Оперативни системи 2

Факултет техничких наука Косовска Митровица

Драгиша Миљковић

Имплементација система датотека

Имена датотека – подсећање

Различите врсте имена су прикладније у различитим контекстима:

inode – индексни чвор

- јединствено име за сваку датотеку у датом систему
- у оквиру инода се чувају метаподаци о датотеци: величина, права приступа, итд.

path – путања

- људима је лако да их упамте
- датотеке су организоване хијерархијски

file descriptor – дескриптор датотеке

- ово су целобројне вредности
- помоћу њих се избегавају честе претраге директоријума
- потребно је памтити више помераја за следеће читање или уписивање

API датотека – подсећање

```
int fd = open(char *path, int flag, mode_t mode)

read(int fd, void *buf, size_t nbyte)

write(int fd, void *buf, size_t nbyte)

close(int fd)
```

Имплементација система датотека

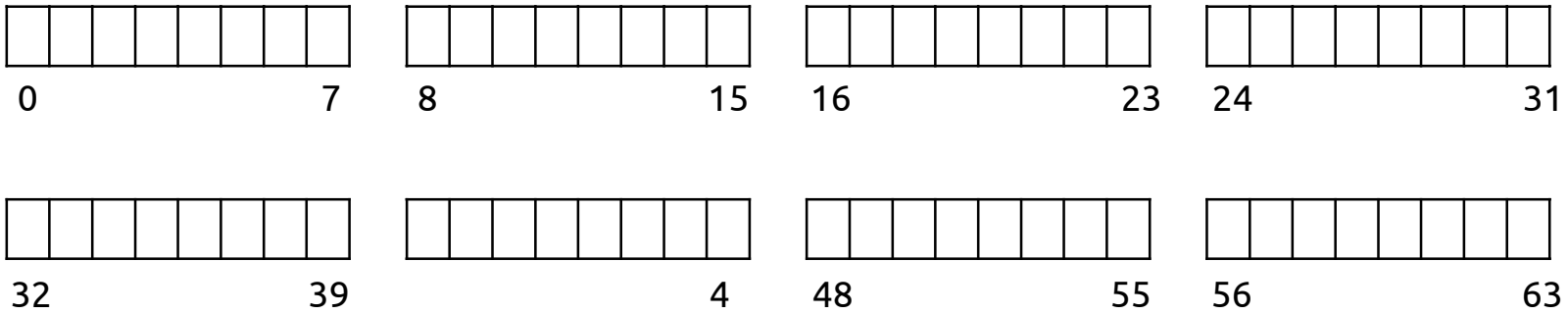
- Које типове **структура података** користи систем датотека?
- Како систем датотека организује своје **податке** и **метаподатке**?
- Потребно је разумевање **метода за приступ систему датотека**
  - open(), read(), write(), итд.

Општа организација

Дађемо општи пример организације структуре података система датотека.

Диск је подељен на **низ блокова**.

- Величина блока је (у овом примеру, а и најчешће у пракси) четири килобајта.
- Блокови су адресирани од 0 до N-1.
- Желимо да помоћу неке структуре мапирамо датотеке у блокове на диску.



Пример – провера величине блока

Windows

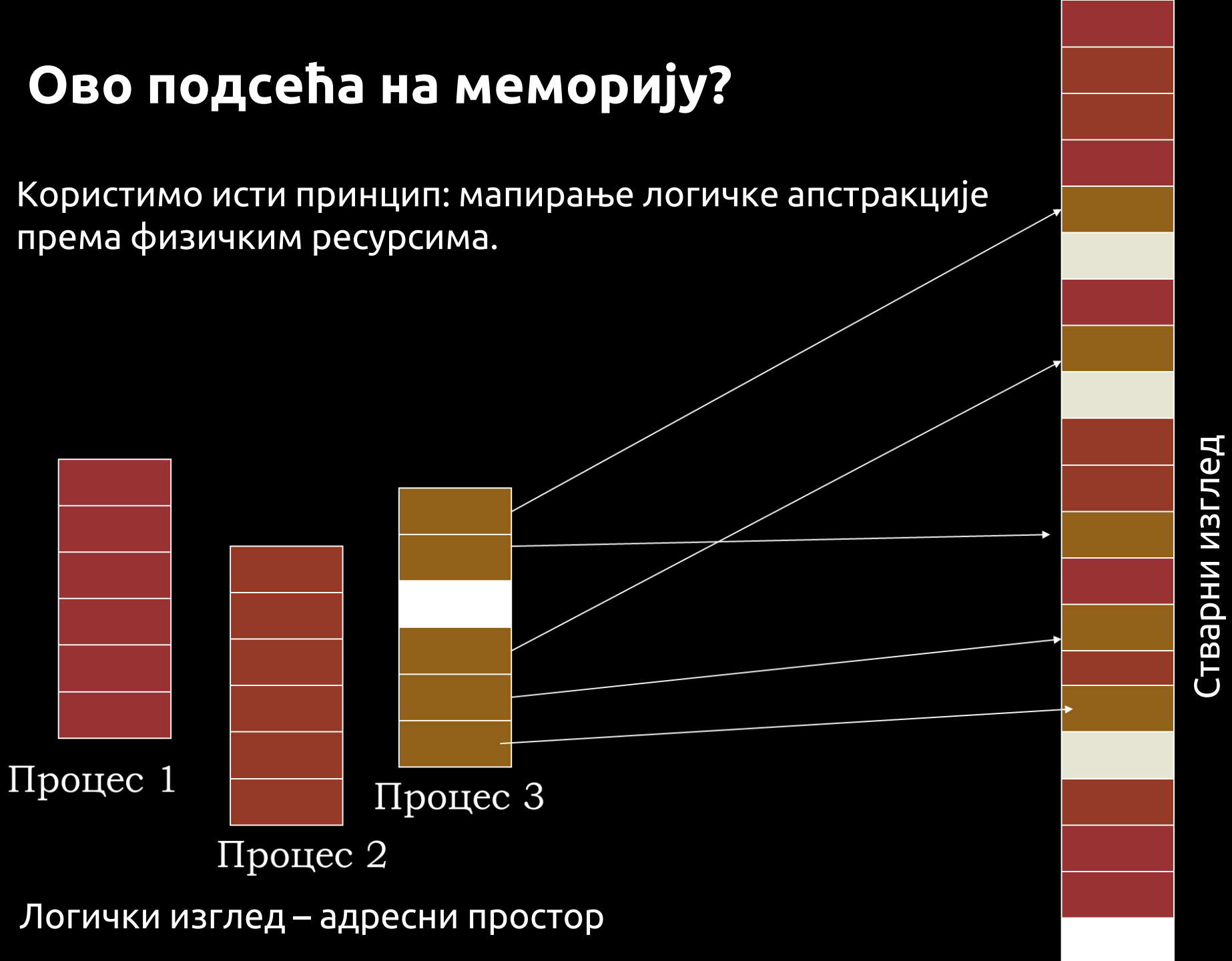
```
PS C:\Users\despot> fsutil fsinfo ntfsInfo C:
NTFS Volume Serial Number :        0xd2163c4a163c3239
NTFS Version      :                3.1
LFS Version       :                2.0
Total Sectors     :                932.994.531  (444,9 GB)
Total Clusters    :                116.624.316  (444,9 GB)
Free Clusters     :                 15.340.312  ( 58,5 GB)
Total Reserved Clusters :               5.553  ( 21,7 MB)
Reserved For Storage Reserve :              0  (  0,0 KB)
Bytes Per Sector  :                512
Bytes Per Physical Sector :        512
Bytes Per Cluster :                4096
Bytes Per FileRecord Segment    :  1024
Clusters Per FileRecord Segment :  0
```

Linux

```
d@d:~$ sudo blockdev --getbsz /dev/sda
4096
d@d:~$
```

Ово подсећа на меморију?

Користимо исти принцип: мапирање логичке апстракције према физичким ресурсима.



Логички изглед – адресни простор

Стратегије алокације

Има много различитих приступа:

- Додела континуалног простора (*Contiguous*)
- Везивање региона (*Extent-based*)
- Уланчавање блокова (*Linked*)
- Мапе датотека (*File-allocation Tables*)
- Метода индексних блокова (*Indexed*)
- Метода вишенивоских индексних блокова (*Multi-level Indexed*)

Стратегије алокације – наставак

Питања:

- Колико изабрани приступ доводи до фрагментације (и интерне и екстерне)? Тј. колико слободног простора „пропада“?
- Да ли се допушта да датотека мења величину током времена?
- Какве су перформансе код секвенцијалног приступа?
- Колико брзо се блокови података проналазе код насумичног приступа?
- Колико се простора одваја за метаподатке?

Додела континуалног простора (*Contiguous allocation*)

Алоцирање сваке датотеке се обавија тако да оне заузму континуалан простор, тј. низ узастопних блокова.

Метаподаци се чувају на почетку сваке датотеке.

ОС алоцира простор тако што проналази довољно велики комад слободног простора. (Пре тога мора и да предвиди величину датотеке; колико простора треба да заузме?)

Додела континуалног простора – наставак

Фрагметација – има много спољашње фрагментације, неопходно је периодично радити дефрагментацију.

Повећавање датотеке – није могуће без њеног премештања.

Време претраге код секвенцијалног приступа – одличне перформансе.

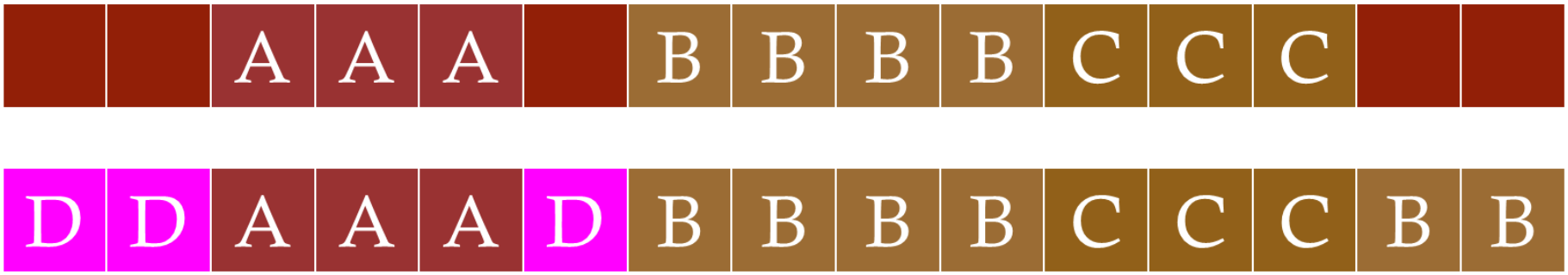
Колико се брзо рачуна насумичан приступ – једноставно рачунање.

Метаподаци – не заузимају много места.

Везивање региона (*Extent-based*)

Алоцира се (мали број) узастопних региона (тј. опсега) за сваку датотеку.

Метаподаци – мали низ (обично 2 до 6) који означавају сваки од опсега. Сваки члан низа чува почетни блок и величину.



Везивање региона (*Extent-based*) – наставак

Фрагметација – има спољашње фрагментације, мада мање од претходног приступа.

Повећавање датотеке – могуће је повећавати датотеку, све док има слободних додатних опсега.

Време претраге код секвенцијалног приступа – и даље пружа добре перформансе.

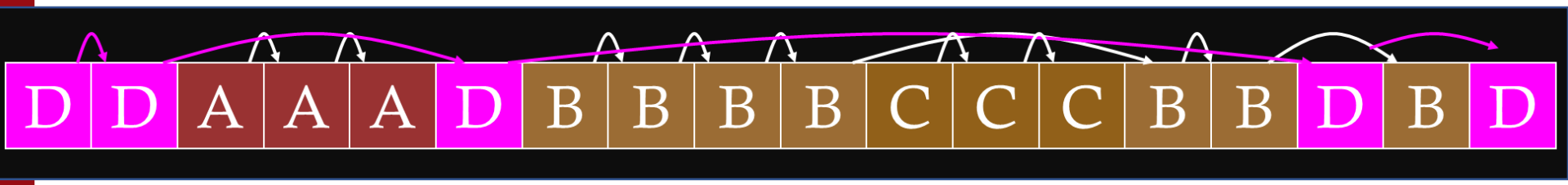
Колико се брзо рачуна насумичан приступ – и даље је једноставно рачунање.

Метаподаци – ни овде не заузимају превише места.

Уланчавање блокова (*Linked*)

Алоцира се уланчана листа група блокова фиксне величине.

Метаподаци се налазе на почетку првог блока датотеке, а такође и сваки блок садржи показивач на следећи блок.



Уланчавање блокова (*Linked*) – наставак

Фрагметација – нема спољашње фрагментације. Да ли има унутрашње?

Повећавање датотеке – лако је повећавати датотеку.

Време претраге код секвенцијалног приступа – ово зависи од тога какви су подаци.

Колико се брзо рачуна насумичан приступ – много, много споро.

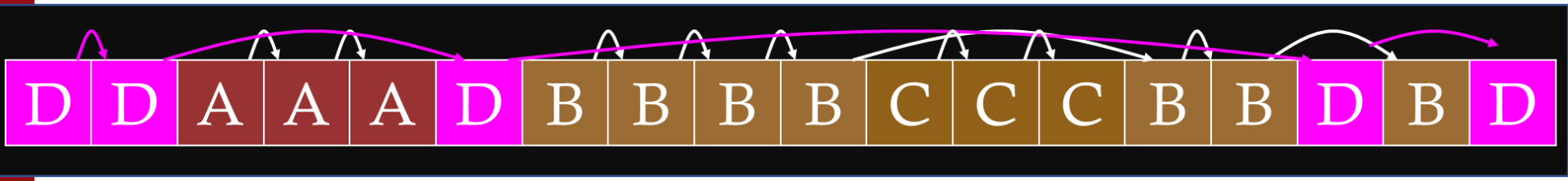
Метаподаци – у сваком блоку се троши по један показивач.

Један од начина да се неке од ових ствари поправе је пажљив избор величине блока (пазите, блок не мора да буде једнак величини сектора, може бити већи).

Мапе датотека (*File-allocation Tables*)

Ово је варијација уланчане алокације. Овде се све информације везане за уланчану структуру чувају у посебној табели на диску (**FAT**).

Метаподаци су лоцирани у првом блоку датотеке.



Нацртајте FAT табелу за дати диск. Сваки блок има један ред у табели, а у њој треба да буду смештене информације да ли је блок алоциран, као и показивач на следећи блок.

Мапе датотека (*File-allocation Tables*) – наставак

У односу на уланчани приступ, овде је мана што је неопходно читање са две локације на диску за свако појединачно читање.

Koja je mogућа optimizacija?

- **Кеширање** FAT табеле у главној меморији.
  - Предност кеширања – ово умногоме побољшава насумичан приступ.

Метода индексних блокова (*Indexed*)

За сваку датотеку се алоцирају блокови фиксне величине.

Метаподаци – ово је низ показивача на блокове.

- Простор за показиваче се алоцира у тренутку прављења датотеке.

Метода индексних блокова (*Indexed*) – наставак

Предности

- Нема спољашње фрагментације
- Датотеке могу лако да буду повећаване све до максималне величине
- Подржава насумичан приступ

Мане

- Много података се троши на метаподатке, тј. превише се простора губи на непотребне показиваче (а и већина датотека је мале величине, па је ово неефикасно).

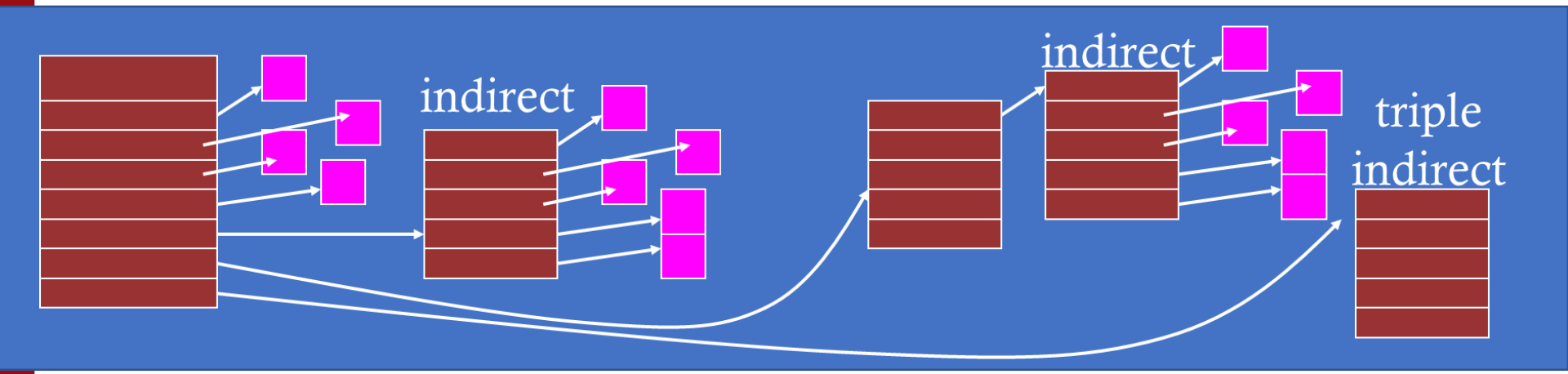
Метода вишенивоских индексних блокова (*Multi-level Indexed*)

Ово је варијација методе индексних блокова.

Динамички се алоцира хијерархија показивача на блокове (у складу са потребом).

Метаподаци – мали број показивача који се статички алоцирају, као и додатни показивачи на блокове показивача (алоцирани по потреби).

Пример – Линуксови системи датотека ext2, ext3



Метода вишенивоских индексних блокова (*Multi-level Indexed*) – наставак

Поређење са методом индексних блокова:

Предности

- Не троши се простор на непотребне показиваче
- И даље је могућ брз приступ малим датотекама.
- Датотеке могу да расту до огромне величине.

Мане

- Да би се израчунала адреса, мора се читати индиректни блок показивача. Дакле, имамо додатно читање.
  - Решење: индиректне блокове треба чувати **кеширане** у главној меморији.

Везивање великог броја региона (*Extent-based*)

Код модерних система датотека је могуће алоцирати вишеструке узастопне регионе за сваку датотеку.

Ови опсези су организовани у вишенивоске структуре у облику стабла.

- Сваки лист стабла чува адресу почетка и величину региона.
- У случајевима када нема много региона, овакав приступ минимизује сувишне трошкове за метаподатке.
- Допушта да датотеке расту без ограничења фиксним бројем региона.

Везивање великог броја региона (*Extent-based*) – наставак

Фрагметација – и интерна и екстерна су прихватљиво мале.

Повећавање датотеке – лако је повећавати датотеку.

Време претраге код секвенцијалног приступа – и даље даје добре перформансе.

Колико се брзо рачуна насумичан приступ – ово зависи од величине датотеке.

Метаподаци – нема много трошкова.

