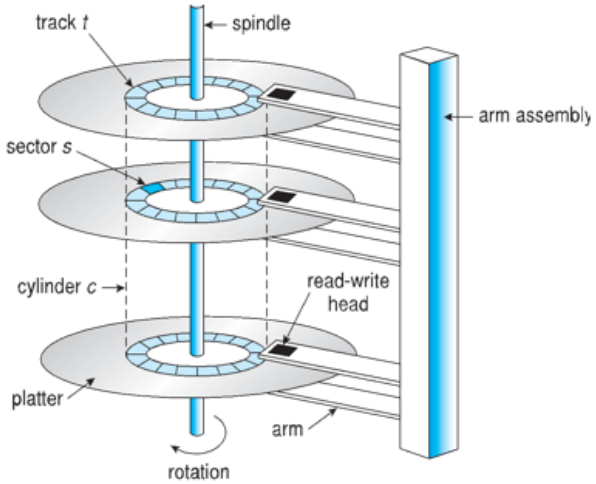
סוגי זיכרון:

זיכרון לא נדיף – זיכרון אלקטרוני – כמו הזיכרון הראשי אבל כשמנתקים מהחשמל הוא לא מאבד את הזיכרון – הגישה אליו כמו מטריצה ולכן הוא עובד מהר יותר כי אין לו חלקים מכניים.

HDD – סדר גודל של סיבוב זה 60 סיבובים בשנייה. באיזה מהירות הוא מעביר מידע ממנו למחשב? לכל התקן יש קונטרולר – זה המוח האלקטרוני שמנהל את קריאת הנתונים ואחר כך מעביר אותם למחשב.

- seek time – זמן שלוקח להזיז את המיקום של הקריאה למקום הנכון.
- transfer time – זמן שלוקח להעביר למחשב עצמו – לזיכרון הראשי.



יכול להיות כמה דיסקים – הראשים קריאה כתיבה. כולם זזים באותו זמן.

Sector – בזיכרון הראשי אפשר לקרוא ביט אחד. בזיכרון הזה אי אפשר – צריך לקרוא לפחות 512 ביט בבת אחת ואז לכתוב את כלום – זה נקרא סקטור. זה קבוצה של ביטים.

Cylinder – קוטר – איך ממלאים זיכרון של קובץ אחד? ממלאים היקף כלשהו ואז ממלאים באותו ההיקף רק בדיסק אחר – כדי שבהמשך קריאת הקובץ לא נצטרך להזיז את הראש, שזה לוקח הכי הרבה זמן.

ביצועים לדיסק – עיכוב זמן גישה = זמן חיפוש ממוצע + latency ממוצע.

- latency – הזמן שלוקח להמתין עד שהסקטור הרצוי יסתובב מתחת לראש הקריאה. למשל, אם יש 60 סקטורים (סיבובים), אז הזמן הזה הוא חצי מסיבוב – כלומר 1/120 שניות ≈ 8.3 מילישניות.

NVM – nonvolatile memory devices:

SSD – Solid-state disk:

- יותר אמין – כי אם HDD נופל מקומה שלישית 😊 הוא עלול להיהרס, כי אם הדיסקים המסתובבים נפגמים יכול להיות בעיה. ב-SSD אין חלקים נעים.

- חיסרון – זה יותר יקר פר יחידה.
- יש לו אורך חיים קצר יותר – מספר הכתיבות במשך החיים שלו מוגבל. כמה כתיבות הוא יכול לעשות הרבה פחות.
- הוא יותר קטן מ-HDD – קשה לבנות דיסק בגודל 20 טרה.
- הוא מהר מאוד ולכן לפעמים התקשורת – האפיק בין המחשב לדיסק – איטי מדי עבורו.
- אין בו חלקים זזים – לא צריך להזיז ראש קריאה כתיבה ואין סיבובים ולכן זה יותר מהיר.

