694
B 1695
B 1696
B 1697
B 1698
B 1699
B 1700
B 1701
B 1702
B 1703

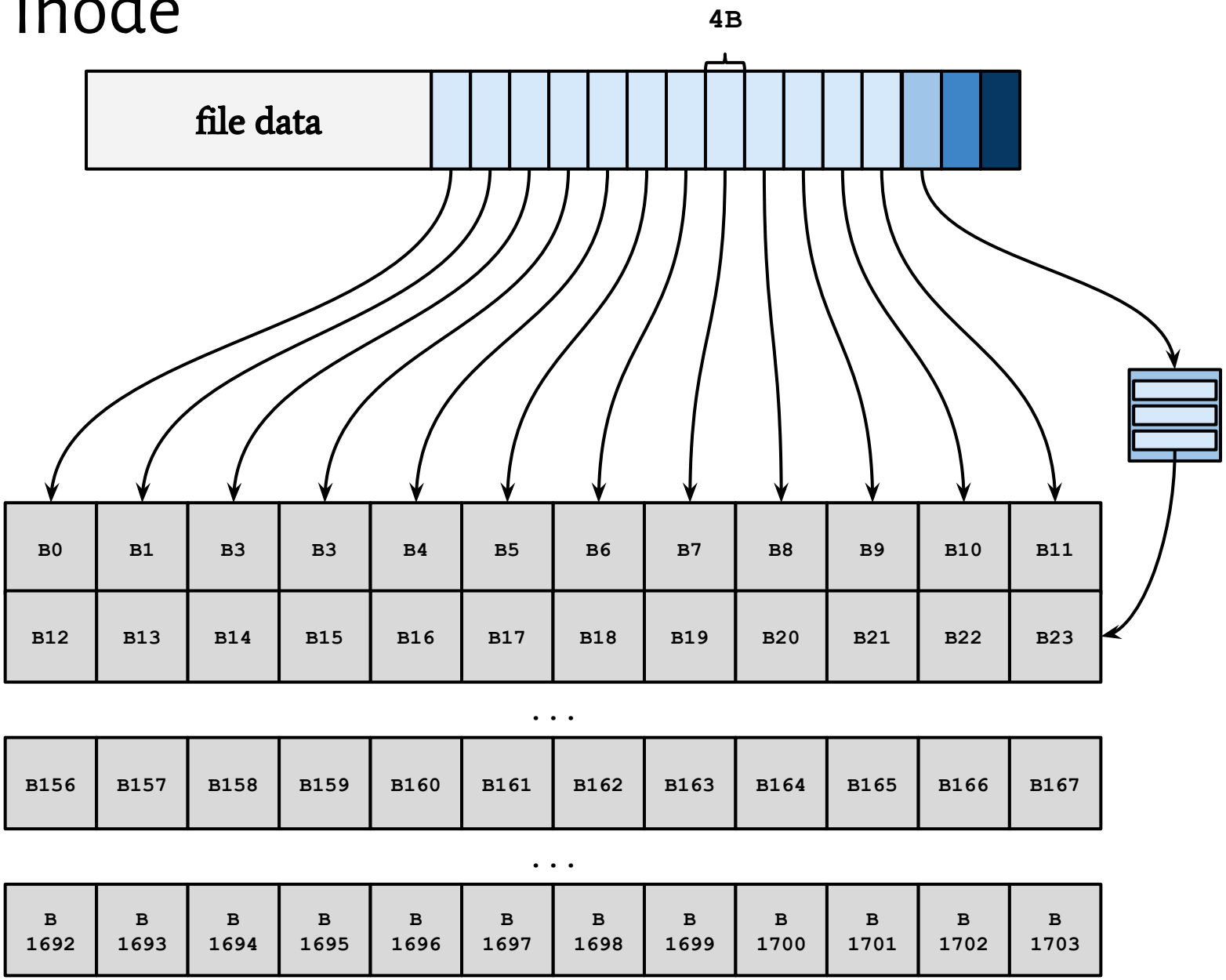
Inode


file data
B0
B1
B3
B3
B4
B5
B6
B7
B8
B9
B10
B11
B12
B13
B14
B15
B16
B17
B18
B19
B20
B21
B22
B23
. . .
B156
B157
B158
B159
B160
B161
B162
B163
B164
B165
B166
B167
. . .
B 1692
B 1693
B 1694
B 1695
B 1696
B 1697
B 1698
B 1699
B 1700
B 1701
B 1702
B 1703

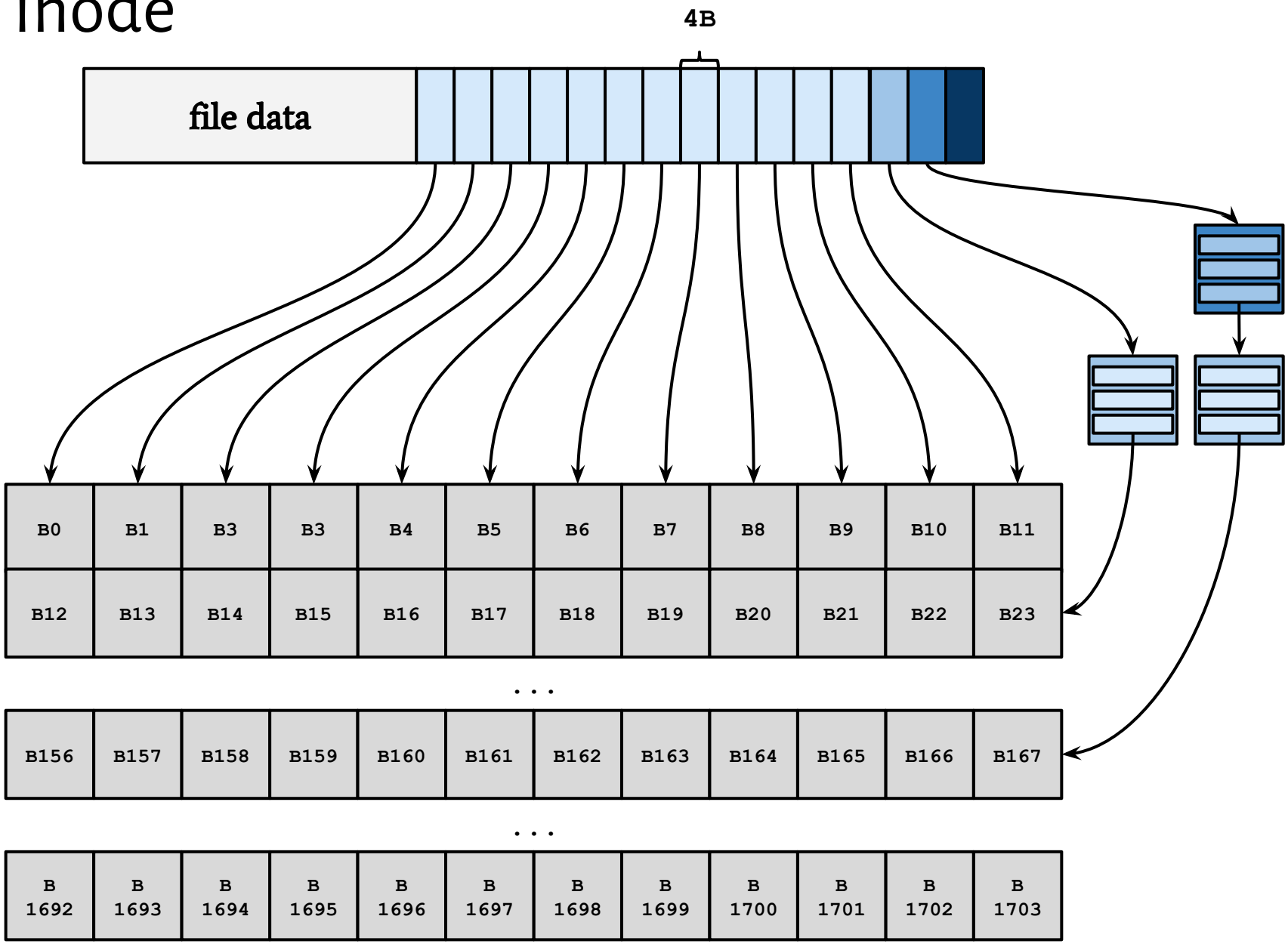
Inode


4B
file data
B0
B1
B3
B3
B4
B5
B6
B7
B8
B9
B10
B11
B12
B13
B14
B15
B16
B17
B18
B19
B20
B21
B22
B23
. . .
B156
B157
B158
B159
B160
B161
B162
B163
B164
B165
B166
B167
. . .
B 1692
B 1693
B 1694
B 1695
B 1696
B 1697
B 1698
B 1699
B 1700
B 1701
B 1702
B 1703

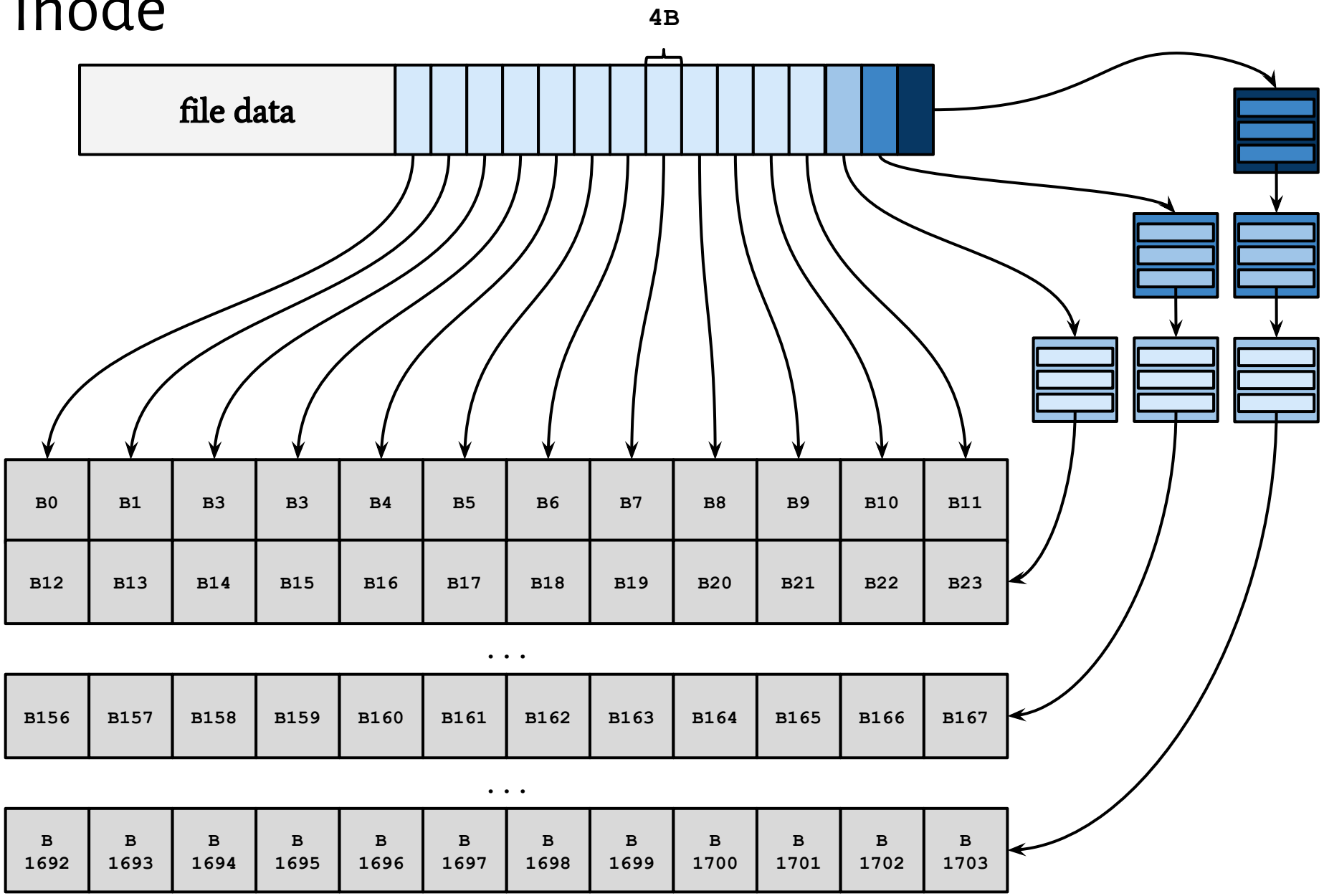
Inode


4B
file data
B0
B1
B3
B3
B4
B5
B6
B7
B8
B9
B10
B11
B12
B13
B14
B15
B16
B17
B18
B19
B20
B21
B22
B23
. . .
B156
B157
B158
B159
B160
B161
B162
B163
B164
B165
B166
B167
. . .
B 1692
B 1693
B 1694
B 1695
B 1696
B 1697
B 1698
B 1699
B 1700
B 1701
B 1702
B 1703

