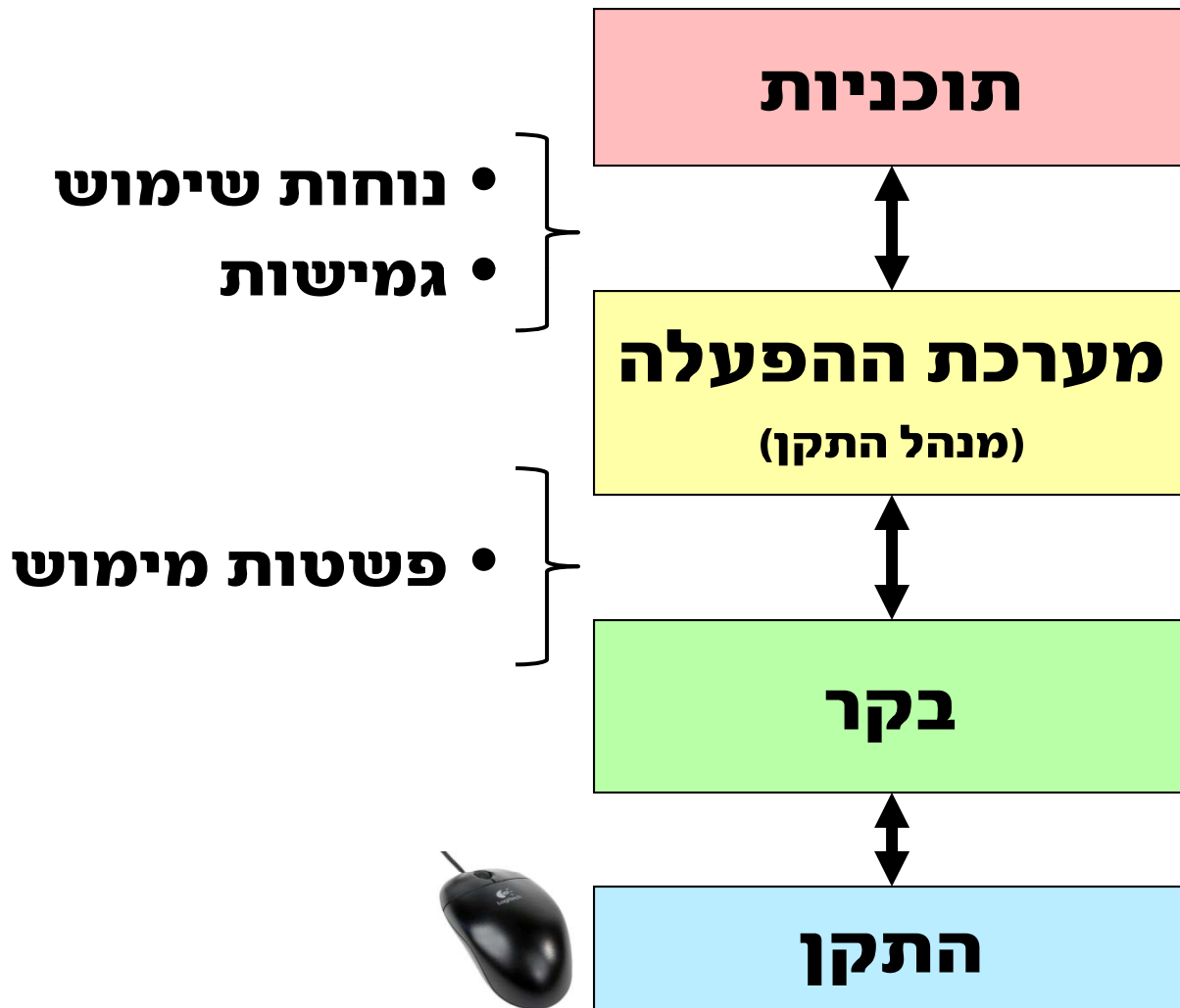


פרק 2

קלט/פלט

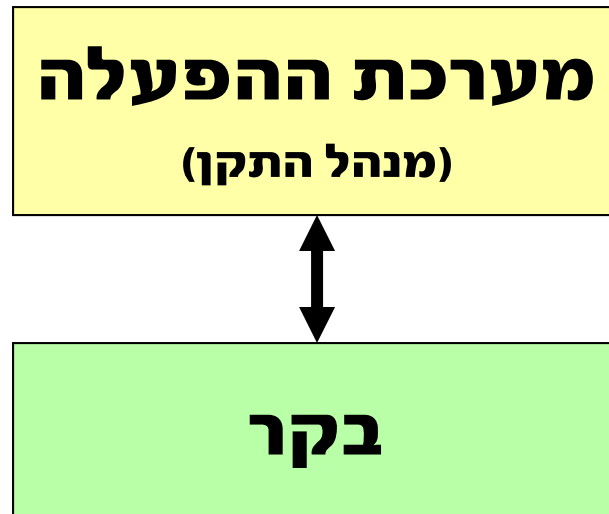
