תרגול 13

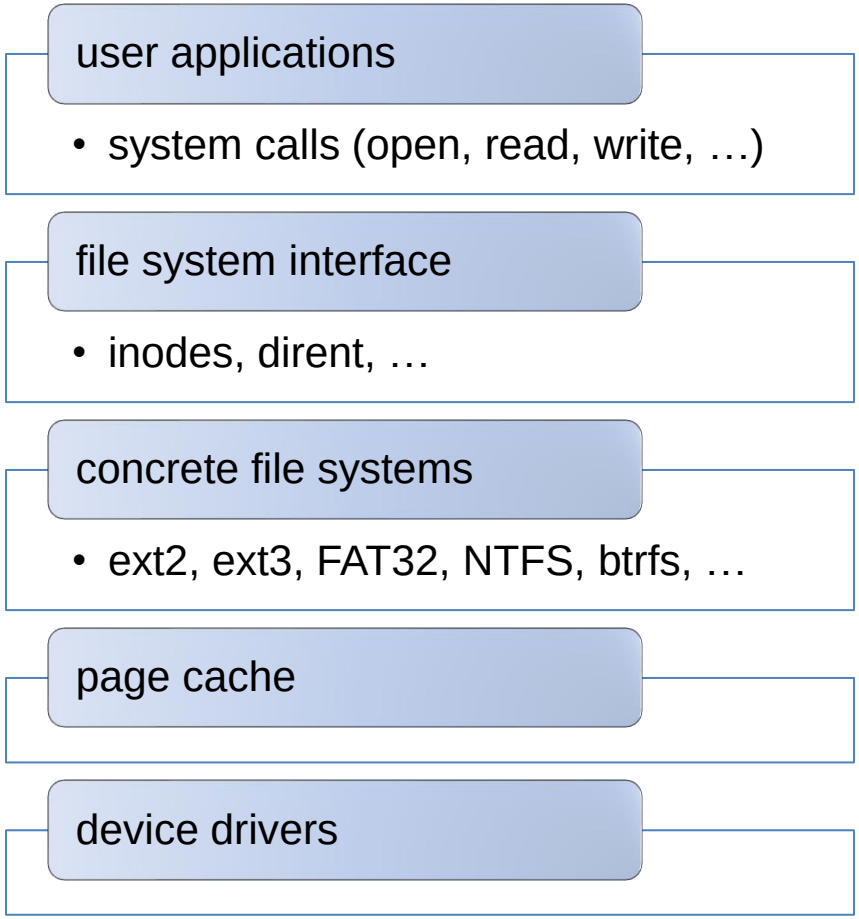
ממשק מערכת הקבצים בלינוקס

Very Simple File System (VSFS)

File Allocation Table (FAT)

TL;DR

- מצד אחד, בלינוקס "everything is a file".
  - regular files, directories, links, sockets, pipes, fifos, …
- מצד שני, קיימים מגוון סוגי קבצים, אמצעי אחסון פיזיים, ומערכות קבצים המציגות ממשקים שונים.
- לינוקס מוסיפה שכבת אבסטרקציה כדי להציג למשתמש ממשק אחיד ונוח למערכת הקבצים.



ממשק מערכת הקבצים בלינוקס

תזכורת: מהי מערכת הפעלה?

- מערכת תוכנה אשר אחראית על ניהול החומרה עבור המשתמשים.

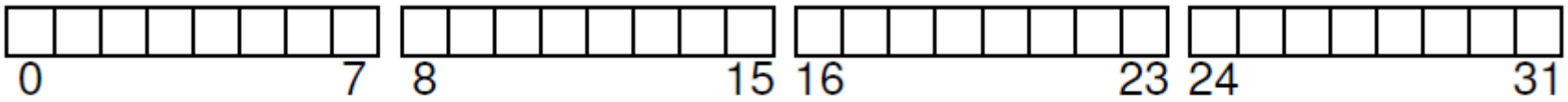
- מערכת ההפעלה מספקת אבסטרקציות לשלושת רכיבי החומרה המרכזיים:

רכיב חומרה	⇔	אבסטרקציה עיקרית
מעבד	⇔	תהליך
זיכרון פיזי	⇔	מרחב זיכרון וירטואלי
דיסק	⇔	מערכת הקבצים

מהו דיסק?

- כונן דיסק קשיח (HDD = hard disk drive) הוא התקן אחסון עמיד.
  - כלומר המידע נותר על הדיסק גם אם אין מתח, ובפרט גם לאחר כיבוי המחשב.

- מבחינת מערכת ההפעלה, הדיסק הוא מערך של **סקטורים**.
- סקטור == 512 בתים רציפים המתחילים בכתובת מיושרת.



- למרות שהדיסק פועל ביחידות של סקטורים (512 בתים), מערכת ההפעלה מנהלת קבצים ביחידות גדולות יותר – **בלוקים** בגודל 4KB.

מדוע?

מהו סקטור?

- הדיסק מבטיח למערכת ההפעלה כי פעולת כתיבה של סקטור בודד היא אטומית – הסקטור נכתב לדיסק בשלמותו, או לא נכתב בכלל.
- אין אטומיות ביחידות גדולות יותר מסקטור בודד.
  - כתיבה של בלוק בגודל 4KB יכולה להיקטע באמצע (למשל בגלל נפילת מתח) ואז המשתמש יראה תוצאות כתיבה חלקית.
- בגלל מבנה הדיסק, גישה סדרתית לסקטורים סמוכים מהירה יותר (בערך פי 100) מגישה אקראית לסקטורים המפוזרים בדיסק.
- שימו לב: המונחים סקטורים ובלוקים מתבלבלים לפעמים...

מהו קובץ?

- מערך של בתים (ללא מבנה מיוחד).

- אוסף הפעולות על קבצים בלינוקס ניתן ע"י קריאות המערכת:
- creat() – יצירת קובץ חדש.
- open() – פתיחת קובץ קיים (או יצירת קובץ חדש).
- read() – קריאה מתוך קובץ פתוח.
- write() – כתיבה לקובץ פתוח.
- close() – סגירת קובץ פתוח.
- unlink() – מחיקת קישור לקובץ (ואולי גם את הקובץ עצמו אם זה הקישור האחרון).
- ועוד עשרות רבות של קריאות מערכת...

תכונות של קבצים

- מערכת הקבצים שומרת לכל קובץ גם תכונות נוספות (metadata) במבנה הנקרא inode. תכונות לדוגמה:

1. inode number – מזהה יחודי לקובץ.
2. גודל הקובץ.
3. הרשאות גישה – למי מותר לקרוא/ לכתוב/להריץ את הקובץ.
4. חותמות זמן – הזמן האחרון בו קראו/כתבו מהקובץ.
5. מיקום בדיסק – הסקטורים בדיסק המרכיבים את הקובץ.

- בהמשך נראה איך מערכות קבצים שונות שומרות את המידע הזה.

שימו לב: ה-inode אינו שומר את שם הקובץ כי לאותו קובץ יכולים להיות מספר שמות שונים באמצעות קישורים (links).
שם הקובץ (ליתר דיוק, שם הקישור) נשמר בתיקיה המכילה אותו.

קריאת המערכת stat()

```
user@ubuntu:~$ touch file
user@ubuntu:~$ stat file
  File: file
  Size: 0     Blocks: 0     IO Block: 4096   regular empty file
Device: 801h/2049d   Inode: 18483362    Links: 1
Access: (0644/-rw-r--r--)  Uid: (1000/user)  Gid: (1000/user)
Access: 2020-01-12 10:44:38.006213596 +0200
Modify: 2020-01-12 10:44:38.006213596 +0200
Change: 2020-01-12 10:44:38.006213596 +0200
user@ubuntu:~$ echo -n "a" >> file
user@ubuntu:~$ stat file
  File: file
  Size: 1     Blocks: 8     IO Block: 4096   regular file
Device: 801h/2049d   Inode: 18483362    Links: 1
Access: (0644/-rw-r--r--)  Uid: (1000/user)  Gid: (1000/user)
Access: 2020-01-12 10:44:38.006213596 +0200
Modify: 2020-01-12 10:45:01.588753840 +0200
Change: 2020-01-12 10:45:01.588753840 +0200
```

למה מספר הבלוקים הוא 8?

מהי תיקיה?