Inode


4B
file data
B0
B1
B3
B3
B4
B5
B6
B7
B8
B9
B10
B11
B12
B13
B14
B15
B16
B17
B18
B19
B20
B21
B22
B23
. . .
B156
B157
B158
B159
B160
B161
B162
B163
B164
B165
B166
B167
. . .
B 1692
B 1693
B 1694
B 1695
B 1696
B 1697
B 1698
B 1699
B 1700
B 1701
B 1702
B 1703

Inode


4B
file data
B0
B1
B3
B3
B4
B5
B6
B7
B8
B9
B10
B11
B12
B13
B14
B15
B16
B17
B18
B19
B20
B21
B22
B23
. . .
B156
B157
B158
B159
B160
B161
B162
B163
B164
B165
B166
B167
. . .
B 1692
B 1693
B 1694
B 1695
B 1696
B 1697
B 1698
B 1699
B 1700
B 1701
B 1702
B 1703

Zadanie C1

rozmiar pliku - **15000 B**

rozmiar bloku - **1 KiB**

rozmiar fragmentu - **1000 B**

Zadanie C1

rozmiar pliku - **15000 B**

rozmiar bloku - **1 KiB**

rozmiar fragmentu - **1000 B**

Ile operacji dyskowych wymaga **wczytanie** fragmentu, którego **względny logiczny adres to 6000**?

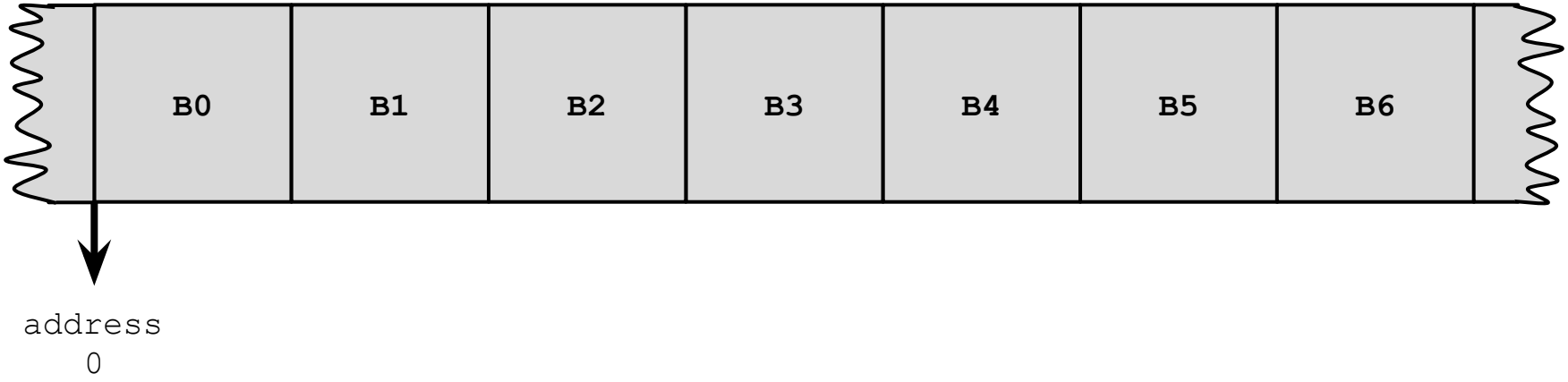
Zadanie C1

rozmiar pliku - **15000 B**

rozmiar bloku - **1 KiB**

rozmiar fragmentu - **1000 B**

Ile operacji dyskowych wymaga **wczytanie** fragmentu, którego **względny logiczny adres to 6000**?



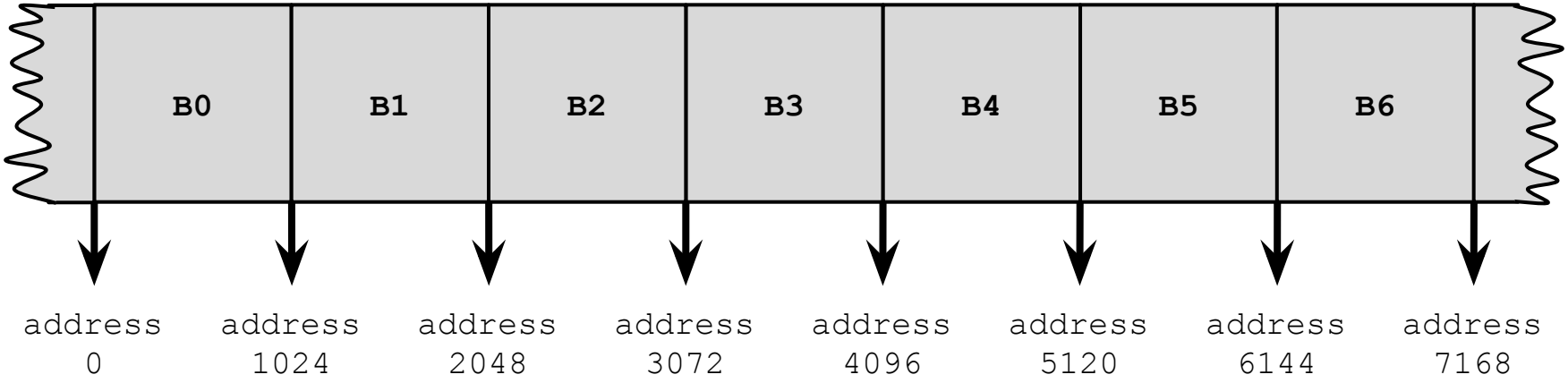
Zadanie C1

rozmiar pliku - **15000 B**

rozmiar bloku - **1 KiB**

rozmiar fragmentu - **1000 B**

Ile operacji dyskowych wymaga **wczytanie** fragmentu, którego **względny logiczny adres to 6000**?



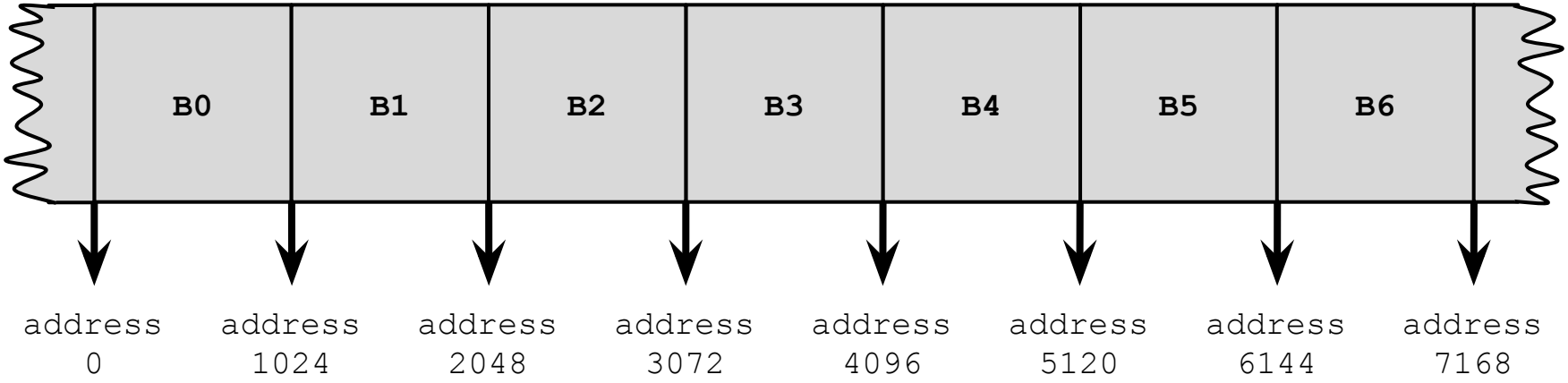
Zadanie C1

rozmiar pliku - **15000 B**

rozmiar bloku - **1 KiB**

rozmiar fragmentu - **1000 B**

Ile operacji dyskowych wymaga **wczytanie** fragmentu, którego **względny logiczny adres to 6000**?

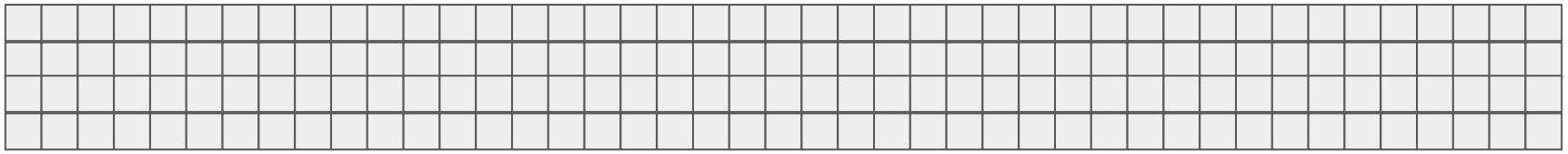


Consecutive blocks of a file are not necessarily next to each other physically.

Btw: defragmented files

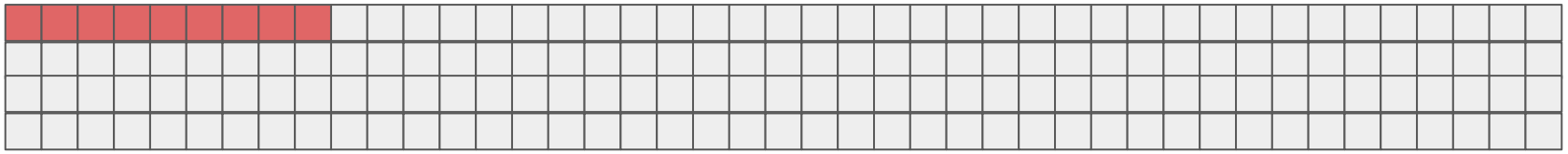
Btw: defragmented files

Microsoft's old FAT file system doesn't attempt to arrange files intelligently:



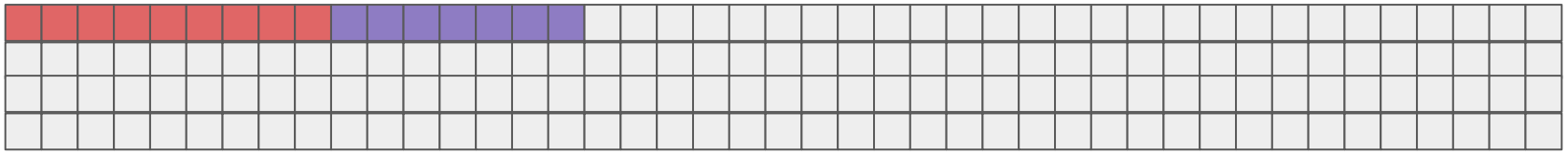
Btw: defragmented files

Microsoft's old FAT file system doesn't attempt to arrange files intelligently:



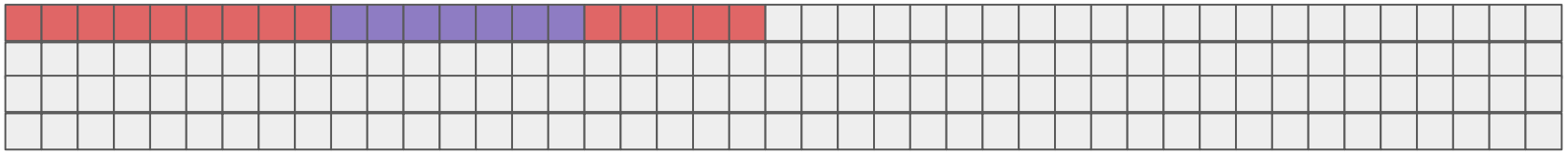
Btw: defragmented files

Microsoft's old FAT file system doesn't attempt to arrange files intelligently:

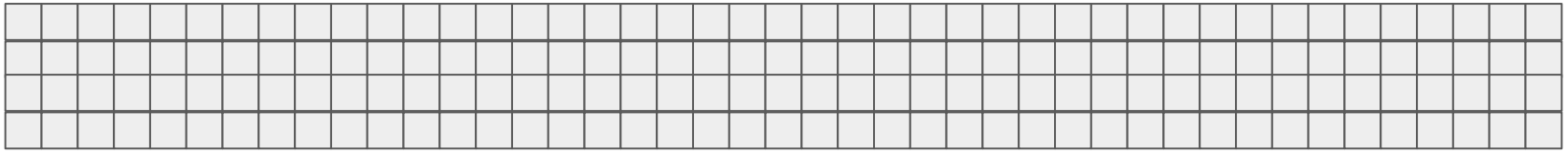


Btw: defragmented files

Microsoft's old FAT file system doesn't attempt to arrange files intelligently:

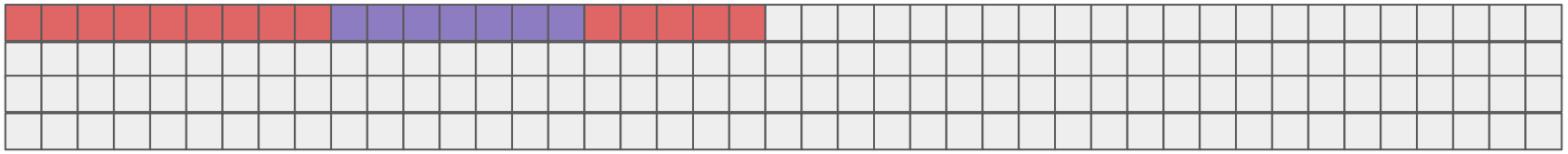


Microsoft's newer NTFS file system (Windows XP and 2000) tries to be a bit smarter:

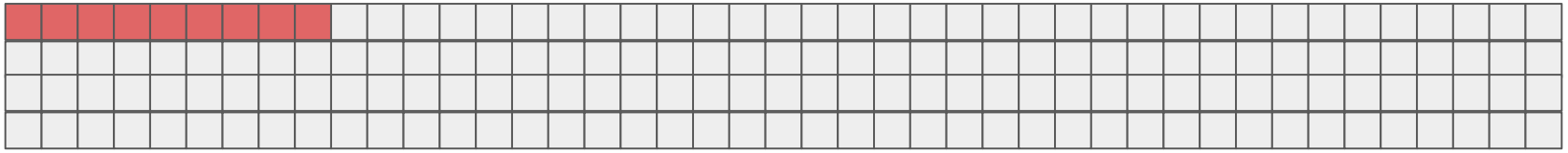


Btw: defragmented files

Microsoft's old FAT file system doesn't attempt to arrange files intelligently:

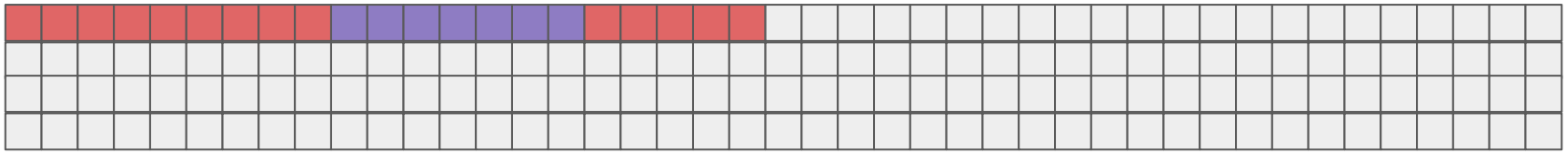


Microsoft's newer NTFS file system (Windows XP and 2000) tries to be a bit smarter:

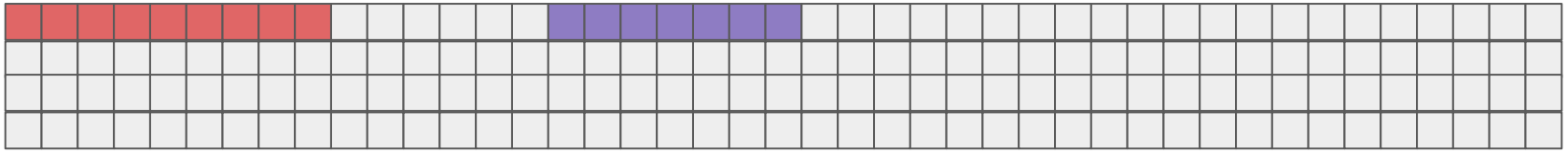


Btw: defragmented files

Microsoft's old FAT file system doesn't attempt to arrange files intelligently:

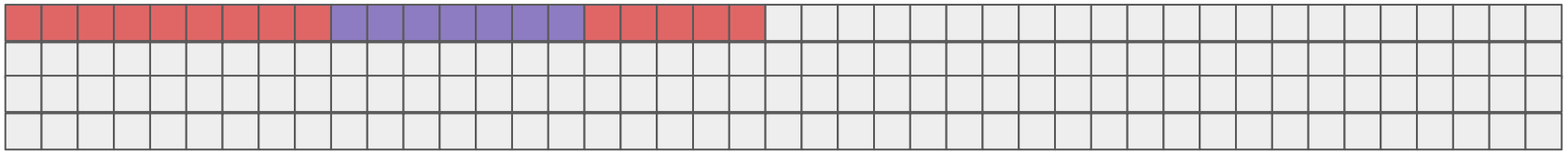


Microsoft's newer NTFS file system (Windows XP and 2000) tries to be a bit smarter:

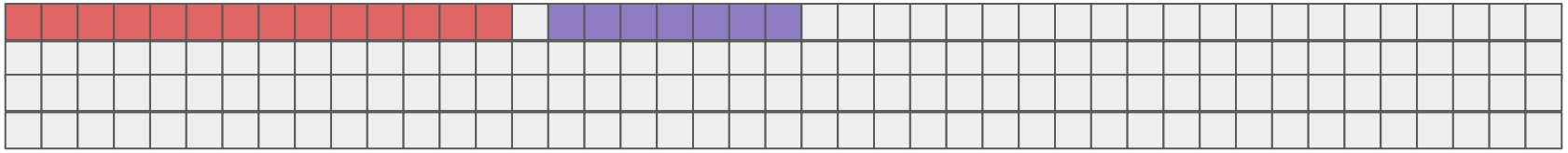


Btw: defragmented files

Microsoft's old FAT file system doesn't attempt to arrange files intelligently:

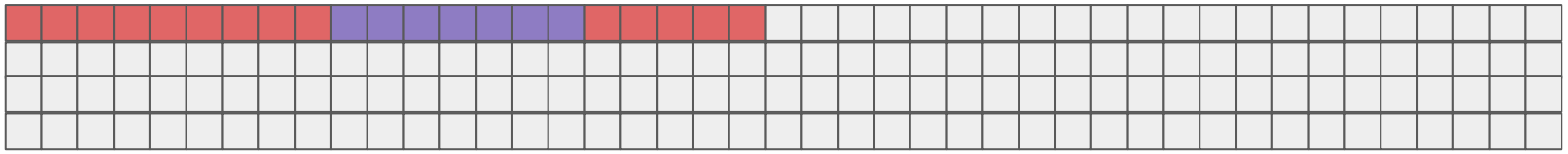


Microsoft's newer NTFS file system (Windows XP and 2000) tries to be a bit smarter:

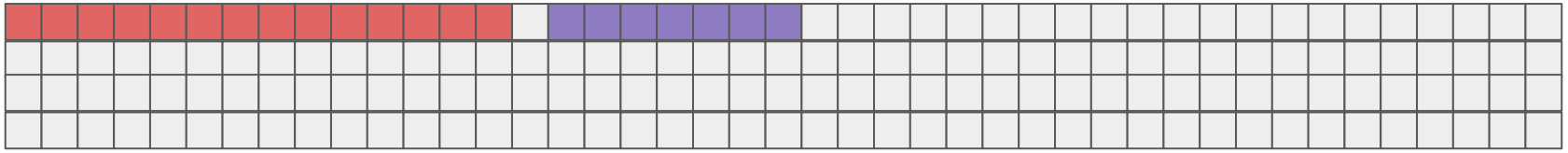


Btw: defragmented files

Microsoft's old FAT file system doesn't attempt to arrange files intelligently:



Microsoft's newer NTFS file system (Windows XP and 2000) tries to be a bit smarter:



Linux file systems (ext2, ext3, ext4) scatter different files all over the disk and reallocate them immediately if defragmentation occurs.

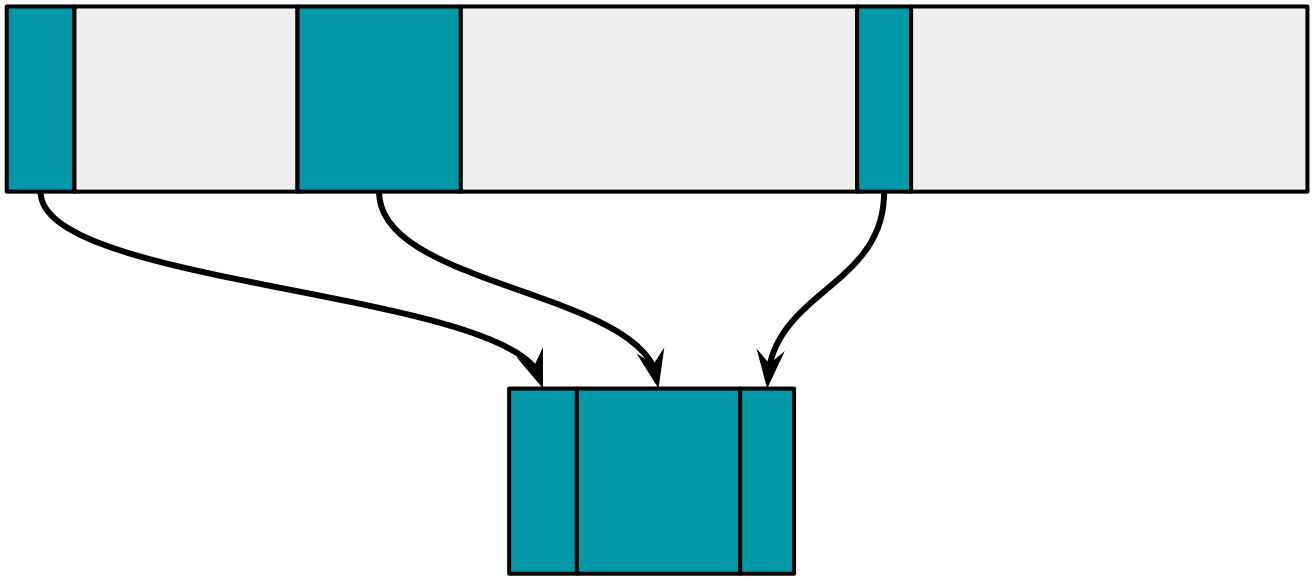
https://www.howtogeek.com/115229/htg-explains-why-linux-doesnt-need-defragmenting/