פעם היה טייפ מגנטי – הוא מאוד אמין ולכן משתמשים בו בבנקים לזיכרון ארוך טווח – כי זמן גישה מאוד ארוך (לא למבחן).

איך דיסקים בנויים:
דיסקים – אין הבדל בין SSD ל-HDD – מערכת ההפעלה רואה אותם כמערך חד ממדי - כבלוקים לוגיים.

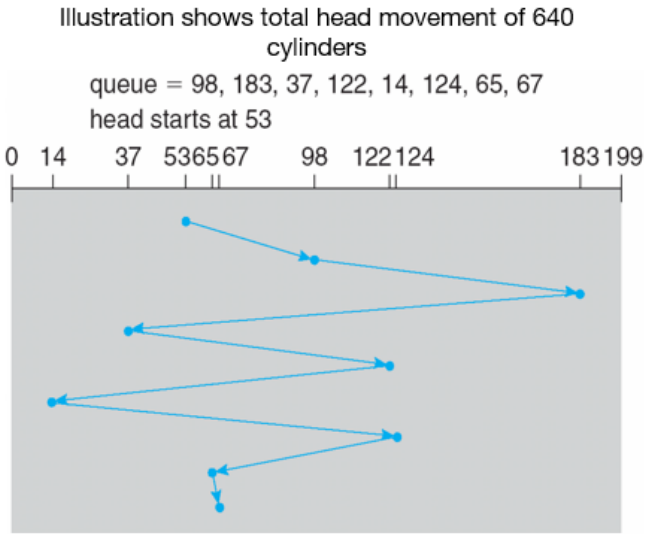
רק לקרנל מותר לגשת לדיסק – כי אחרת כל מתכנת יכול לגשת לנתונים ולמחוק אותם - לדרוס אותם.
low-level format – מחלק ליחידות של 512 ביט שזה נקרא גם בלוקים לוגיים – כל פעם אפס הוא במקום אחר (זה לוגי - לא קבוע).

מיפוי של מספר לוגי לבלוק הוא מורכב – צריך להבין איזה דיסק, איזה היקף, איזה בלוק. מערכת ההפעלה לא עושה את זה כי היא רואה את זה כמערך חד ממדי – הקונטרולר של הדיסק הוא זה שמחשב את זה ואחר כך הוא שם את זה במקום הנכון.

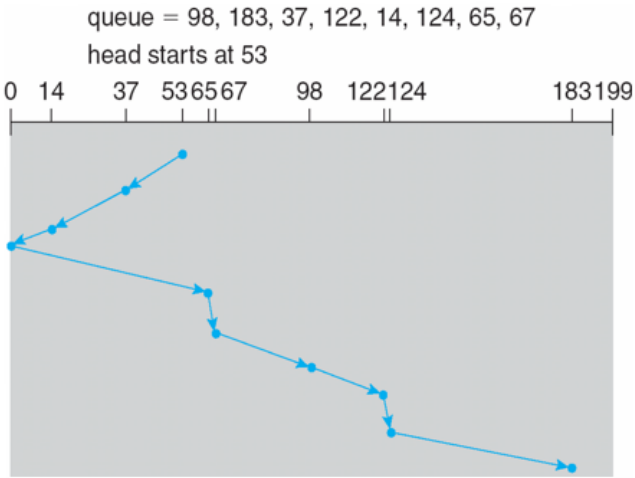
תזמון דיסק (נמצא רק ב-HDD) – כי רק שם הגישה שונה בין כל מקום בדיסק – כי יש זמן הזזת הראש קריאה כתיבה.
מעריכים שזמן חיפוש שווה למרחק ההזזה – אם מזיזים רבע מהדיסק אז זה יהיה רבע מזמן ההזזה הכללי.

