



TEL AVIV UNIVERSITY

# מערכות הפעלה

מרצה: ערן טרומר  
סמסטר א' תשע"ב

## הרצאה 8

# מערכות קבצים

## שני מנגנונים, שם אחד

❖ מיפוי ממרחב שמות מדרגי קריא לבני אדם (מחרוזות) לעצמים של מערכת ההפעלה:

▪ `/usr/bin/ls`

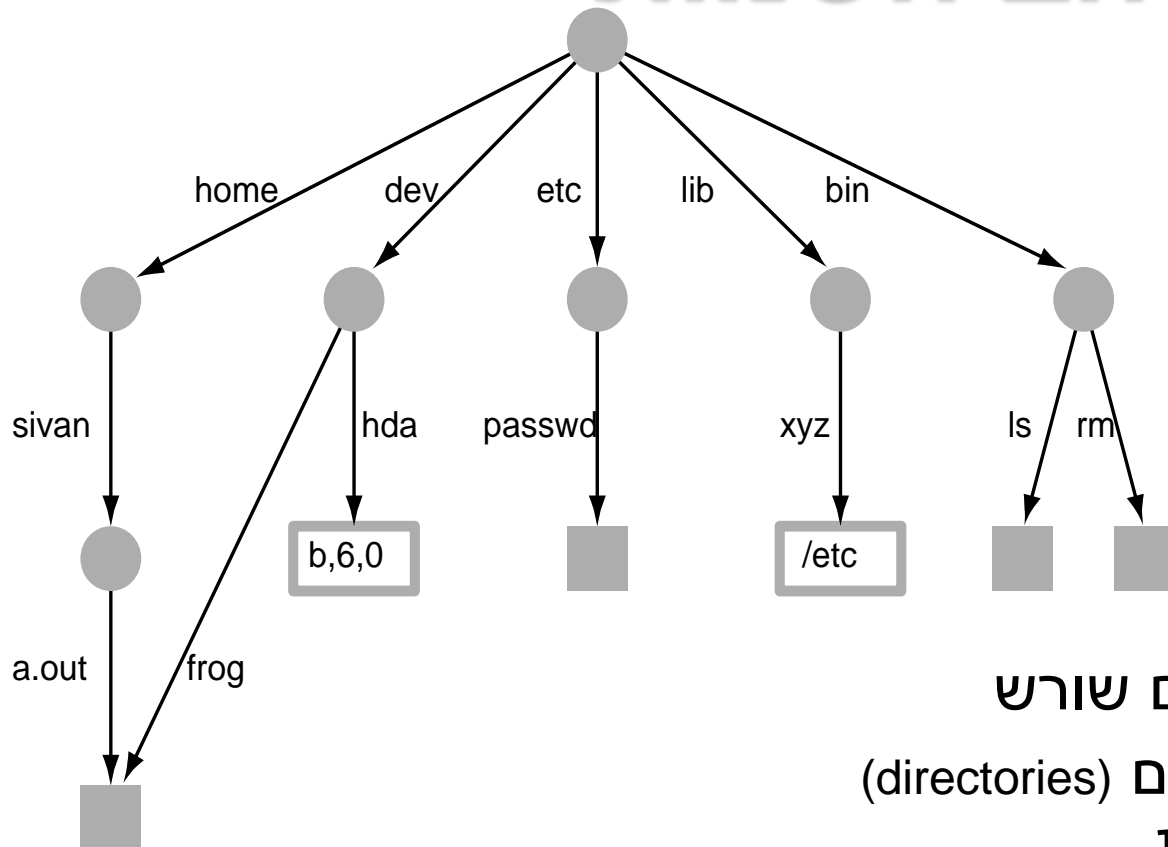
▪ `/dev/lp0`

▪ `C:\winnt\system32\kernel32.dll`

❖ מנגנון לשמירת ואחזור קבצים על התקני זיכרון חיצוניים (דיסקים, תקליטורים, זכרון הבזק וכו')

▪ במערכות הפעלה מסוימות (מקינטוש, חלונות NT/2000) קובץ יכול להיות מורכב ממספר רצפים; אנו נתעלם מאפשרות זו, שמייצגת בעיקרון סוג שונה מעט של מדריך.

# מרחב השמות



❖ עץ או גרף חסר מעגלים עם שורש

❖ צמתים פנימיים הם מדריכים (directories)

= תיקיות (folders) = ספריות.