Btw: sparse files

Create a sparse file:

```
$ dd of=sparse-file bs=1k seek=5120 count=0
```

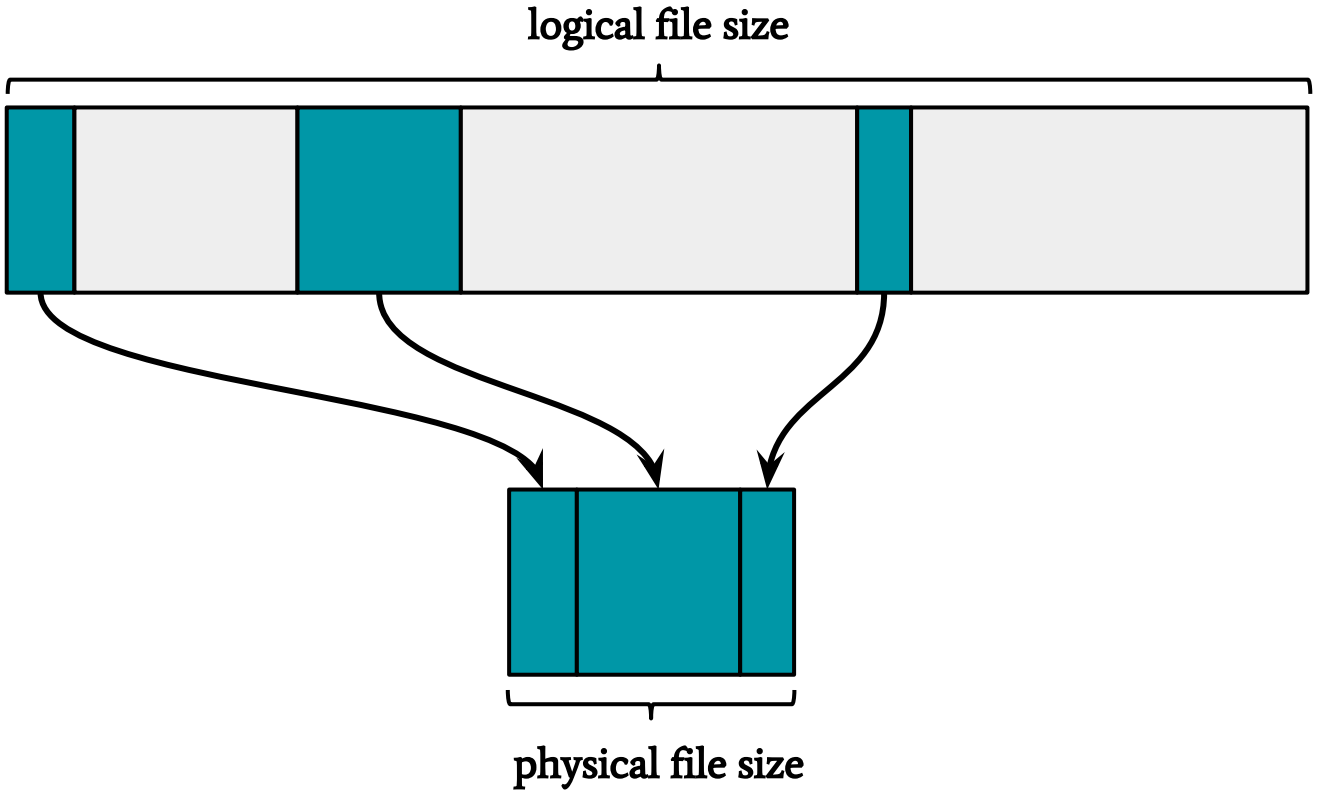
Btw: sparse files



Create a sparse file:

```
$ dd of=sparse-file bs=1k seek=5120 count=0
```

Btw: sparse files



Create a sparse file:

```
$ dd of=sparse-file bs=1k seek=5120 count=0
```

du command prints the occupied space, while **ls** prints the apparent size

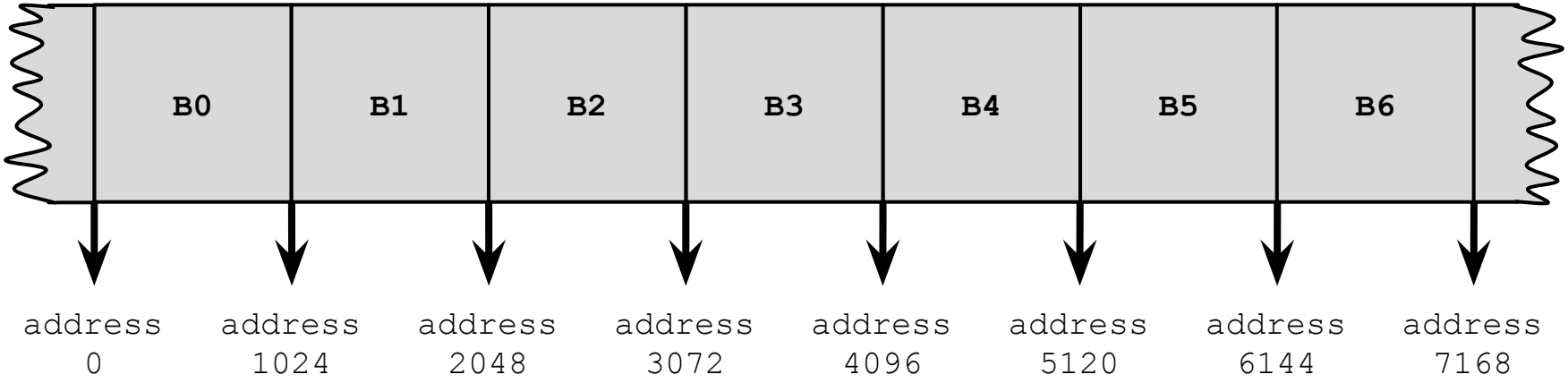
Zadanie C1

rozmiar pliku - **15000 B**

rozmiar bloku - **1 KiB**

rozmiar fragmentu - **1000 B**

Ile operacji dyskowych wymaga **wczytanie** fragmentu, którego **względny logiczny adres to 6000**?



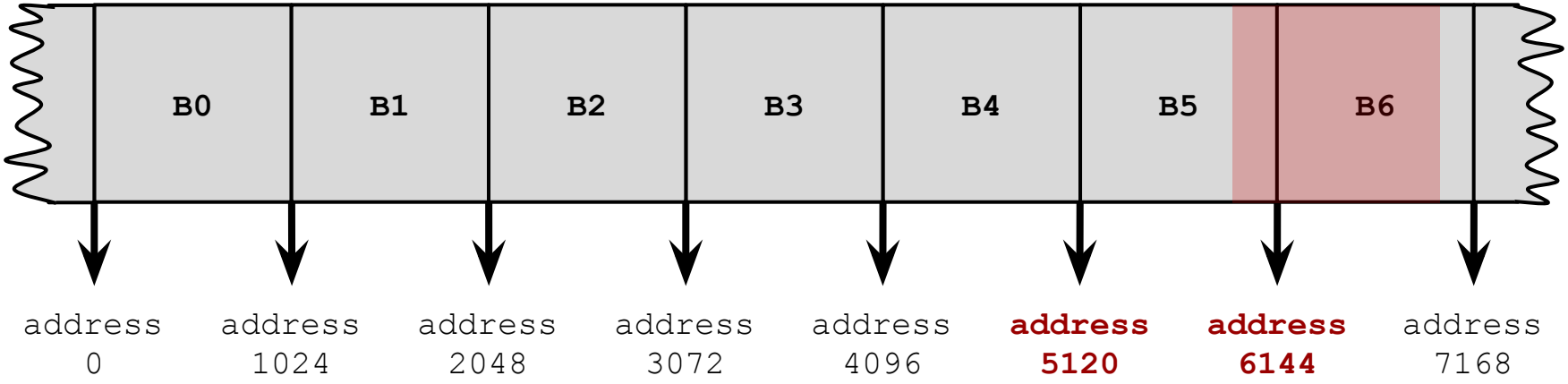
Zadanie C1

rozmiar pliku - **15000 B**

rozmiar bloku - **1 KiB**

rozmiar fragmentu - **1000 B**

Ile operacji dyskowych wymaga **wczytanie** fragmentu, którego **względny logiczny adres to 6000**?



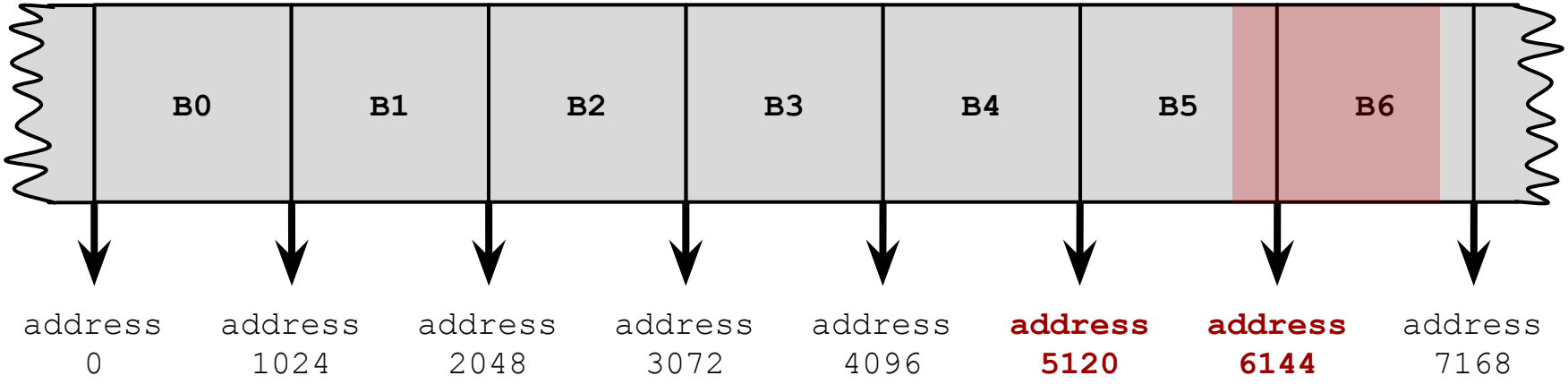
Zadanie C1

rozmiar pliku - **15000 B**

rozmiar bloku - **1 KiB**

rozmiar fragmentu - **1000 B**

Ile operacji dyskowych wymaga **wczytanie** fragmentu, którego **względny logiczny adres to 6000**?



Wczytanie fragmentu wymaga **dwóch** operacji dyskowych read.

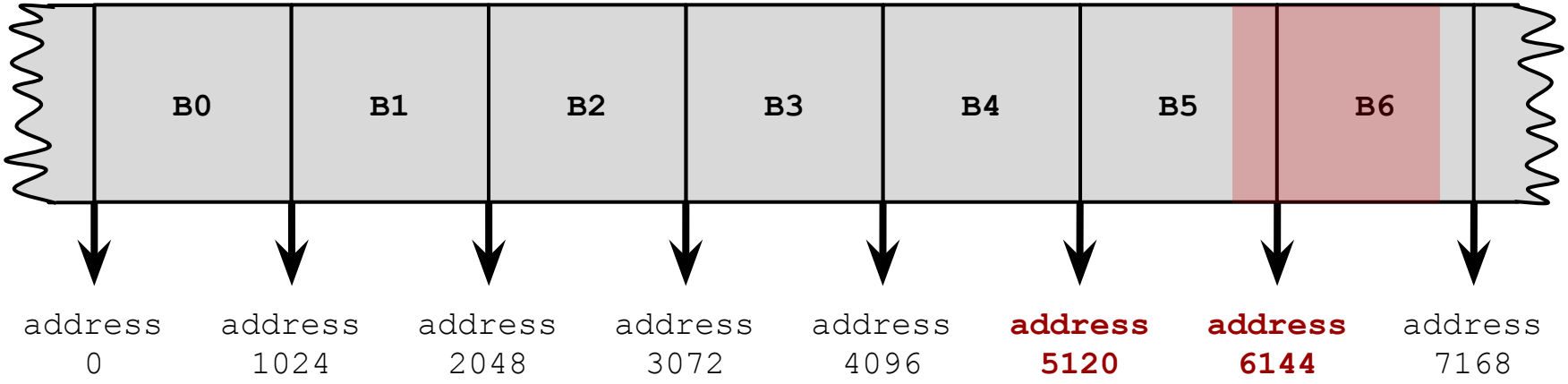
Zadanie C1

rozmiar pliku - **15000 B**

rozmiar bloku - **1 KiB**

rozmiar fragmentu - **1000 B**

Ile operacji dyskowych wymaga **zapisanie** fragmentu, którego **względny logiczny adres to 6000**?