זמני גישה:

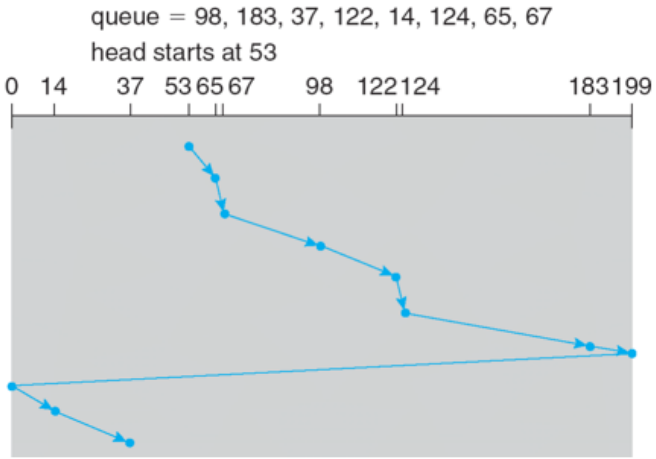
- FCFS – קודם הוא זז לפי סדר המחכים בתור הגישה – סך ההזזות ("אורך של כל קו"). תחשבו על שליח וולט – הוא הולך לכתובות לפי הסדר, קודם לכתובת הראשונה ואז לשנייה – לפי הסדר. זמן חישוב הוא המרחק הכללי שהוא נסע (דוגמא של לויטן … 😂).



- אלגוריתם הסריקה – SCAN – כמו מעלית שזזה בכיוון מסוים ומטפלת בכל מה שבדרך שלה. – הולך לכיוון מסוים (נגיד לכיוון מטה – לכיוון אפס), וכל מספר שהוא עובר הוא מטפל בו. הוא סורק מצד אחד לצד שני של הדיסק – ילך לאפס אפילו שהוא לא צריך.



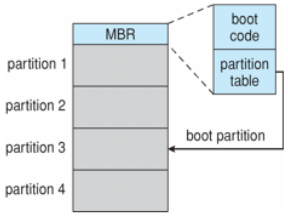
- C-SCAN – סריקה מעגלית (מחזורית) – תמיד סורק באותו כיוון. הוא הולך בכיוון אחד ואז חוזר לסוף בכיוון ההפוך בלי לטפל בשום דבר וממשיך באותו כיוון. הוא תמיד ייתן תוצאה פחות טובה מ-SCAN.
איך מחשבים – קצה גבוה פחות כיוון ראש ועד האורך הכללי של הזיכרון ועוד המיקום הסופי של הראש.
למה משתמשים? הוא נותן זמנים יותר אחידים – כי ב-SCAN רגיל זמן טיפול בקצוות הוא יותר איטי, לעומת זאת ב-C-SCAN הזמן אחיד.



- SSTF – Shortest Seek Time First – לא מתחשב בסדר ההגעה – הולך למקום הכי קרוב אליו. מנסים להזיז כמה שפחות. לא תמיד נותן את הזמן הכי טוב – כי יכולה להיות סדרה של קפיצות מימין לשמאל שיקחו הרבה זמן. לשיטה הזו יש חיסרון של "הרעבה" – בקשות רחוקות יכולות לחכות המון זמן אם כל הזמן מגיעות בקשות קרובות יותר.

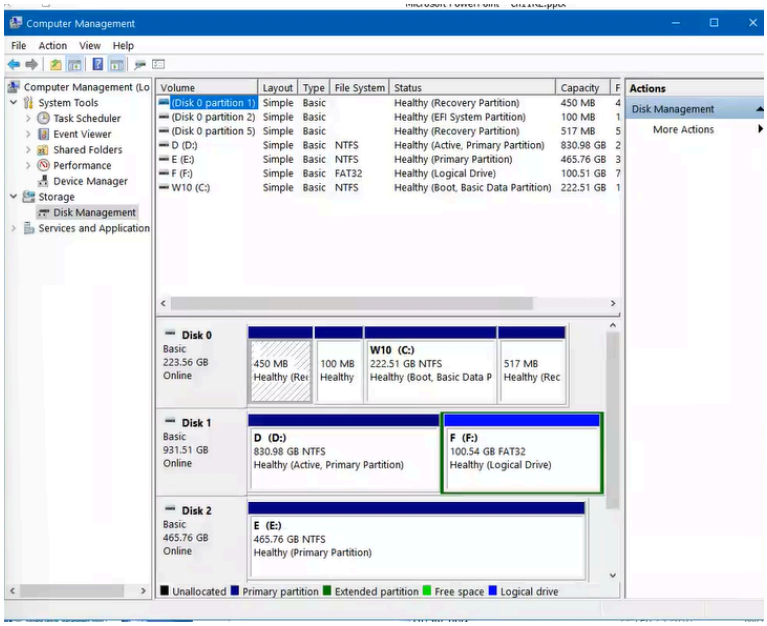
בכל סקטור יש קוד לניהול טיפול שגיאות.