❖ העלים הם קבצים או עצמים אחרים (התקנים, צינורות)

❖ מצביע סימבולי הוא עלה מסוג מיוחד; נדון בו בהמשך

❖ ביוניקס/לינוקס לקשתות יש שמות

❖ התייחסות לעצמים על ידי שרשור שמות הקשתות לאורך המסלול מהשורש

# מרחב השמות בחלונות 2000/NT

- ❖ לעצמים עצמם יש שמות, לא לקשתות
- ❖ לעצם יכולים להיות מספר שמות נרדפים (למשל למימוש שמות של 8.3 תווים לקבצים)
- ❖ מרחב השמות כולל גם עצמים ששמורים במערכת הקבצים (כלומר בדיסק) וגם עצמים נדיפים כמו מנעולים ואירועים
- ❖ שמות קבצים במרחב השמות האחד הזה מתפענחים כאילו התחילו ב- "??" \ " ועצמים נדיפים כאילו התחילו ב-  
"\BaseNamedObjects"

# פעולות על מרחב השמות

- ❖ יצירת עלה מסוג מסוים ומצביע אליו ממדריך כלשהו  
(creat, open, mknod, socket)
- ❖ יצירת מדריך ומצביע אליו (mkdir)
- ❖ יצירת מצביע נוסף לעצם קיים (link)
- ❖ יצירת מצביע סימבולי (symlink)
- ❖ מחיקת מצביע; העצם נמחק כאשר אין מסלול מהשורש אליו ואינו בשימוש על ידי תהליך (unlink)
- ❖ שינוי הרשאות גישה לעצם (chown, chmod)
- ❖ שתילת ועקירת מערכת קבצים שלמה בצומת (mount, unmount)
- ❖ פענוח שם למזהה פנימי והודעה למערכת ההפעלה שהעצם בשימוש (open), והודעה על סיום שימוש בעצם (close)
- ❖ קריאת תוכן מדריך או חיפוש במדריך (opendir, readdir)

# פענוח שמות (open)

❖ פירוק שם למרכיבים שמופרדים ב- \ או ב- / ,

❖ למשל, `/usr/bin/ls`

❖ מעקב אחרי המסלול במרחב השמות שמוביל לעצם

❖ בכל שלב בפענוח מערכת ההפעלה מחפשת את המרכיב

הנוכחי של השם במדריך

❖ לכל תהליך נשמר מדריך נוכחי (current working directory)

שהוא נקודת ההתחלה למסלולים שאינם מתחילים בשורש.

❖ מצביעים מיוחדים: `.` ו-`..`

# מצביעים סימבוליים

- ❖ עלים במרחב השמות שמכילים שם של עצם אחר
- ❖ כאשר מנגנון הפענוח מגיע למצביע סימבולי, הערך של המצביע משורשר לסיומת השם המתפענח, והפענוח מתחיל מחדש:
  - למשל, `/lib/xyz` הוא מצביע סימבולי ל- `etc`
  - כאשר מגיעים למצביע בפענוח של `/lib/xyz/passwd`, משרשרים את תוכן המצביע לסיומת `passwd`
  - הפענוח מתחיל מחדש עם השם המשורשר `etc/passwd`
- ❖ ניתן ליצור מצביע סימבולי גם אם העצם המוצבע לא קיים, וניתן למחוק את העצם המוצבע

# נקודות הצבה

- ❖ מצביע מצומת במרחב השמות לשורש של מרחב שמות אחר
- ❖ מאפשר "לשתול" מערכת קבצים מדיסק, מחיצה, או מחשב מרוחק במרחב השמות
- ❖ כאשר תהליך הפענוח מגיע לנקודת הצבה, התהליך ממשיך עם הסיומת של השם מהשורש של המערכת המוצבת.
- ❖ ביוניקס/לינוקס: טבלה נדיפה במערכת ההפעלה
- ❖ בחלונות 2000: עצם ששמור בדיסק (נקודת פענוח)
- ❖ נקודת פענוח יכולה גם לגרום להפעלת שגרה שממשיכה את תהליך הפענוח