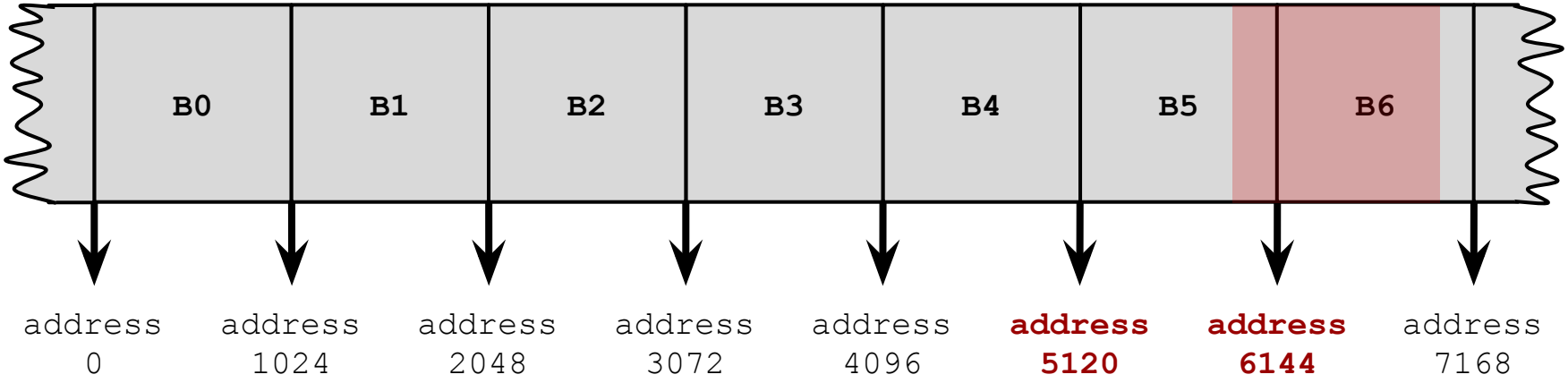
Zadanie C1

rozmiar pliku - **15000 B**

rozmiar bloku - **1 KiB**

rozmiar fragmentu - **1000 B**

Ile operacji dyskowych wymaga **zapisanie** fragmentu, którego **względny logiczny adres to 6000**?



Zapisanie fragmentu wymaga **czterech** operacji dyskowych: dwóch read i dwóch write.

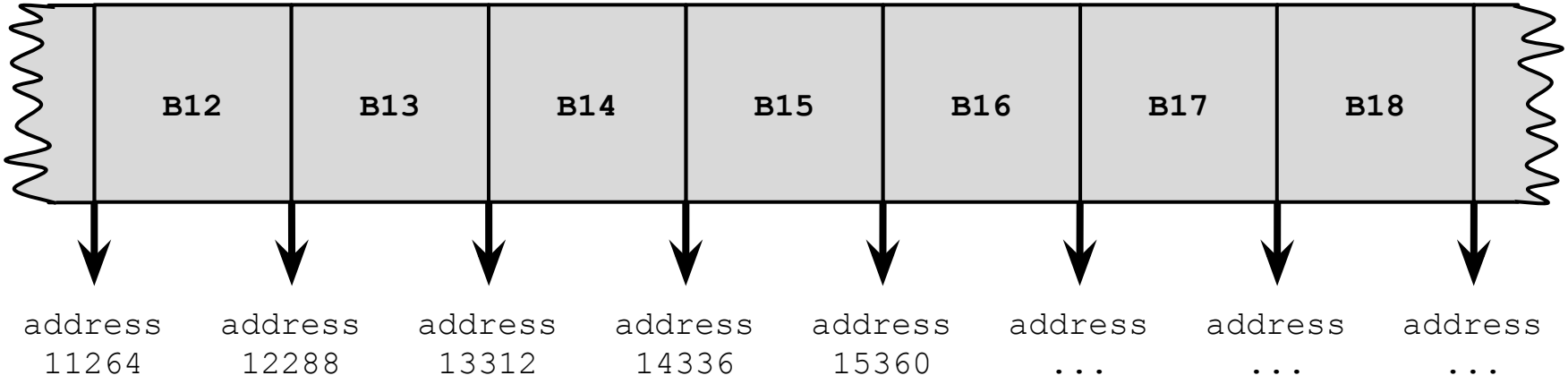
Zadanie C1

rozmiar pliku - **15000 B**

rozmiar bloku - **1 KiB**

rozmiar fragmentu - **1000 B**

Ile operacji dyskowych wymaga **wczytanie** fragmentu, którego **względny logiczny adres to 14000**?



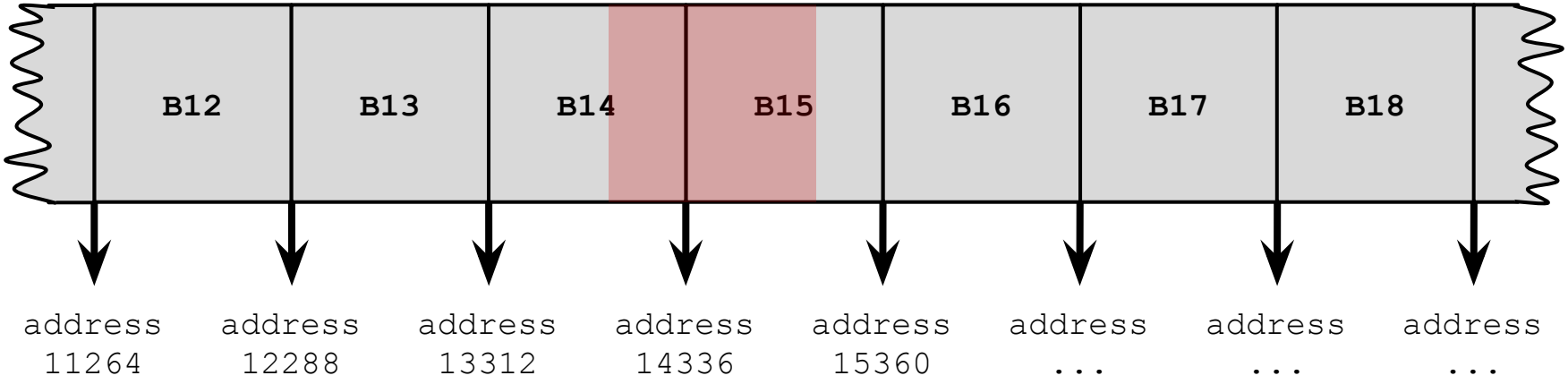
Zadanie C1

rozmiar pliku - **15000 B**

rozmiar bloku - **1 KiB**

rozmiar fragmentu - **1000 B**

Ile operacji dyskowych wymaga **wczytanie** fragmentu, którego **względny logiczny adres to 14000**?



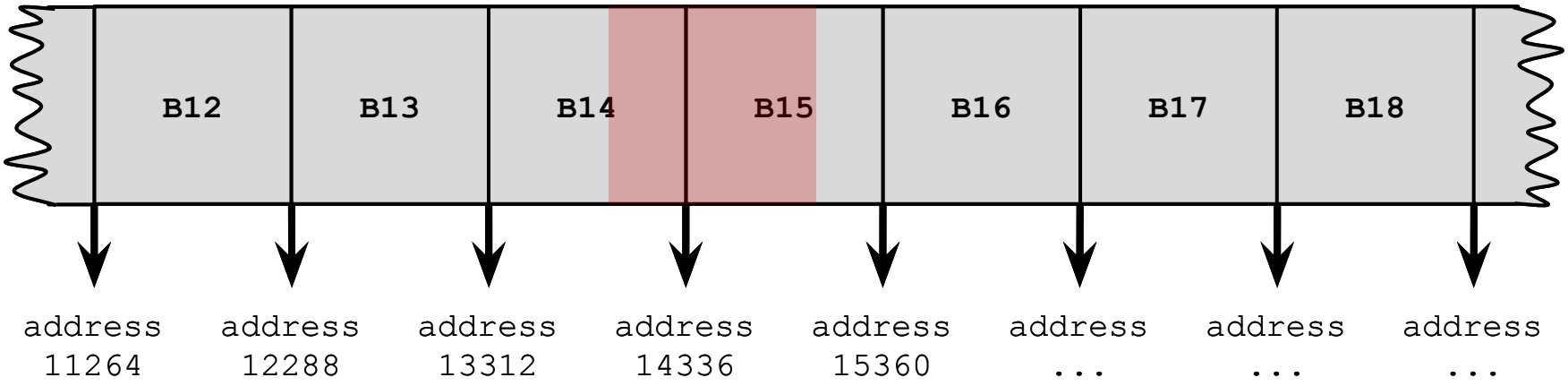
Zadanie C1

rozmiar pliku - **15000 B**

rozmiar bloku - **1 KiB**

rozmiar fragmentu - **1000 B**

Ile operacji dyskowych wymaga **wczytanie** fragmentu, którego **względny logiczny adres to 14000**?



Odczytanie fragmentu wymaga **trzech** operacji dyskowych read:

jednej dla bloku pośredniego + dwóch dla bloków z danymi

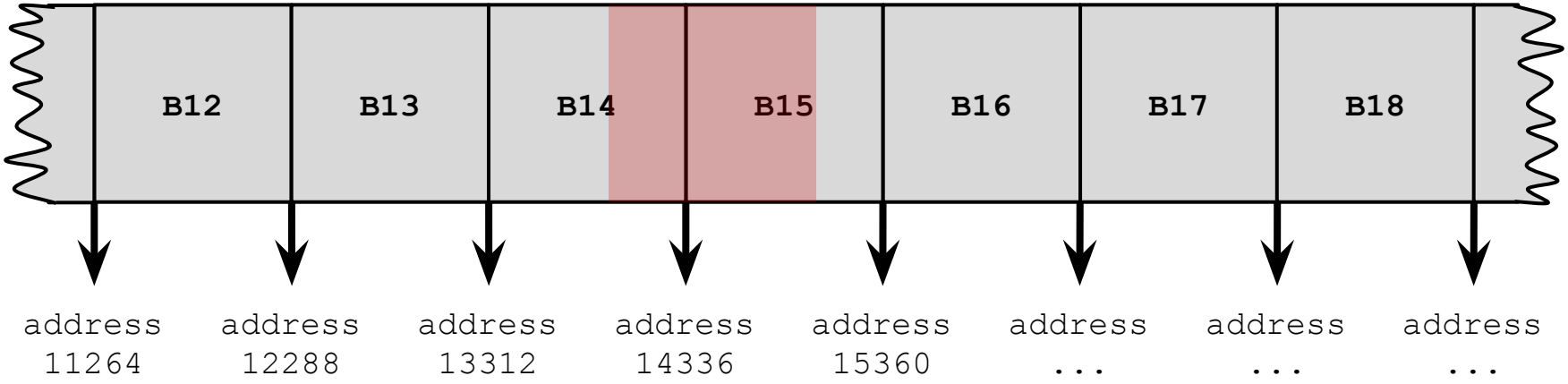
Zadanie C1

rozmiar pliku - **15000 B**

rozmiar bloku - **1 KiB**

rozmiar fragmentu - **1000 B**

Ile operacji dyskowych wymaga **zapisanie** fragmentu, którego **względny logiczny adres to 14000**?



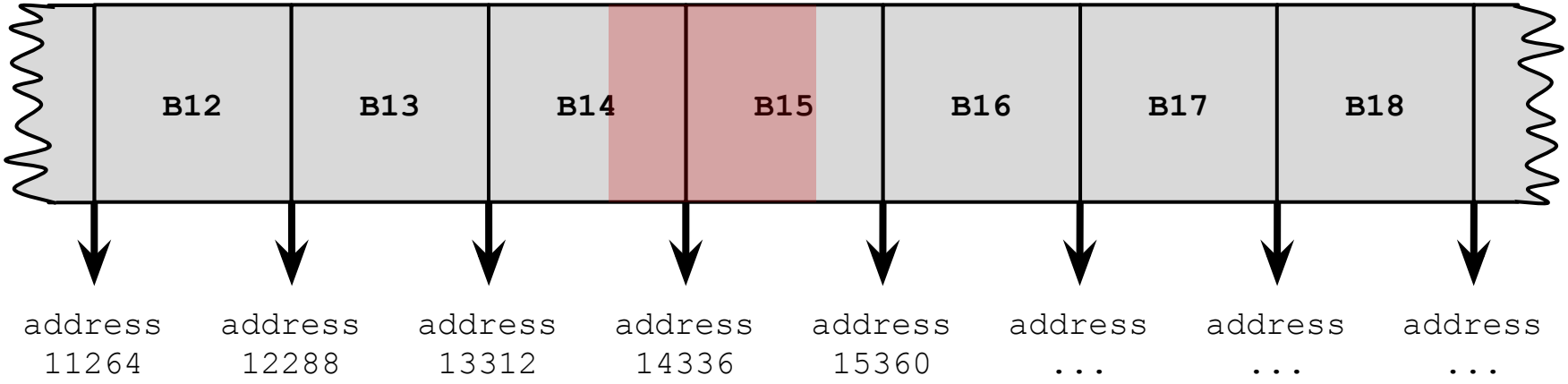
Zadanie C1

rozmiar pliku - **15000 B**

rozmiar bloku - **1 KiB**

rozmiar fragmentu - **1000 B**

Ile operacji dyskowych wymaga **zapisanie** fragmentu, którego **względny logiczny adres to 14000**?