Low-level-formatting – יוצרים סקטורים ומחלקים את הדיסק לסקטורים, בדרך כלל של 512 ביט (חצי קילו בייט).
צריך לחלק את הדיסק לחלקים ואחר כך לבצע logical formatting (מחיצות) כדי שנוכל ליצור מערכת קבצים.



Booting from secondary storage in Windows

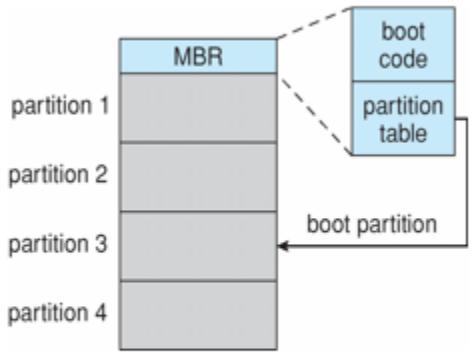
כל מחיצה נתפסת כדיסק לוגי בפני עצמה – כאילו איזה חלק מתוך הדיסק הוא "שלי". אם כולה – אז יש מחיצה אחת.
אפשר לראות איך כל דיסק מחולק למחיצות – כל אחת נתפסת כדיסק לוגי בפני עצמו – כך רואים בסייר הקבצים.
המחיצה היא שטח מסוים בדיסק שמחשב מסתכל עליו כדיסק בפני עצמו. פירמוט הוא פר מחיצה – וכל מחיצה יכולה להיות מערכת קבצים שונה לחלוטין.



(צילום מסך של המחשב של לויטן - רואים פה איך כל דיסק מחולק לחלקים)

מערכת קבצים יכולה להחליט שהיא לא רוצה לחלק קובץ בגודל בית אחד אלא ל-cluster (אשכול) – קבוצה של סקטורים בערך 4 קילובייט – כי זה לא משתלם להזיז את ראש הקריאה כתיבה בשביל כל כך מעט מידע. (יחידת ההקצאה המינימלית – 4KB).

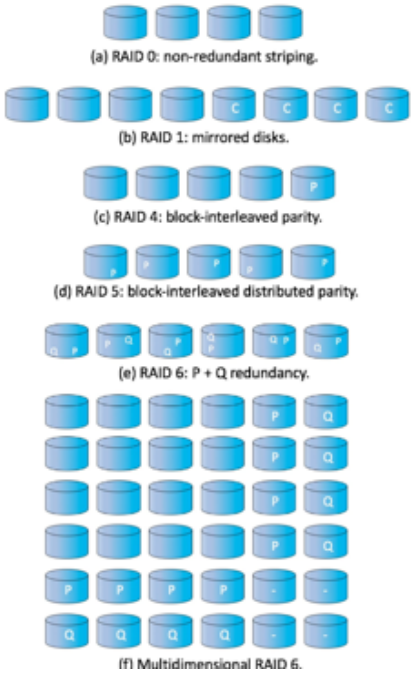
בכל דיסק יש ROOT partition – היא תמיד מחוברת למערכת ההפעלה. לינוקס מאפשרת לא לחבר מחיצות מסוימות – ואז למשל וירוס לא יכול לגשת למערכות קבצים לא מחוברות.