# מחיקת קבצים ומעגלים

- ❖ אם מערכת ההפעלה הייתה מתירה ליצור מעגלים בגרף, קשה היה לאפיין ביעילות עצמים שלא ניתן להגיע אליהם מהשורש
- ❖ ניתן לסמן את כל העצמים שניתן להגיע אליהם ולמחוק את השאר, אז זהו תהליך יקר
- ❖ מערכות הפעלה מונעות יצירת מעגלים ומוחקות עצמים שמספר ההצבעות אליהם יורד ל-0; מכיון שאין מעגלים ואין עצמים שאין אליהם הצבעות, ניתן להגיע לכל עצם שיש אליו הצבעות
  - חשוב לספור גם "הצבעות" לקובץ מתהליכים רצים
- ❖ דרך פשוטה למנוע היוצרות מעגלים: איסור על יותר ממצביע אחד למדריכים
- ❖ תוכניות לטיפול בקבצים צריכות להיזהר ממעגלים שכוללים מצביעים סימבוליים (למשל תוכניות גיבוי או יצירת אינדקסים)

**שמירת ואחזור קבצים**

# סמנטיקה של גישה לקבצים

❖ עקביות (consistency) בגישה ממספר תהליכים:

- כתיבות וקריאות הן אטומיות, כלומר לא רואים מידע חלקי
- כתיבה לקובץ נראית מייד לתהליכים אחרים

עקביות סדרתית (sequential consistency): ניתן לסדר את כל הכתיבות והקריאות בסדר שלם וקריאה תמיד רואה את הקובץ לאחר הכתיבה האחרונה

❖ עמידות (durability) למרות נפילות:

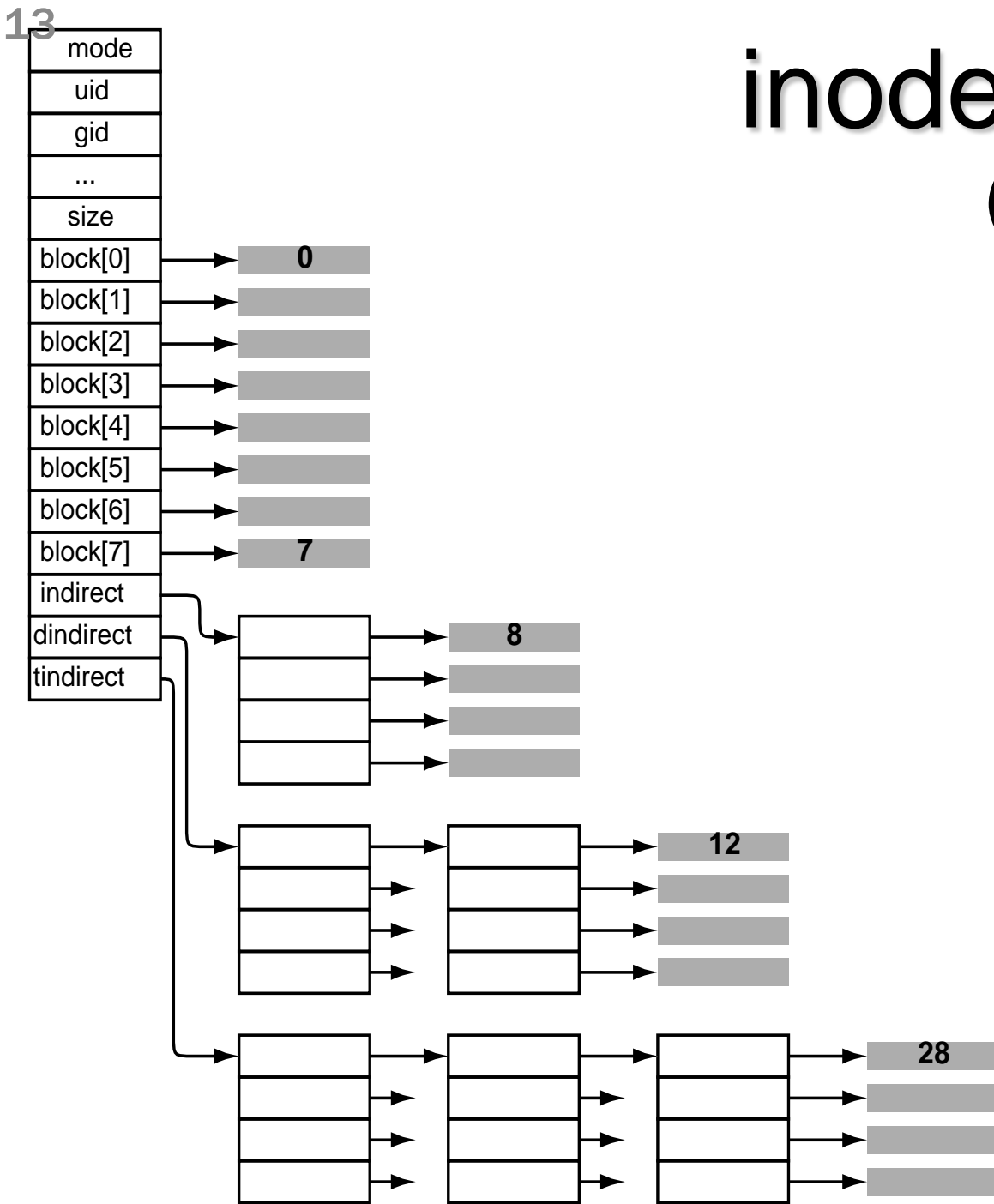
- כתיבה רגילה אינה מבטיחה עמידות אם המחשב ייפול
- סגירה של קובץ מבטיחה (לפעמים) עמידות לכתיבות של התהליך
- ניתן להבטיח עמידות על ידי פתיחת הקובץ בצורה מיוחדת או על ידי קריאות מערכת מיוחדות
- הבטחת עמידות פוגעת בביצועים (יותר כתיבות ולפי סדר שרירותי)