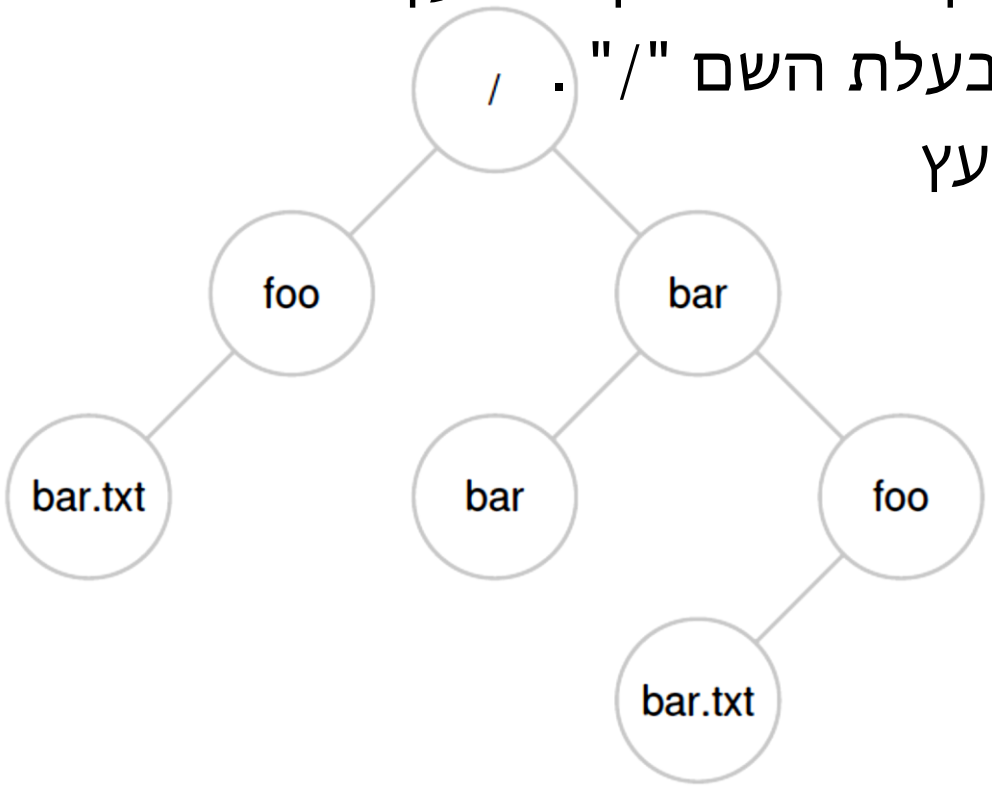
- סוג מיוחד של קובץ המיוצג ע"י מערך של רשומות.
- כל רשומה היא מיפוי:
  name → inode number

name	inode number
.	33
..	15
foo	5
bar	806
	-1
	-1
	-1

- כל תיקיה מכילה תמיד שתי רשומות מיוחדות:
- "." (נקודה) – מצביע לתיקיה הנוכחית.
- ".." (שתי נקודות) – מצביע לתיקיית האב.

היררכית מערכת הקבצים

- כאמור, תיקיות בלינוקס מצביעות לקבצים ו/או תיקיות נוספים.
- לינוקס מארגנת את כל הקבצים והתיקיות לעץ יחיד.
- שורש העץ הוא התיקיה בעלת השם "/".
- ניתן להתייחס לקבצים בעץ לפי הנתיב האבסולוטי (absolute pathname) או לפי הנתיב היחסי (relative pathname) לתיקיה הנוכחית.



פעולות על תיקיות

- ניתן ליצור תיקיה חדשה באמצעות קריאת המערכת mkdir().
- ניתן להסיר תיקיה ריקה באמצעות קריאת המערכת rmdir().

- לא ניתן לקרוא/לכתוב לקובץ תיקיה באמצעות קריאות המערכת read()/write() מכיוון שפרטי המימוש של תיקיות מוסתרים מהמשתמש.
  - מערכות קבצים שונות מממשות תיקיות באופן שונה, כפי שנראה בהמשך.

- ניתן לקרוא קובץ תיקיה (כלומר לקרוא את רשימת הקבצים השייכים לתיקיה) רק באמצעות קריאת המערכת getdents().
- ניתן לכתוב לקובץ תיקיה רק באמצעות יצירה של קובץ חדש או קישור חדש בתוכה.

בלינוקס יש שני סוגי קישורים (links)

hard link

ln src dst

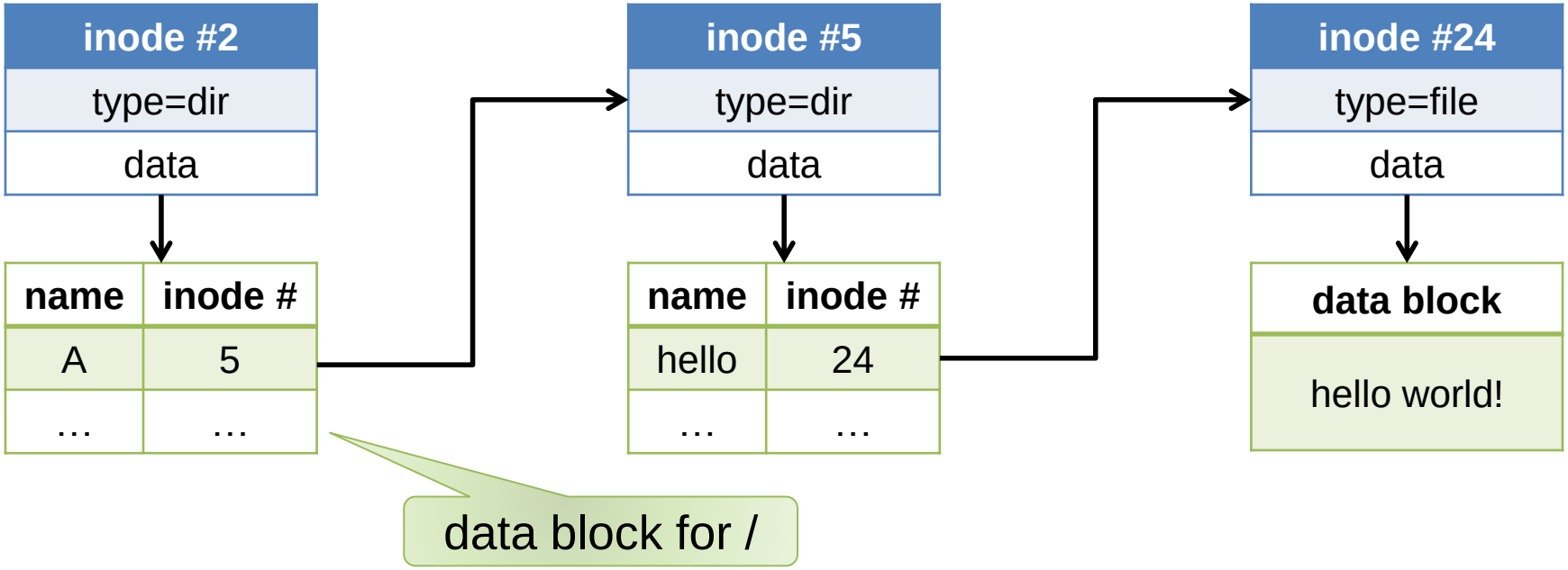
- קישור קשיח הוא שם נרדף לקובץ המקורי כי הוא מצביע ישירות ל-inode של הקובץ המקורי.
- כתיבה דרך הקישור כותבת לקובץ אליו הוא מצביע.
- מחיקת הקישור תקטין את מונה הקישורים של הקובץ (כפי שנשמר ב-inode).
- הקובץ יימחק מהדיסק רק כאשר כל ה-hard links אליו יימחקו.