Booting from secondary storage in Windows

MBR – יש בו טבלה של איפה כל מחיצה מתחילה ונגמרת. הוא גם יודע לבצע boot – הוא יודע איפה במחיצה מוגדר ה-boot שבה אמור להיות הטוען עליה – bootstrap loader – הוא יטען את הקוד של מערכת ההפעלה.

<u>RAID – מערך דיסקים זולים:</u>

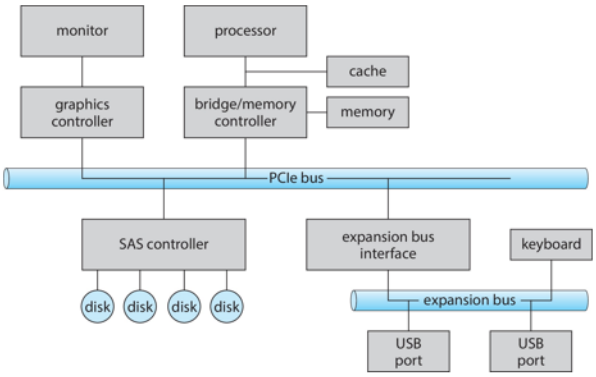


- RAID 0 – יש 4 דיסקים – אפשר במקביל לכתוב ל-4 דיסקים בבת אחת. מתקבל כאילו דיסק אחד בגודל גדול יותר וגם זמן גישה יותר מהיר כי אפשר לעבוד במקביל. מיועד להגדלת מהירות ונפח.
- RAID 1 – כותבים לדיסק (למשל 4KB) וגם שומרים גיבוי באותו גודל – מיועד להגברת אמינות.
- RAID 4 – יש 4 דיסקים ודיסק חמישי לקוד תיקון שגיאות.

יש שני סוגי RAID – או הגברת נפח ומהירות, או הגברת נפח ואמינות.

מצגת 12:
איך נתונים מועברים מהמחשב לדיסק:
חומרה:

- אמצעי קלט פלט שונים – מקלדת, עכבר, זיכרון.
- PORT – נקודת החיבור להתקן – יכול להיות פיזי ויש לו כתובות. לכל שקע יש כתובת בזיכרון.
- BUS – ערוץ התקשורת שקיים.
- Controller – לכל התקן יש מוח – הקונטרולר שלו. המערכת מתעסקת רק איתו ולא עם ההתקן עצמו.



ה-BUS עצמו יכול לבצע פעולות מורכבות יותר ואז זה נקרא CHANNEL – הוא יכול לתזמן וכו'.

פעולות קלט/פלט כתובות PORT:
המעבד כותב לכתובת מסוימת בזיכרון – בטווח כתובות מסוים – ואז זה ממופה לפי התקן. ההתקן יודע איפה לגשת ואז לא צריך את ה-BUS מהכיוון של המעבד.

I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)

הטווחים בזיכרון שההתקנים מחוברים אליהם.
כל ההתקנים יותר איטיים מהמעבד – איך המעבד מתמודד עם זה?
יש RAM סינכרוני שידוע כמה זמן ייקח לדיסק לבצע פעולה. אבל בפועל אי אפשר לדעת. אז איך המעבד יודע?

שיטה ראשונה – POLLING – יש אוגר מצב ואם הוא לא אפס זה אומר שההתקן עדיין באמצע פעולה – המעבד עושה BUSY WAITING וזה חבל.
לכן עושים פסיקות – interrupt – המעבד מקבל פסיקה.

יש 3 סוגים של פסיקות:

- פסיקות תוכנה.
- פסיקות TRAP – של מערכת ההפעלה (הקרנל).
- INTERRUPT – פסיקה חיצונית.