Odczytanie fragmentu wymaga **pięciu** operacji dyskowych:

read dla bloku pośredniego + dwóch read bloków z danymi + dwóch write

Zadanie C1

rozmiar pliku - **15000 B**

rozmiar bloku - **1 KiB**

rozmiar fragmentu - **1000 B**

Ile operacji dyskowych wymaga **zapisanie** fragmentu, którego **względny logiczny adres to 14000**?

Odczytanie fragmentu wymaga **pięciu** operacji dyskowych:

read dla bloku pośredniego + dwóch read bloków z danymi + dwóch write

W jakiej sytuacji uzyskamy najmniejszą liczbę operacji dyskowych?

Zadanie C1

rozmiar pliku - **15000 B**

rozmiar bloku - **1 KiB**

rozmiar fragmentu - **1000 B**

Ile operacji dyskowych wymaga **zapisanie** fragmentu, którego **względny logiczny adres to 14000**?

Odczytanie fragmentu wymaga **pięciu** operacji dyskowych:

read dla bloku pośredniego + dwóch read bloków z danymi + dwóch write

W jakiej sytuacji uzyskamy najmniejszą liczbę operacji dyskowych?

W sytuacji, gdy jest to pierwszy zapis do bloku o numerze 14 i nie został jeszcze przydzielony blok pośredni.

Zadanie C1

rozmiar pliku - **15000 B**

rozmiar bloku - **1 KiB**

rozmiar fragmentu - **1000 B**

Ile operacji dyskowych wymaga **zapisanie** fragmentu, którego **względny logiczny adres to 14000**?

Odczytanie fragmentu wymaga **pięciu** operacji dyskowych:

read dla bloku pośredniego + dwóch read bloków z danymi + dwóch write

W jakiej sytuacji uzyskamy najmniejszą liczbę operacji dyskowych?

W sytuacji, gdy jest to pierwszy zapis do bloku o numerze 14 i nie został jeszcze przydzielony blok pośredni.

Liczba operacji dyskowych:

Zadanie C1

rozmiar pliku - **15000 B**

rozmiar bloku - **1 KiB**

rozmiar fragmentu - **1000 B**

Ile operacji dyskowych wymaga **zapisanie** fragmentu, którego **względny logiczny adres to 14000**?

Odczytanie fragmentu wymaga **pięciu** operacji dyskowych:

read dla bloku pośredniego + dwóch read bloków z danymi + dwóch write

W jakiej sytuacji uzyskamy najmniejszą liczbę operacji dyskowych?

W sytuacji, gdy jest to pierwszy zapis do bloku o numerze 14 i nie został jeszcze przydzielony blok pośredni.

Liczba operacji dyskowych: **trzy** operacje write.

Zadanie C2

rozmiar pliku - **2 GiB = 2^{31} B**

rozmiar bloku - **1 KiB**

Ile **maksymalnie** bloków dyskowych może być potrzebne do zapamiętania tego pliku?

Zadanie C2

rozmiar pliku - **2 GiB = 2^{31} B**

rozmiar bloku - **1 KiB**

Ile **maksymalnie** bloków dyskowych może być potrzebne do zapamiętania tego pliku?

➔ liczba bloków z danymi = 2^{31} B / 2^{10} B = **2^{21}**

➔ liczba adresów do bloków fizycznych w bloku pośrednim = 2^{10} / 2^{2} = 2^{8}

➔ liczba bloków pośrednich pierwszego stopnia potrzebna do zaadresowania bloków z danymi = $\lceil (2^{21} - 12) / 2^{8} \rceil$ = **2^{13}**

Uwaga: jeden z bloków pośrednich będzie zawierał nie 2^{8} adresów, ale $(2^{8} - 12)$, bo pierwszych 12 bloków jest adresowanych w i-węźle.

➔ liczba bloków pośrednich drugiego stopnia = $\lceil (2^{13} - 1) / 2^{8} \rceil$ = **2^{5}**

➔ liczba bloków pośrednich trzeciego stopnia = $\lceil 2^{5} / 2^{8} \rceil$ = **1**

W sumie potrzebne jest **$(2^{21} + 2^{13} + 2^{5} + 1)$** bloków.

Zadanie C2

rozmiar pliku - **2 GiB = 2^{31} B**

rozmiar bloku - **1 KiB**

Ile **minimalnie** bloków dyskowych może być potrzebne do zapamiętania tego pliku?

Zadanie C2

rozmiar pliku - **2 GiB = 2^{31} B**

rozmiar bloku - **1 KiB**

Ile **minimalnie** bloków dyskowych może być potrzebne do zapamiętania tego pliku?

Załóżmy, że plik jest dziurawy i jego jedyne dane są zapisane na samym końcu - w ostatnim bloku. Wówczas potrzebujemy tylko jednego bloku, który pomieści dane zapisane pod adresem logicznym $2^{31} - 2^{10}$ oraz trzech bloków pośrednich, które będą trzymać informację o adresie fizycznym tego bloku. W sumie potrzebujemy więc **czterech** bloków.

Zadanie C3

Gdy rozmiar pliku w i-węźle systemu ext2 był opisywany liczbą 32-bitową i jej najstarszy bit był zarezerwowany, plik mógł mieć wielkość co najwyżej 2 GiB.

We współczesnych systemach ext2 i ext3 32-bitowe pole i-węzła o nazwie i-blocks przechowuje **rozmiar pliku wyrażony w liczbie 512 bajtowych porcji**.

Jak zmieniło się ograniczenie na rozmiar pliku?

Co jeszcze ogranicza rozmiar pliku?

Co ogranicza rozmiar partycji?

Zadanie C3

Gdy rozmiar pliku w i-węźle systemu ext2 był opisywany liczbą 32-bitową i jej najstarszy bit był zarezerwowany, plik mógł mieć wielkość co najwyżej 2 GiB.

We współczesnych systemach ext2 i ext3 32-bitowe pole i-węzła o nazwie i-blocks przechowuje **rozmiar pliku wyrażony w liczbie 512 bajtowych porcji**.

Jak zmieniło się ograniczenie na rozmiar pliku?

Teraz możemy zapisać rozmiar pliku równy aż 2^9 B * 2^{32} = 2^{41} B.

Co jeszcze ogranicza rozmiar pliku?

Co ogranicza rozmiar partycji?

Zadanie C3

Gdy rozmiar pliku w i-węźle systemu ext2 był opisywany liczbą 32-bitową i jej najstarszy bit był zarezerwowany, plik mógł mieć wielkość co najwyżej 2 GiB.

We współczesnych systemach ext2 i ext3 32-bitowe pole i-węzła o nazwie i-blocks przechowuje **rozmiar pliku wyrażony w liczbie 512 bajtowych porcji**.

Jak zmieniło się ograniczenie na rozmiar pliku?

Teraz możemy zapisać rozmiar pliku równy aż 2^9 B * 2^{32} = 2^{41} B.

Co jeszcze ogranicza rozmiar pliku?

Liczba bloków pośrednich (jest tylko jeden blok trzeciego stopnia).

Co ogranicza rozmiar partycji?

Zadanie C3

Gdy rozmiar pliku w i-węźle systemu ext2 był opisywany liczbą 32-bitową i jej najstarszy bit był zarezerwowany, plik mógł mieć wielkość co najwyżej 2 GiB.

We współczesnych systemach ext2 i ext3 32-bitowe pole i-węzła o nazwie i-blocks przechowuje **rozmiar pliku wyrażony w liczbie 512 bajtowych porcji**.

Jak zmieniło się ograniczenie na rozmiar pliku?

Teraz możemy zapisać rozmiar pliku równy aż 2^9 B * 2^{32} = 2^{41} B.

Co jeszcze ogranicza rozmiar pliku?

Liczba bloków pośrednich (jest tylko jeden blok trzeciego stopnia).

Co ogranicza rozmiar partycji?

Przestrzeń adresowa bloków fizycznych (4 B na adres bloku → 2^{32} bloków)

Zadanie C3

Gdy rozmiar pliku w i-węźle systemu ext2 był opisywany liczbą 32-bitową i jej najstarszy bit był zarezerwowany, plik mógł mieć wielkość co najwyżej 2 GiB.

We współczesnych systemach ext2 i ext3 32-bitowe pole i-węzła o nazwie i-blocks przechowuje **rozmiar pliku wyrażony w liczbie 512 bajtowych porcji**.

Uzupełnij tabelkę:

block size (bs)	max file size	max partition size
1 KiB = 2^{10} B		
2 KiB = 2^{11} B		
4 KiB = 2^{12} B		
8 KiB = 2^{13} B		

Zadanie C3

Gdy rozmiar pliku w i-węźle systemu ext2 był opisywany liczbą 32-bitową i jej najstarszy bit był zarezerwowany, plik mógł mieć wielkość co najwyżej 2 GiB.

We współczesnych systemach ext2 i ext3 32-bitowe pole i-węzła o nazwie i-blocks przechowuje **rozmiar pliku wyrażony w liczbie 512 bajtowych porcji**.

Uzupełnij tabelkę:

block size (bs)	**max file size**	**max partition size**
1 KiB = 2^{10} B	2^{34} B	$2^{32} * 2^{10} = 2^{42}$
2 KiB = 2^{11} B	2^{38} B	$2^{32} * 2^{11} = 2^{43}$
4 KiB = 2^{12} B	2^{41} B	$2^{32} * 2^{12} = 2^{44}$
8 KiB = 2^{13} B	2^{41} B	$2^{32} * 2^{13} = 2^{45}$

max file size = min(12 + (bs/4) + (bs/4)^2 + (bs/4)^3) * bs, 2^{41})