soft / symbolic link

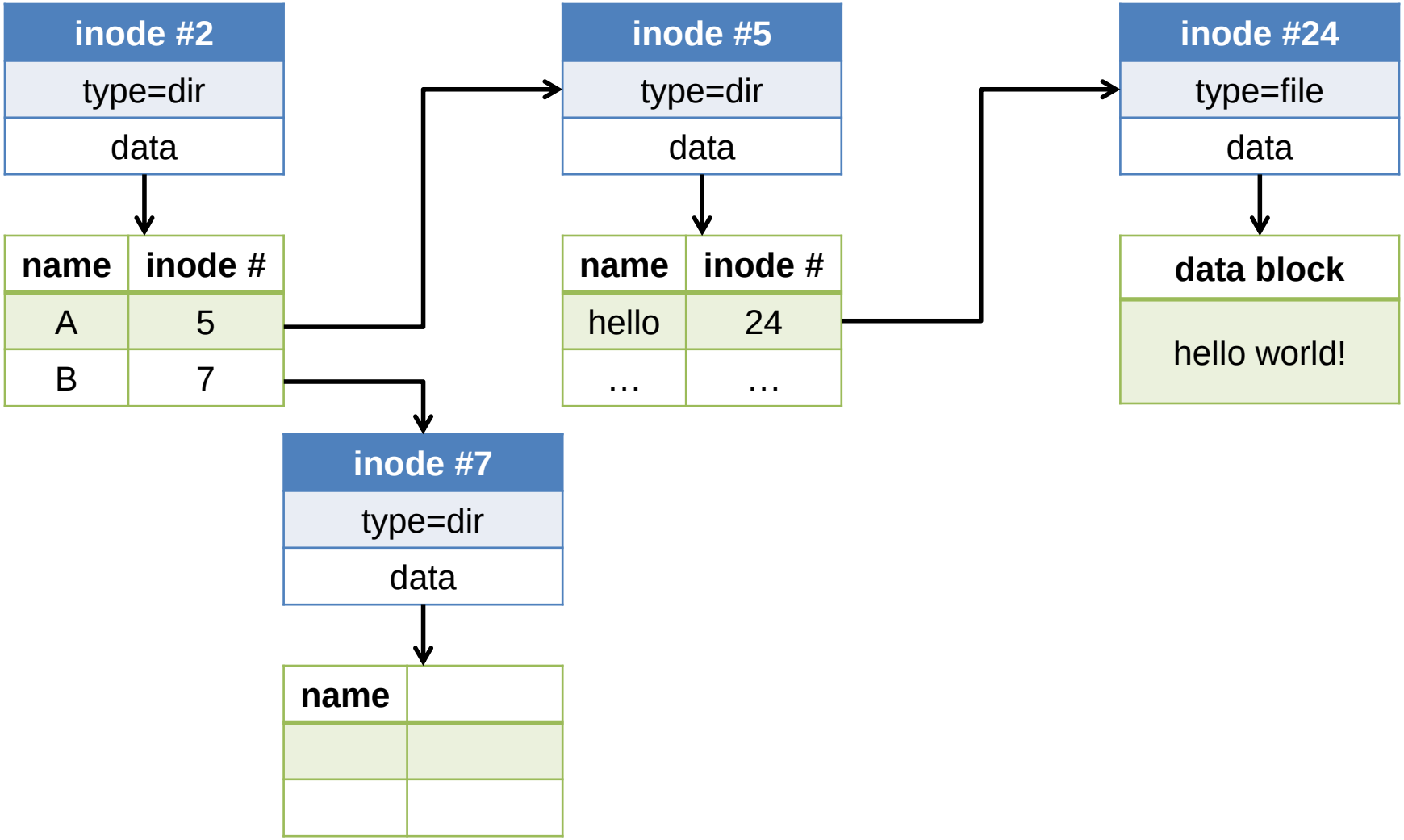
ln -s src dst

- קישור סימבולי הוא קובץ חדש עם inode נפרד מזה של הקובץ המקורי.
- כתיבה דרך הקישור כותבת לקובץ אליו הוא מצביע.
- מחיקת הקישור (באמצעות הפקודה rm) לא תמחק את הקובץ המוצבע.
- אפשר ליצור קישורים סימבוליים גם לקובץ שלא קיים.