НАПОМЕНА

У примеру који следи, ми ћемо радити са системом датотека код којег се користи метода вишенивоских индексних блокова.

Ово је једноставан приступ.

Могуће је направити комплексније системе датотека на основу оваквих структура.

Карактеристике диска који је узет за пример

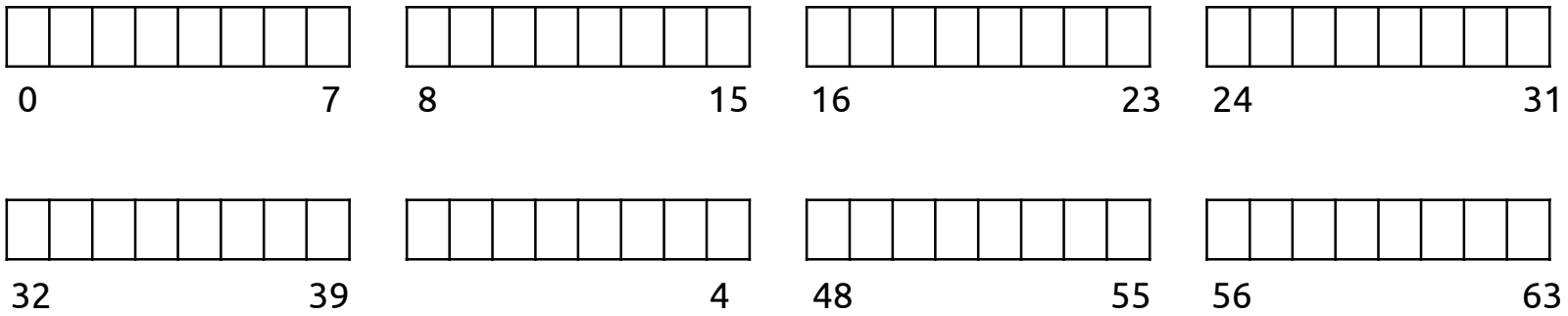
Структуре које се налазе на нашем примеру диска:

- блокови података – *data block*
- табела индексних чворова – *inode table*
- индиректни блокови – *indirect block*
- директоријуми
- мапа битова за податке – *data bitmap*
- мапа битова за индексне чворове – *inode bitmap*
- *superblock*

Општа организација

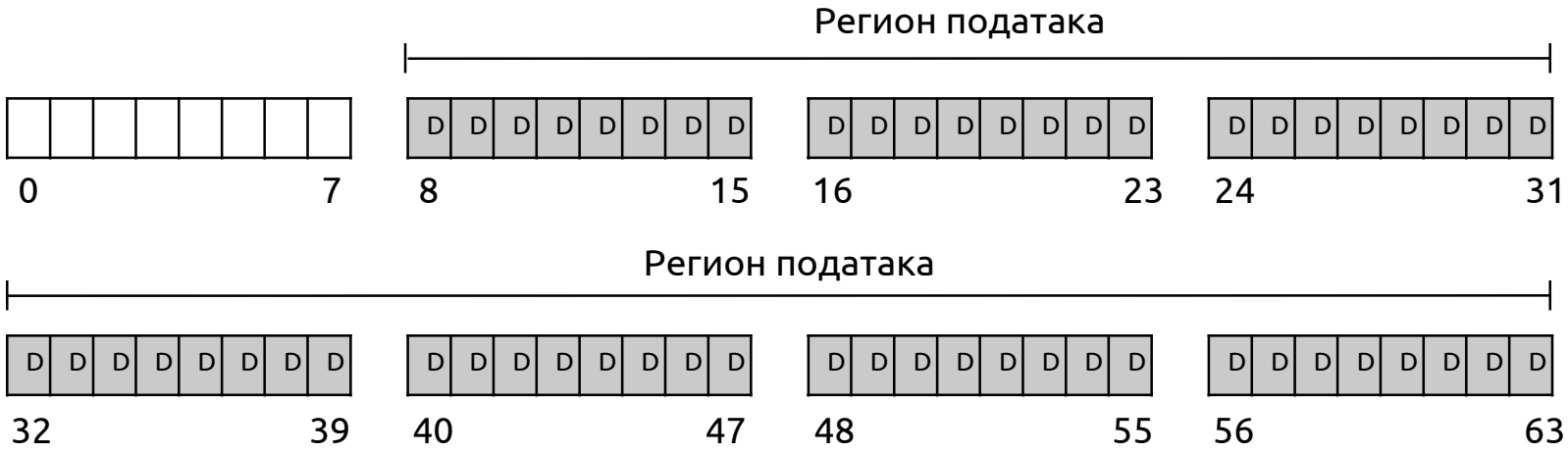
Враћамо се примеру с почетка. Диск је подељен на низ од 64 **блокова**.

- Узећемо да је величина блока четири килобајта.



Регион података у систему датотека

Резервише се **регион података** у коме ће се чувају кориснички подаци.



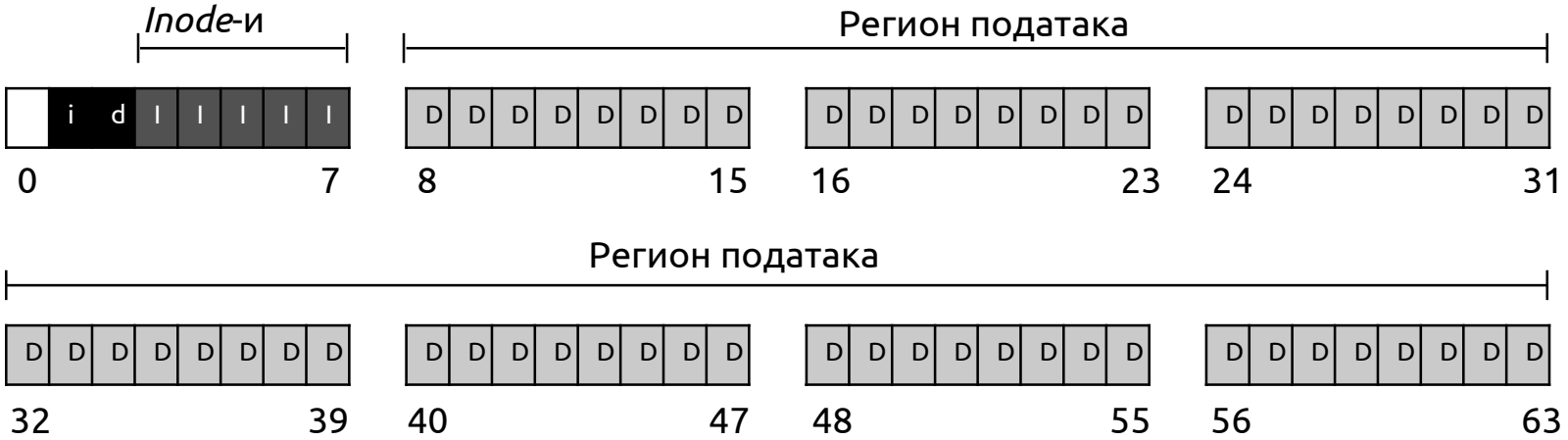
Систем датотека мора да прати који блокови података чине неку датотеку, величину те датотеке, ко је њен власник, итд.

Како сачувати индексни чвор (**inode**) у систему датотека?

Inode табела у систему датотека

Резервише се део простора за чување табеле индексних чворова (**inode table**). Она садржи низ *inode*-а.

- Нпр. *inode* табела је смештена у блоковима 3 – 7, а величина *inode*-а је 256 бајтова.
  - У блок од 4KB може да се смести 16 индексних чворова, резервисали смо пет блокова за њих. Дакле – систем датотека садржи 80 индексних чворова. (Ово је максимални број датотека на нашем диску.)

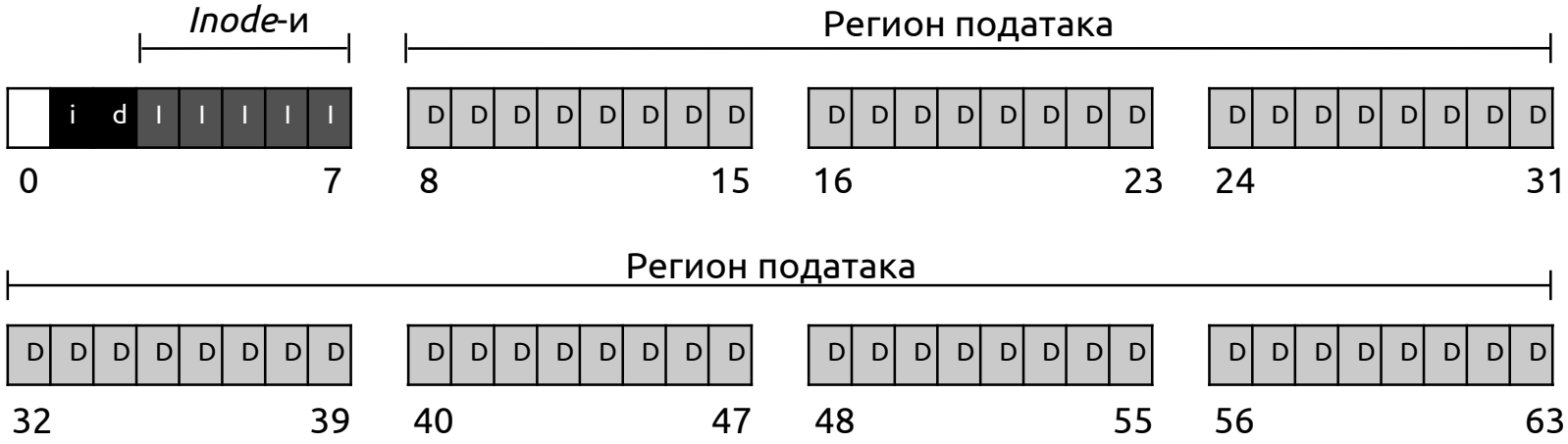


Алокационе структуре

Оне служе да се прати да ли су индексни чворови или блокови података слободни или алоцирани.

Може да се користи ***bitmap*** (мапа битова), овде вредност сваког бита указује да ли је тај чвор односно блок слободан (нула – 0) или заузет (јединица – 1).

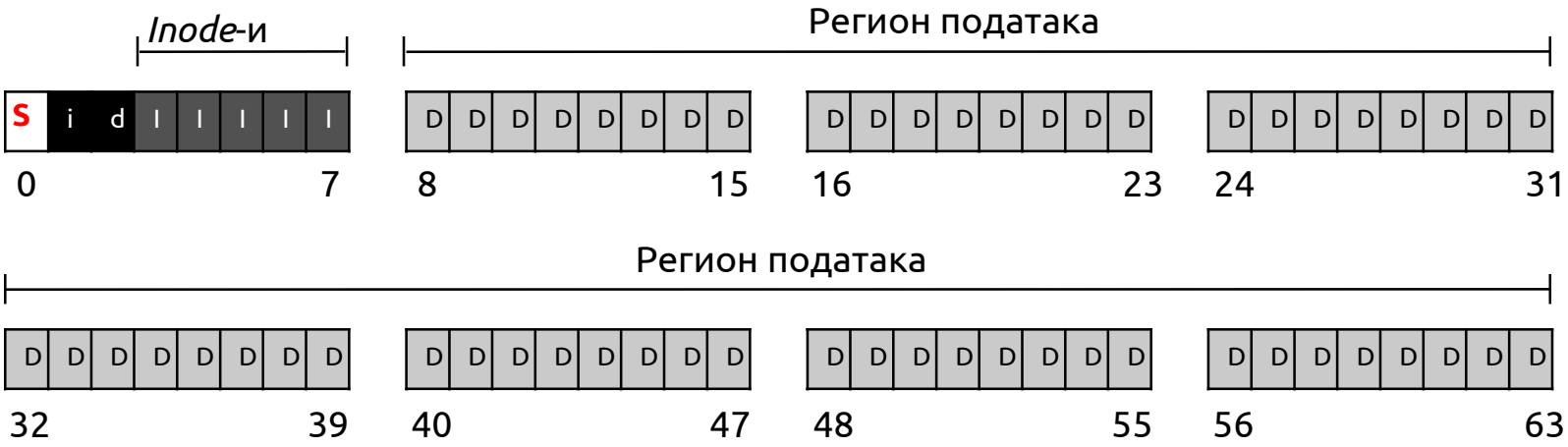
- ***Data bitmap*** (мапа битова података): за регионе података,
- ***Inode bitmap*** (мапа битова индексних чворова): за inode табелу.



Супер блок

Супер блок садржи **опште информације о одређеном систему датотека**.

- Нпр. у њему се чува број индексних чворова, почетна локација у inode табели, итд.

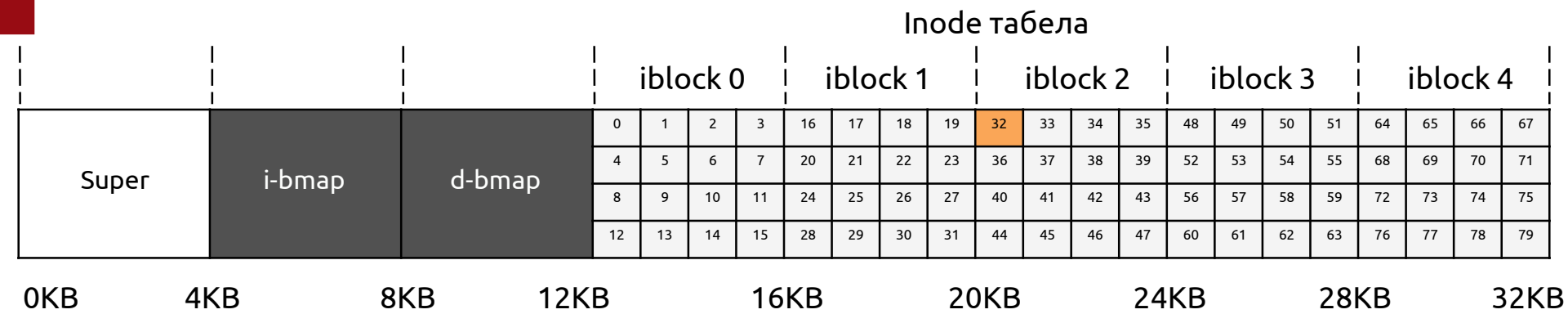


Дакле, када се прикачи систем датотека, ОС прво прочита супер блок и иницијализује разне информације о том систему датотека.

Организација датотеке: *inode*

Сваки индексни чвор се референцира преко **inode** броја (*index node*).

- Помоћу *inode* броја, систем датотека може да израчуна где се индексни чвор налази на диску.
- Нпр. *inode* број 32
  - Израчуна се офсет у оквиру *inode* региона (32 x sizeof(inode) = 32 x 256 бајтова = 8192 бајтова).
  - Додавање почетне адресе у *inode* табелу (12KB) + *inode* регион (8KB) = 20KB



Организација датотеке: inode (наставак)

Диск се не адресира на нивоу бајта, већ преко сектора хард-диска.

$$\text{блок} = \frac{(inode\text{Број} * sizeof(inode))}{\text{величина блока}}$$

$$\text{сектор} = \frac{((\text{блок} * \text{величинаБлока}) + inode\text{ПочетнаАдреса})}{\text{величина сектора}}$$

inode

Size	Name	What is this inode field for?
2	mode	can this file be read/written/executed?
2	uid	who owns this file?
4	size	how many bytes are in this file?
4	time	what time was this file last accessed?
4	ctime	what time was this file created?
4	mtime	what time was this file last modified?
4	dtime	what time was this inode deleted?
2	gid	which group does this file belong to?
2	links_count	how many hard links are there to this file?
4	blocks	how many blocks have been allocated to this file?
4	flags	how should ext2 use this inode?
4	osd1	an OS-dependent field
60	block	a set of disk pointers (15 total)
4	generation	file version (used by NFS)
4	file_acl	a new permissions model beyond mode bits
4	dir_acl	called access control lists

Поједностављен приказ *inode*-а система датотека Ext2

Inode

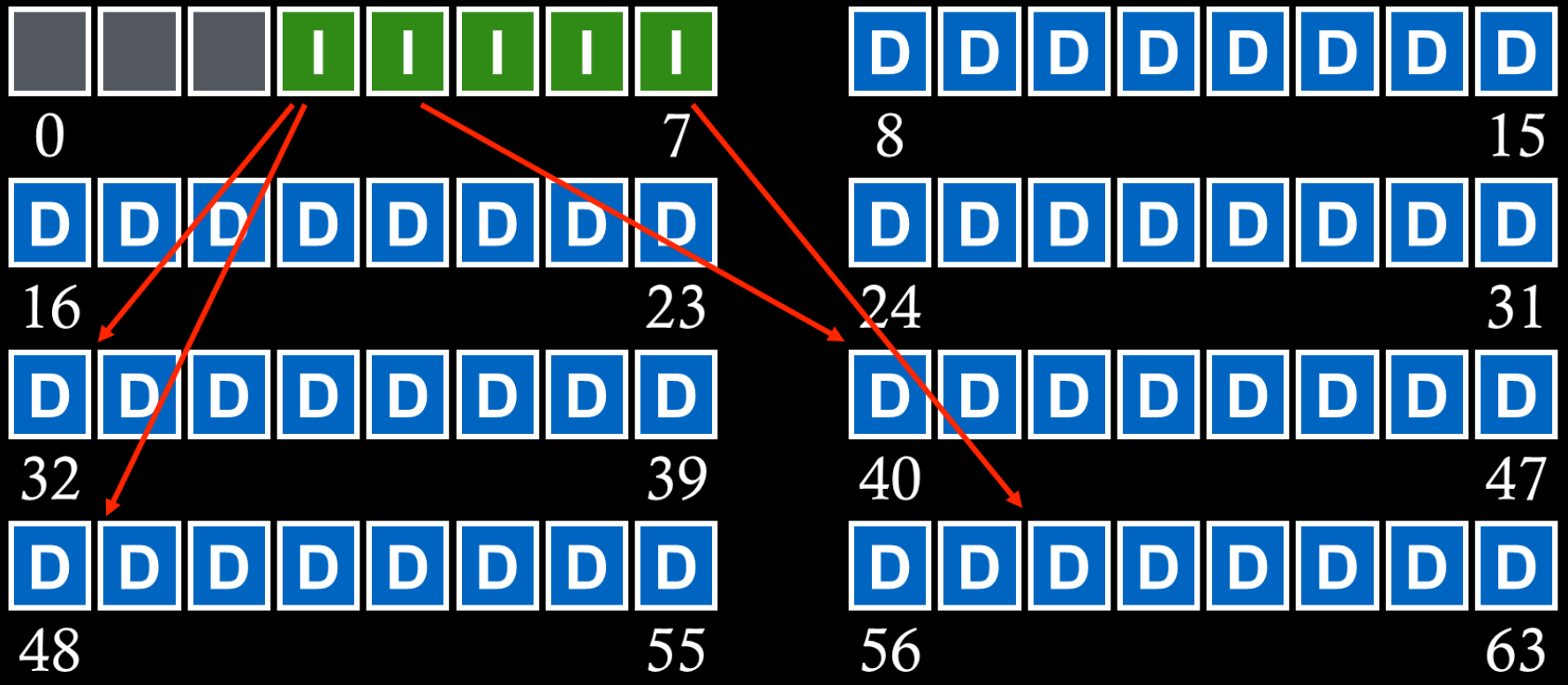
Директни показивачи

Индиректни показивачи (у блоку)

- Нпр. блокови од 4KB и величина адресе од 4B, ово даје могућност за смештање 1024 показивача, рецимо да имамо и 12 директних показивача; дакле – та датотека може бити максималне величине (12+1024)*4K тј. 4144KB
- За датотеке веће од 4144KB се морају користити три нивоа показивача.