מבנה ומטרות מערכת הקלט-פלט



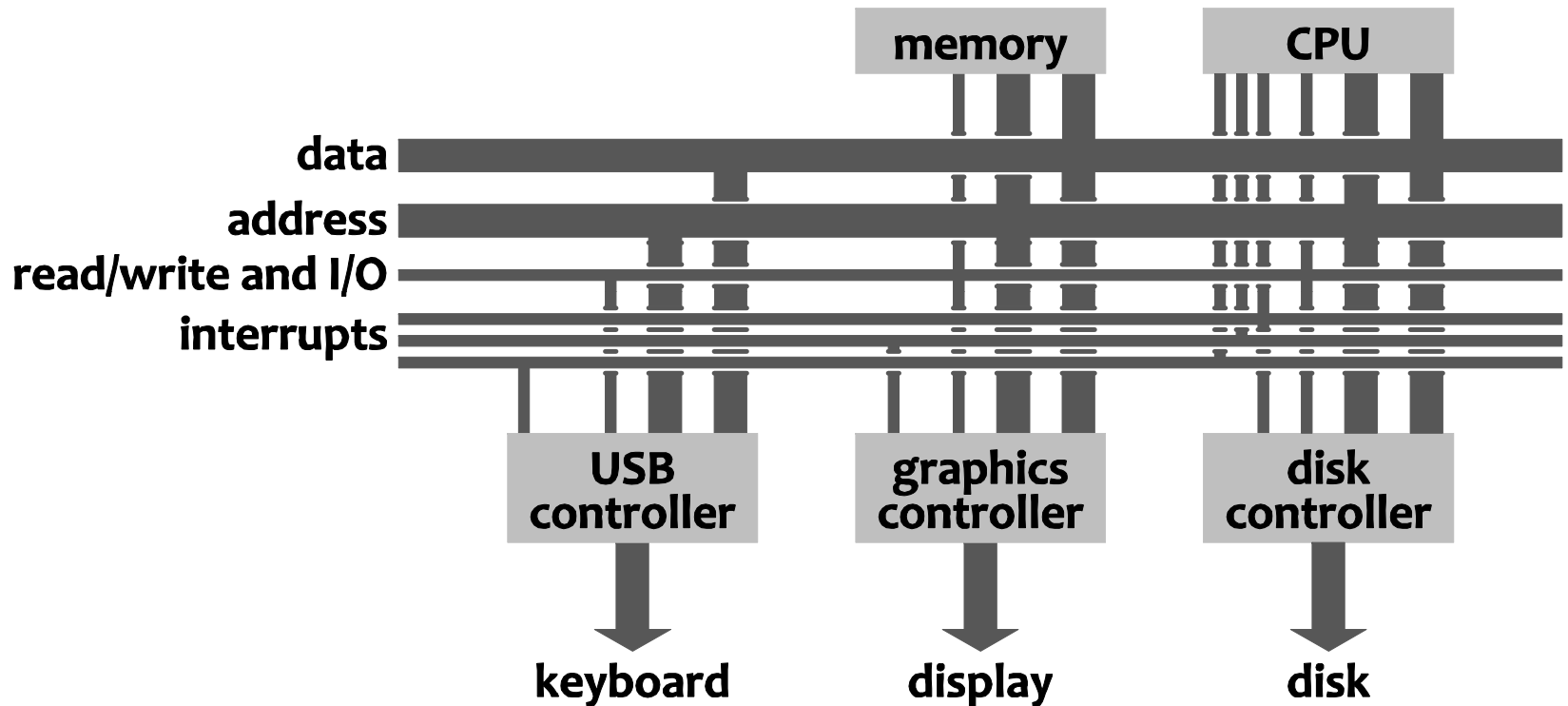
תמיד:

- זמן תגובה
- קצב העברה
- משאבים מזעריים
- אמינות

תקשורת עם בקרים



מנגנוני תקשורת עם בקרים



❖ דגימה (polling)

❖ פסיקות (interrupts)

❖ העברת נתונים ישירות מ/אל הזיכרון (direct memory access, DMA)

דגימה

❖ מערכת ההפעלה דוגמת את הבקר מדי פעם על מנת לבדוק האם קרא אירוע שמצריך תקשורת עם הבקר (הגיעו נתונים, הבקר מוכן לקבל פקודות נוספות, וכדומה)

דגימה

❖ מערכת ההפעלה דוגמת את הבקר מדי פעם על מנת לבדוק האם קרא אירוע שמצריך תקשורת עם הבקר (הגיעו נתונים, הבקר מוכן לקבל פקודות נוספות, וכדומה)

חסרונות משמעותיים:

❖ דגימה כמעט שאינה בשימוש. דגימה איטית מדי עלולה להוביל לאובדן נתונים

▪ פסיקת שעון רק 100 פעמים בשניה בהרבה מערכות

❖ דגימה כשלא קורה כלום מבזבזת משאבי מחשב

❖ כמעט לא בשימוש

פסיקות

- ❖ הבקר מודיע למעבד בעזרת קו תקשורת מיוחד בפס שקרא אירוע שמצריך תקשורת עמו. המעבד מגיב בהפעלת שגרת פסיקה של מערכת ההפעלה שמטפלת באירוע
- ❖ תגובה כמעט מיידיית
- ❖ תקורה בשמירת ושחזור מצב המעבד: לא כדאי לטפל בנפרד בפיסות קטנות של נתונים
- ❖ בשימוש נרחב מאוד

אבחנה בין פסיקות שונות

- ❖ כאשר מספר בקרים משתפים קו פסיקה, מערכת ההפעלה צריכה לברר איזה בקר הפעיל את הפסיקה
- ❖ ניתן לחסוך את הבדיקה על ידי שימוש במספר קווי פסיקה בפס
- ❖ במעבדים עם כניסת פסיקה אחת (למשל מעבדי IA32) משתמשים בבקר פסיקות שיש לו כמה כניסות פסיקה ויציאה אחת. בקר הפסיקות מודיע למעבד שאחד הבקרים הפעיל פסיקה ומערכת ההפעלה יכולה לברר בעזרתו איזה בקר הפעיל את הפסיקה

מניעת הפעלה רקורסיבית של שגרות פסיקה

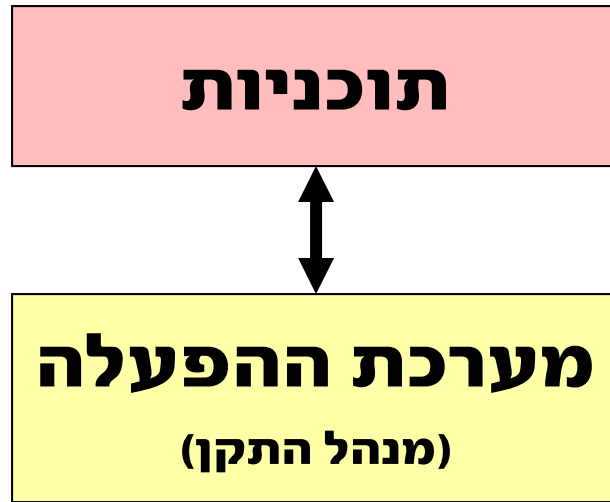
- ❖ פסיקה משעה את השגרה שרצה ומפעילה את שגרת הפסיקה
- ❖ רצוי לא להשעות את שגרת הפסיקה עצמה ולהפעילה שוב אם מתרחשת פסיקה נוספת לפני שהיא חוזרת
- ❖ הפעלה רקורסיבית עלולה לגרום להשחתת מבני נתונים (עוד על כך בפרק 5)
- ❖ מערכות הפעלה פשוטות משעות את הטיפול בפסיקות כאשר שגרת פסיקה רצה
- ❖ מערכות הפעלה מתקדמות יותר מסוגות פסיקות על פי עדיפויות ומשעות רק פסיקות בעדיפות נמוכה יותר מזו המטופלת