# מיפוי קבצים להתקן האחסון

- ❖ מערכת ההפעלה מחלקת קבצים לבלוקים בגודל קבוע ושומרת אותם בדיסק, לא בהכרח ברצף
- ❖ מנגנון המיפוי מוצא את המיקום הפיזי בדיסק של בלוק מסוים ברצף הנתונים של הקובץ (היכן הבלוק ה-17 של הקובץ?)
- ❖ התעקשות על הקצאה ברצף בלוקים פיזי מפשטת את המיפוי
  - אבל גורמת לפיצול חיצוני (external fragmentation) – הרבה חורים, קטנים מדי לקובץ גדול),
  - לא מאפשרת הגדלה של קובץ ללא הזזה אם אין אחריו חור
- ❖ מיפוי של בלוקים גדולים מקטין את התקורה של המיפוי ואת גודל מבנה הנתונים, ומשפר את ביצועי הדיסק
  - אבל בלוקים גדולים מבזבזים יותר מקום בבלוק האחרון של כל קובץ (פיצול פנימי – internal fragmentation)
  - לפעמים אפשר לדחוס כמה זנבות לקובץ אחד
- ❖ מדריכים מאוחסנים באופן דומה, כרשימה של (מספר inode, שם)

# מנגנון מיפוי: inodes

(מנגנון יוניקס מסורתי)



# מיפוי קבצים ב-NTFS

- ❖ קובץ מיוצג על ידי רשומה בקובץ master file table (MFT)
- ❖ רשומה מכילה את
  - תחילת רצף הנתונים (קובץ קטן ישמר בתוך הרשומה)
  - נתונים אודות הקובץ: שמו או שמותיו, הרשאות גישה, זמן יצירה, ...
  - מערך של מצביעים לבלוקים של הקובץ בדיסק
- ❖ אם אין מקום במערך המצביעים לכל הבלוקים משתמשים ברשומת המשך

# מיפוי קבצים ב-FAT

- ❖ בדיסק (מחיצה) שמורה טבלת הקצאה אחת של מצביעים שגודלה כמספר הבלוקים בדיסק
- 0: קובץ מיוצג על ידי מספר הבלוק הראשון שלו
- ❖ בכל איבר בטבלת ההקצאה שמור מצביע לבלוק הבא של הקובץ
- 23: 47
- ❖ הבלוקים הפנויים משורשרים לרשימה גם הם
- ❖ מיפוי בקובץ ארוך דורש זמן פרופורציוני למספר הבלוקים בקובץ (מעבר על רשימה מקושרת)
- ❖ הקובץ בדוגמה שמור בבלוקים 47, 23, 54

ex.doc → 54

0:

1:

...

23: 47

...

47: -1

...