Други приступ је коришћење **extents** показивача (нпр. ext4 систем датотека, повезане листе (нпр. FAT систем датотека), или било која друго структура података).

Индексни чворови



Индексни чвор

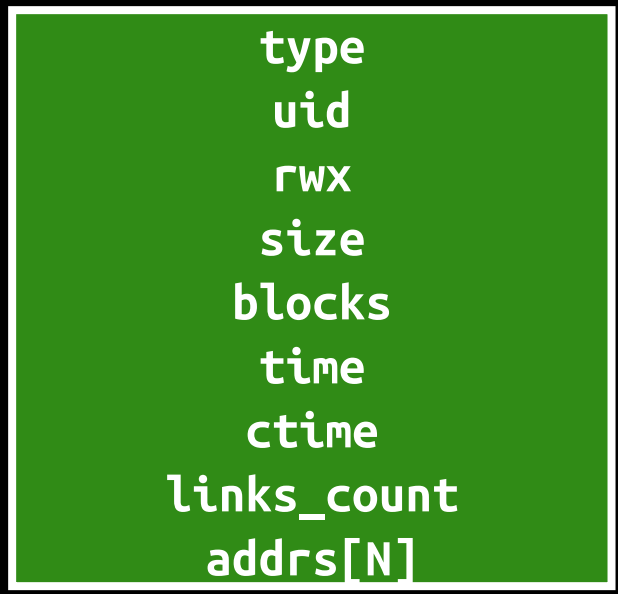
Претпоставимо да је индексирање на једном нивоу (тј. да имамо само показиваче на блокове података).

Која је највећа могућа величина датотеке?

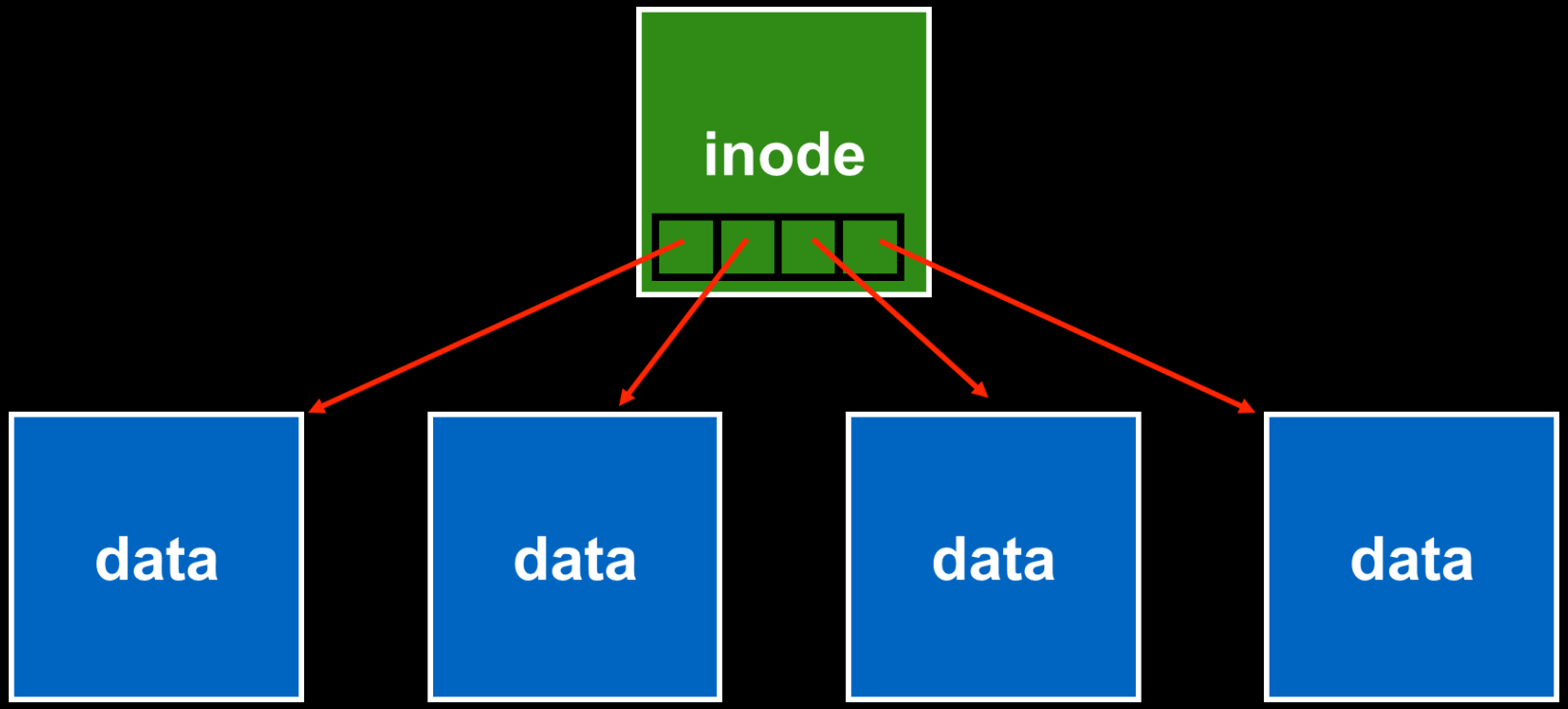
Уколико су *inode*-и величине 256 бајтова (и ако претпоставимо да сви могу бити употребљени као показивачи), и уколико имамо 4-бајтно адресирање:

256 / 4 = 64

64 * 4K = 256KB

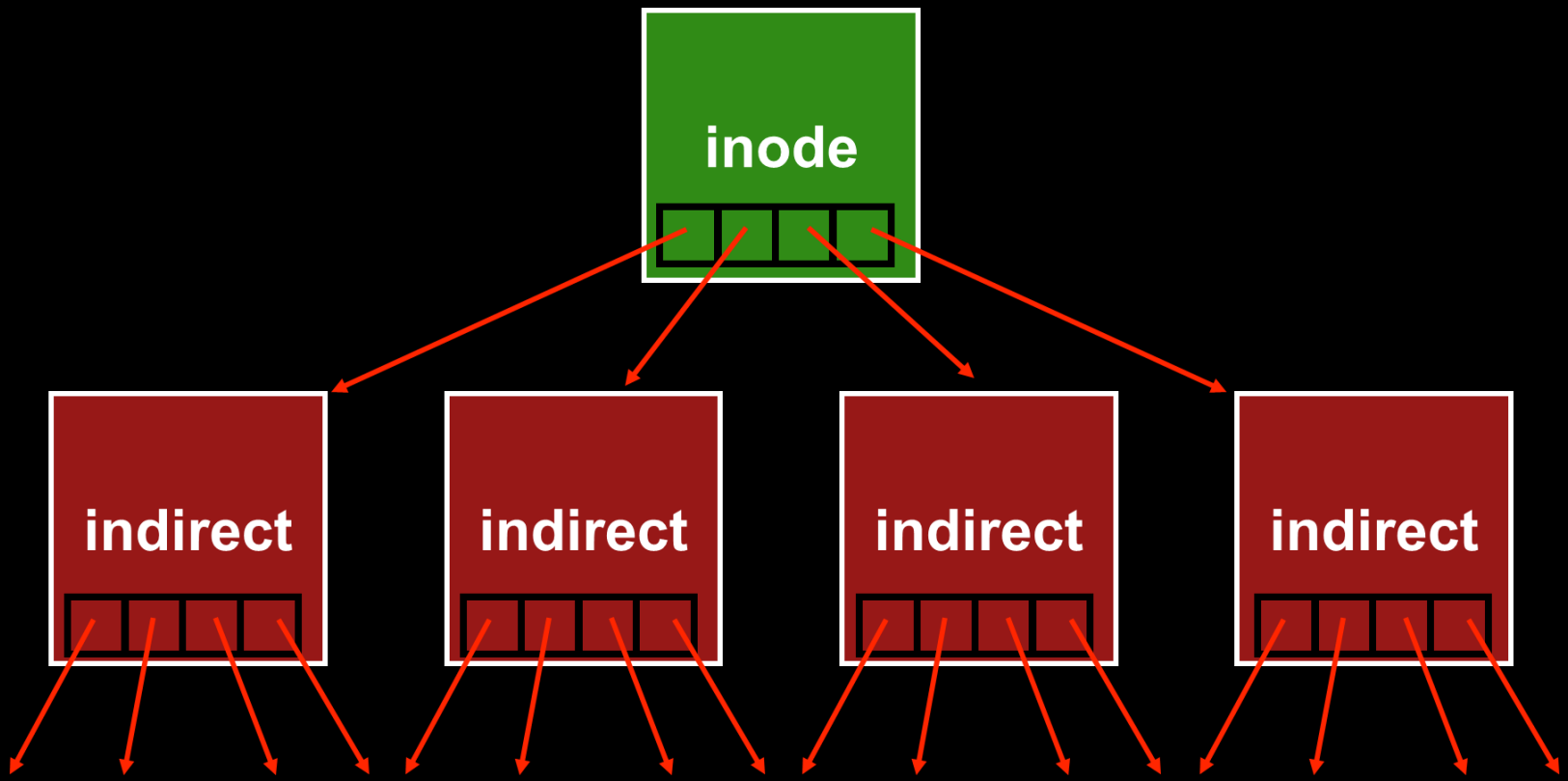


Једнонивоски индексни блокови



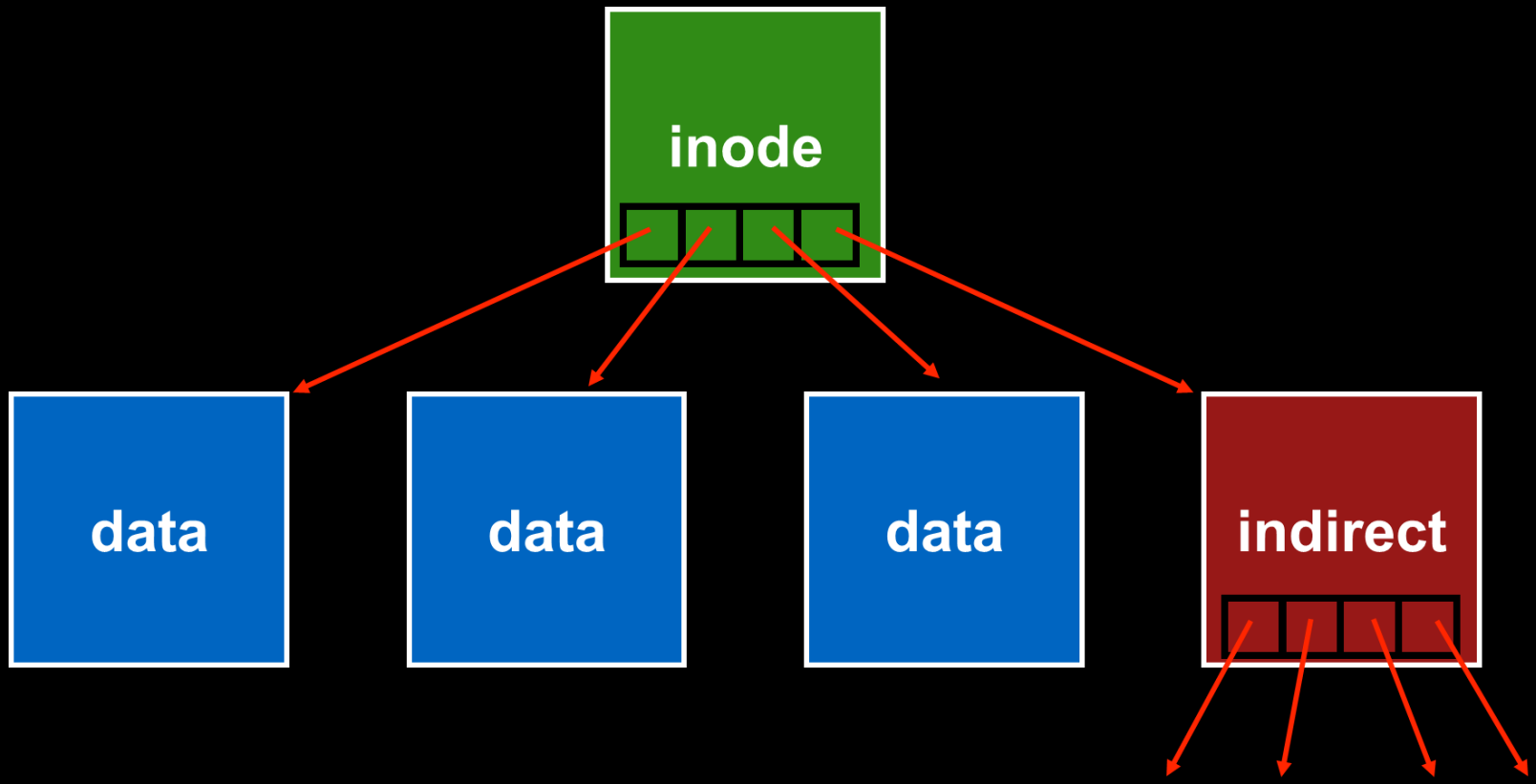
Вишенивоски индексни блокови

Индиректни блокови се чувају у обичним блоковима података.



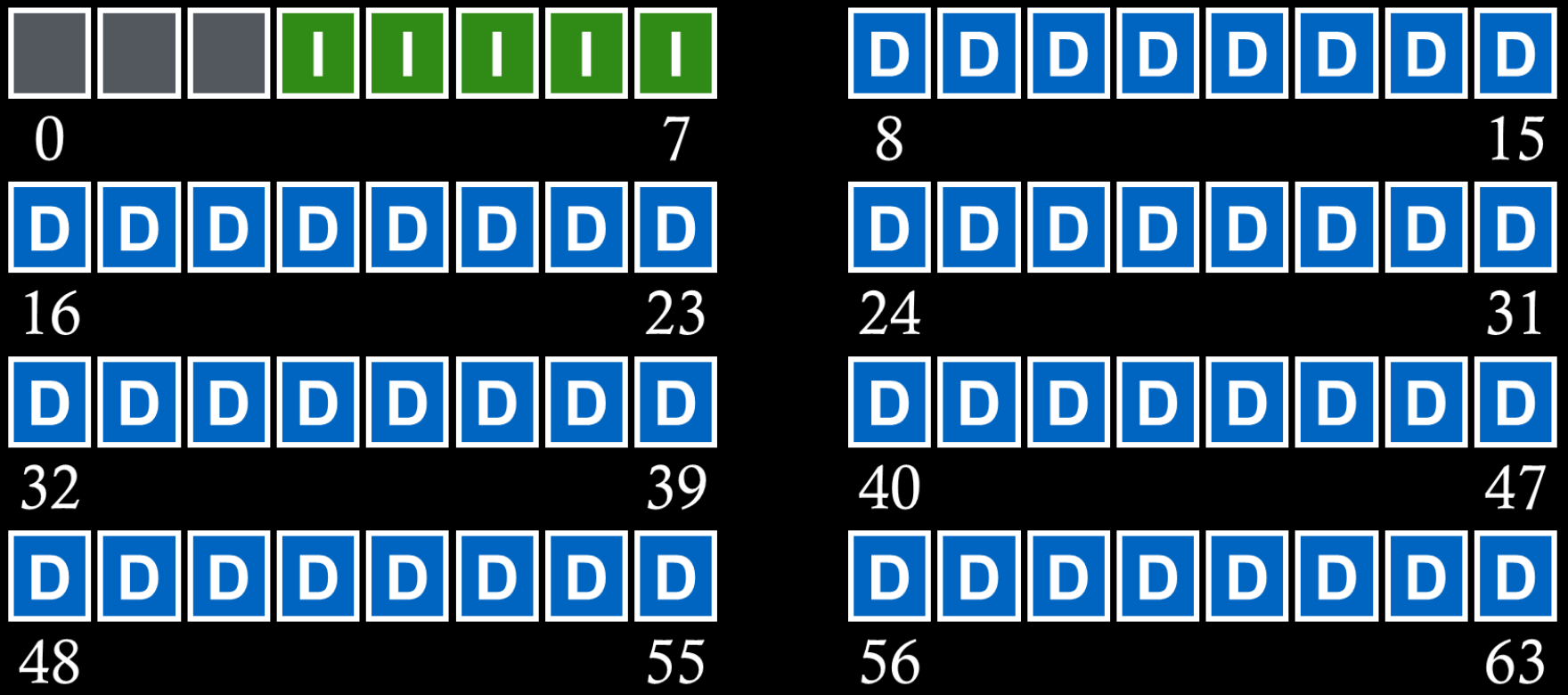
Шта уколико желимо да овај приступ оптимизујемо за мале датотеке?