הפעלה דחויה של שגרות

- ❖ אם הטיפול בפסיקה ארוך, עדיף לבצע כמה שיותר מהטיפול מחוץ לשגרת הפסיקה, על מנת שלא לדחות טיפול בפסיקות אחרות
- ❖ מערכת ההפעלה מבצעת בשגרת הפסיקה את הטיפול שיש לבצע מייד ואורזת ייצוג של שאר הטיפול במבנה נתונים שמייצג הפעלה דחויה של שגרה
- ❖ מערכת ההפעלה מבצעת את הקריאות הדחויות לאחר הטיפול בפסיקות אך לפני החזרה לתוכנית המשתמש
- ❖ זהו למעשה מנגנון תזמון זעיר שמבטיח טיפול מהיר באירועים דחופים (פסיקות)

גישה ישירה לזיכרון

- ❖ מערכת ההפעלה מורה לבקר להעביר בלוק גדול של נתונים בין התקן חיצוני (דיסק, רשת) ובין הזיכרון. הבקר מעביר את הנתונים ללא התערבות נוספת של המעבד
- ❖ יתרון: קצב העברה גבוה ללא עומס על המעבד
- ❖ דורש ניהול זיכרון זהיר כדי שהתקנים יכתבו/יקראו רק מהכתובות הנכונות
- ❖ בשימוש נרחב מאוד בבקרי דיסקים ורשתות תקשורת מהירות
- ❖ בעבר העלות של בקר DMA היתה משמעותית, כיום זה שיקול רק לעיתים רחוקות



תקשורת עם מנהלי התקנים

מנהלי התקן (device drivers)

- ❖ מנהל התקן הוא מודול תוכנה עם ממשק סטנדרטי ששולט על התקן חומרה מסוים, כמו בקר דיסקים מסוג מסוים או בקר רשת מסוג מסוים
- ❖ השגרה שמופעלת על ידי קריאת מערכת קוראת למנהל ההתקן המתאים
- ❖ לעיתים מערכת ההפעלה קוראת למנהל התקן באופן לא ישיר: תוכנית קוראת מקובץ ומערכת ההפעלה קוראת למנהל ההתקן של הדיסק על מנת לחפש ולקרוא את הנתונים
- ❖ לא כל מנהל התקן שולט על חומרה; לפעמים מנהל התקן שולט על התקן וירטואלי, למשל מערך דיסקים

ממשק של מנהל התקן

- ❖ שגרת פסיקה שמופעלת כאשר ההתקן מפעיל פסיקה (interrupt handler or interrupt service routine)
- ❖ שגרה לאתחול ההתקן (initialize)
- ❖ שגרות להתחברות והתנתקות (close, open); משאבים, הרשאות
- ❖ שגרות לקריאת וכתיבת נתונים (write, read)
- ❖ שגרה להזזת מצביע הקלט/פלט בהתקני גישה ישירה (seek)
- ❖ העברת פקודות מיוחדות להתקן או למנהל (ioctl), למשל קצב שידור
- ❖ ממשק אחר למנהלי התקן של בקרי רשת תקשורת ושל בקרי תצוגה גרפית

דוגמה: סיפורה של גישה לדיסק

- ❖ תוכנית מבצעת קריאת מערכת - קריאה מקובץ (שכבר נפתח)
- ❖ מערכת ההפעלה מפעילה את שגרת read של מנהל ההתקן
 - מנהל ההתקן שולח הוראת בערוץ פלט אל ההתקן: "קרא N בלוקים ממקום M וכתוב את תוכנם לזיכרון בכתובת A"
- ❖ ... המתנה (אולי פעילה) ...
- ❖ ההתקן קורא מידע מהמדיה ומעביר אותו בגישה ישירה לזכרון
- ❖ ההתקן יוצר פסיקת חומרה
- ❖ המעבד קופץ לשגרת הפסיקה של מערכת ההפעלה
 - מערכת ההפעלה מריצה את שגרת הפסיקה של מנהל ההתקן
 - שגרה דחויה של מנהל ההתקן מעתיק את החלק הרלוונטי של המידע בכתובת A אל הזיכרון של התוכנית
 - מנהל ההתקן מדווח למערכת ההפעלה על סיום פעולת הקריאה
- ❖ מערכת ההפעלה חוזרת מקריאת המערכת והתוכנית ממשיכה לרוץ