54: 23

# מדיניות הקצאת בלוקים לקבצים

- ❖ איזה בלוקים כדאי להקצות לקובץ?
- ❖ רצוי למפות רצפים ארוכים בקובץ לרצפים פיזיים ארוכים משום שיישומים קוראים לעיתים קרובות קבצים שלמים ברצף או חלקים גדולים שלהם, וקריאת רצף בלוקים פיזי יעילה יותר
- ❖ רצוי להקצות קבצים שמשתמשים בהם ביחד קרוב בדיסק על מנת למזער את הצורך בהזזת זרוע הדיסק, למשל קבצים מאותה ספרייה
- ❖ מערכות קבצים כמו FAT ששומרות את הבלוקים הפנויים ברשימה מקצות מתחילת הרשימה, ובמשך הזמן קבצים מקבלים בלוקים אקראיים פחות או יותר ← ביצועים גרועים



# הקצאה חכמה ב-FFS

❖ Fast File System – BSD Unix

❖ הקצאה חכמה טיפוסית: רצפים פיזיים ארוכים וקבצים מאותה ספריה קרובים בדיסק

❖ הדיסק מחולק לקבוצות גלילים עוקבים; בכל קבוצה שומרים

- את הפרמטרים של מבנה הדיסק ומבנה מערכת הקבצים, לשרידות
- מאגר של inodes כדי שה-inodes ישמרו קרוב לקבצים שלהם
- מערך סיביות לייצוג הבלוקים הפנויים בקבוצת הגלילים
- בלוקים של נתונים

❖ מדיניות הקצאה דו-שלבית:

- בחירת קבוצת גלילים שבה יוקצה הבלוק הדרוש לקובץ
- בחירת הבלוק הספיציפי בקבוצה

# FFS: פרטי ההקצאה

## ❖ בחירת קבוצת גלילים:

- אם הקובץ קיים ובקבוצה שבו הוא מוקצה יש מקום, והקובץ תופס פחות מרבע ממנה, נקצה באותה קבוצה, אחרת נמצא קבוצה אחרת עם תפוסה נמוכה
- אם הקובץ חדש, נקצה בקבוצה שמכילה את המדריך שבו הקובץ

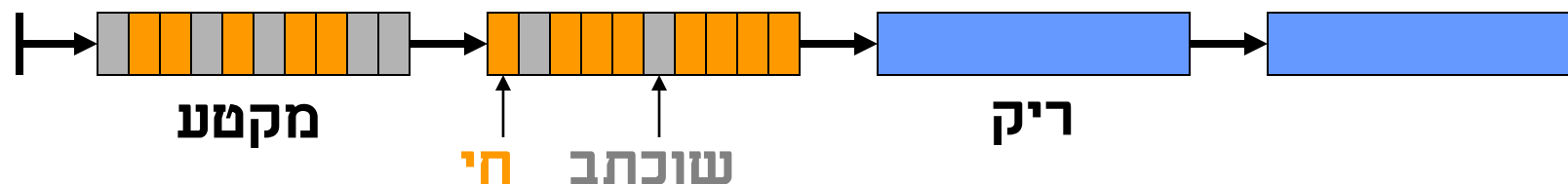
## ❖ בחירת בלוק:

- הבלוק הקרוב ביותר מבחינה סיבובית של הדיסק לבלוק האחרון בקובץ, אם הוא פנוי, אחרת משתמשים במדיניות משנית
- ❖ מערכת הקבצים מנסה למפות אשכולות של 64 KB של נתונים לבלוקים רצופים פיזית על מנת למרב את קצב ההעברה

# סיכום FFS

- ❖ ביצועים מצוינים בסביבות שבהן יש הרבה העברת נתונים מכיון שהזרוע זזה מעט (הקצאה בקבוצות גלילים) ומכיון שלא ממתינים הרבה לסיבוב הדיסק (הקצאה מיטבית של בלוקים והעברה של אשכולות שלמים
- ❖ ביצועים פחות טובים בסביבות שבהן יוצרים ומוחקים קבצים בתכיפות: להקצאת בלוקים אין השפעה אבל תכונות אחרות של FFS מגבילות את הביצועים