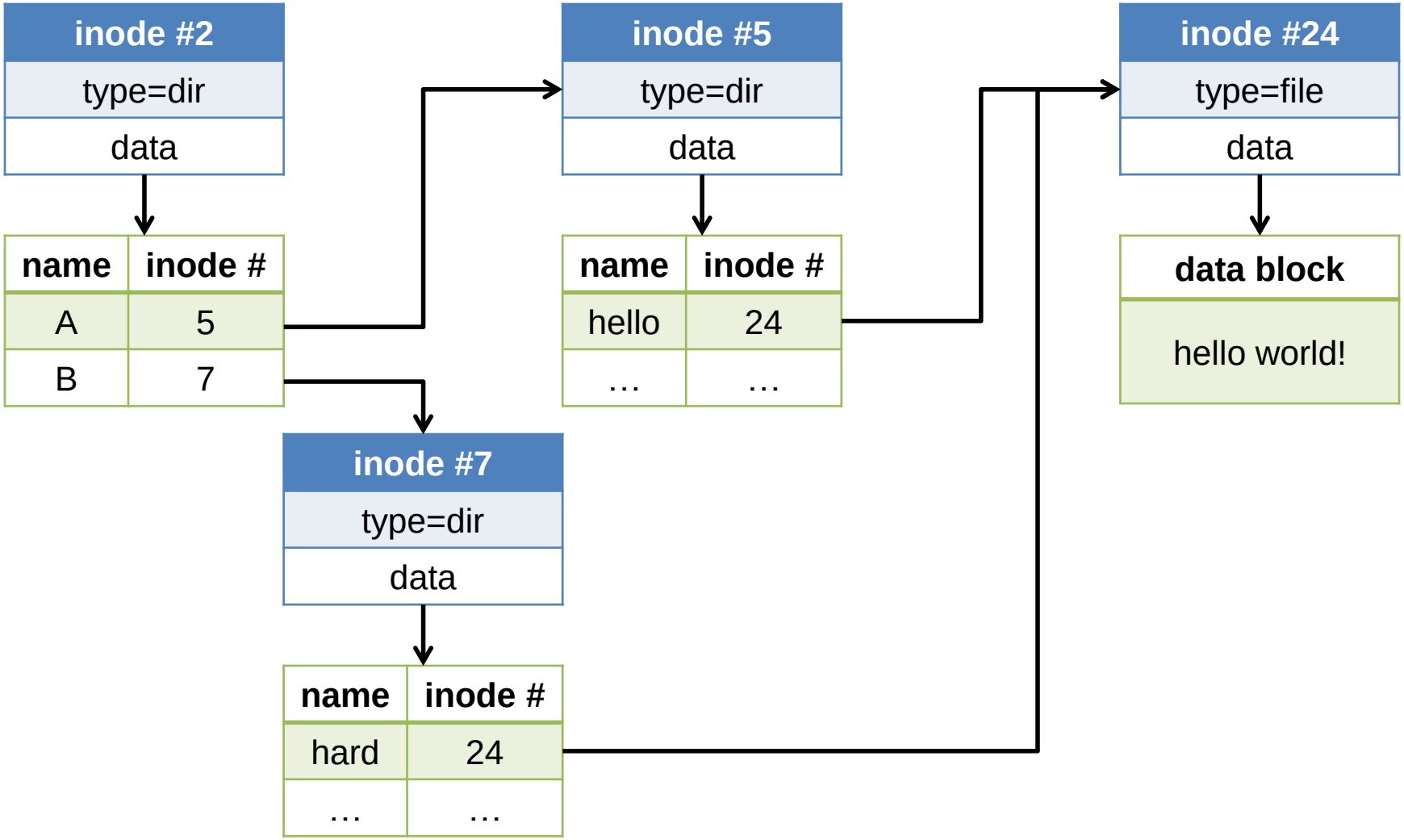
דוגמה: קישורים



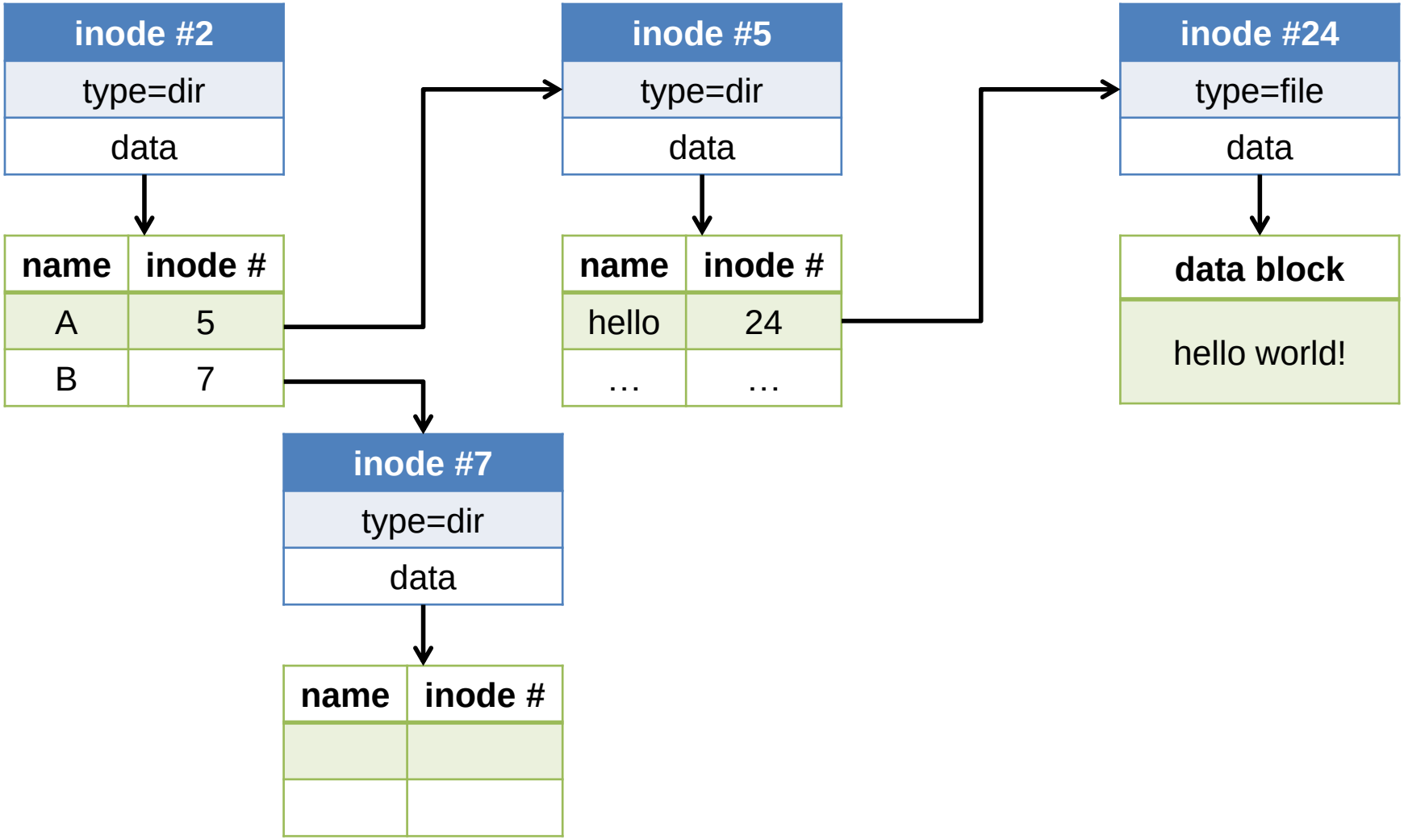
>> mkdir /B


inode #2
type=dir
data
name
inode #
A
5
B
7
inode #5
type=dir
data
name
inode #
hello
24
…
…
inode #24
type=file
data
data block
hello world!
inode #7
type=dir
data
name

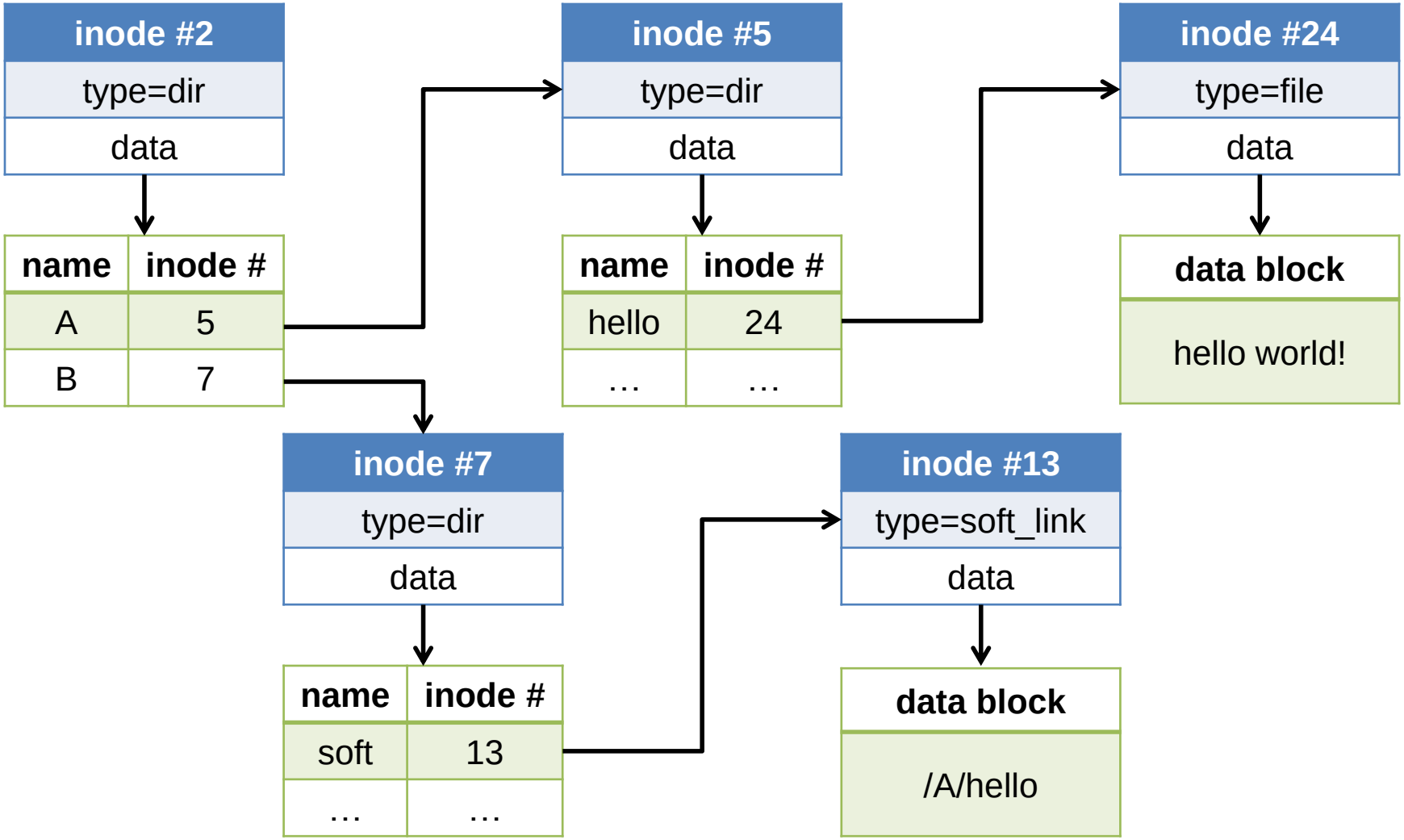
>> ln /A/hello /B/hard



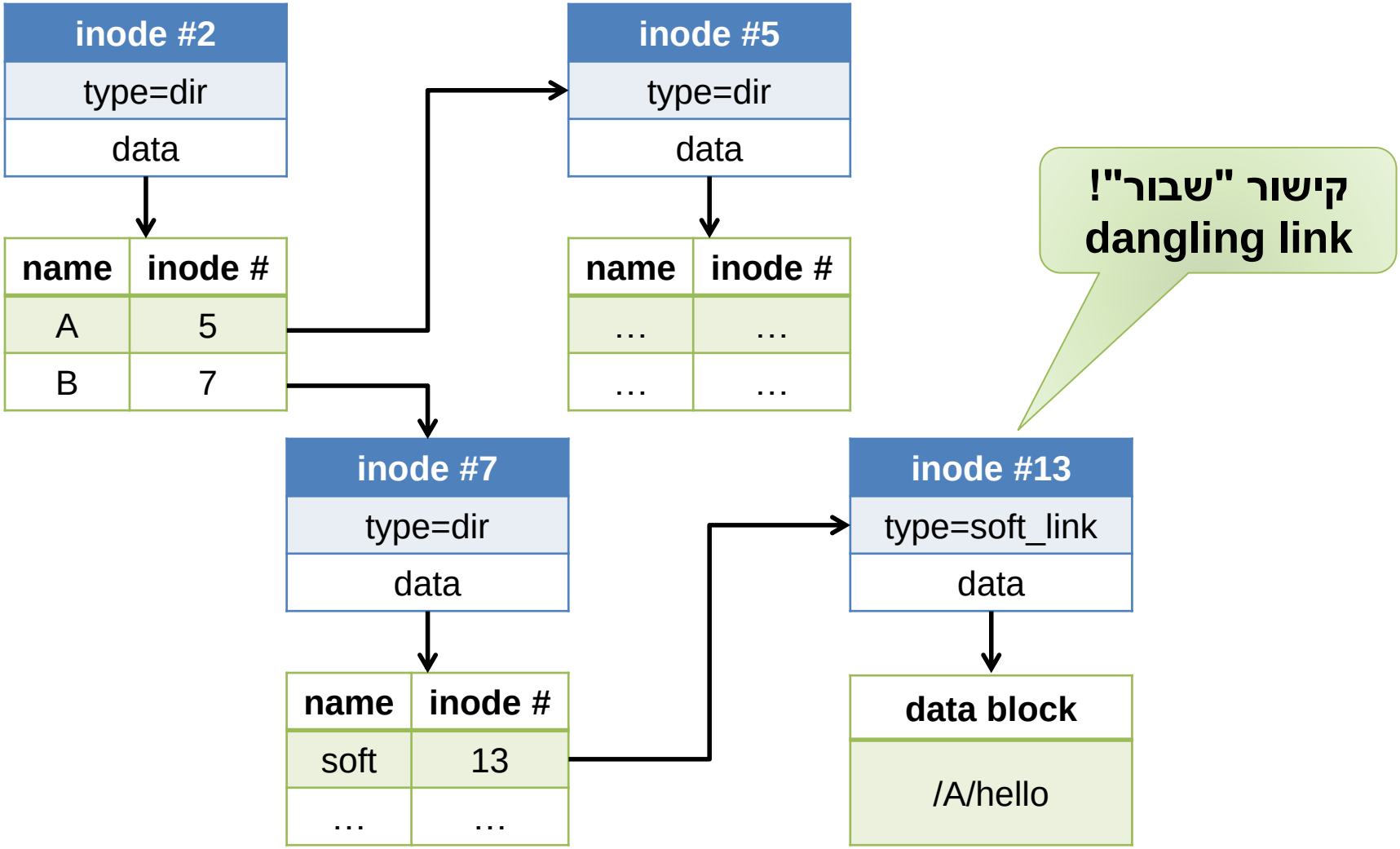
>> rm /B/hard


inode #2
type=dir
data
name
inode #
A
5
B
7
inode #5
type=dir
data
name
inode #
hello
24
…
…
inode #24
type=file
data
data block
hello world!
inode #7
type=dir
data
name
inode #

>> ln -s /A/hello /B/soft



>> rm /A/hello


inode #2
type=dir
data
name
inode #
A
5
B
7
inode #5
type=dir
data
name
inode #
…
…
…
…
inode #7
type=dir
data
name
inode #
soft
13
…
…
inode #13
type=soft_link
data
data block
/A/hello
קישור "שבור"!
dangling link

השוואה בין סוגי קישורים בלינוקס

קישורים קשים

✓ לא יכולים להיות "שבורים".