הטמנה וחציצה

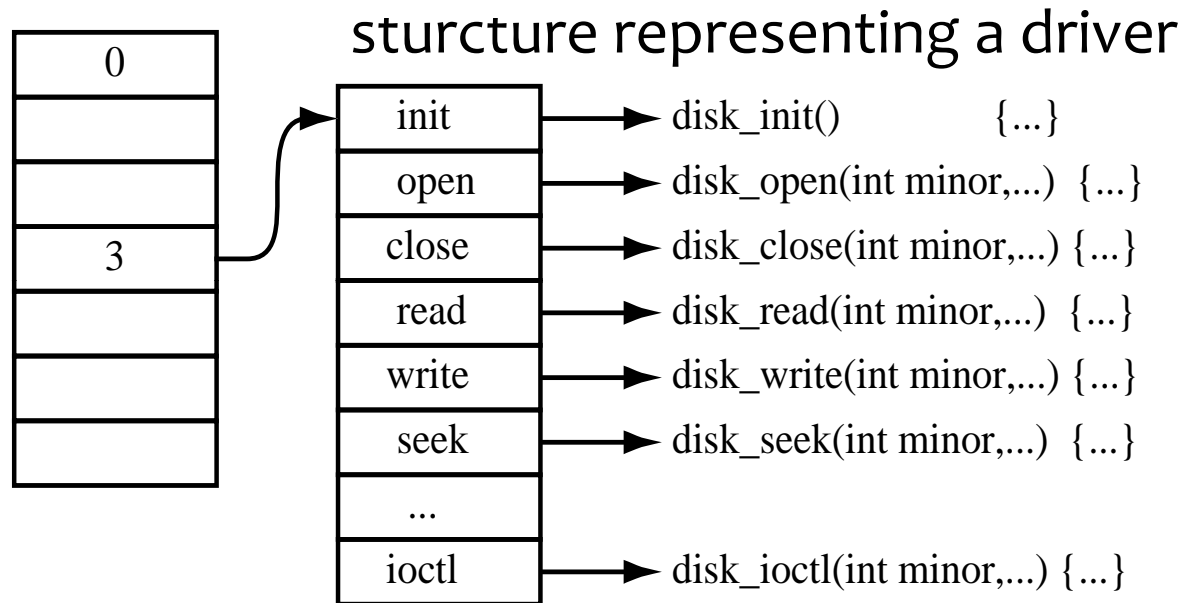
- ❖ **הטמנה (caching):** מערכת ההפעלה שומרת בזיכרון הראשי בלוקים של נתונים שנקראו לאחרונה מהתקני זיכרון חיצוניים
- ❖ גישה חוזרת לבלוק מחזירה את תוכנו מהמטמון ללא גישה להתקן החיצוני
- ❖ הטמנה (וגם read-ahead) חוסכת גישות להתקן חיצוני איטי
- ❖ **חציצה (buffering):** מערכת ההפעלה שומרת בזיכרון הראשי בלוקים שמיועדים להיכתב להתקני זיכרון חיצוניים ומעכבת את הכתיבתם להתקן
- ❖ כתיבה חוזרת לאותו בלוק משנה את תוכן הבלוק בחוצץ; רק הנתונים המאוחרים יכתבו להתקן החיצוני
- ❖ חציצה (עיכוב כתיבה) חוסכת גישות שמשכתבות נתונים ומאפשרת לכתוב להתקן כאשר הוא פנוי

מנהלי התקן עם וללא הטמנה

- ❖ מערכת ההפעלה מנהלת מאגר חוצצים מרכזי (מטמון) עבור מנהלי התקן של התקני זיכרון חיצוניים
- ❖ התקנים שאינם התקני זיכרון (מקלדת, עכבר, מדפסת, רשת תקשורת) אינם זקוקים להטמנה, אבל חלקם זקוקים לחציצה
- ❖ חציצה מאפשרת לקרוא נתונים שמגיעים מההתקן (מקלדת למשל) גם אם תוכנית אינה מבצעת קריאה באותו רגע ממש, ולכתוב נתונים בקצב שמתאים להתקן (מדפסת, מודם)
- ❖ בלינוקס/יוניקס יש שתי טבלאות של מנהלי התקן: עם הטמנה (block) וללא הטמנה (character), חוץ ממנהלי התקן לבקרי רשת ולבקרי תצוגה גרפית