# גישה אחרת: LFS



Log-structured File System ❖

❖ הדיסק מחולק למקטעים בגודל קבוע (כמיליון בתים)  
שמשורשרים לרשימה

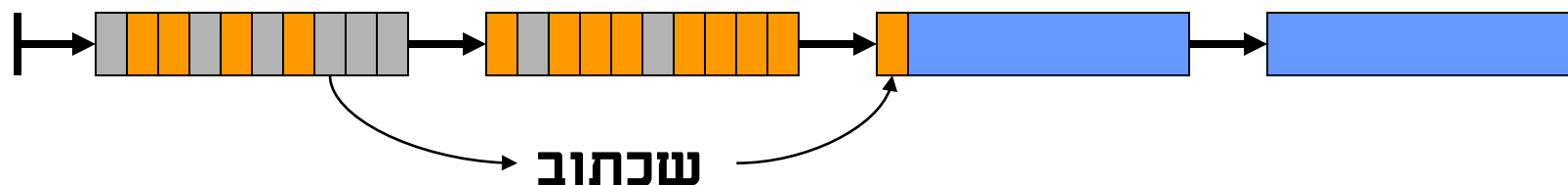
❖ נתונים נכתבים על מקטעים כאילו הרשימה הייתה מגילה  
אינסופית; נתונים נכתבים ומשוכתבים רק בסוף הרשימה

❖ תחילת המגילה מכילה בלוקים "חיים" (חלק מקבצים) ובלוקים  
של קבצים שנמחקו ובלוקים ששוכתבו

❖ סוף הרשימה ריק

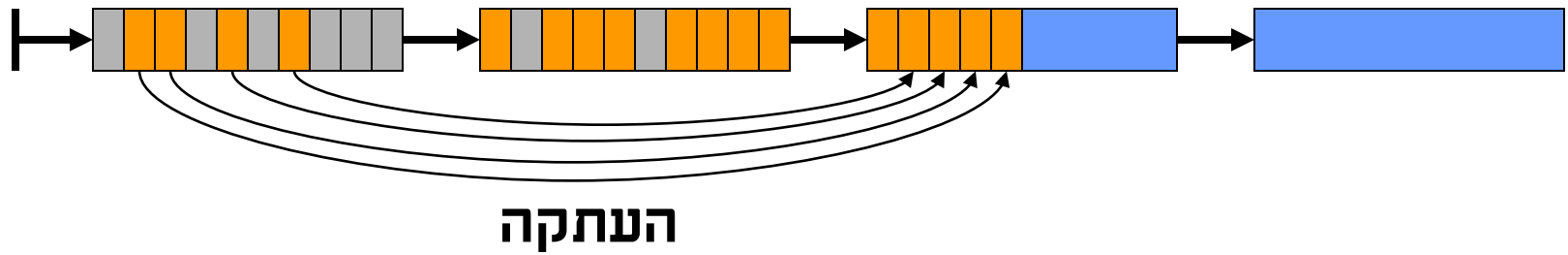
❖ תהליך מיוחד דואג שתמיד יהיה מקטעים ריקים בסוף הרשימה

# שכתוב בלוק ב-LFS

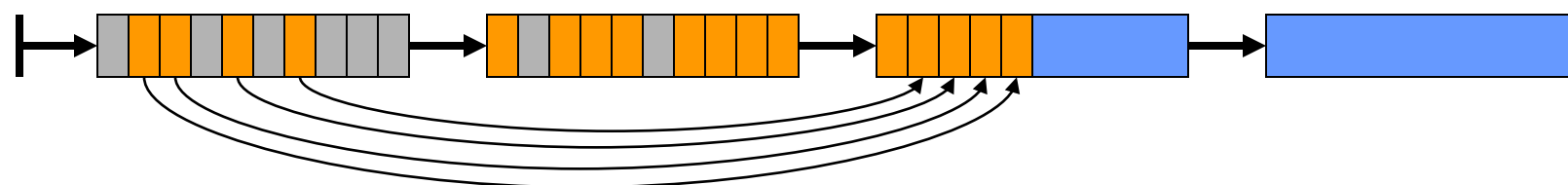


- ❖ בלוקים משוכתבים לסוף המגילה
- ❖ כאשר משכתבים בלוק, צריך לשכתב גם את ה-inode או בלוק המצביעים שמצביע עליו, משום שמקומו השתנה
- ❖ שרשרת השכתובים נעצרת ב-inode משום שמדריכים מצביעים ל-inodes לוגיים שטבלה ממפה למיקום בדיסק
- ❖ בלוקים אינם נכתבים אחד אחד אלא בקבוצות, כאשר מצטבר מקטע שלם או כאשר חייבים לכתוב בלוק לקובץ