× לא יכולים להצביע על תיקיות – ראו שקופית הבאה.

× לא יכולים לקשר בין שתי מערכות קבצים שונות (הקיימות על שני דיסקים נפרדים, למשל).

מדוע?

קישורים רכים

✓ יכולים לקשר בין שתי מערכות קבצים שונות (הקיימות על שני דיסקים נפרדים למשל).

✓ יכולים להצביע על תיקיות.

× יכולים להיות "שבורים".

× עלולים ליצור מעגלים במערכת הקבצים.

הצבעה לתיקיות – מה הבעיות?

בלבול קשרי המשפחה

- אם קישורים יצביעו לתיקיות:

```
>> cd /A
>> ln /A loop
```

- אז לא ניתן להגדיר תיקיית אב (parent directory) אחת ויחידה לכל קובץ במערכת.
- בדוגמה מעלה: /A היא תיקיית האב של loop, אך כך גם /A/loop .

אינסוף מסלולים לאותו קובץ

- פעולות מסוימות עלולות להיקלע לרקורסיה אינ[illegible]פית, למשל הדפסת כל ה[illegible]בצים תחת התיקיה הנו[illegible]ת:

/A/loop
/A/loop/loop
/A/loop/loop/l[illegible]

הבעיה הזו עדיין קיימת בקישורים רכים.
לינוקס מונעת אותה באמצעות זיהוי מעגלים בגרף היררכיית הקבצים.

VERY SIMPLE FILE SYSTEM (VSFS)

Very Simple File System (VSFS)

- כעת נבנה בצורה הדרגתית מערכת קבצים פשוטה בשם VSFS.
- מערכת הקבצים הזו היא גרסה מפושטת של מערכת הקבצים המקורית של UNIX ולכן היא מציגה באופן פשוט את המושגים הבסיסיים של מערכות קבצים בלינוקס.

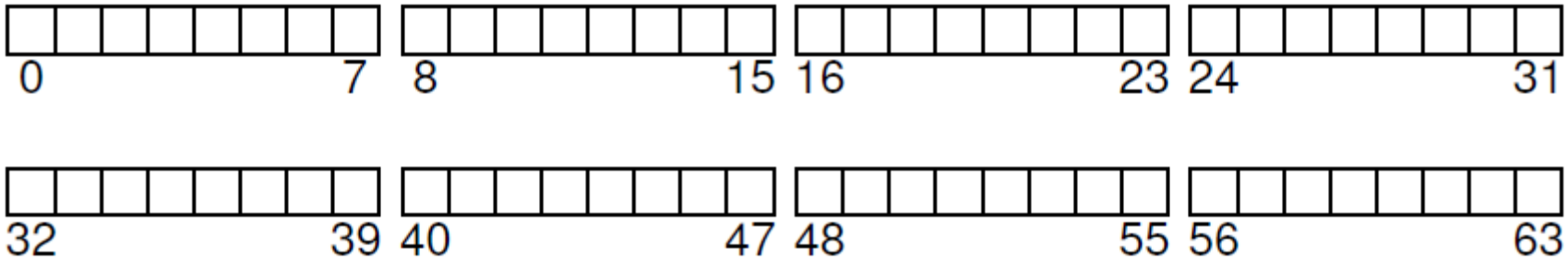
- מערכת הקבצים ממומשת בתוכנה בלבד, ללא תמיכת חומרה מיוחדת מעבר למה שראינו. נבחן את מערכת הקבצים לפי:

1. **מבני הנתונים** – איך מערכת הקבצים מארגנת את המידע על הדיסק?
2. **אופני הגישה** – איך קריאות המערכת open(), read(), write() פועלות על מבני הנתונים הללו?

שלב 1: חלוקה לבלוקים

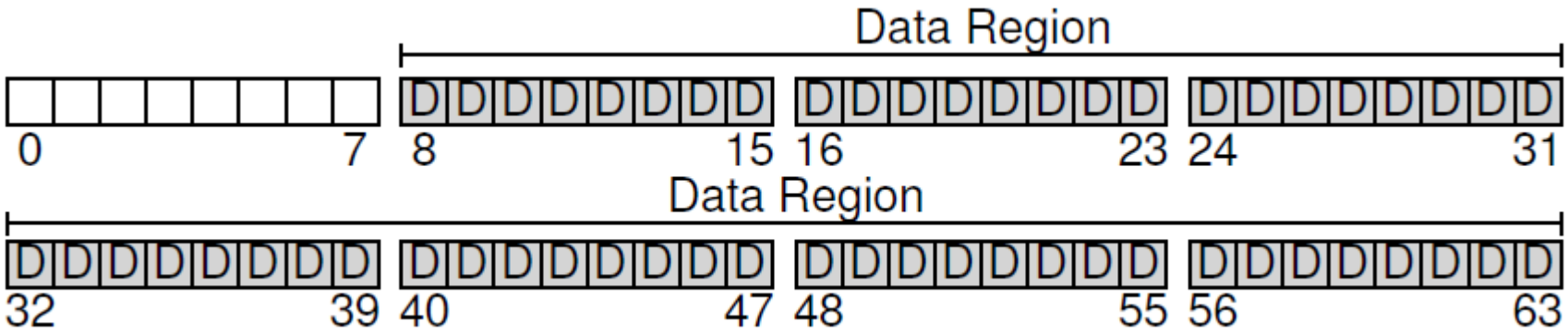
- כאמור, למרות שהדיסק פועל ביחידות של סקטורים (512 בתים), מערכת ההפעלה מנהלת קבצים ביחידות של בלוקים (4096 בתים).

- לצורך הדוגמה נניח כי הדיסק מכיל 64 בלוקים בלבד.
- ← הדיסק בגודל 256KB.



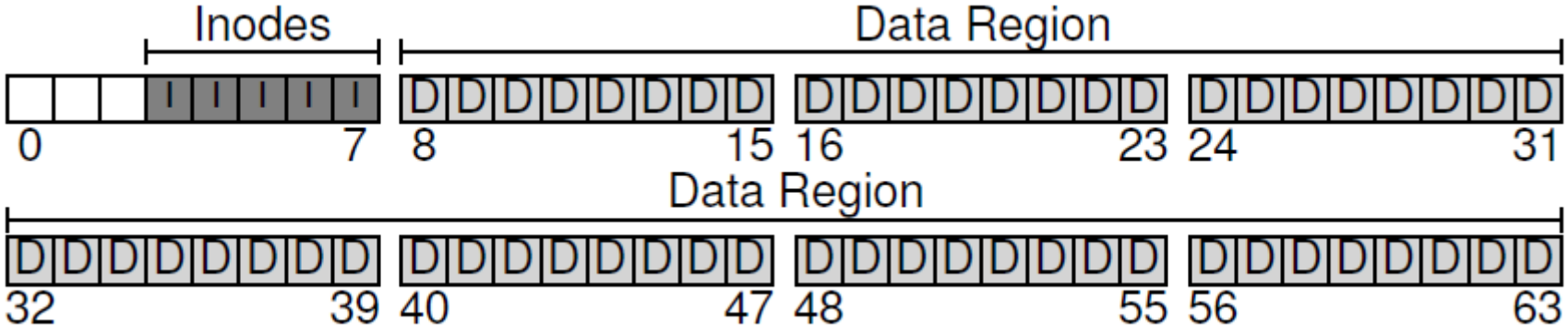
שלב 2: אזור הנתונים (data region)

- מה צריך לשמור במערכת הקבצים?
- הדבר הראשון הוא, כמובן, הנתונים של המשתמשים.
- במערכת הקבצים שלנו נניח כי האיזור המוקדש לנתונים של המשתמשים (data region) הוא 56 בלוקים.
  - כל מערכת קבצים תשאף כמובן להקדיש חלק גדול ככל הניתן מהדיסק לאיזור הנתונים.