התייחסות להתקנים ביוניקס

block devices



- ❖ התקן מיוצג על ידי שני מספרים: ראשוני שמייצג את מנהל ההתקן שמטפל בהתקן ומשני שמייצג את ההתקן עבור המנהל
- ❖ תוכניות מתייחסות להתקן על ידי גישה לקובץ מיוחד שמכיל מזהה טבלת מנהלים (עם/ללא הטמנה) ואת שני המספרים
- ❖ ניתן להתייחס להתקן כאל קובץ, להתחבר אליו ולקרוא/לכתוב נתונים

מנהלי התקן בלינוקס/יוניקס

❖ מנהל ההתקן קיים וההתקן עצמו קיים:

```
# ls -l /dev/lp0
crw-rw---- 1 root daemon 6,0 may 5, 1988 /dev/lp0
# cat < /dev/lp0
skfjkl...
```

❖ מנהל ההתקן קיים, אבל אין התקן פיזי:

```
# cat < /dev/lp0
cat: -: Input/output error
```

❖ הקובץ המיוחד קיים, אבל אין מנהל התקן בטבלה:

```
# cat < /dev/lp0
bash: /dev/lp0: No such device
```

מסלול השגרות במערכת ההפעלה

```
...  
fd = open("/dev/lp0",...); // returns 4  
read(fd, ...);
```

↑ user mode
↓ kernel mode

file descriptor 4 for process 1234 → special file (6,0)

read system call:

```
call character_devices[6].read(0,...)
```

device driver for lp:

```
int read(...) { retrieve bytes from buffer }
```

```
void isr(void) { read bytes from controller to buffer }
```

עוד קבצים מיוחדים של מנהלי התקן

```
# ls -ld /dev/*
```

```
brw-rw---- 1 root root          11,  0 Nov  7 18:47 /dev/cdrom
crw-r----- 1 root kmem         10, 144 Nov  7 18:47 /dev/nvram
brw-rw---- 1 root disk           8,  0 Nov  7 18:47 /dev/sda
crw--w---- 1 root tty            4,  0 Nov  7 18:47 /dev/tty0
crw--w---- 1 root tty            4,  1 Nov  7 18:47 /dev/tty1
crw-rw---- 1 root root        188,  0 Nov  7 18:47 /dev/ttyUSB0
crw-rw---- 1 root audio         14,  4 Nov  9 18:47 /dev/audio
brw-rw---- 1 root disk       253,  0 Nov  9 18:47 /dev/dm-0
crw-rw-rw- 1 root root           1,  5 Nov  7 18:47 /dev/zero
crw-rw-rw- 1 root root           1,  8 Nov  7 18:47 /dev/random
```

(partial list)

יצירת הקבצים המיוחדים בלינוקס

- ❖ הקבצים המיוחדים ב-dev/, המובילים אל טבלאות מנהל ההתקן הם הממשק מולו עובדות רוב התוכניות וחלק ממ"ה
- ❖ הקבצים עצמם נוצרים על ידי תוכנית שירות המהווה חלק ממערכת ההפעלה במובן הרחב (לא בגרעין)
- ❖ בעבר: סקריפט בעליית המערכת יצר את כל הקבצים האפשריים
- ❖ כיום: מנגנון מסובך (udev) עם אוסף כללים: "אם מנהל התקן X מצא התקן של יצרן Y מדגם Z אז צור קובץ מיוחד F אשר מצביע אליו"
- ❖ תוכניות רגילות לא מודעות למנגנונים הללו

קישור בין ההתקן למנהל ההתקן

❖ התקנה ידנית

❖ חיבור אוטומטי

- בחיבורי התקנים מודרניים (למשל USB) ההתקן מתאר את עצמו למנהל התקן גנרי (יכולות סטנדרטיות, מספר יצרן ומספר דגם)
 - מנהל ההתקן ספיציפי (למשל לכרטיס קול מסויים) מספר למערכת ההפעלה באיזה מזהים הוא תומך
 - כאשר מחברים התקן חדש הנמצא ברשימה, מערכת ההפעלה מפעילה את מנהל ההתקן המתאים
- ❖ חיפוש אוטומטי באינטרנט של מנהל התקן לפי מזהה שלו

התקנים כקבצים: שימושים (קוריוזים?)