Вишенивоски индексни блокови



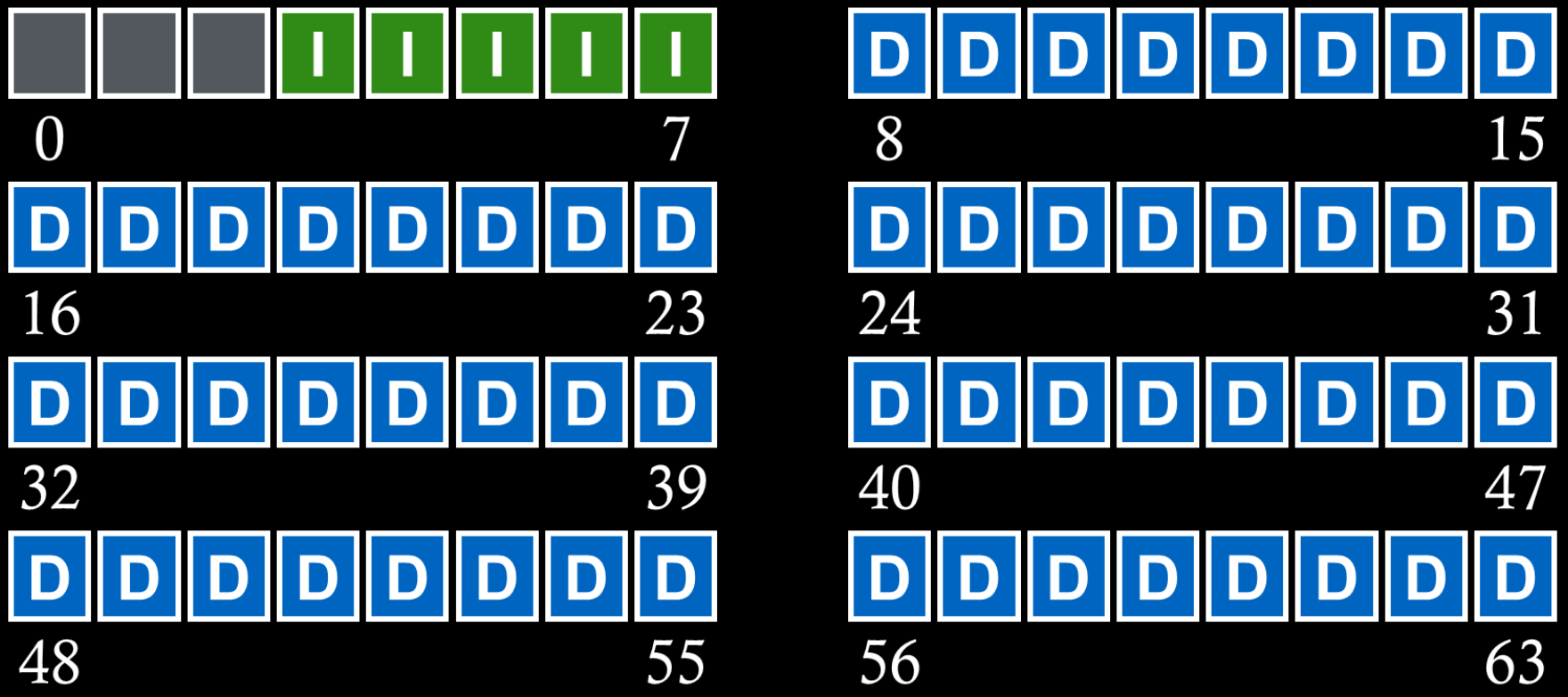
Овај приступ је бољи за смештање малих датотека!

Пример



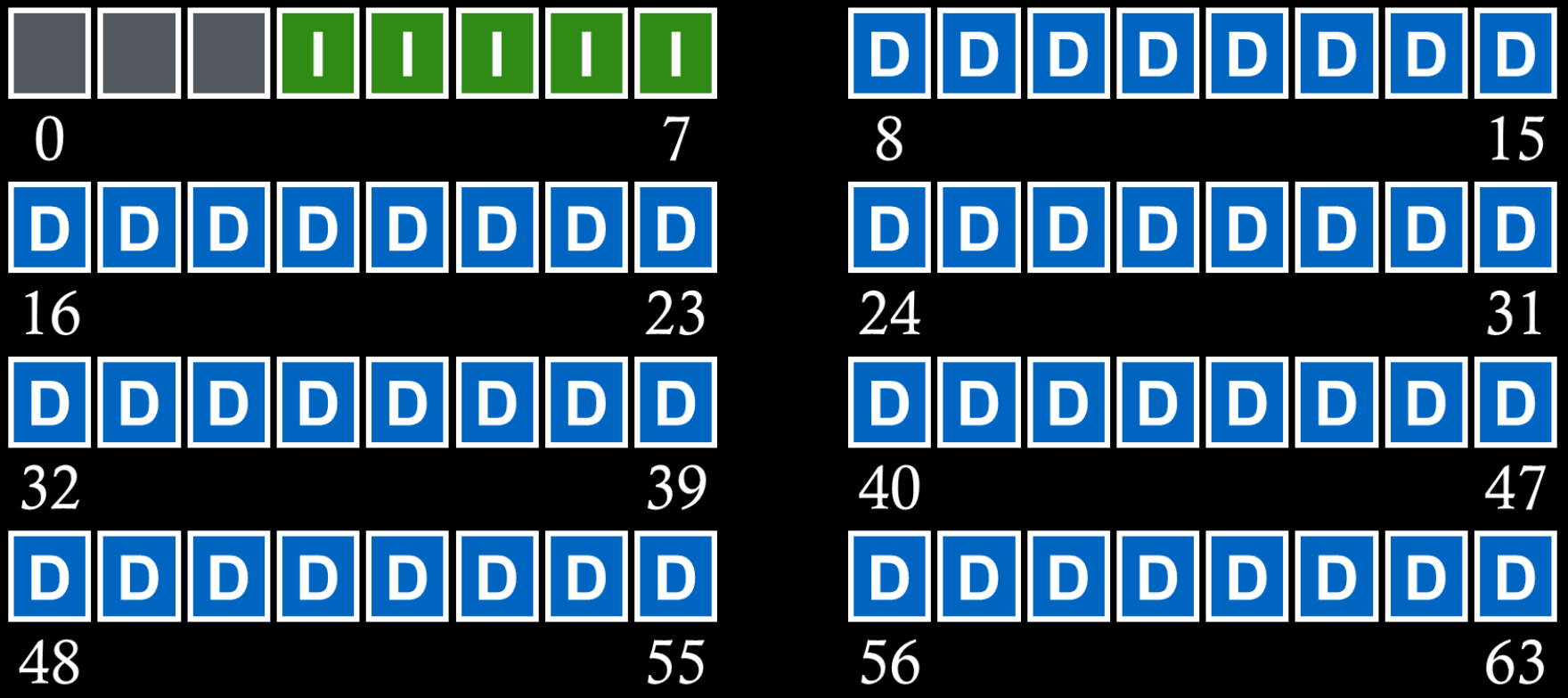
Нека је величина индексног чвора 256 бајтова (дакле – имамо их по 16 у сваком блоку). Колики је офсет за индексни чвор под бројем **0**?

Пример



Нека је величина индексног чвора 256 бајтова (16 у сваком блоку). Колики је офсет за индексни чвор под бројем **4**?

Пример



Нека је величина индексног чвора 256 бајтова (16 у сваком блоку). Колики је офсет за индексни чвор под бројем **40**?

Неке карактеристике система датотека

Неки закључци који су изведени на основу проучавања система датотека и хард-дискова, а које треба имати у виду током пројектовања система датотека		
Већина датотека је мала	→	Грубо посматрано, уобичајена величина датотека је 2KB
Просечна величина датотека има тенденцију раста	→	Тренутни просек је око 200KB
Највећи део бајтова диска отпада на велике датотеке	→	Највећи део простора диска заузима мали број великих датотека
Системи датотека садрже велики број датотека	→	Просек је скоро сто хиљада датотека
Системи датотека су обично допола пуни	→	Иако се просечна величина дискова повећава, они су и даље просечно 50% попуњени
Директоријуми обично имају мало датотека	→	Многи од њих немају много датотека, обично 20 или мање

Директоријуми

Има различитих приступа на различитим системима датотека.

Најчешћи приступ је:

- Директоријуми се чувају у оквиру блокова података;
- Велики директоријуми користе вишеструке блокове података;
- Битови у оквиру индексних чворова се користе да би се назначила разлика између директоријума и датотека.

Могу се користити различите структуре за организовање датотека

- Листе
- Б-стабла…

Организација директоријума

inum	reclen	strlen	name
5	12	2	.
2	12	3	..
12	12	4	foo
13	12	4	bar
24	36	28	foobar_is_a_pretty_longname

Сваки директоријум има и **.** и **..** показиваче (на себе и директоријум изнад).

Брисање датотеке се обавља као позив ***unlink*()**

Најчешће се директоријум посматра као посебна врста датотеке

Могуће је и другачије организовати садржај директоријума, нпр. Б-стабла

Управљање слободним простором

Како можемо да пронађемо слободне блокове података или слободне индексне чворове?

- *Bitmap*
- *Free list* (листа слободних локација)
- *B-tree* (бинарно стабло)

Приступање подацима – читање

`open("/foo/bar", ORDONLY)`

Табела читања датотеке. Време расте одозго надоле.

	data bitmap	inode bitmap	root inode	foo inode	bar inode	root data	foo data	bar data[0]	bar data[1]	bar data[2]
open(bar)			read			read				
				read			read			
					read					
read()					read					
					write			read		
read()					read					
					write				read	
read()					read					
					write					read

Приступање подацима – уписивање

Табела уписивања датотеке. Време расте одозго надоле.

	data bitmap	inode bitmap	root inode	foo inode	bar inode	root data	foo data	bar data[0]	bar data[1]	bar data[2]
create (/foo/bar)		read write	read	read	read write	read	read	write		
write()	read write				read write			write		
write()	read write				read write				write	
write()	read write				read write					write

Ефикасност - кеширање и баферовање

Како избећи превелики број I/O операција за основне операције над датотекама?

- Код **читања** – користити **кеширање**
- Код **уписивања** – користити **баферовање**

За ово нам је потребна DRAM меморија.

Одлагање уписивања нам помаже тако што се избегава (непотребно) уписивање података који ће убрзо бити поново измењени, или који ће убрзо бити обрисани, а и могуће је боље распоредити уписивања.

Потребно је наћи меру колико дуго треба баферовати податке пре уписивања, затим колико података баферовати…

- Обично се чека између 5 и 30 секунди.

Сажетак

Видели смо једноставан пример система датотека, његове основне структуре, као и основне операције.

У наставку ћемо видети како да **ефикасно** алоцирамо меморију тако да добијамо што боље перформансе хард-диска, као и како да управљамо отказима хард-дискова.

Локалност и FFS систем датотека

Системи датотека

Локални системи датотека које ћемо обрадити

- **FFS**: *Fast File System*
- **LFS**: *Log-Structured File System* (системи датотека засновани на логовима) *– радимо на следећем блоку*

UNIX оперативни систем

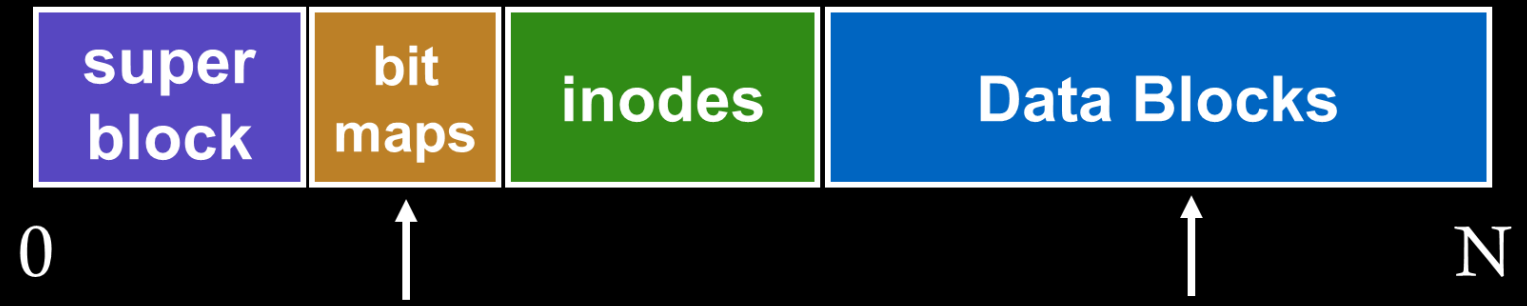
Структуре података

Једноставан је и подржава основне апстракције.

Лако је користити систем датотека.

Али су перформансе лоше!

Основна структура



- Једноставан је и подржава основне апстракције.
- Лако је користити систем датотека.
- Али су перформансе лоше!

Проблеми код UNIX оперативних система

Лоше перформансе. Unix систем датотека је третирао диск као RAM меморију (тј. као меморију са случајним приступом).

Проблем: **Фрагментација**. Пример блокова са случајним приступом и четири датотеке.

- Блоковима података за сваку од датотека се може приступити проласком напред и назад кроз диск јер су они смештени узастопно.

A1	A2	B1	B2	C1	C2	D1	D2

- Датотеке b и d су потом обрисане:

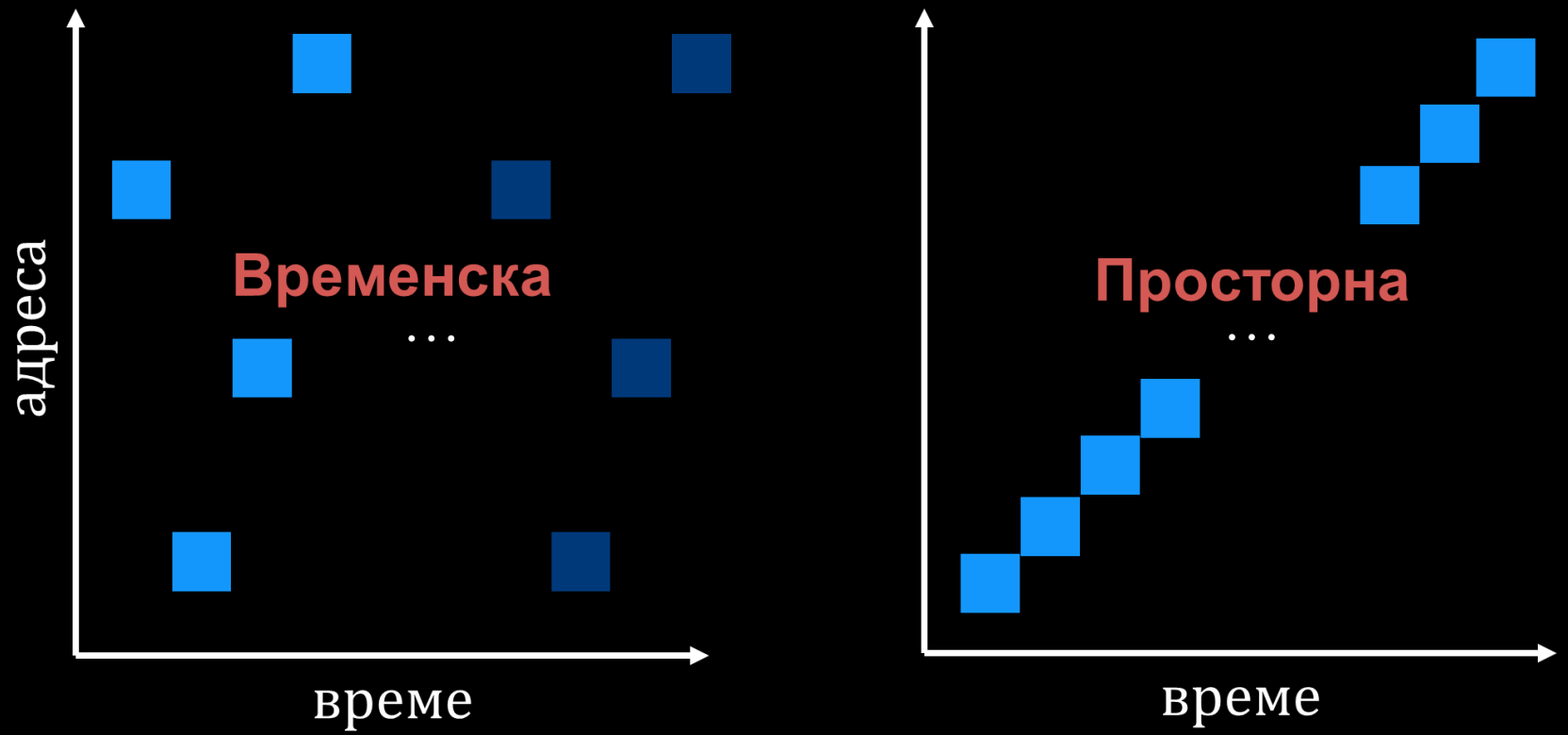
A1	A2			C1	C2		

- Креирана је датотека E и смештена је у слободне блокове (који нису узастопни):

A1	A2	E1	E2	C1	C2	E3	E4

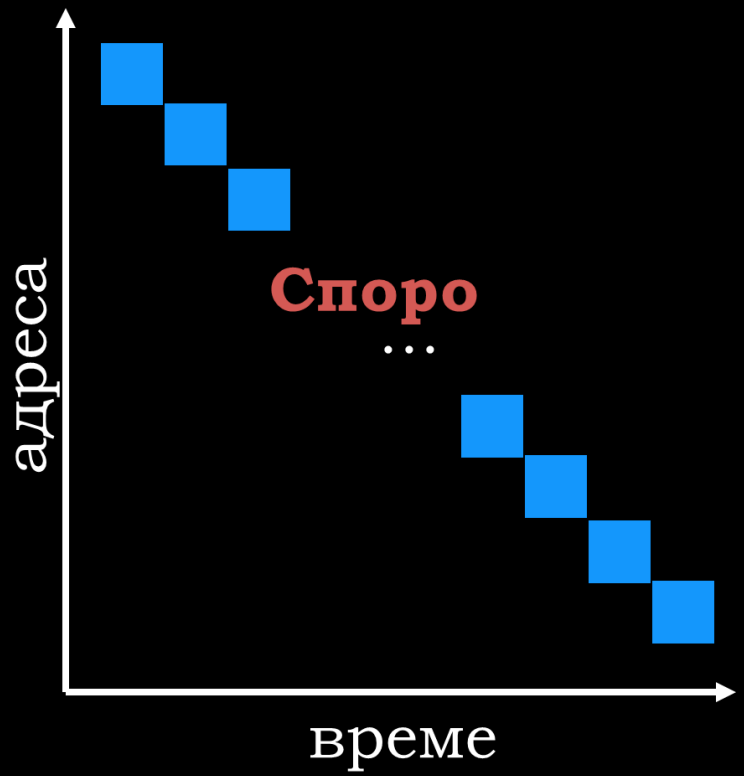
Проблем: Премала величина блока.

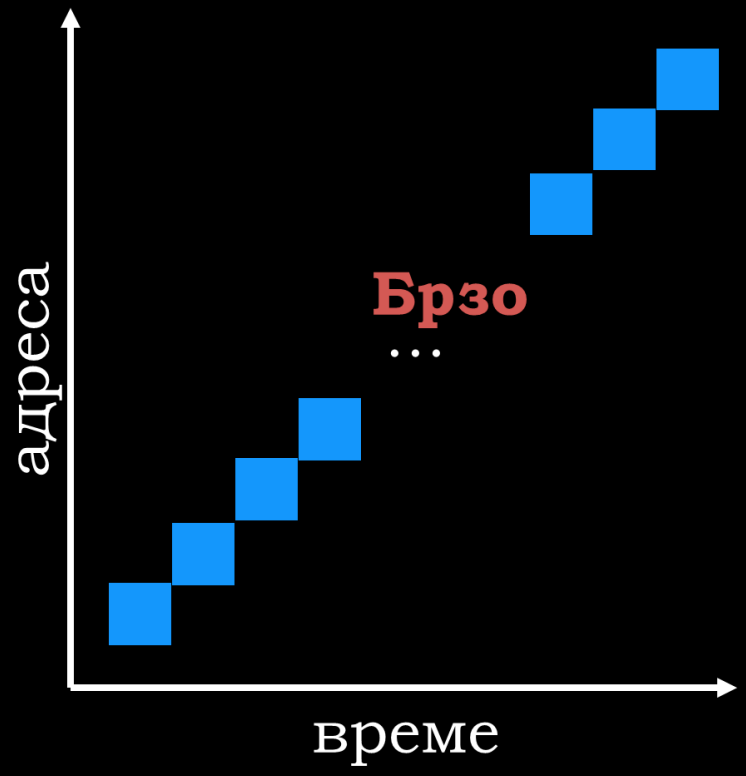
Присећање: типови локалности



- Који тип локалности је најкориснији код хард-диска?