שלב 3: טבלת ה-inodes

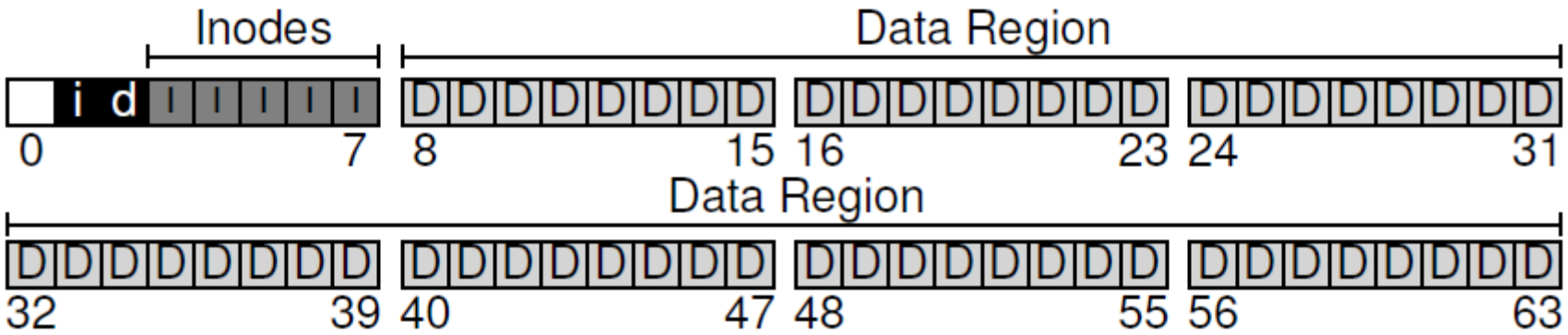
- כאמור, מערכת ההפעלה צריכה לשמור גם inode לכל קובץ.
  - ה-inode שומר אינפורמציה נוספת (metadata) על כל קובץ, למשל גודל הקובץ, הרשאות גישה, חותמות זמן, ועוד.
- כל ה-inodes נשמרים במערך רציף בגודל 5 בלוקים בדיסק.
- אם נניח כי גודל inode הוא 128 בתים, מה מספר הקבצים המקסימלי האפשרי במערכת הקבצים VSFS?



שלב 4: מערכי ביטים (bitmaps)

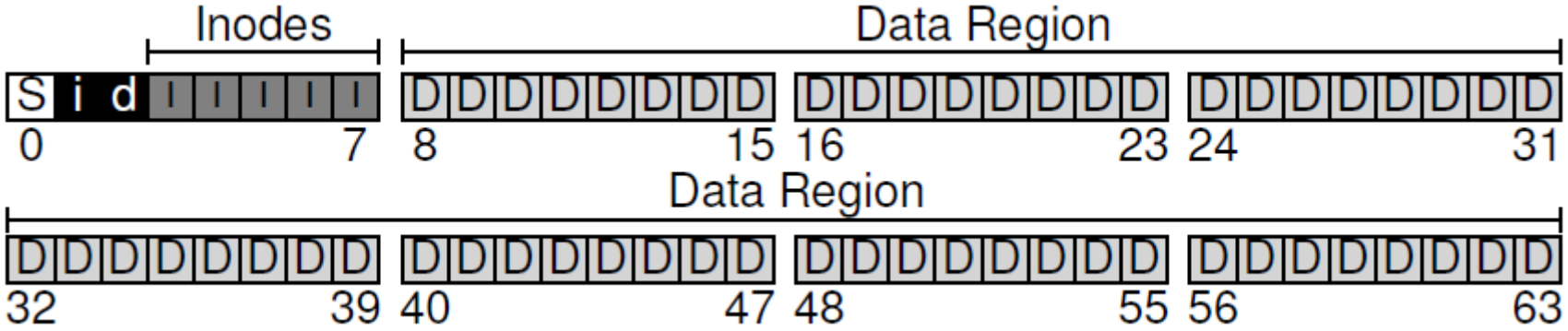
- מערכת ההפעלה צריכה כמובן לעקוב אחר הבלוקים הפנויים באיזור הנתונים ואחר ה-inodes הפנויים בטבלה.
- VSFS משתמשת בשני מערכי ביטים (bitmaps) פשוטים כדי לסמן האם האובייקט המתאים פנוי (0) או תפוס (1).
- לכל מערך ביטים נקדיש בלוק אחד בדיסק.

האם זה מספיק?



שלב 5: סופרבלוק (superblock)

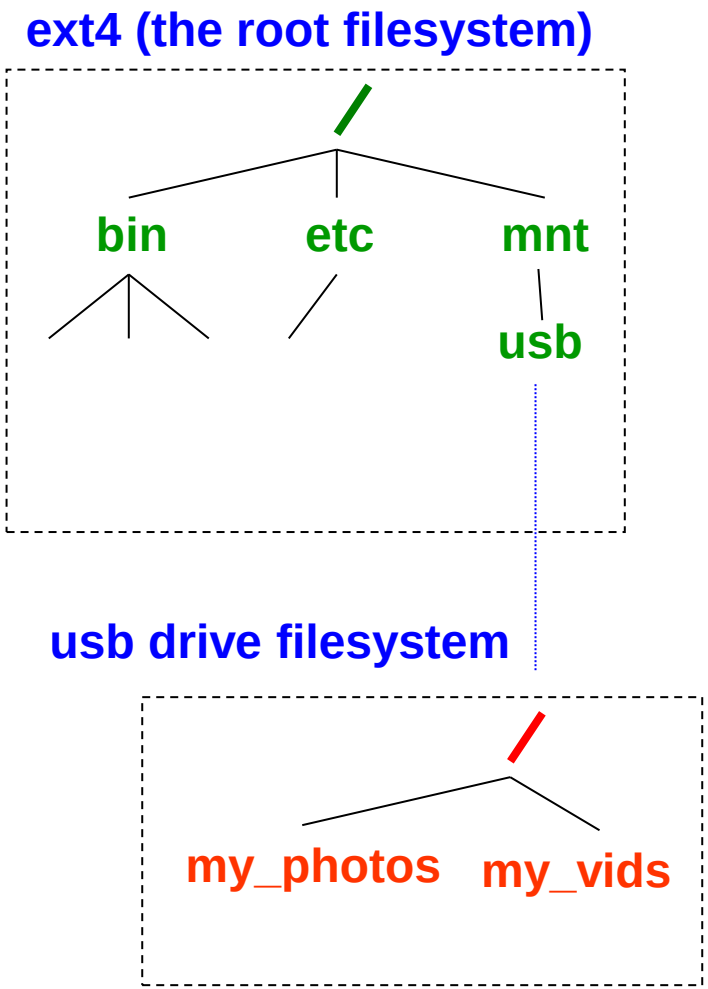
- הבלוק הראשון בדיסק ייקרא הסופרבלוק, והוא ישמור מידע כללי על מערכת הקבצים, למשל:
  - מה מספר הבלוקים שלה? מה מספר ה-inodes שלה?
  - היכן מתחילה טבלת ה-inodes?
  - היכן נמצא ה-inode של תיקיית השורש "/"?
- בעת הרכבה של מערכת הקבצים, מערכת ההפעלה תקרא את הסופרבלוק לזיכרון וכך תדע כיצד לגשת לקבצים.



הרכבה וניתוק של מערכות קבצים

- לינוקס מאפשרת להרכיב (mount) מערכות קבצים שונות **להיררכיה אחת בצורת עץ**.
- פעולת ההרכבה נעשית על-גבי תיקייה קיימת (לרוב ריקה) במערכת הקבצים הנקראת נקודת ההרכבה (mount point).
  - לאחר ההרכבה לא ניתן לגשת לקבצים ולתיקיות שהיו תחת תיקיה זו לפני ההרכבה.
  - לאחר הניתוק ניתן יהיה לגשת שוב לקבצים ולתיקיות שהיו תחת נקודת ההרכבה לפני ההרכבה.
  - אם הרכבנו כמה פעמים, אז בכל ניתוק נחזור למערכת הקבצים שהייתה לפני הניתוק.
- הרכבה וניתוק של מערכת קבצים נעשות באמצעות קריאות המערכת **mount()** ו-**unmount()** ודורשות הרשאות מתאימות.

דוגמה: הרכבה של מערכת קבצים



- שורש העץ הוא התיקייה "/" השייכת למערכת הקבצים ext4.
- נרצה להרכיב את מערכת הקבצים שיושבת בהתקן חיצוני מסוג disk-on-key.
- תיקיית ההרכבה /mnt/usb תשמש כתיקיית השורש של מערכת הקבצים החדשה.
- לאחר ההרכבה ניתן לגשת למידע של ההתקן בצורה פשוטה, למשל ע"י גישה ל- /mnt/usb/my_photos .