יש גם מושג של חריגות – למשל חלוקה באפס.
בעקבות כל פסיקה הקרנל נכנס לפעולה כלשהי – לפי מי שיזם את זה:

- מעבד → TRAP
- חיצוני → INTERRUPT

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

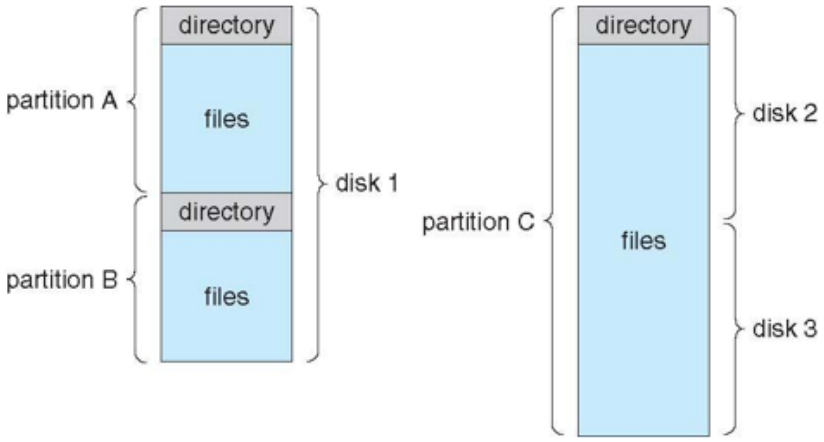
דוגמאות לפסיקות אפשריות

מצגת 13:
מה שקורה ב-open – המערכת מאתרת את הקובץ בדיסק כדי שנוכל לקרוא אותו. מוסרים את שם הקובץ ואת הנתיב שלו. הפעולה לוקחת זמן כי צריך למצוא את הקובץ. אחרי שהיא עושה את הפעולה הזאת היא יודעת את המידע על מיקום הקובץ.

לקובץ יש 3 מבטים:

1. איך שהקוד רואה אותו – במרחב המשתמש.
2. הקובץ בנוי מרשומות – כדי לדעת כמה לקרוא.
3. יש תו שמסמן סוף רשומה – EOF.

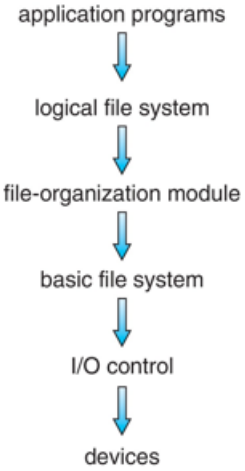
Sequential access – גישה לפי הסדר, רציפה.
Direct access – גישה ישירה למקום מסוים בקובץ.



יש דיסק מחולק ל-2 מחיצות – מה השתנה(ממה שלמדנו בחומר הקודם)? שבכל partition יש מערכת קבצים. אפשר גם ליצור מחיצה אחת משני דיסקים. אי אפשר להשתמש במחיצה לפני שיוצרים מערכת קבצים (אפשר אבל כמידע גולמי - אי אפשר לסדר את המידע) .

Directory – תוכן עניינים – מציין איפה כל קובץ נמצא במחיצה.

מצגת 14:



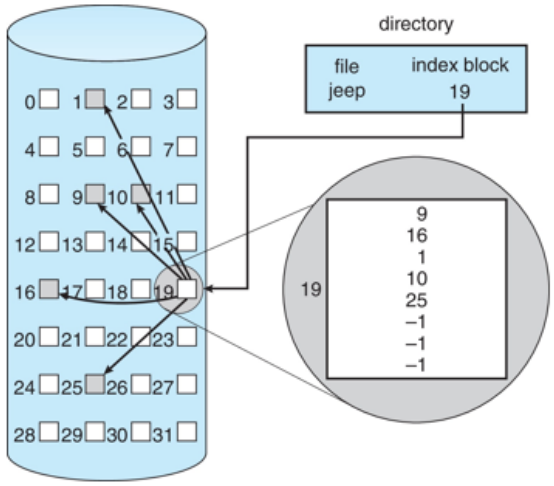
האפליקציה רואה את הקובץ כרשומות. מערכת ההפעלה רואה אותו כבלוקים. בפועל בזיכרון זה יכול להיות מוקצה לא ברצף.