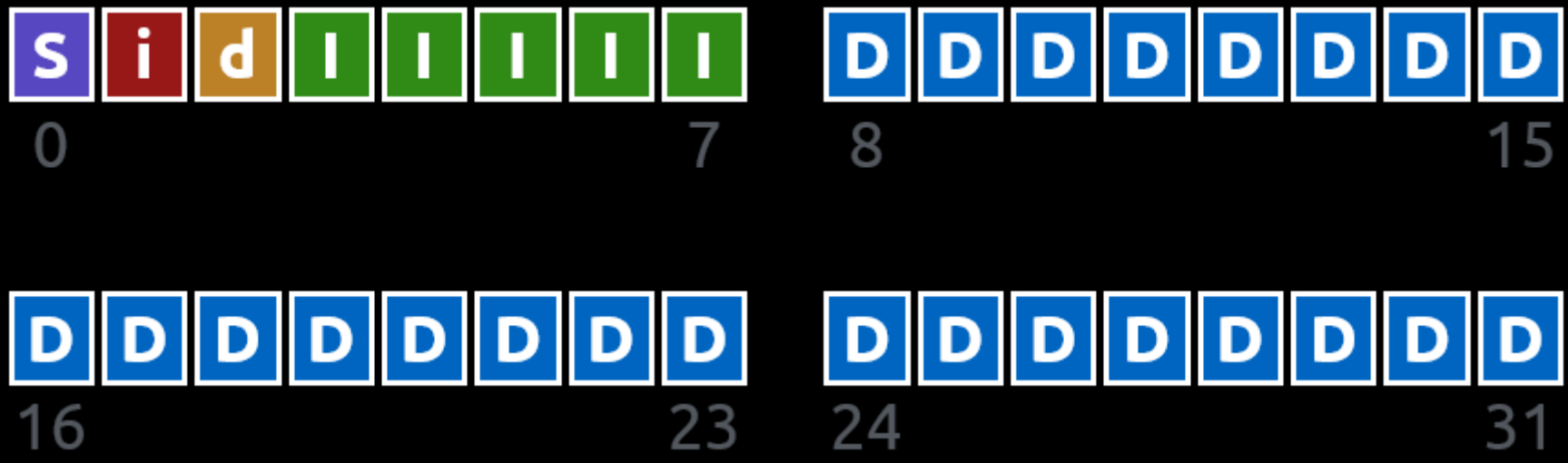
Важан је поредак





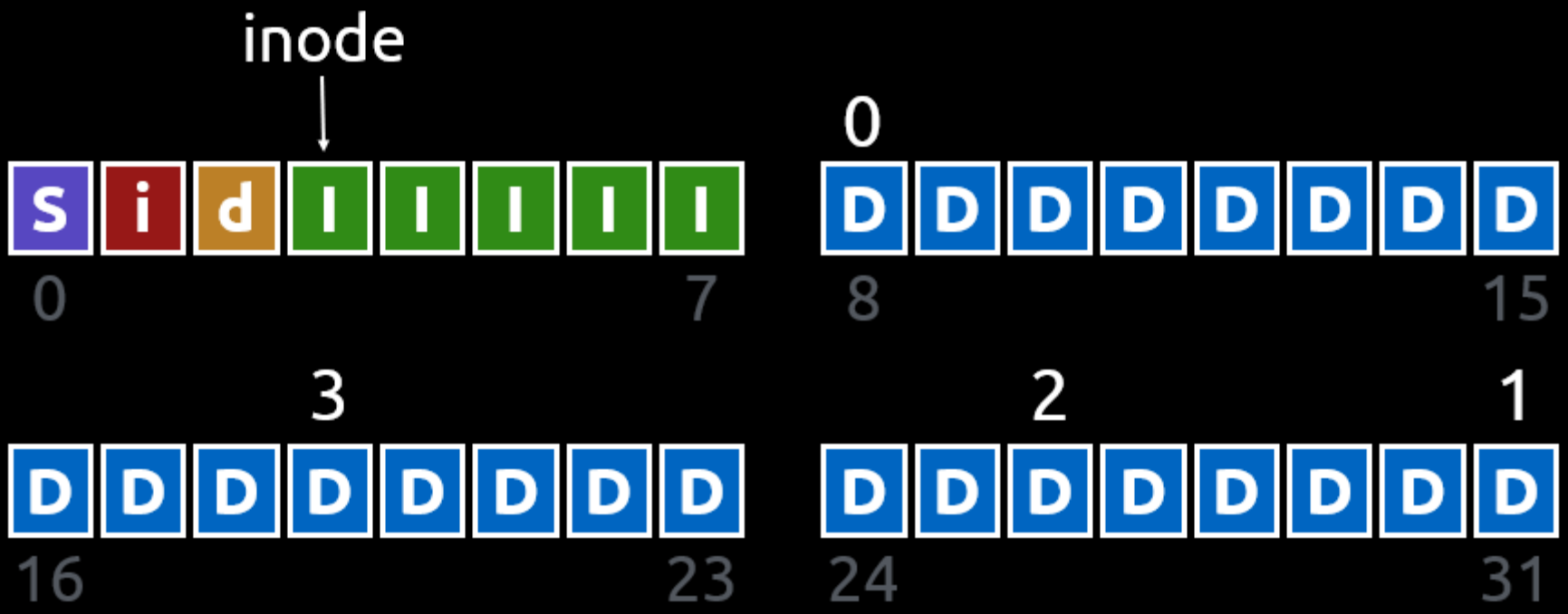
- Ово се може искористити код распоређивача операција над хард-диском

Како одабрати индексни чвор и блок података



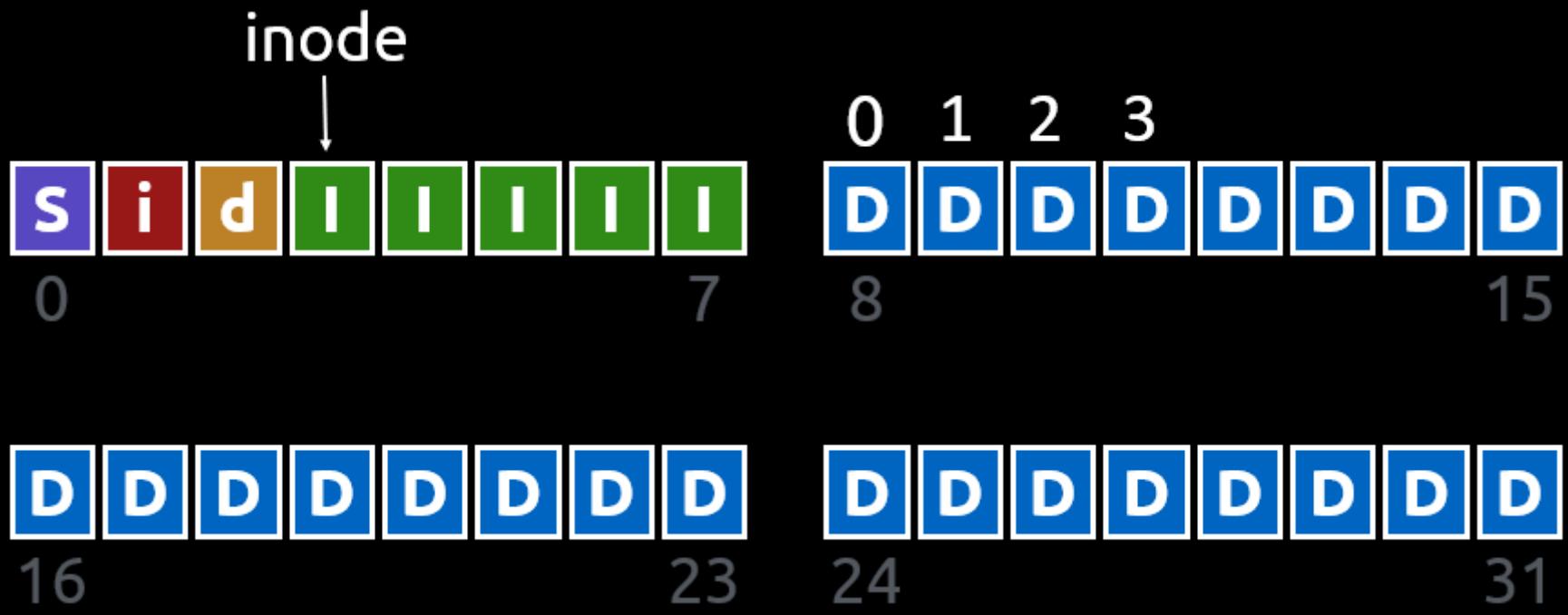
- Претпоставимо да су сви слободни, које треба да одаберемо?

Лош распоред блокова датотеке



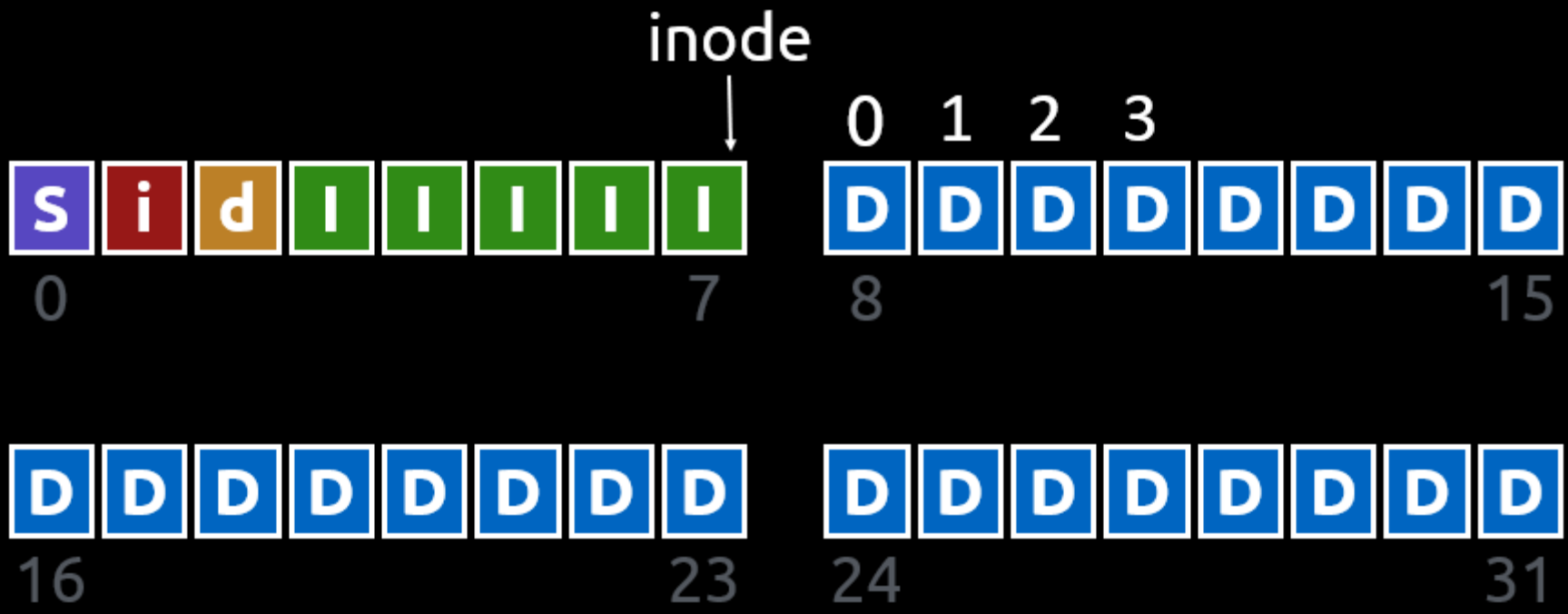
- Овај распоред нема много смисла.

Бољи распоред блокова датотеке



- Ово је много боље, али да ли може ово да се додатно поправи???

Бољи распоред блокова датотеке



- Ово би било идеално, али јасно је да се ово не може радити са свим датотекама

FFS: „свест“ о диску је решење

FFS (*Fast File System*) је 1980-их дизајниран на универзитету Беркли.

Дизајниран је тако да структуре система датотека и правила алокације имају „**свест**“ о диску, и да се тако побољшају перформансе.

- Одржан је исти API за рад са системом датотека (`open()`, `read()`, `write()`, итд).
- Али промењена је унутрашња имплементација тих системских позива.

Зашто је потребна та **свест** о диску? Следи објашњење…

Стари системи датотека

- Ово је изворни UNIX систем датотека.
- Блокови података од 512 бајтова, листа слободних локација се чува индексним чворовима и блоковима података.
- Када се упореди остварена пропусност за секвенцијална читања/уписивања са максималним могућим брзинама које су декларисане за тај хард-диск, види се да је у неким случајевима дошло до тога да је остварено само **2%** од могућег.

Старење система датотека

Перформансе система датотека опадају са временом

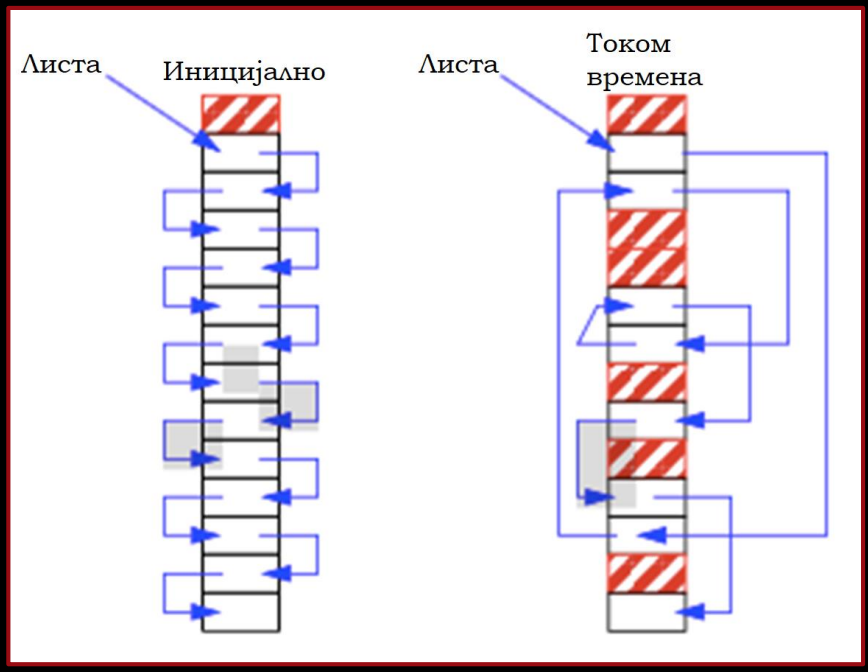
- На новом систему датотека се остварује око 17,5% максималне пропусности
- Након неколико недеља коришћења диска то опадне на 3%

Проблем је што систем датотека са временом постаје све више фрагментисан

- Све теже је пронаћи довољно велики комад меморије у листи

Нека решења?

- Повремено радити дефрагментацију,
- Сортирање листе слободних локација.



Величина блока у систему датотека

Како величина блока утиче на перформансе?

- Шта ако је удвостручимо?
- Резултат: перформансе се и више него удвостручују!

Зашто ово повећава перформансе?

- Логички суседни блокови обично нису физички суседни, а овако имамо двоструко мање претрага и ротација диска.

Зашто се перформансе више него удвоструче?

- Мањи блокови изискују коришћење више индиректних блокова.

Закључак о старим системима датотека

Листа слободних локација постаје хаотична -> све више се приступа насумичним локацијама

Блокови су мали (512 бајтова)

Блокови су лоше распоређени

- Индексни чворови и подаци су далеко једни од других
- Повезани индексни чворови нису близу једни другима
  - За које то индексне чворове кажемо да су повезани? Они који су у истом директоријуму!

Остварује се само **2%** могућих перформанси

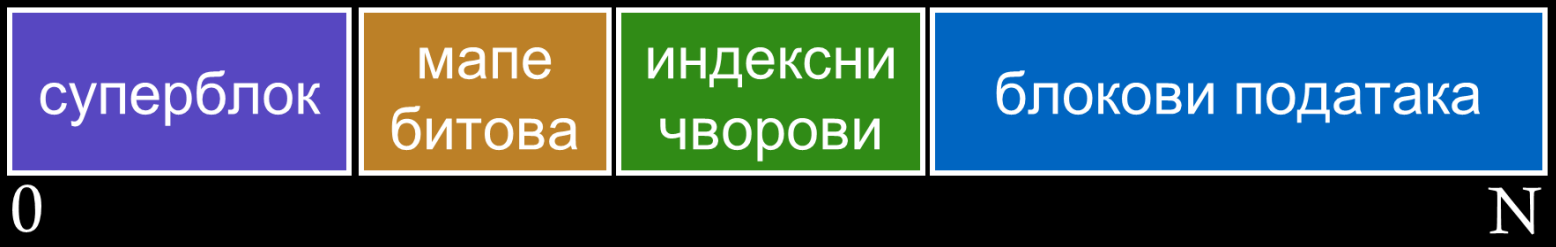
Проблем: стари системи датотека су се понашали према диску као да је то RAM!

FFS: „свест“ о диску – наставак

Основна питања код дизајнирања система датотека:

- Где на диску треба сместити метаподатке и податке?
- Како се могу искористити велики блокови без превеликог трошења простора (тј. унутрашње фрагментације)?