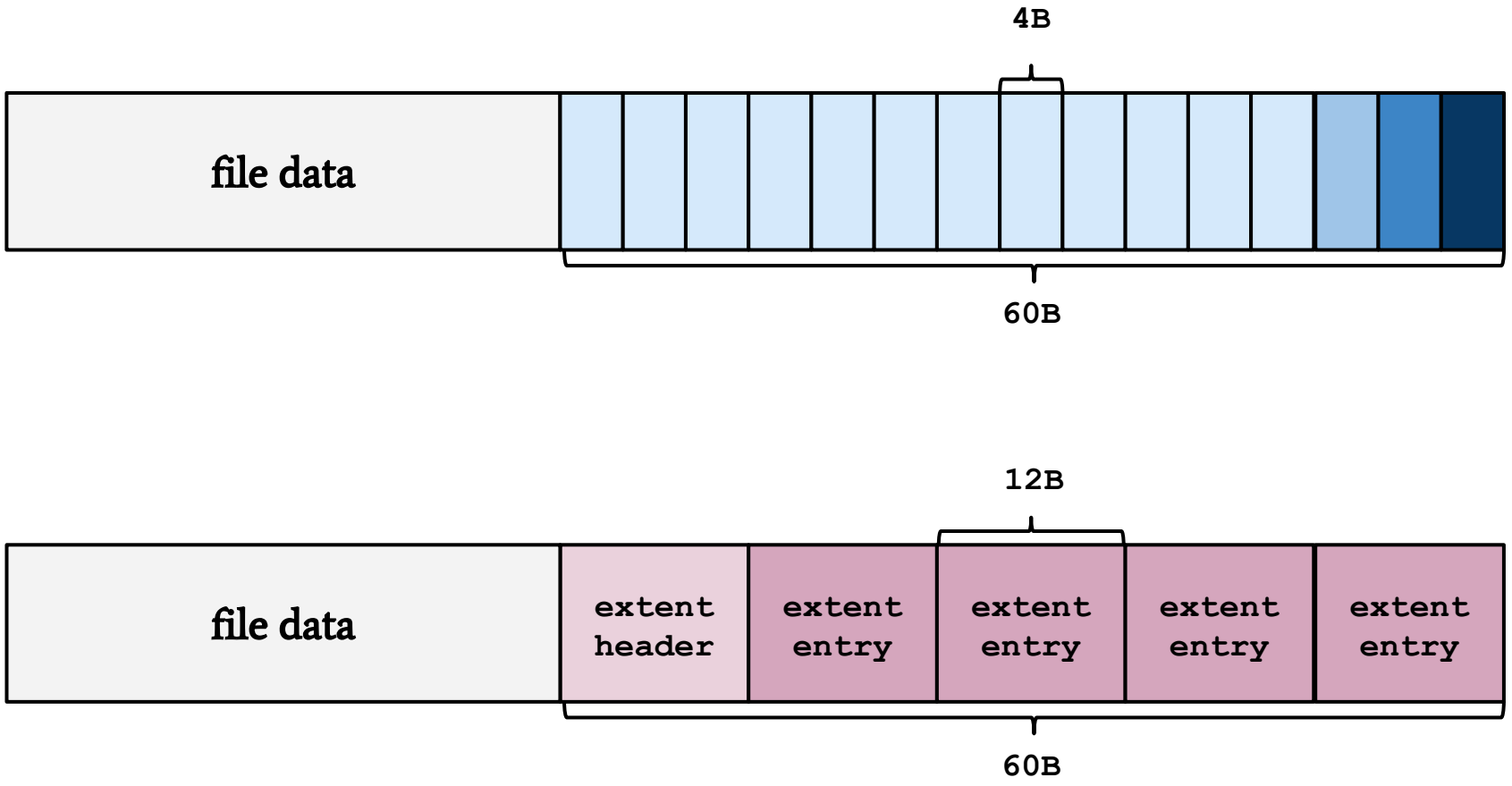
Ext4

★ **64 bits** to keep the size of a file in bytes.

★ **48 bits** to address physical blocks.

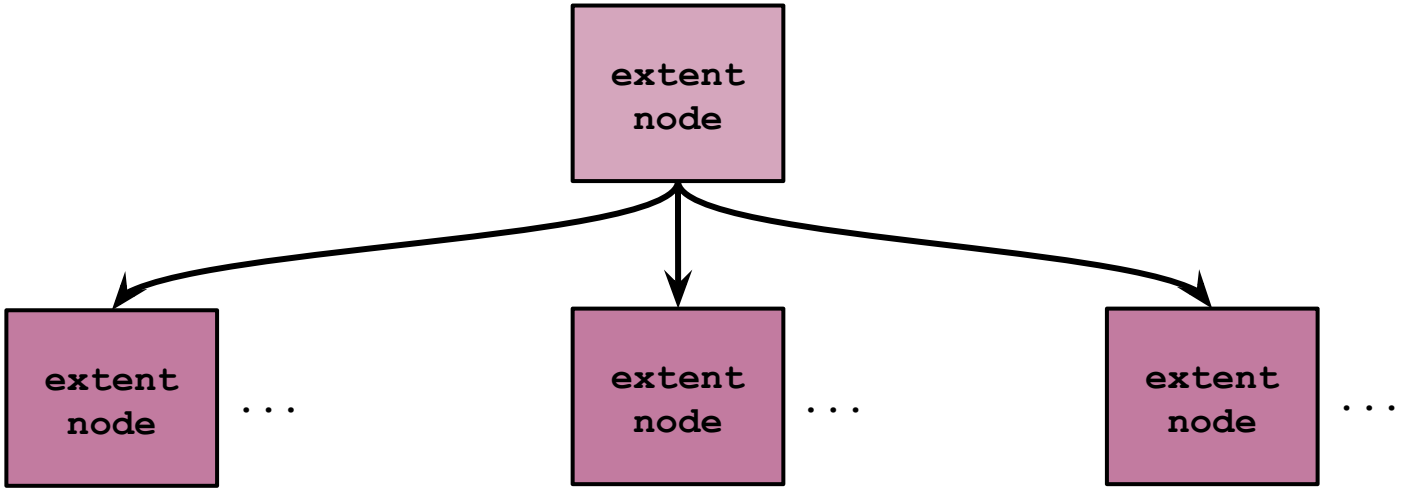
https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout#The_Contents_of_inode.i_block

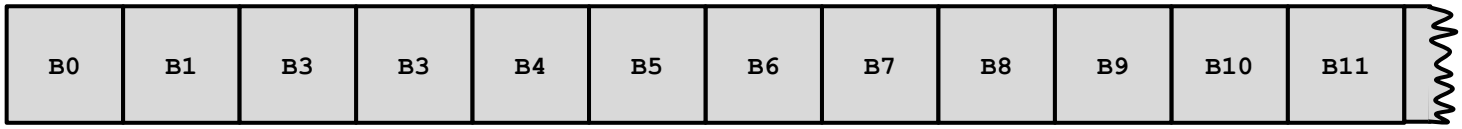
Ext4



https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout#The_Contents_of_inode.i_block

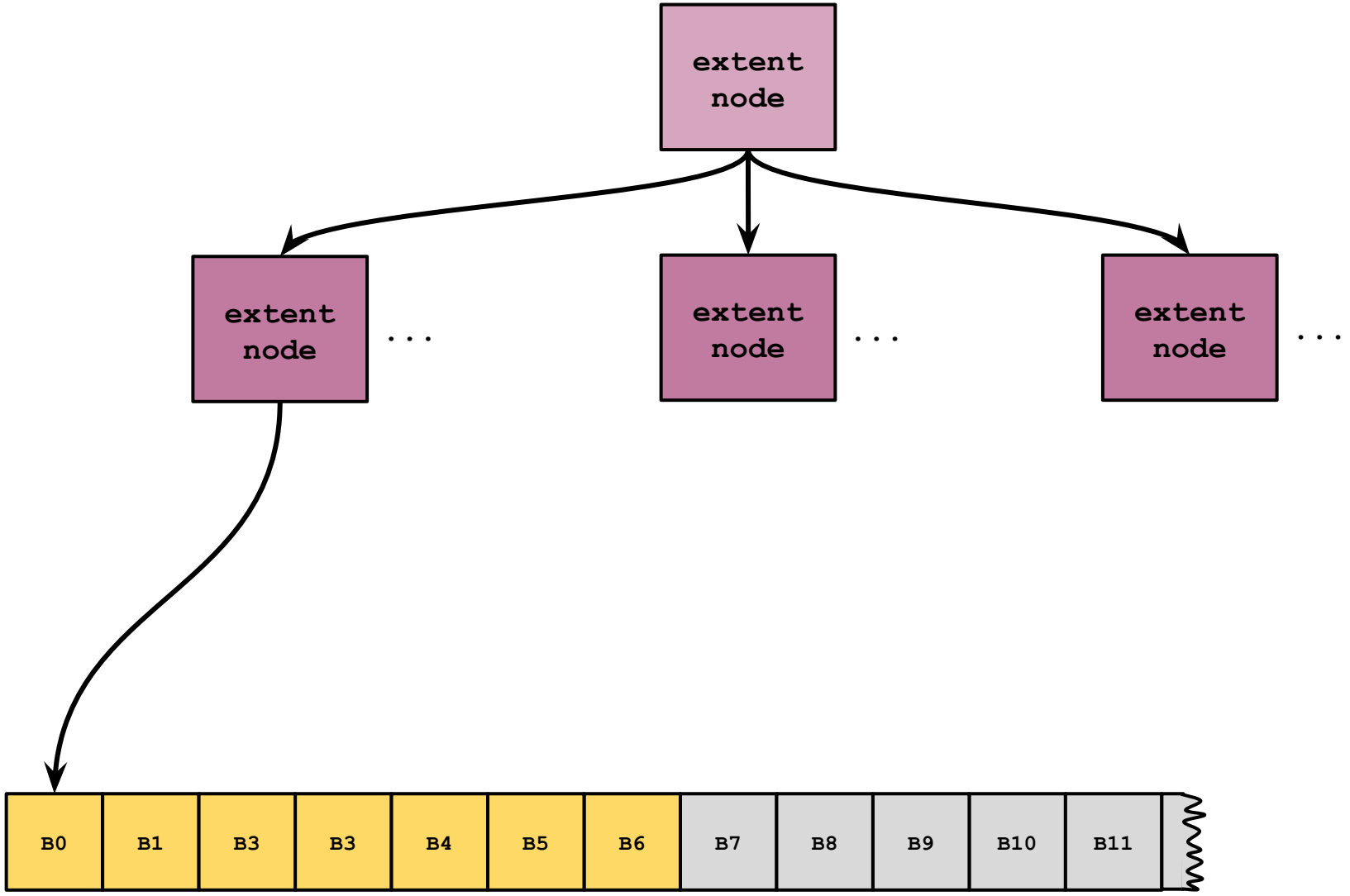
Ext4





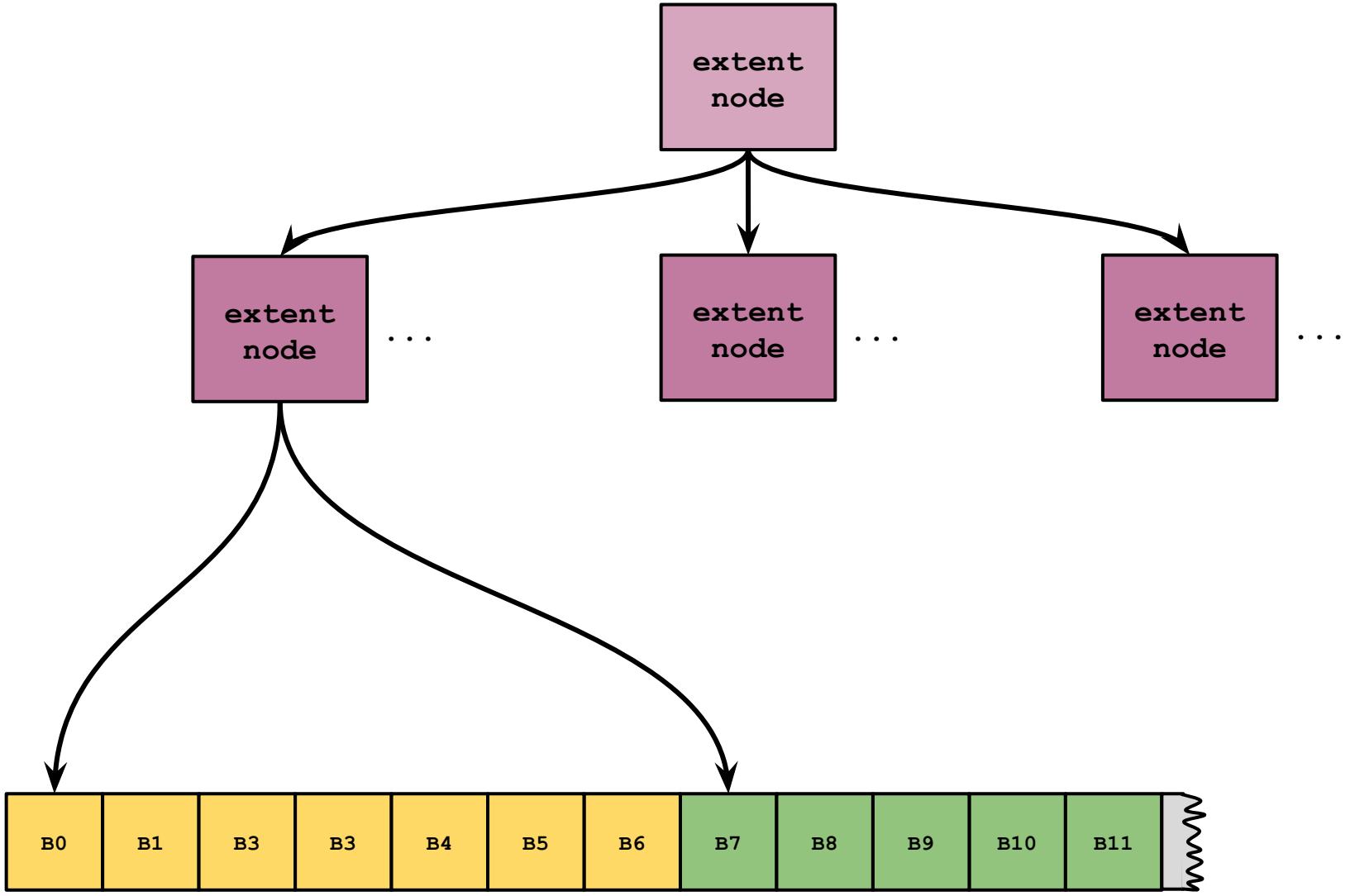
https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout#The_Contents_of_inode.i_block