מציאת inode לפי מספרו

- בהינתן מספר inode ניתן לחשב בדיוק היכן הוא נמצא על הדיסק.
- ספציפית, הסקטור בו נמצא ה-inode מחושב באמצעות:

```
inodeAddr = inodeTableAddr + (inodeNumber x inodeSize)
            |-----in bytes-----|          |--in bytes--|

sector = inodeAddr / sectorSize
```

- לאחר שמערכת ההפעלה מצאה את ה-inode, יש בידיה את כל המידע כדי להמשיך ולקרוא את הקובץ.

ext2 inode

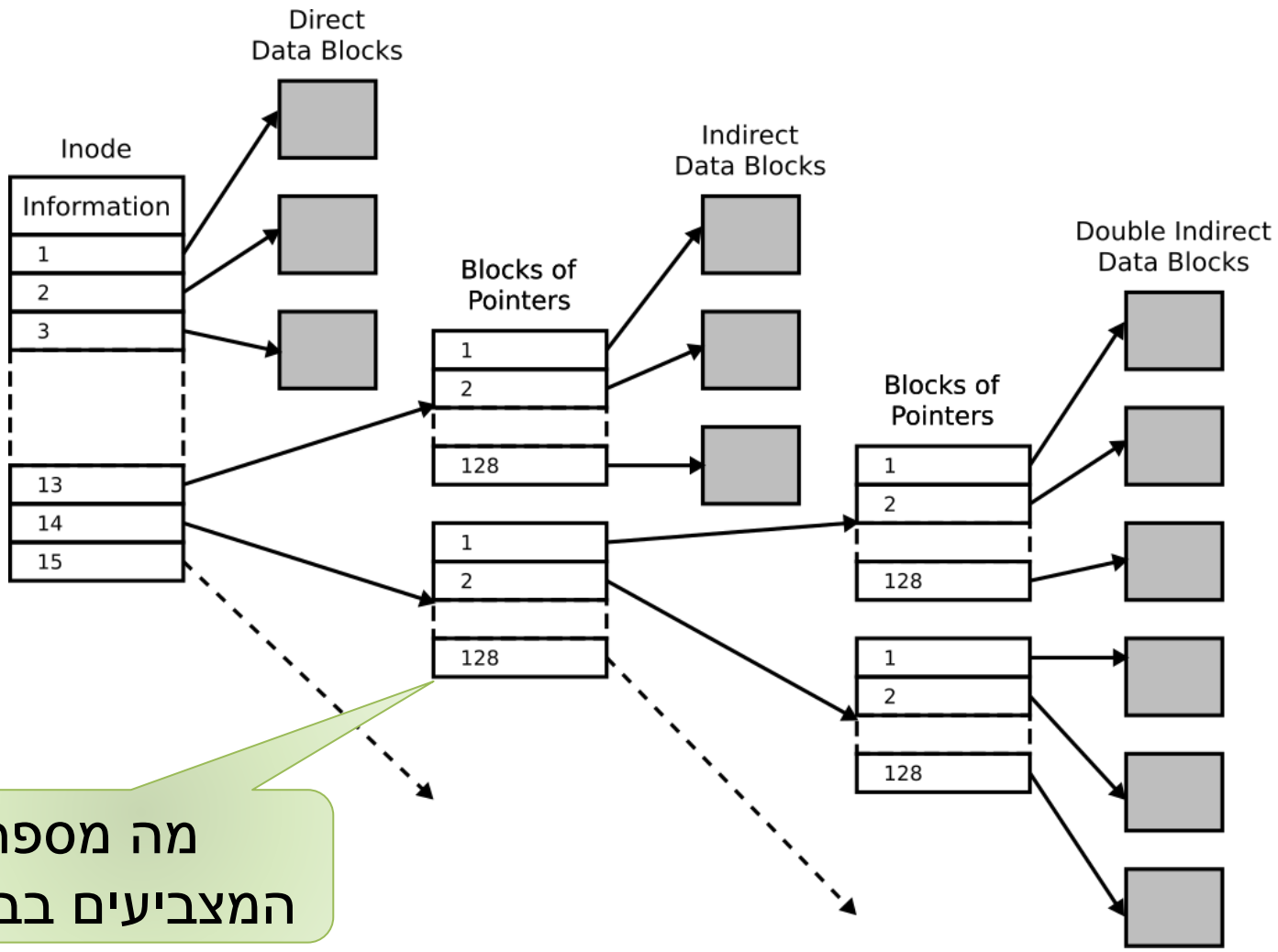
Size	Name	What is this inode field for?
2	mode	can this file be read/written/executed?
2	uid	who owns this file?
4	size	how many bytes are in this file?
4	time	what time was this file last accessed?
4	ctime	what time was this file created?
4	mtime	what time was this file last modified?
4	dtime	what time 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

איך להצביע לבלוקים המרכיבים את הקובץ?

- הדרך הפשוטה ביותר היא לשמור בתוך ה–inode את המצביעים.
  - מצביע לבלוק הוא בסך הכל מספר של בלוק בדיסק.

- דוגמה: ext2 inode שומר 12 מצביעים ישירים.
  מה המגבלה על גודל הקובץ שניתן לכסות עם מצביעים ישירים?
- כדי להתגבר על המגבלה הנ"ל, ניתן להוסיף מצביע לא ישיר – מצביע לבלוק שיכיל עוד מצביעים ישירים.
- וגם מצביע לא ישיר כפול – מצביע לבלוק שמכיל מצביעים לא ישירים.
- וגם מצביע לא ישיר משולש – הבנתם את הרעיון...
- את הבלוקים הנוספים של המצביעים שומרים באיזור הנתונים כי הוא האיזור הגדול ביותר.

מצביעים לבלוקים



מה מספר
המצביעים בבלוק?

אופני הגישה בקריאה מקובץ

- כעת נניח כי מערכת הקבצים VSFS מורכבת וכי הסופרבלוק כבר נמצא בזיכרון.
- שאר הנתונים (קבצים, תיקיות, inodes, ...) עדיין בדיסק.

- כמה פעמים ניגש הקוד הבא לדיסק?

```
int fd = open("/foo/bar", O_RDONLY);
read(fd, buffer, 4096);
read(fd, buffer, 4096);
read(fd, buffer, 4096);
```

מהלך פתיחת+קריאת קובץ

	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					
								read		
					write					
read()										
					read					
									read	
					write					
read()										
					read					
										read
					write					

שלבי קריאת המערכת open()

1. קריאת המערכת open() צריכה למצוא את ה-inode של הקובץ bar כדי לבדוק הרשאות גישה. חיפוש הקובץ מתחיל בתיקיית השורש, אשר ה-inode שלה נמצא במקום ידוע בדיסק.
2. ה-inode של תיקיית השורש מצביע לבלוקים המרכיבים אותה. מערכת ההפעלה תקרא את הבלוקים האלה ותחפש כניסה בשם foo - הכניסה הזאת מצביעה ל-inode של התיקיה foo.
3. מערכת ההפעלה תקרא את ה-inode של foo כדי לבדוק את הרשאות הגישה לתיקיה הזו.
4. מערכת ההפעלה תקרא את הבלוקים המרכיבים את התיקיה foo ותחפש בהם כניסה בשם bar.
5. השלב האחרון של open() יהיה לקרוא את ה-inode של הקובץ bar ולוודא הרשאות גישה אליו.

שלבי קריאת המערכת read()

1. קריאת המערכת read() תיגש קודם ל-inode של הקובץ bar כדי למצוא את הבלוקים המרכיבים אותו.
2. לאחר מכן מערכת ההפעלה תקרא את הבלוק המבוקש מהדיסק.
3. לבסוף מערכת ההפעלה תכתוב ל-inode כדי לעדכן את חותמת הזמן של הגישה האחרונה לקובץ.

אופני הגישה בכתיבה לקובץ

- שוב נניח כי מערכת הקבצים VSFS מורכבת וכי הסופרבלוק כבר נמצא בזיכרון.
- שאר הנתונים (קבצים, תיקיות, inodes, ...) עדיין בדיסק.

- כמה פעמים ניגש הקוד הבא לדיסק?

```
int fd = open("/foo/bar",
     O_CREAT | O_WRONLY | O_TRUNC);
write(fd, buffer, 4096);
write(fd, buffer, 4096);
```

מהלך יצירת+כתיבת קובץ

	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							
						read				
				read						
							read			
		read								
		write								
							write			
					read					
					write					
				write						
write()					read					
	read									
	write									
								write		
					write					
write()					read					
	read									
	write									
									write	
					write					

למה קריאה
ואז כתיבה?

שלבי יצירת קובץ חדש

יצירת קובץ חדש דומה לפתיחת קובץ קיים בתוספת השלבים הבאים:

1. קריאת ה-inode bitmap כדי למצוא inode פנוי לקובץ החדש.
2. כתיבה חזרה ל-inode bitmap כדי לסמן את ה-inode התפוס.
3. איתחול ה-inode של הקובץ החדש באמצעות קריאה+כתיבה: ה-inode קטן יותר מסקטור שלם (128 בתים לעומת 512 בתים) ולכן כדי לאתחל אותו צריך לקרוא את כל הסקטור המכיל את ה-inode ואז לכתוב אותו חזרה לדיסק.
4. כתיבה לבלוקים המכילים את התיקיה foo כדי להוסיף את הכניסה המתאימה לקובץ החדש bar.
5. כתיבה ל-inode של התיקיה foo כדי לעדכן את חותמות הזמן שלה.