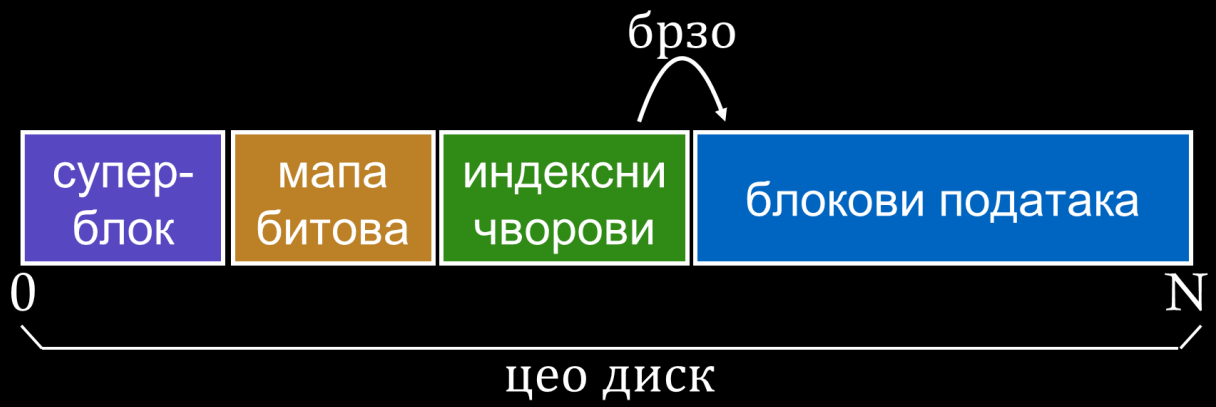
Техника распоређивања 1 – мапе битова

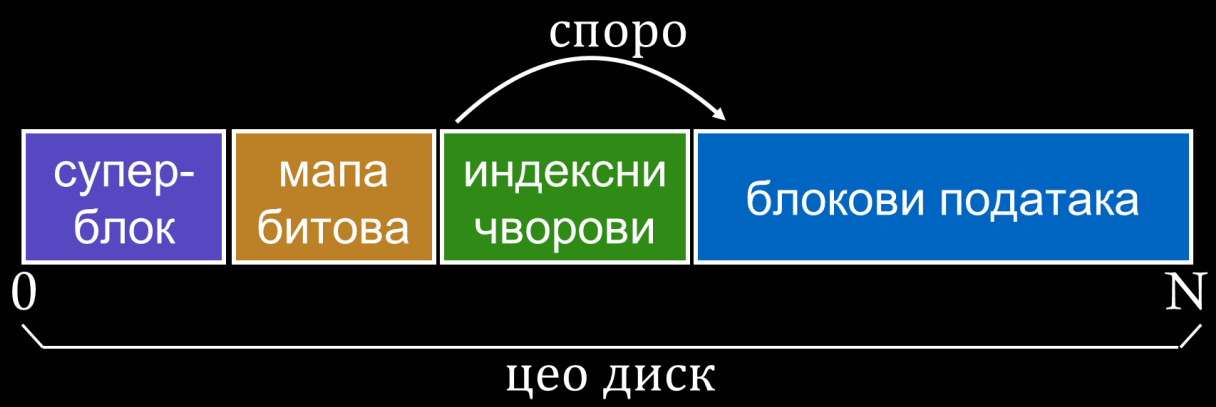


- Коришћење мапа битова уместо листе слободних локација.
- Омогућава већу брзину, као и глобалан поглед.
- Брже се проналазе слободни делови диска

Техника распоређивања 2 – групе

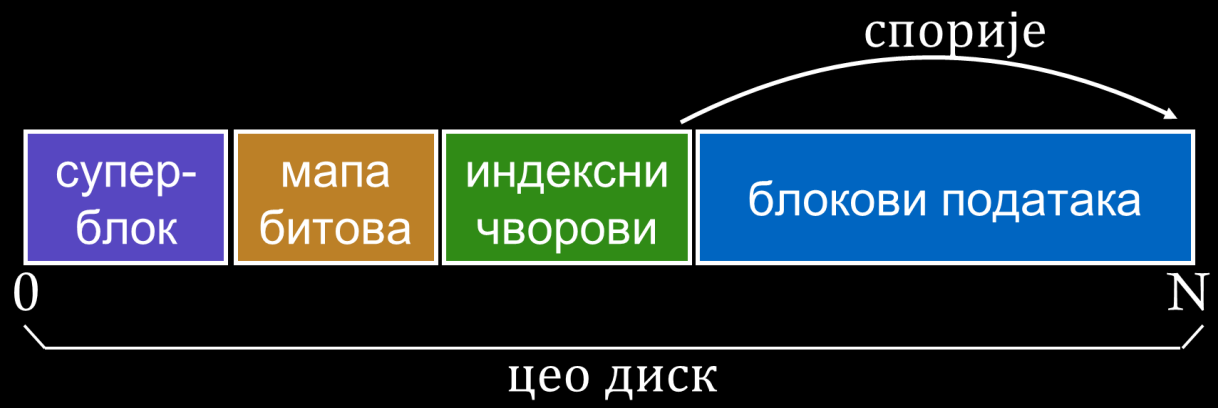
- Како да држимо индексне чворове близу података?

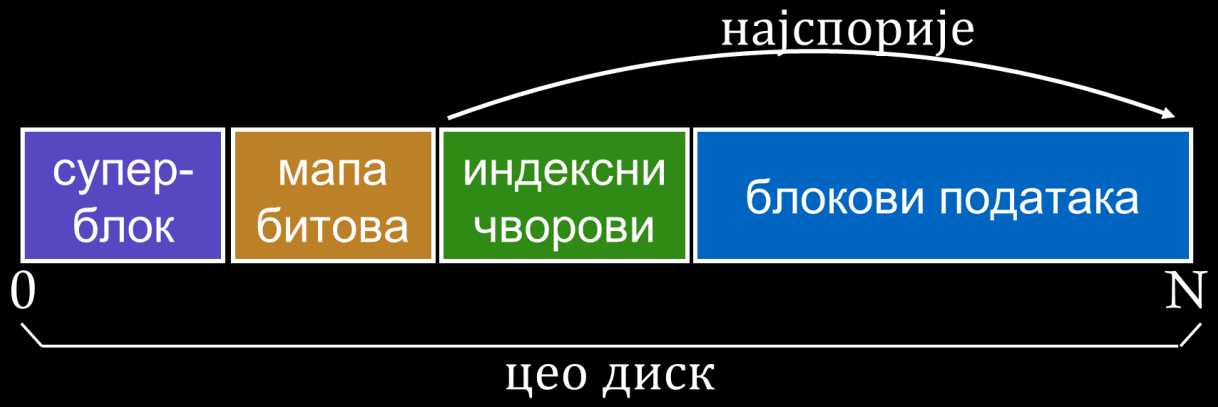




Техника распоређивања 2 – групе

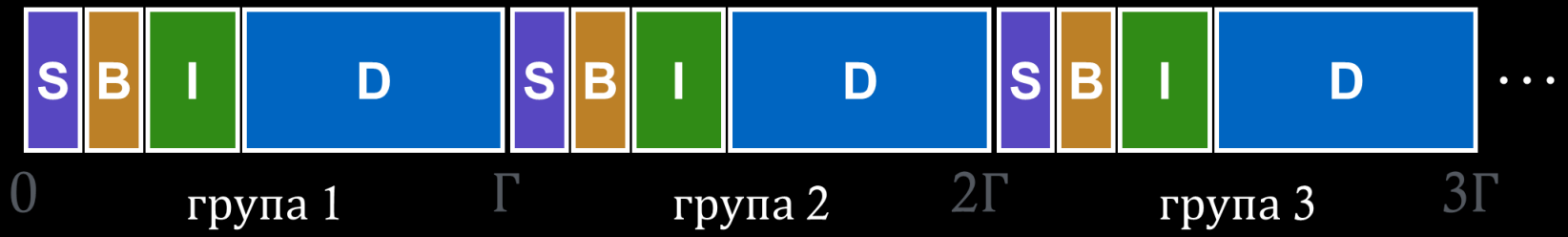
- Како да држимо индексне чворове близу података?





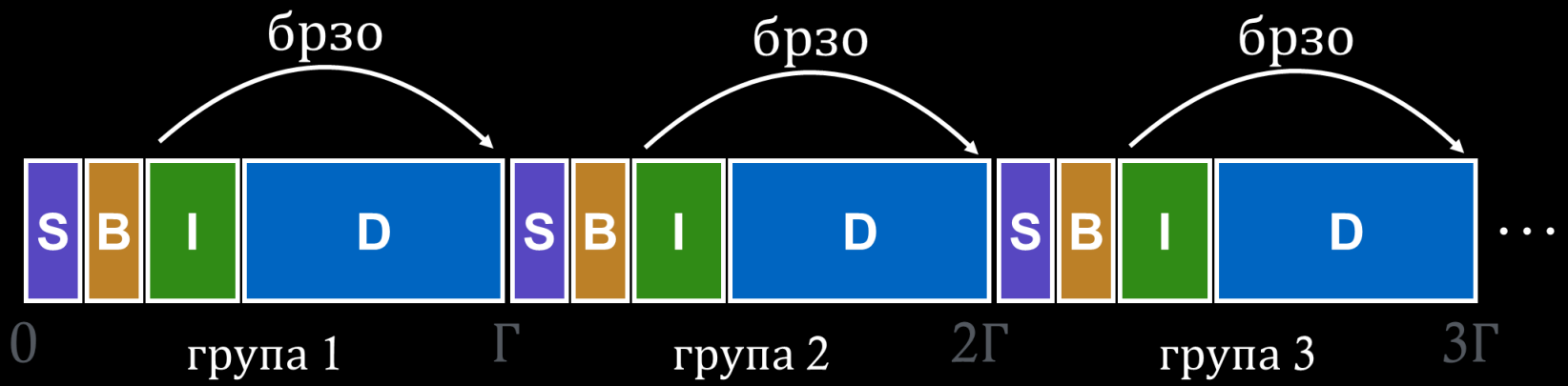
Техника распоређивања 2 – групе

- Како да држимо индексне чворове близу података?
- Одговор: Користимо групе блокова које распоредимо по диску. Треба покушавати да се подаци и њихови индексни чворови сместе у исту групу.



Техника распоређивања 2 – групе

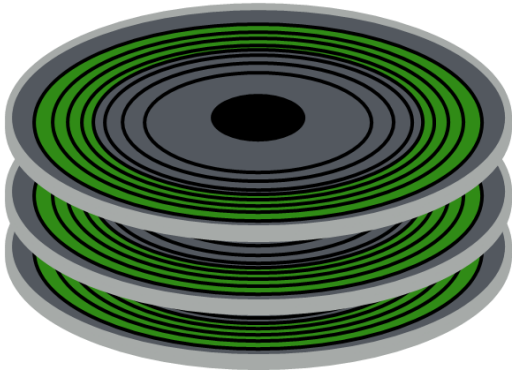
- Стратегија: алоцирати индексне чворове и блокове података у оквиру исте групе.



Организовање структуре: група цилиндара

FFS дели диск на мноштво **група** које чине неке стазе хард-диска на свим плочама, а које су једнако удаљене од центра (дакле, паралелне су, па им главе приступају истовремено). (**Цилиндри!**)

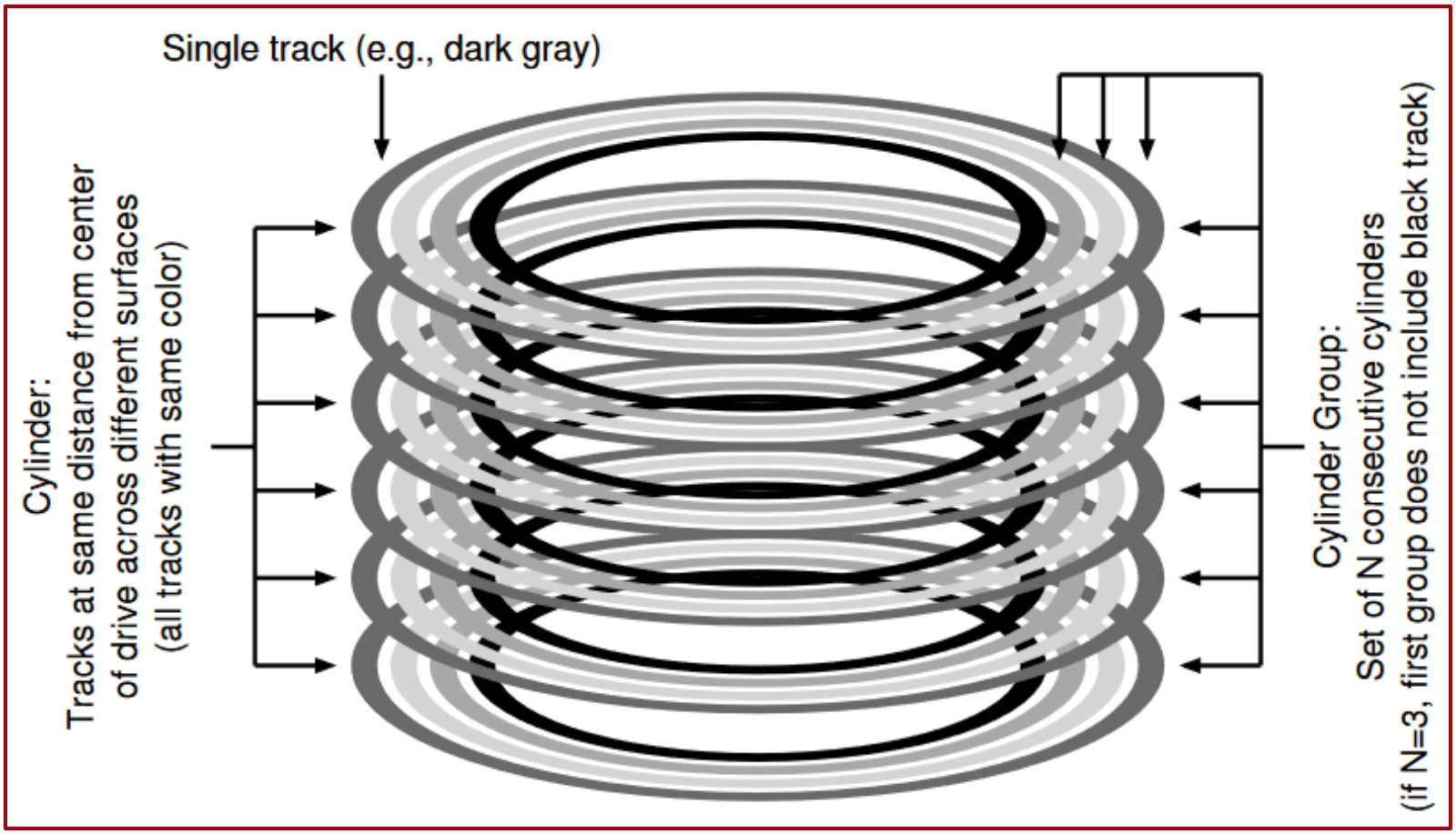
Код модерних система (нпр. на ext2, ext3, ext4) групе цилиндара се називају групе блокова. (Јер ОС хард-диск не види као цилиндре, већ искључиво као блокове.)

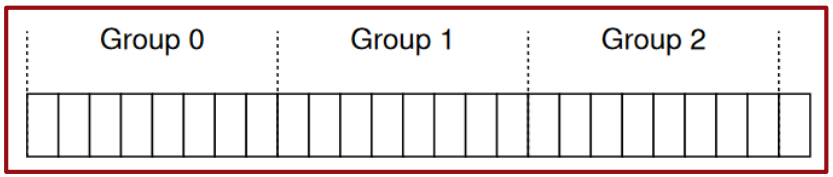


Ове групе се користе да би се побољшале перформансе претраге.

- Смештањем две датотеке у оквиру исте групе.
- Тако код приступања једном блоку за другим неће бити потребе за претрагом диска.
- FFS треба да алоцира датотеке и директоријуме у оквиру сваке од ових група.

Цилиндри


Single track (e.g., dark gray)
Cylinder:
Tracks at same distance from center
of drive across different surfaces
(all tracks with same color)
Cylinder Group:
Set of N consecutive cylinders
(if N=3, first group does not include black track)


Group 0
Group 1
Group 2


S
ib
db
Inodes
Data

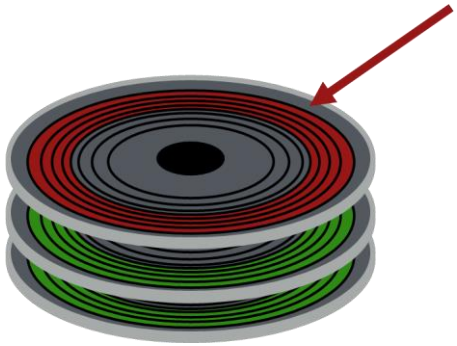
Организовање структуре: група цилиндара (наставак)

S	ib	db	Индексни чворови	Подаци

Структуре података постоје за сваку од група цилиндара.

Прави се и копија супер блока (да би се постигла већа поузданост (ако се нешто деси некој копији))

- Нпр. шта ако је (као што је то било код старих система) суперблок само на горњој плочи?



- inode мапа битова и мапа битова података (да би се пратило који су слободни индексни чворови и блокови података)
- индексни чворови и блокови података

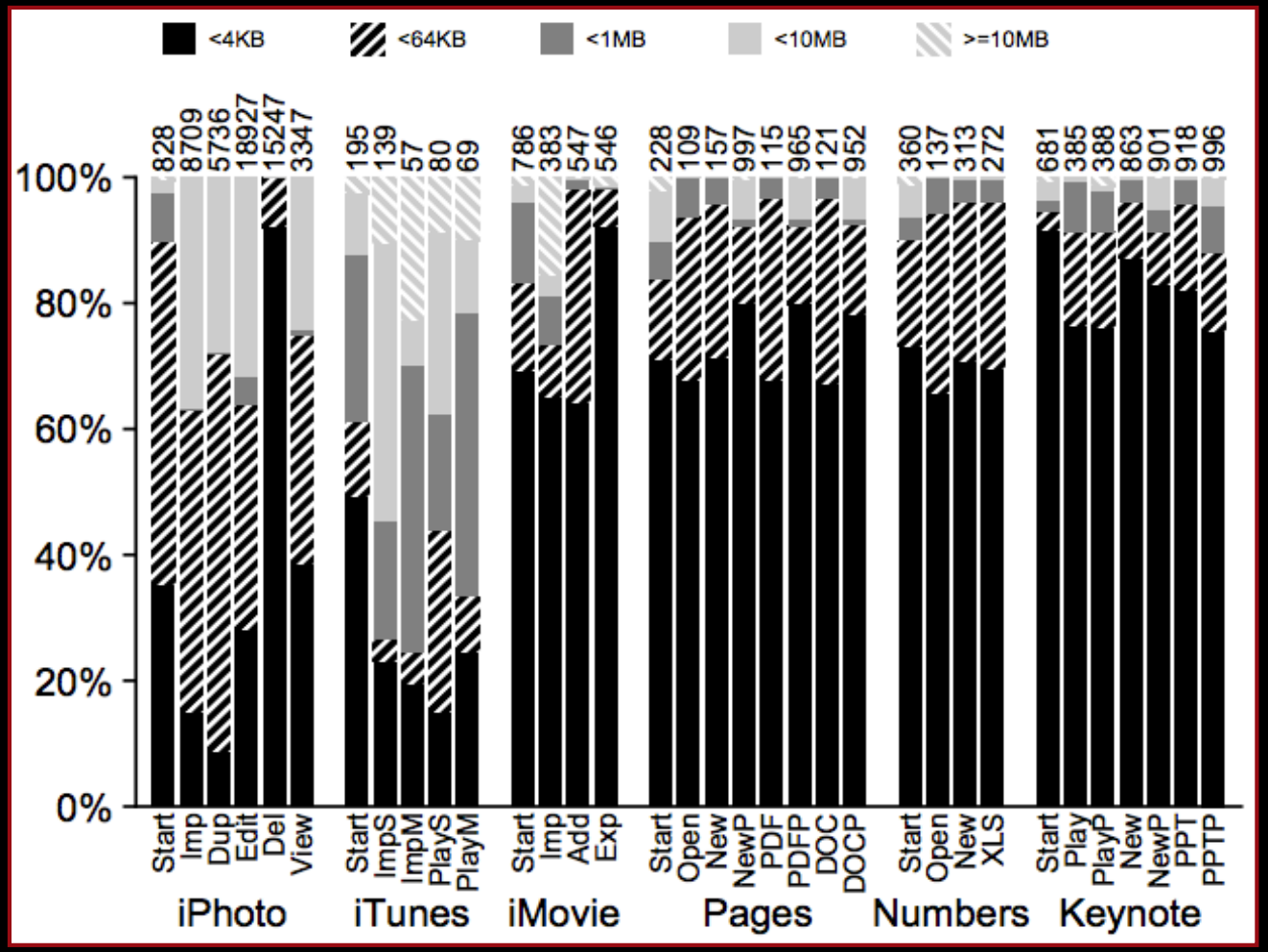
Техника распоређивања – величина блокова

- Као што је већ речено, код старих система датотека је удвостручавање величине блокова доводило да двоструко бољих перформанси.
- Стратегија: одабрати величину блока тако да нема потребе читати више од два индиректна блока (тј. два нивоа) да би се прочитао један блок података.