Ext4



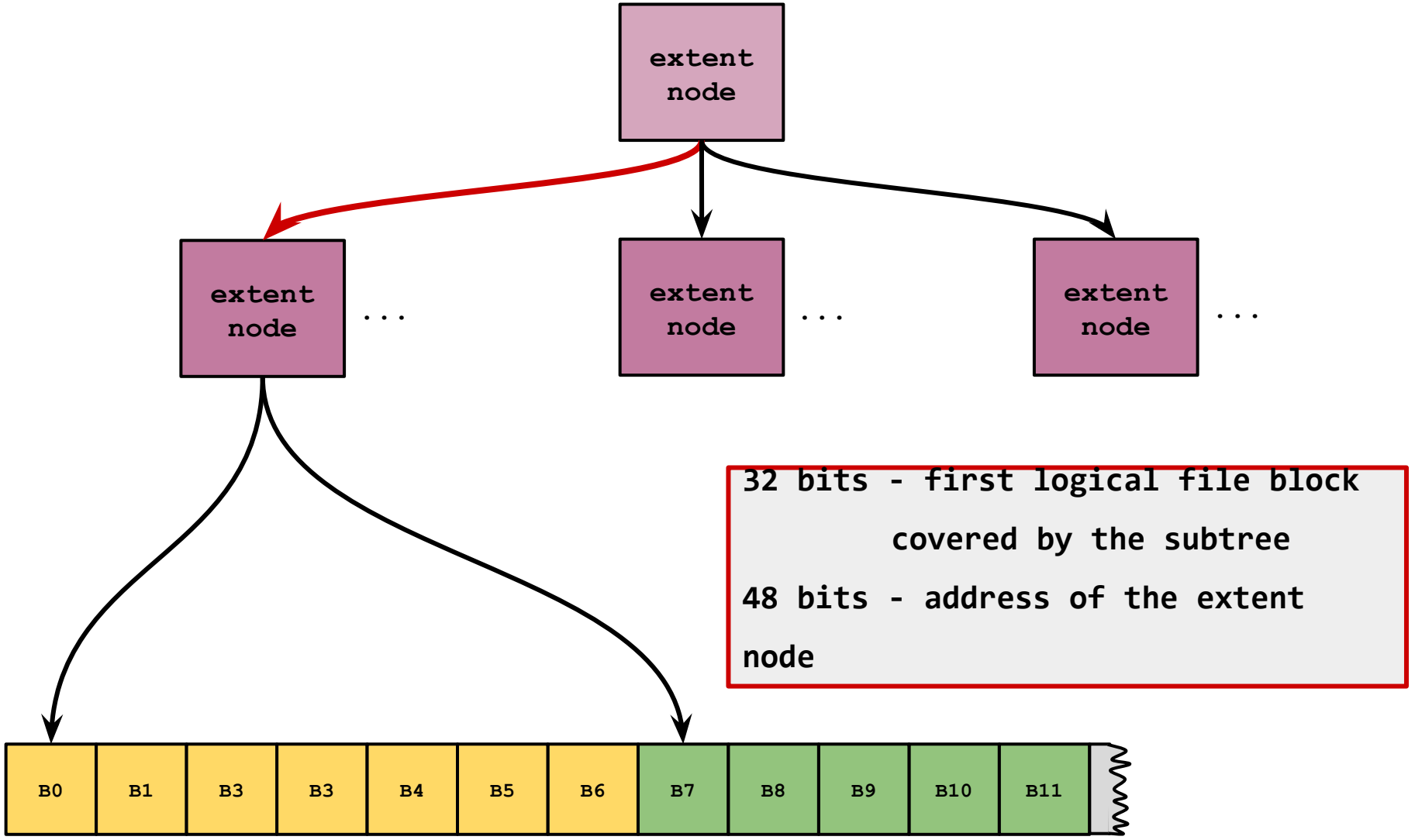
https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout#The_Contents_of_inode.i_block

Ext4



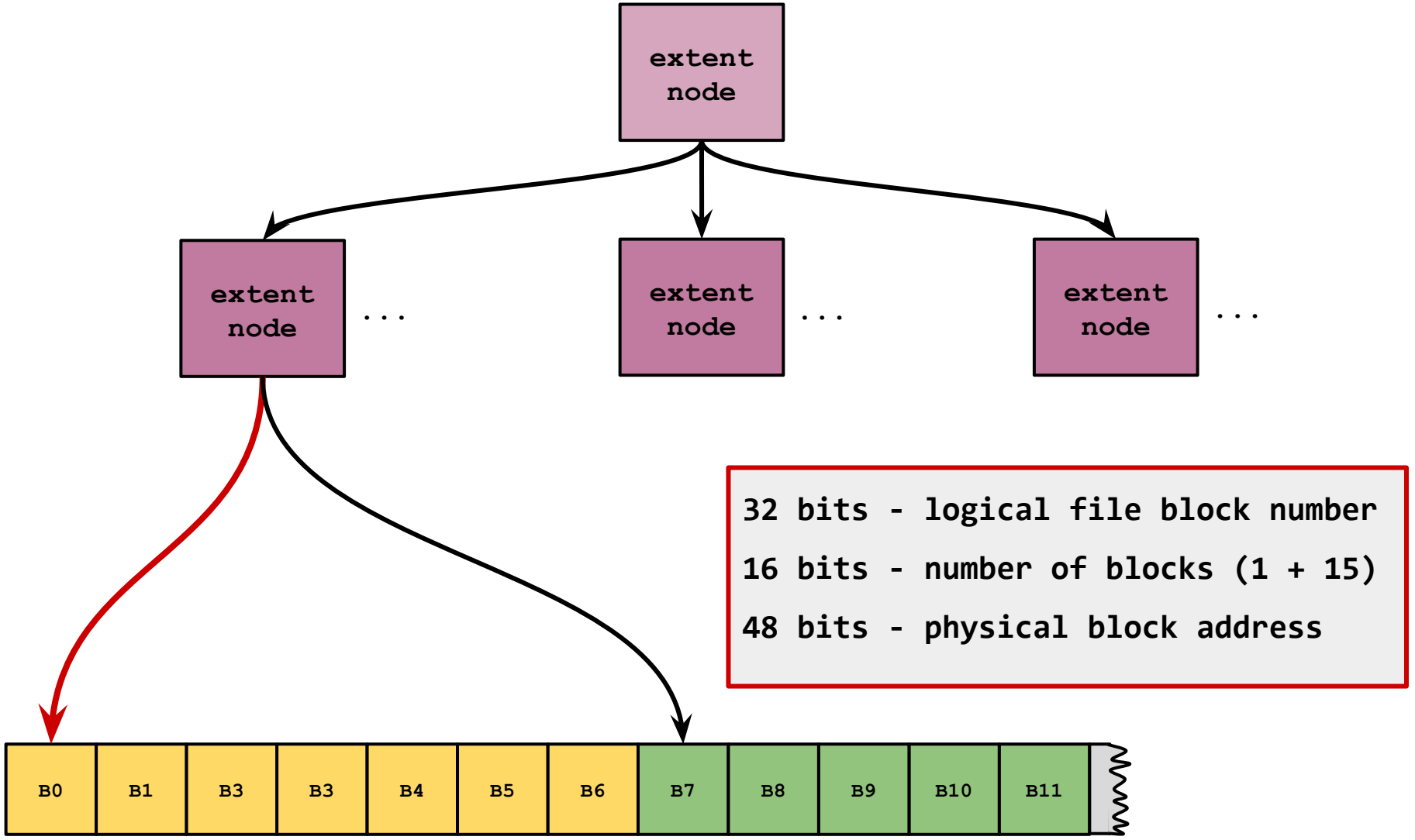
https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout#The_Contents_of_inode.i_block

Ext4



https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout#The_Contents_of_inode.i_block

Ext4



https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout#The_Contents_of_inode.i_block

Zadanie D1

Wiedząc, że w systemie ext4 numery logiczne bloków są 32-bitowe, a numery fizyczne bloków są 48-bitowe, wypełnij tabelkę jak w zadaniu 3 dla rozmiaru bloku 1, 2 i 4 KiB.

block size (bs)	max file size	max partition size
1 KiB = 2^{10} B		
2 KiB = 2^{11} B		
4 KiB = 2^{12} B		
8 KiB = 2^{13} B		

Zadanie D1

Wiedząc, że w systemie ext4 numery logiczne bloków są 32-bitowe, a numery fizyczne bloków są 48-bitowe, wypełnij tabelkę jak w zadaniu 3 dla rozmiaru bloku 1, 2 i 4 KiB.

block size (bs)	max file size	max partition size
1 KiB = 2^{10} B	$2^{32} * 2^{10} = 2^{42}$	$2^{48} * 2^{10} = 2^{58}$
2 KiB = 2^{11} B	$2^{32} * 2^{11} = 2^{43}$	$2^{49} * 2^{10} = 2^{59}$
4 KiB = 2^{12} B	$2^{32} * 2^{12} = 2^{44}$	$2^{50} * 2^{10} = 2^{60}$
8 KiB = 2^{13} B	$2^{32} * 2^{13} = 2^{45}$	$2^{51} * 2^{10} = 2^{61}$

W praktyce rozmiar pliku ogranicza przestrzeń adresowa bloków logicznych pliku:

max file size = $\min(2^{32} * bs, 2^{64})$

max partition size = $2^{48} * bs$

Zadanie D2

Ustalmy, że rozmiar bloku wynosi 4 KiB. Porównaj, ile bloków dyskowych jest potrzebne do zapamiętania pełnego pliku (bez dziur) o rozmiarze 512 MiB w systemach ext3 i ext4.

Zadanie D2

Ustalmy, że rozmiar bloku wynosi 4 KiB. Porównaj, ile bloków dyskowych jest potrzebne do zapamiętania pełnego pliku (bez dziur) o rozmiarze 512 MiB w systemach ext3 i ext4.

ext3

liczba bloków z danymi = $2^{29} / 2^{12} = \mathbf{2^{17}}$

liczba adresów w bloku = $2^{12} / 2^{2} = 2^{10}$

liczba bloków pośrednich = $\lceil (2^{17} - 12) / 2^{10} \rceil = \mathbf{2^{7}}$

liczba bloków pośrednich II stopnia = $\lceil (2^{7} - 1) / 2^{10} \rceil = \mathbf{1}$

W sumie potrzebujemy $\mathbf{(2^{17} + 2^{7} + 1)}$ bloków.

Zadanie D2

Ustalmy, że rozmiar bloku wynosi 4 KiB. Porównaj, ile bloków dyskowych jest potrzebne do zapamiętania pełnego pliku (bez dziur) o rozmiarze 512 MiB w systemach ext3 i ext4.

ext4

liczba bloków z danymi = $2^{29} / 2^{12} = \mathbf{2^{17}}$

Zakładając, że plik nie jest pofragmentowany, ale zajmuje ciągłą przestrzeń na dysku, nie potrzebujemy dodatkowych bloków do trzymania adresów. Wystarczą nam dwa z czterech *extent entry*, które mieszczą się w i-węźle. (Jedno *extent entry* może wskazywać na ciąg 2^{15} bloków.)

Deep-sea Lizardfish

http://www.iflscience.com/plants-and-animals/this-video-of-a-deepsea-monster-will-give-you-nightmares/