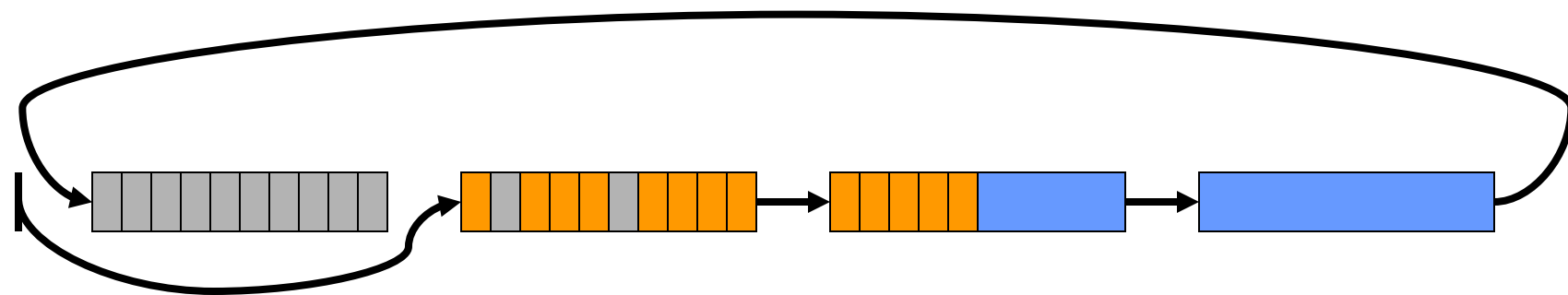
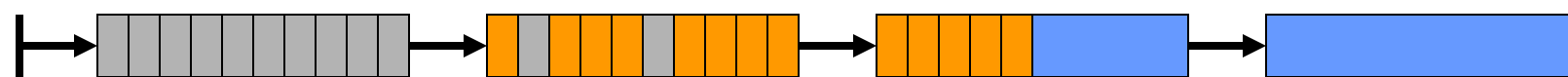
# ניקוי מקטעים ב-LFS



# ניקוי מקטעים ב-LFS



העתקה



❖ זיהוי בלוקים חיים לעומת משוכתבים: כל מקטע מתאר את הזרות (קובץ+מספר בלוק) של כל בלוק בו; ממפים את הבלוק ומשווים

# סיכום LFS

- ❖ כתיבות יעילות מאוד:
  - תמיד לבלוקים רצופים בדיסק
  - בדרך כלל מקטעים שלמים או חלקים גדולים שלהם
- ❖ קריאות מבלוקים שרירותיים בדיסק, אבל
- ❖ אם כתיבה ביחד מנבאת קריאה ביחד, אז קריאות מאזורים קרובים בדיסק
- ❖ ביצועים מצוינים כאשר ביצועי כתיבה חשובים יותר מביצועי קריאה או כאשר יש יותר כתיבות מקריאות (קבצי לוג, למשל)
- ❖ ביצועים גרועים במערכות מרובות משתמשים: כתיבות בו-זמניות לא מנבאות דבר לגבי קריאות



# נפילות והתאוששות

- ❖ השגרות של מערכת הקבצים מניחות הנחות מסוימות אודות מבנה הנתונים על הדיסק; נכונות השגרות תלויה בקיום ההנחות
- ❖ נפילה פתאומית עלולה להשאיר את המערכת במצב לא תקין
- ❖ הנחות שחייבות להתקיים (הפרה היא חוסר עקביות חמור):
  - בלוק שהוא חלק מקובץ אינו מסומן כפנוי
  - בלוק אינו ממופה כחלק משני קבצים או יותר
  - אין מצביע במדריך ל-inode שמסומן כפנוי
  - ...
- ❖ הנחות פחות חשובות שניתן לתקן בזמן שהמערכת פועלת:
  - אין בלוק שאינו ממופה כחלק מקובץ וגם אינו מסומן כפנוי
  - כל inode שאין אליו מצביע, מסומן כפנוי

# גישות להתאוששות

- ❖ כתיבה זהירה
- ❖ עדכונים רכים
- ❖ כתיבה עצלה
- ❖ מערכות מתאוששות
- ❖ שימוש בזיכרון לא נדיף
- ❖ זיהוי הצורך בהתאוששות: הדלקת סיבית מיוחדת במערכת הקבצים מכריזה שהמערכת ירדה באופן מסודר והיא קונסיסטנטית. הכתיבה הראשונה למערכת קבצים היא כיבוי הסיבית. אם היא כבויה, דרושה התאוששות