- Нпр. са величином блока од 4KB, која је највећа могућа величина датотеке код два нивоа индиректности?

```
(ВеличинаБлока/4B)*(ВеличинаБлока/4B)*ВеличинаБлока = 4GB
```

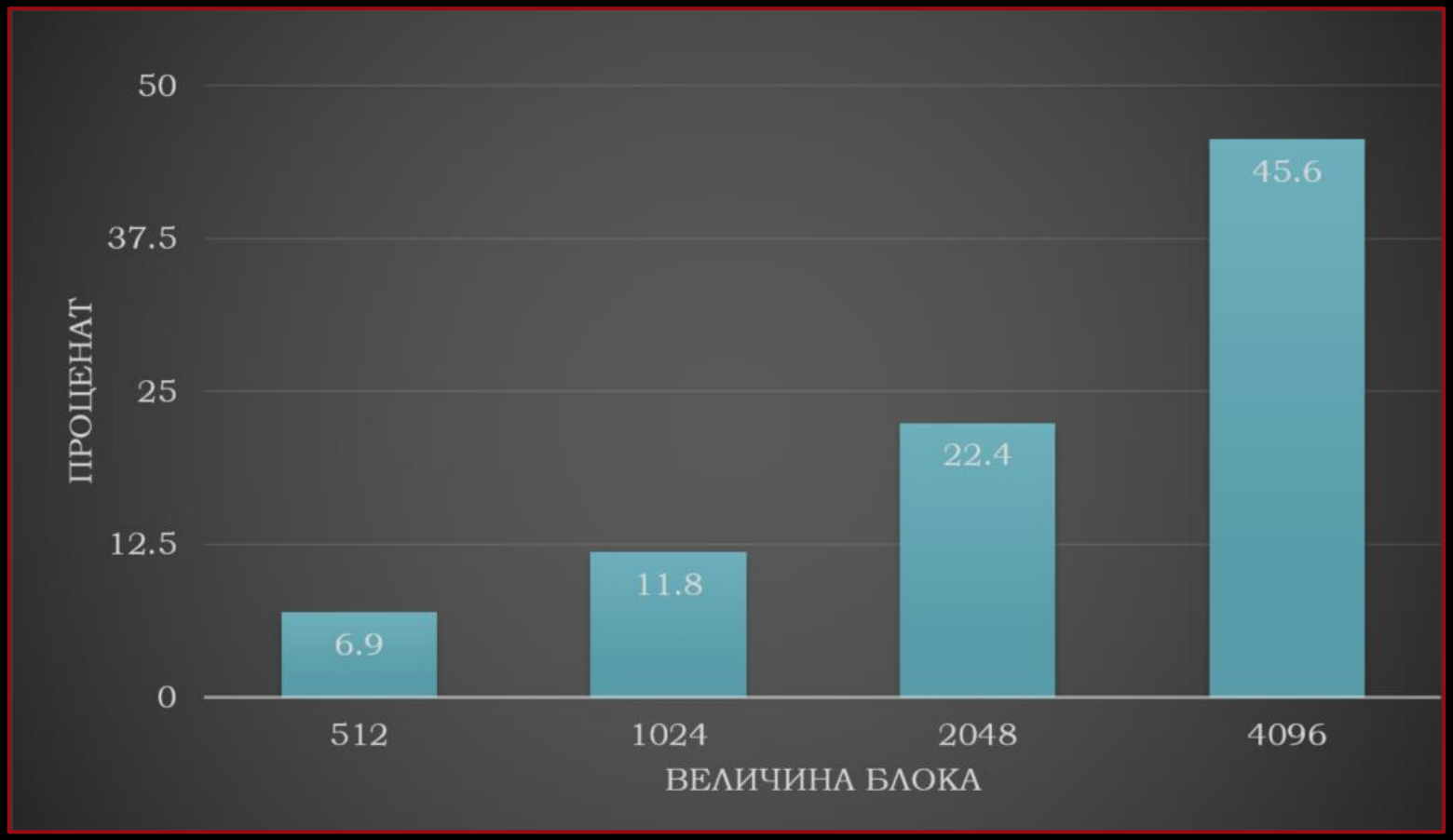
Техника распоређивања – већи блокови



Ако је код старих система датотека удвостручавање величине блокова доводило да двоструко бољих перформанси. Зашто онда не учинити блокове великим?

Зато што је већина датотека мала.

Техника распоређивања – већи блокови



Ако је код старих система датотека удвостручавање величине блокова доводило да двоструко бољих перформанси. Зашто онда не учинити блокове великим?

Зато што је већина датотека мала.

Решење – фрагменти

Хибридни приступ – комбиновање предности које доносе велики блокови са оним које доносе мали.

Коришћење великих блокова када су датотеке довољно велике.

Увођење **фрагмената** за датотеке које користе само део блока података.

> Пазите, фрагментнисане могу бити мале датотеке, или крајњи део велике датотеке (следи пример).

Пример – фрагменти

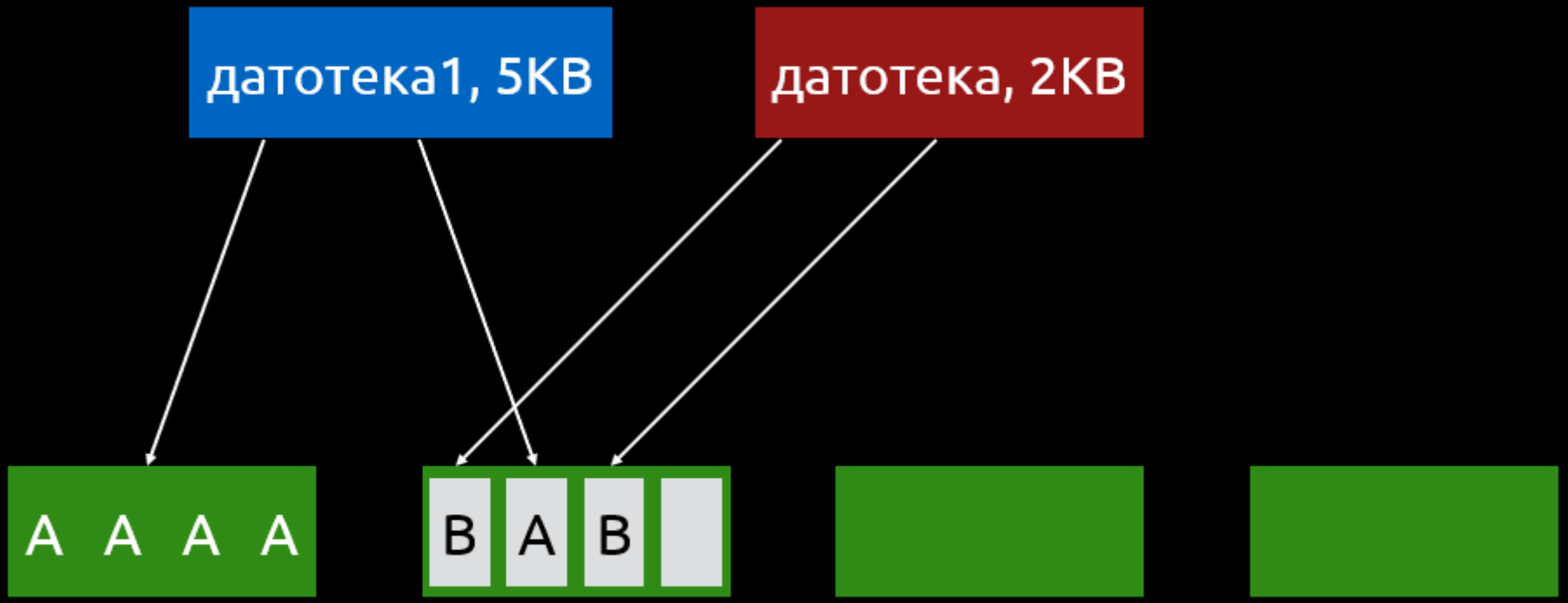
```
Величина блока = 4096
Величина фрагмента = 1024


          битови:  0000  0000  1111  0010
                   blk1  blk2  blk3  blk4
```

О томе да ли адреса указује на блок или на фрагмент нам говори офсет датотеке.

Шта када желимо да датотеке могу да расту?

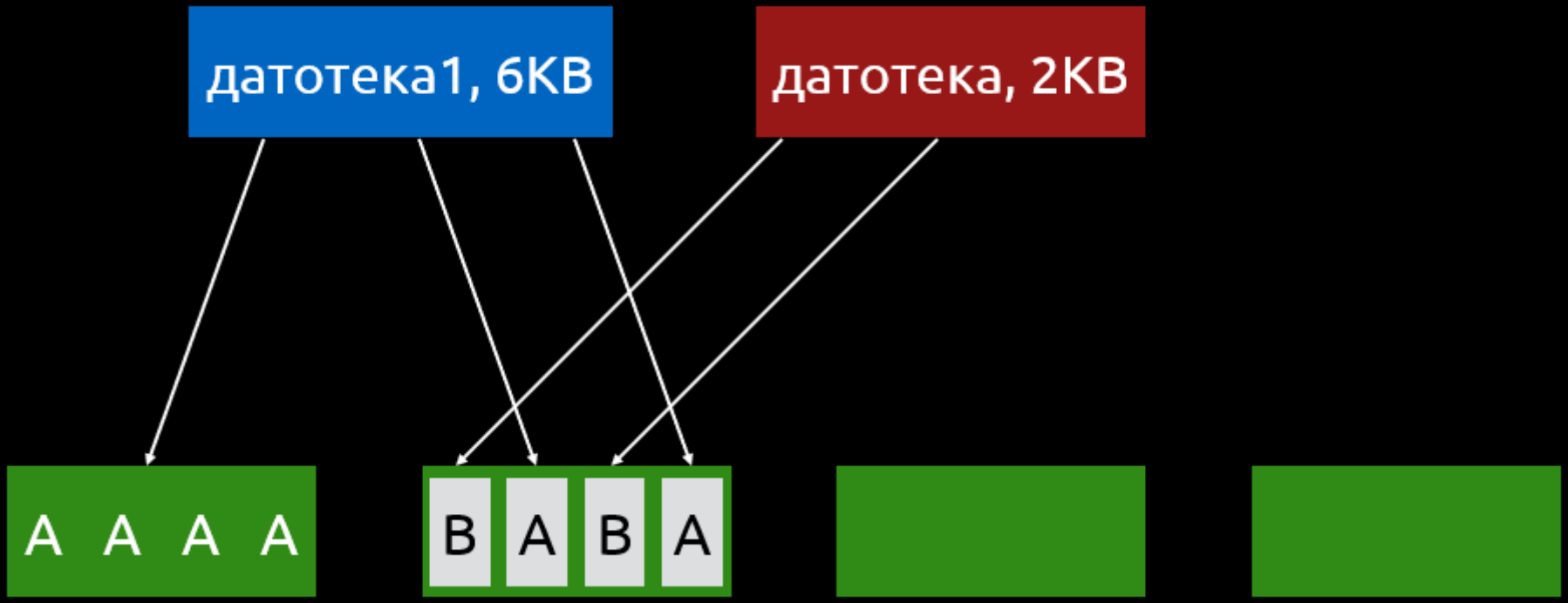
Морамо копирати фрагменте у нови блок (уколико за то нема простора у тренутном блоку).

Пример – фрагменти



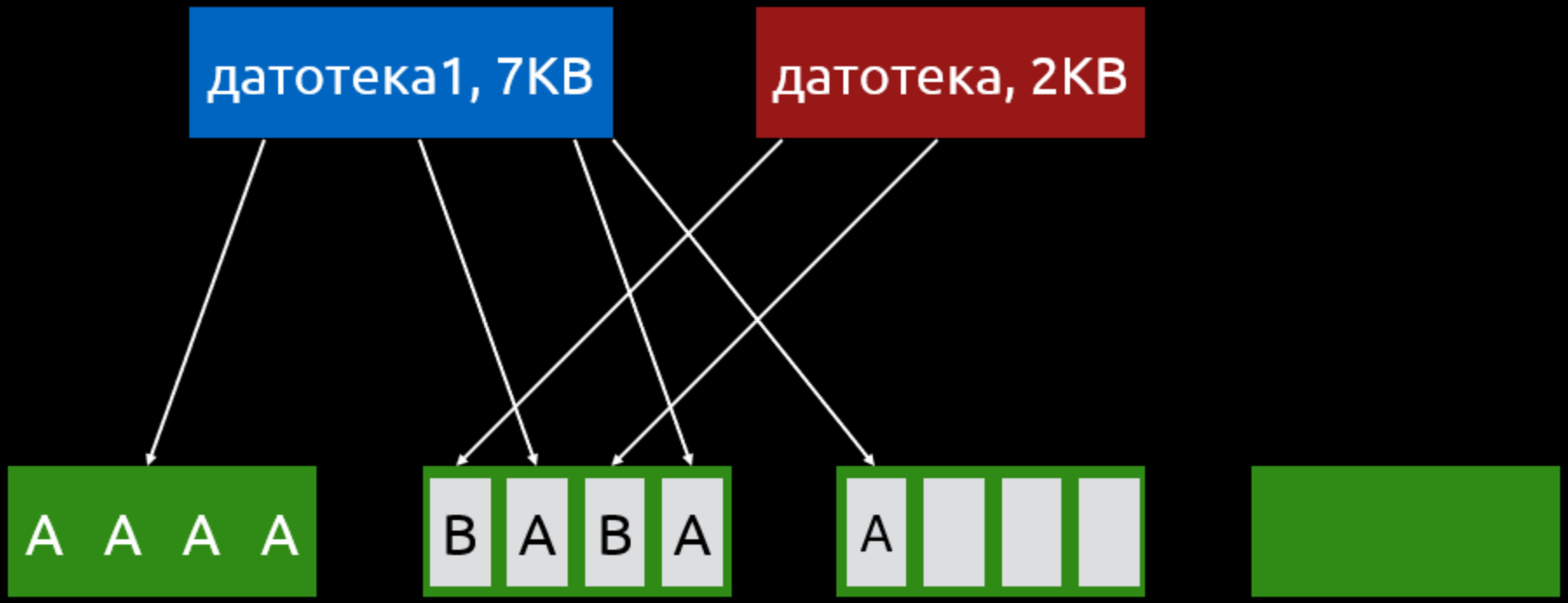
- Рецимо да сада треба да повећамо прву датотеку...

Пример – фрагменти



- Сада још једном повећавамо прву датотеку...

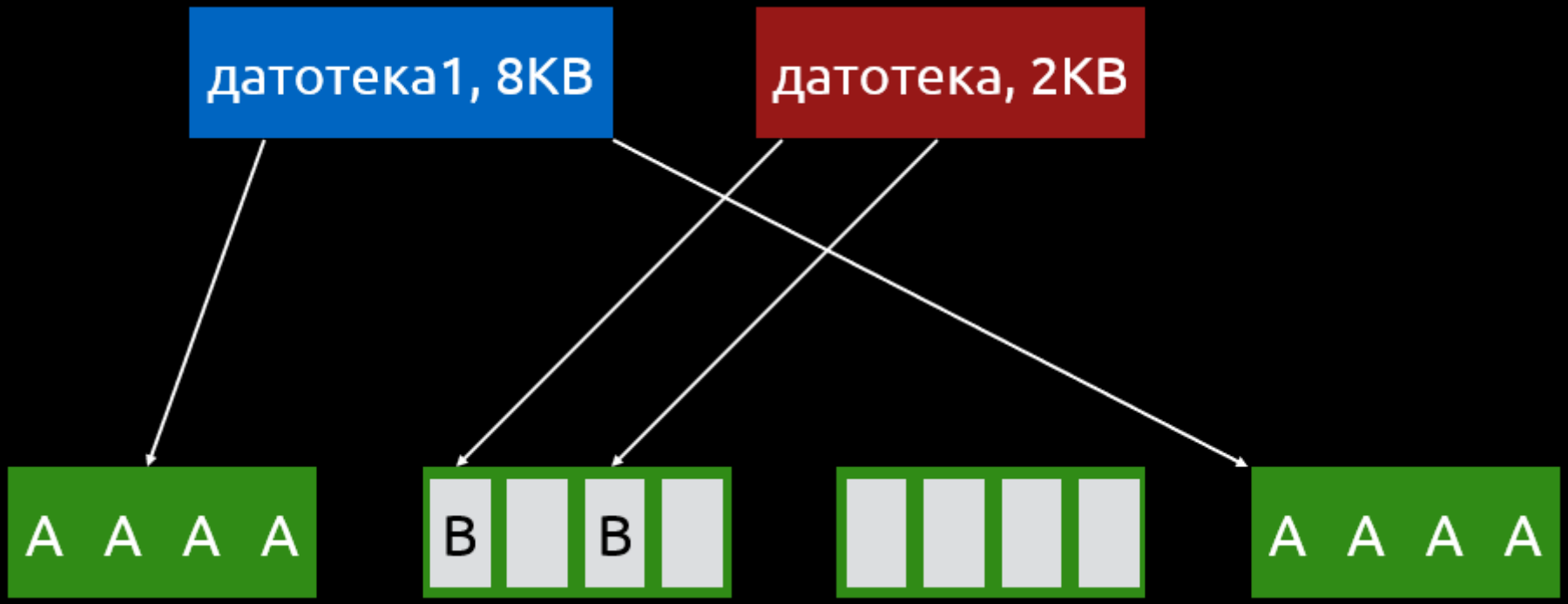
Пример – фрагменти



Опет повећавамо прву датотеку.