שיטות ההקצאה – כמו בזיכרון – הקצאת דפים.
אבל יש יתרון לשמירת קובץ ברצף – כי בדיסק לוקח זמן להזיז את ראש הקריאה. לכן הקצאה רציפה היא הכי מהירה. אחרת, כמו בהקצאת דפים – יש רנדומליות.

צריך לשמור טבלת קבצים כדי לדעת איפה כל קובץ שמור.

קובץ לא חייב להיות שמור ברצף – כדי למנוע "חורים" בזיכרון שלא יהיה ניתן להשתמש בהם.

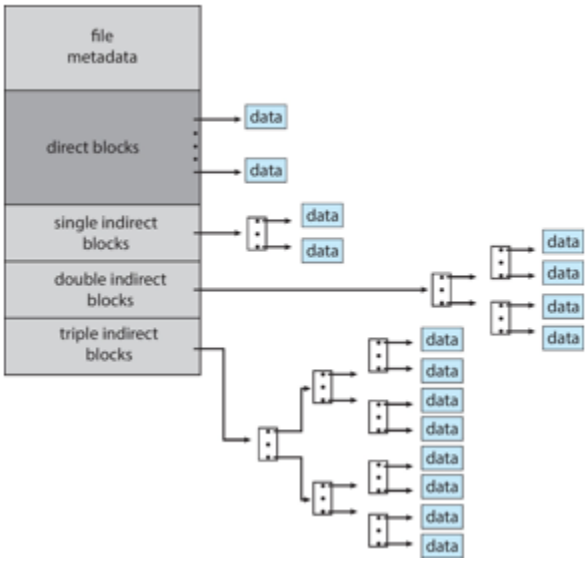
לכן לכל קובץ יש אינדקס – טבלת בלוקים – באילו בלוקים שמרנו אותו.



בלוק 19 בדיסק הוא בלוק של מצביעים – שמצביעים על בלוקים המכילים את הנתונים עצמם (כמו טבלת דפים בזיכרון).

קובץ תופס גם בלוקים של נתונים וגם בלוק של אינדקסים (מצביעים לעוד בלוקים) – ולכן לוקח יותר מקום בדיסק.

בעיה: בשיטה הזו הגודל המקסימלי של קובץ הוא 4MB – כי כל בלוק הוא 4KB וכל כתובת היא 4 בייט – אפשר רק 1000 בלוקים.
לכן יש שיטה שנייה – דומה, עם רשימה מקושרת. יש רמה נוספת של בלוקים – בלוק שמצביע לבלוקים אחרים המכילים מצביעים לבלוקים עם הנתונים. זה מתחלק לפי רמות – תלוי בגודל הקובץ. אם הוא קטן – צריך רק רמה אחת. אם הוא ענק – צריך הרבה רמות. השיטה משולבת – תלוי בגודל הקובץ.



למעשה, כל קובץ עצמו מחולק לרמות:

- החלק הראשון – מצביעים ישירים.
- החלקים הבאים – בלוקים של מצביעים.
- וכן הלאה...

בהקצאה רציפה גישה ישירה מהירה – כי תזוזת הראש היא בגודל הקובץ.
בהקצאה לא רציפה הבלוקים יכולים להיות בכל מקום בדיסק – תזוזת הראש גדולה יותר (בסדר גודל של כל הדיסק).

(אלא אם כן יש מטמון – ואז זה לא רלוונטי.)

כדאי לשמור את בלוקי המצביעים קרוב לבלוקים של הנתונים – כדי לחסוך תזוזות ראש.

file metadata – נתונים על הקובץ – בעלות, הרשאות גישה וכו'.

INODE – מבנה נתונים המתאר קובץ – כולל מצביעים לבלוקים של הנתונים (ישירים, עקיפים, עקיפים כפולים וכו').