שלבי קריאת המערכת write()

- מערכת ההפעלה צריכה כמובן לכתוב את הבלוק המבוקש לדיסק.

אבל אם צריך להקצות בלוק חדש לקובץ אז גם:

1. קריאת ה-data bitmap כדי למצוא בלוק פנוי.
2. כתיבה חזרה ל-data bitmap כדי לסמן את הבלוק התפוס.
3. קריאה + כתיבה ל-inode המתאים כדי לעדכן את המיקום של הבלוק החדש שהוקצה.

inode cache

- ראינו כי קריאות המערכת הנפוצות על קבצים דורשות גישות רבות לדיסק כדי לעדכן את מבני הנתונים.
  - לדוגמה: יצירת קובץ חדש ניגשת 10 פעמים לדיסק.

- כדי לצמצם את כמות הגישות לדיסק, לינוקס שומרת מספר מטמונים:
- מטמון הדפים (page cache) – עבור המידע של איזור הנתונים.
- מטמון inodes – עבור המידע של טבלת ה-inodes.
- מטמונים נוספים עבור ה-bitmaps או מבני נתונים אחרים של מערכת הקבצים.

FILE ALLOCATION TABLE (FAT)

משפחת FAT

- FAT הוא שם כולל למשפחה של מערכות קבצים הקרויות על שם מבנה הנתונים המרכזי בו הן משתמשות: FAT = file allocation table.

- FAT פותחה ע"י חברת מיקרוסופט בשביל מערכת ההפעלה DOS.
- בגלל הפשטות היחסית שלה, היא אומצה במהרה גם ע"י כוננים חיצוניים (floppy disks, CD-ROM).
- FAT עדיין נמצאת בשימוש נרחב בהתקני פלאש (disk-on-key).

- אנחנו נלמד עליה בתור דוגמה נוספת למימוש של מערכת קבצים.

מבנה הדיסק

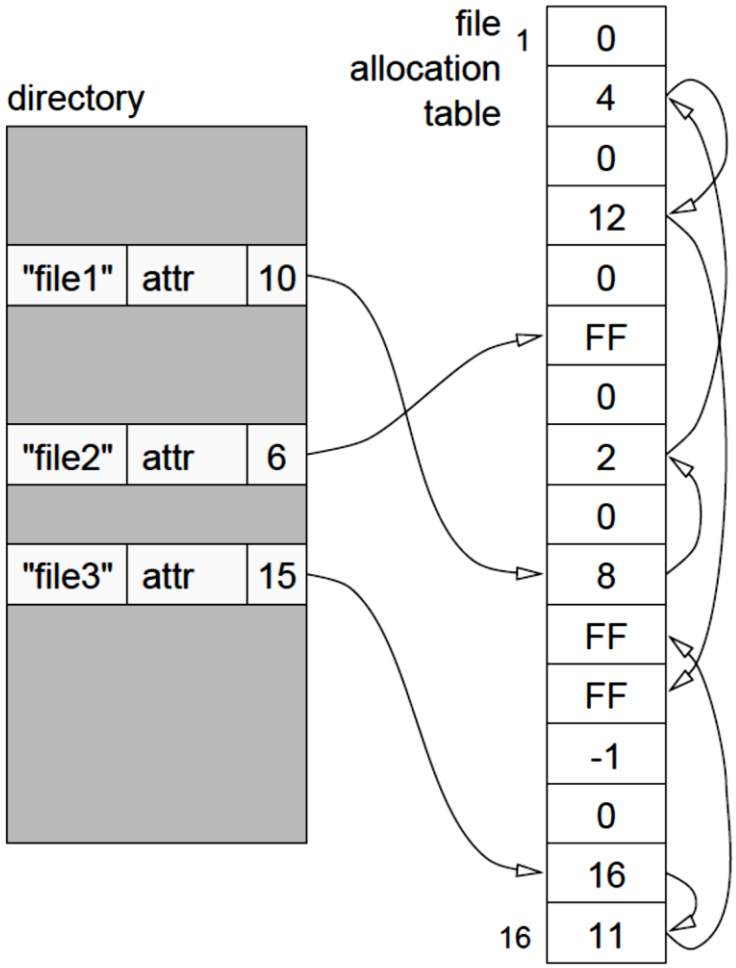
- VSFS שומרת מצביעים לבלוקים של כל קובץ בתוך ה-inode שלו.

- לעומת זאת, מערכת הקבצים FAT שומרת את הבלוקים בצורה של רשימה מקושרת.
- כל הרשימות המקושרת נשמרות בטבלת FAT אשר ממוקמת בתחילת הדיסק. מבנה הדיסק הוא (בערך):

superblock	FAT	data region

- FAT היא טבלה גלובלית לכל הקבצים אשר "מדביקה" את הבלוקים של כל קובץ ליחידה אחת.

File Allocation Table (FAT)



- FAT שומרת כניסה לכל בלוק בדיסק.
- בלוקים ששייכים לקובץ מסוים מצביעים על הבלוק הבא של אותו קובץ.
- FF מסמן שזהו הבלוק האחרון ברשימה (כלומר הבלוק האחרון בקובץ).
- 0 מסמן שזהו בלוק פנוי.
- 1- מסמן שזהו בלוק פגום (כדי לא להשתמש בו).

תיקיות ב-FAT

- תיקיות ב-FAT הם קבצים רגילים אשר מכילים רשומות מסוג:

filename	metadata	starting block

- כל רשומה מכילה את ה-metadata על הקובץ.
- המשמעות: ה-inode כאילו מוטמע בתוך הרשומה.

- לכן מערכות קבצים מסוג FAT לא תומכות בקישורים קשים.

שאלה ממבחן D:

מועד א', סמסטר חורף תש"פ (2019—2020)

נתוני השאלה

- במערכת קבצים כלשהי, שדומה למערכת הקבצים הקלאסית של UNIX, השדה arr ב-inode מכיל מערך של מצביעים:
- inode.arr[0..11] are direct pointers,
- inode.arr[12] is an indirect pointer,
- inode.arr[13] is a double indirect pointer,
- inode.arr[14] is a triple indirect pointer.

- הניחו שה-inode של מערכת הקבצים הינו בגודל 512B (תמיד מיושר לכפולה של 512B).
- כמו כן, הניחו שמערכת הקבצים מורכבת (mounted) על דיסק בגודל 1TB בעל גודל בלוק של 8KB.

סעיף א'

- נסמן ב-N את מספר המצביעים שבלוק מסוג indirect יכול להכיל. מהו ערך ה-N המקסימלי?

- פתרונות לא נכונים:
- להניח את גודל מצביע, למשל להניח כי מצביע הוא 8 בתים בגלל שהארכיטקטורה של המעבד היא 64 ביט.
  - אין קשר בין מערכת הקבצים למעבד. מערכת הקבצים היא יצור תוכנתי בלבד.
- למצוא את N מתוך המשוואה כי גודל הקובץ המקסימלי הוא 1TB:

$$(12+N+N^2+N^3)*8KB=1TB$$

  - אין קשר בין גודל הקובץ המקסימלי לבין נפח הדיסק.
    לדוגמה: קובץ מסוים יכול להיות גדול יותר מנפח הדיסק אם לקובץ יש מספר מצביעים לאותו בלוק בדיסק (למשל אם הקובץ מורכב מהרבה בלוקים זהים).

פתרון סעיף א'

- מספר הבלוקים בדיסק הוא:

1TB / 8KB = 2^40 / 2^13 = 2^27

- ← דרושים 27 ביטים לקידוד אינדקס של בלוק.

מספר המצביעים בבלוק של indirect pointers הוא אם כן:

8KB / 27bit = 2427

  - שימו לב להבדל בין ביט לבית.

- שימו לב: השאלה ביקשה במפורש את N המקסימלי, ולכן הנחנו כי כל נפח הדיסק מוקדש לבלוקים של נתונים. גם בהנחה מציאותית יותר שחלק מהדיסק (לדוגמה 10%) מוקדש לסופרבלוק, inodes , וכן הלאה – עדיין דרושים 27 ביטים לקידוד אינדקס של בלוק, ולכן התשובה נותרת ללא שינוי.

עוד תשובות נכונות

- ניקוד מלא ניתן לסטודנטים שעיגלו את רוחב המצביע (27 ביט) לחזקה שלמה של 2 (כלומר 32 ביט).

- ניקוד מלא ניתן גם לסטודנטים שהוסיפו ביט valid ליד כל מצביע וקיבלו לכן:

$$8KB / 28bit = 2340$$

  - בפועל, אין צורך בביט valid כי ניתן לקודד invalid pointer בעזרת הצבעה לבלוק 0 (שהרי הבלוק הראשון בדיסק מאכלס את הסופרבלוק).

הסוף!

או שאולי זו רק ההתחלה...