Али ово на слици није дозвољено, не смемо имати фрагменте у неколико различитих блокова. Дакле, шта треба урадити?

Пример – фрагменти



Повећавамо прву датотеку.

Копирамо фрагменте у нови блок.

Оптимална величина уписивања

Уписивање података који су мањи од величине блока није ефикасно. Видели сте то и на претходном примеру, на крају смо претходно уписане податке морали да копирамо у нови блок. Замислите да је оваквих уписивања било још…

Решење: нови API који оптимизује величину података који се уписују.

Паметније уписивање

Где би требало уписати нове податке и индексне чворове? Следи објашњење...

Како алоцирати датотеке и директоријуме?

Правило је „**држати повезане ствари заједно**“.

Правила:

1. Директоријуми се смештају близу индексних чворова директоријума.
2. Индексни чворови датотека се смештају близу блокова директоријума.
3. Блокови података се смештају близу индексних чворова.

Проблем: Систем датотека је једно велико стабло.

- Сви директоријуми и датотеке имају заједнички корен.
- Сви подаци у једном систему датотека су на неки начин повезани.

Покушај да се све што је иоле повезано смести једно близу другог не би успео.

Како алоцирати датотеке и директоријуме?

Ближе повезане податке треба смештати близу, а мање повезане далеко једне од других.

Смештање директоријума

- Проналажење групе цилиндара која има мали број алоцираних директоријума и велики број слободних индексних чворова.
- Смештање података директоријума и индексног чвора у ту групу.

Смештање датотека

- Алоцирање блокова података датотеке у исту групу у којој се налази њен индексни чвор.
- Овим се све датотеке смештају у исту групу као њихов директоријум.

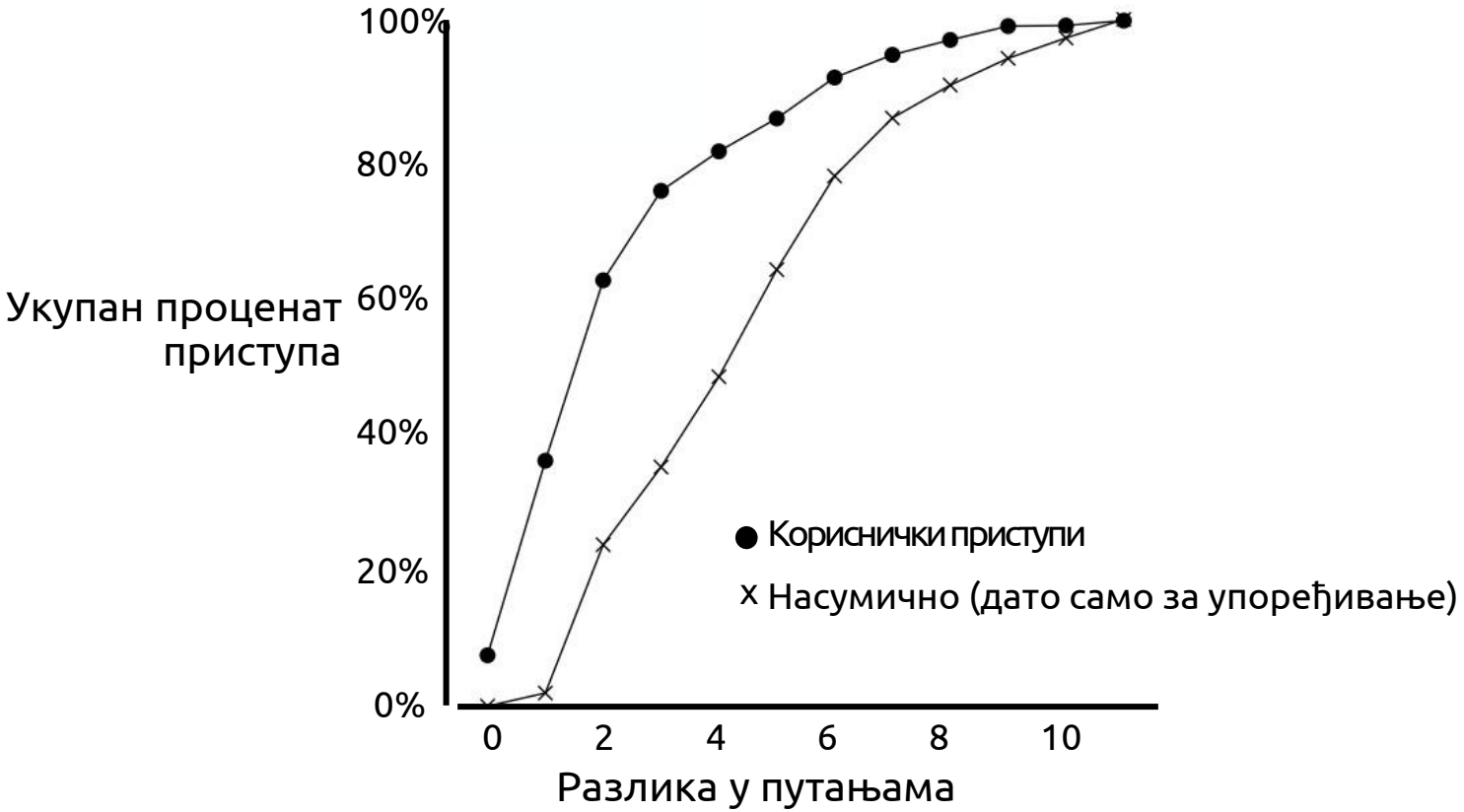
FFS локалност приступа датотекама

На следећем слајду је дата анализа приступа (на основу статистике извучене из софтвера *SEER file manager*-а, а она показује колико су просечно „далеко“ приступи датотекама, тј. колико су удаљени једни од других у оквиру стабла директоријума.

ЛОКАЛНОСТ ЈЕ ВАЖНА!

FFS локалност приступа датотекама – наставак

Овде „разлика у путањама“ представља колико корака су датотеке удаљене једна од других. 0 означава дату датотеку, 1 означава неку датотеку у истом директоријуму, 2 датотеку удаљену један директоријум, итд. А „проценат приступа“ је дат у односу на укупан број приступања датотекама.



Како одабрати које податке сместити у нову групу?

Када се креира нови директоријум, FFS га смешта у нову групу. (Зато се наредба **ls** извршава толико брзо на Линуксу, чак и када има много датотека, све те датотеке у датом директоријуму су једној групи.)

Чега се треба држати?

1. **Индексни чворови датотека –** алоцирају се у истој групи у којој је и директоријум.

2. **Индексни чворови директоријума –** алоцирају се у некој **новој** групи која нема много искоришћених индексних чворова.

3. **Први блок података –** алоцира се близу индексног чвора.

4. **Остали блокови података –** алоцирају се близу претходног блока података.

Пример

Директоријуми (**/**, **/a**, и **/b**)
и четири датотеке (**/a/c**, **/a/d**, **/a/e**, **/b/f**)

```
group inodes     data
    0 /--------- /---------
    1 a--------- a---------
    2 b--------- b---------
    3 c--------- cc--------
    4 d--------- dd--------
    5 e--------- ee--------
    6 f--------- ff--------
    7 ---------- ----------
    ...
```

Лош приступ

```
group inodes     data
    0 /--------- /----------
    1 acde------ accddee---
    2 bf-------- bff-------
    3 ---------- ----------
    4 ---------- ----------
    5 ---------- ----------
    6 ---------- ----------
    7 ---------- ----------
    ...
```

Добар приступ

Изузетак код великих датотека

Једна велика датотека би могла да скоро потпуно попуни групу. То не би оставило простора за мање датотеке.

То би изгледало овако:

G: група блокова

FFS смешта велике датотеке на посебан начин.

- Блокови датотеке се распоређују широм диска.
- Узима се у обзир време преноса великих количина података са диска, као и време претраге.
- Боље је да се диск претражује током читања једне велике датотеке, него да се врши претрага за много малих датотека!

Изузетак великих датотека

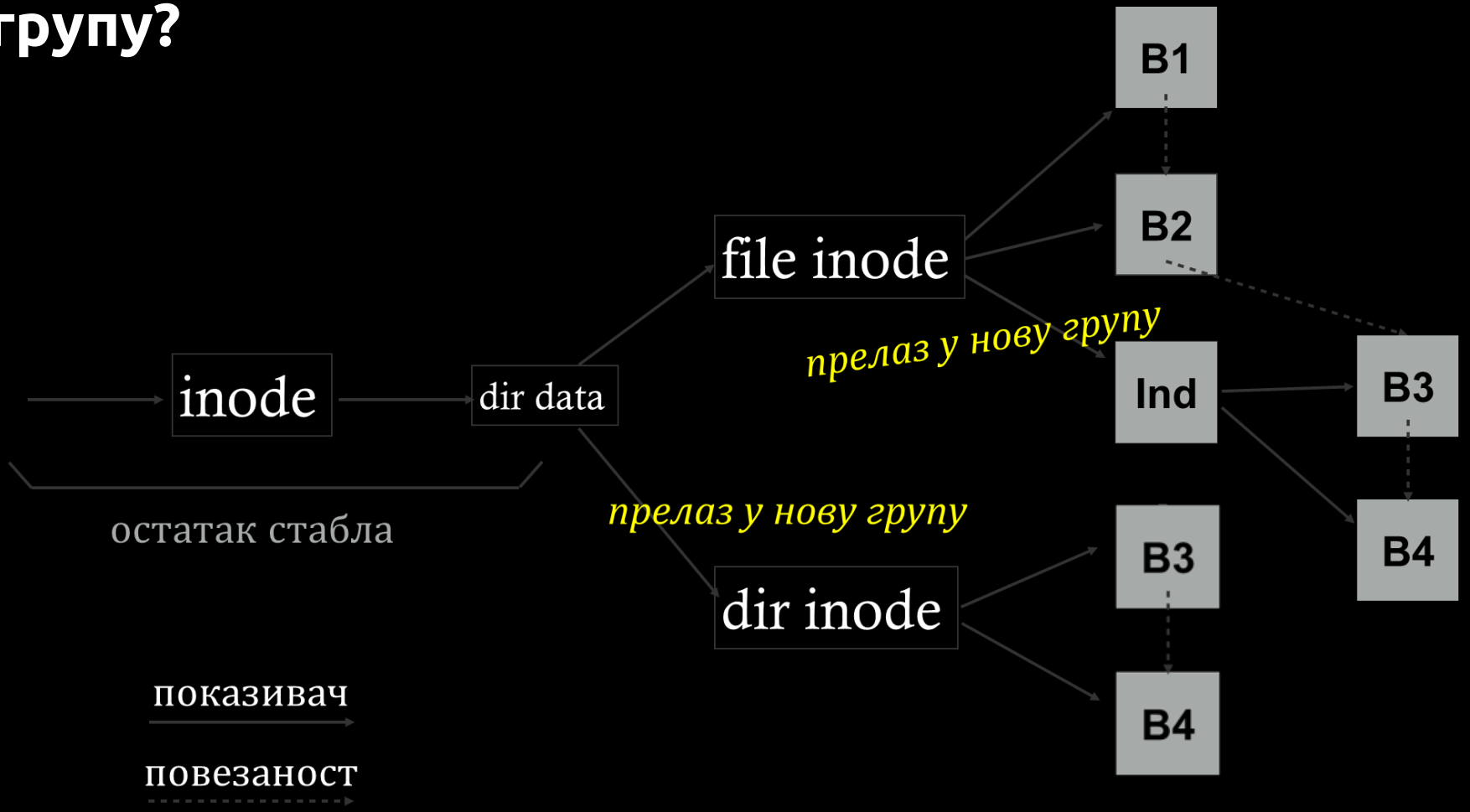
Датотека са претходног слајда би требало да буде смештена на следећи начин:

G: група блокова

G0	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
8 9		**0 1**		**2 3**		**4 5**		**6 7**		

Следи детаљнији пример.

Како одабрати које податке сместити у нову групу?



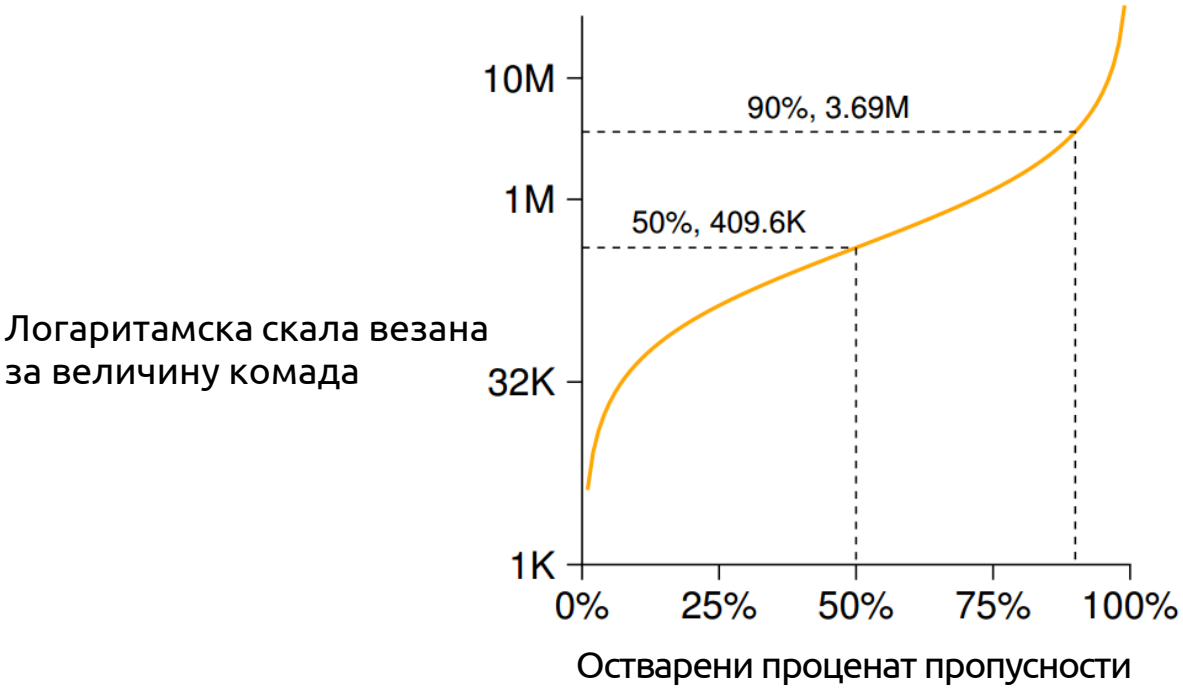
За велику датотеку је потребан индиректни блок.

Почевши од индиректног блока, сви следећи се смештају у нову групу.

Изузетак великих датотека (наставак)

Колико велики треба да буду комади на које се дели велика датотека? (Да би се амортизовали трошкови претраге.)

- Првих дванаест директних блокова се обавезно смештају у исту групу као *inode*.
- Индиректни блок, као и сви блокови на које он указује се смештају у различиту групу.



Допуна правила којих се треба држати

1. **Индексни чворови датотека:** алоцирају се у истој групи у којој је и директоријум.
2. **Индексни чворови директоријума:** алоцирају се у некој **новој** групи која нема много искоришћених индексних чворова.
3. **Први блок података:** алоцира се близу индексног чвора.
4. **Остали блокови података:** алоцирају се близу претходног блока података.

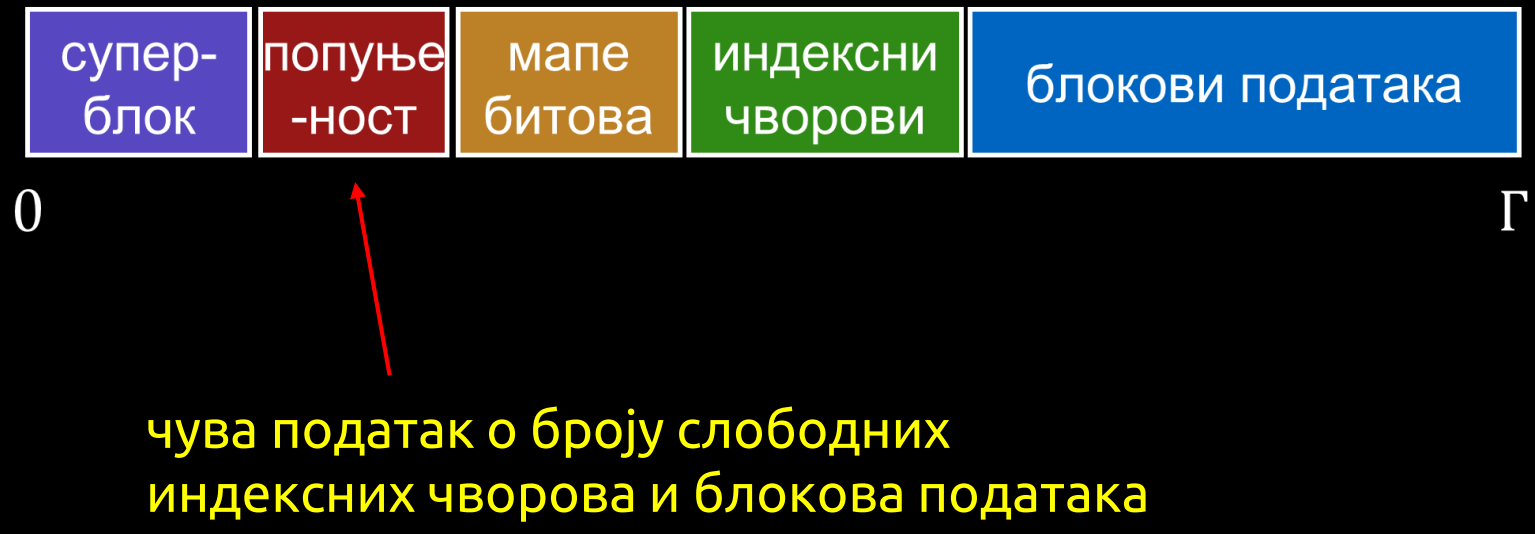
(додатак)

5. **Блокови података великих датотека:** након 48KB (12*4KB), блокови се смештају у другу групу. Након сваког следећег 1MB података, смештај блокове у нову групу (притом се бира група која није много попуњена).

Како одабрати које податке сместити у нову групу?

Како систем датотека зна коју нову групу да одабере током смештања велике датотеке?

Податак о томе колико је група попуњена се може сместити на почетак групе.



Још неке ствари о FFS

- Бафери стаза на хард-диску.
- Омогућио је задавање дугачких имена датотека (она омогућавају „изражајнија“ имена у оквиру система датотека).
- Симболичке везе.
- Атомична операција ***rename***.