❖ ממשק פשוט וסטנדרטי

❖ ניתן להפעיל פקודות קבצים על התקנים

❖ יצירת עותק מושלם של CD לתוך קובץ ISO:

```
cat /dev/cdrom > image.iso
```

❖ דחיסה ושליחה של דיסק קשיח שלם לשרת ברשת:

```
cat /dev/sdb | gzip -c | ssh user@server 'cat > backup'
```

❖ השמעת רעש לבן:

```
cat /dev/random > /dev/audio
```

❖ הקלטת אודיו לגיקים

```
cat /dev/audio > program
```



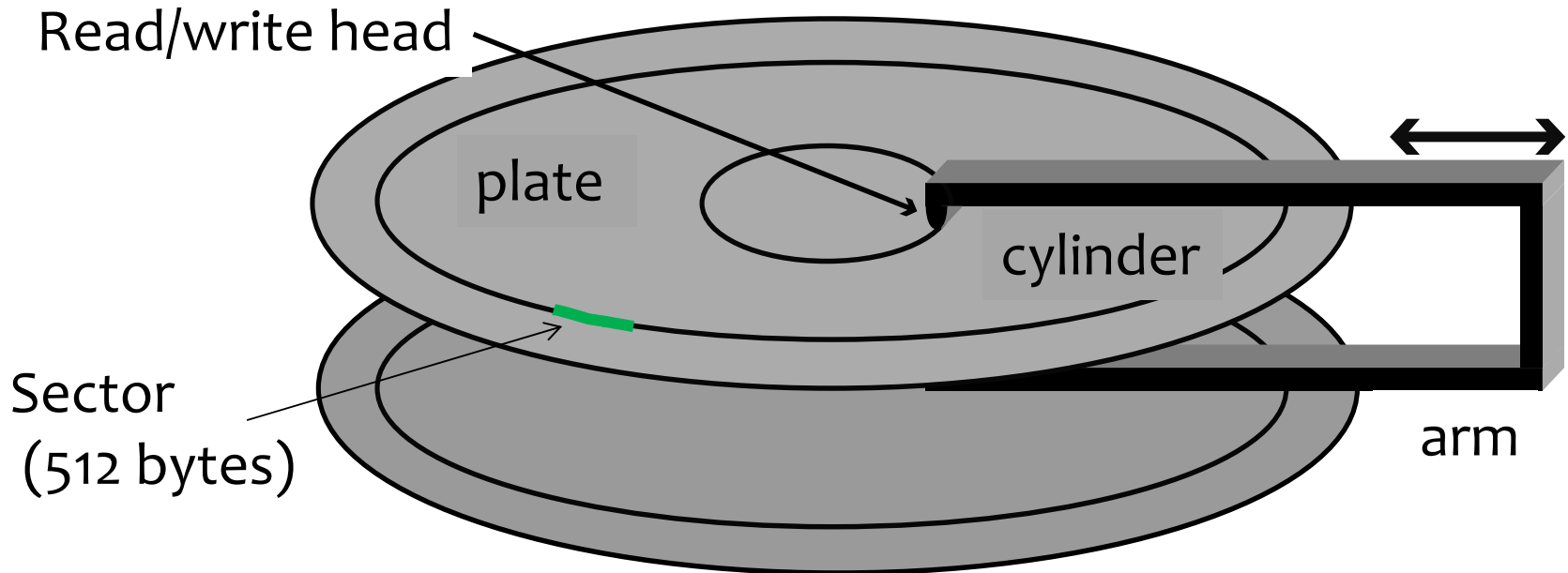

התקנים מעניינים



התקן



דיסקים קבועים



- ❖ משמשים לאחסון קבצים וכהרחבה של הזיכרון הפיזי ולכן ביצועיהם משפיעים ישירות על ביצועי מערכת המחשב כולה
- ❖ מסתובבים במהירות קבועה של 3600-15000 סיבובים בדקה
- ❖ קיבולת של עד 2 TB וקצבי העברה אפקטיביים של 100 MB/s
- ❖ המספרים נכונים לשנת 2011; זמן סיבוב ורוחב פס משתפרים לאט

תזמון דיסקים קבועים

❖ בקשות גישה מצטברות בתור

❖ איזה בקשה לבצע כעת? שאלת מדיניות

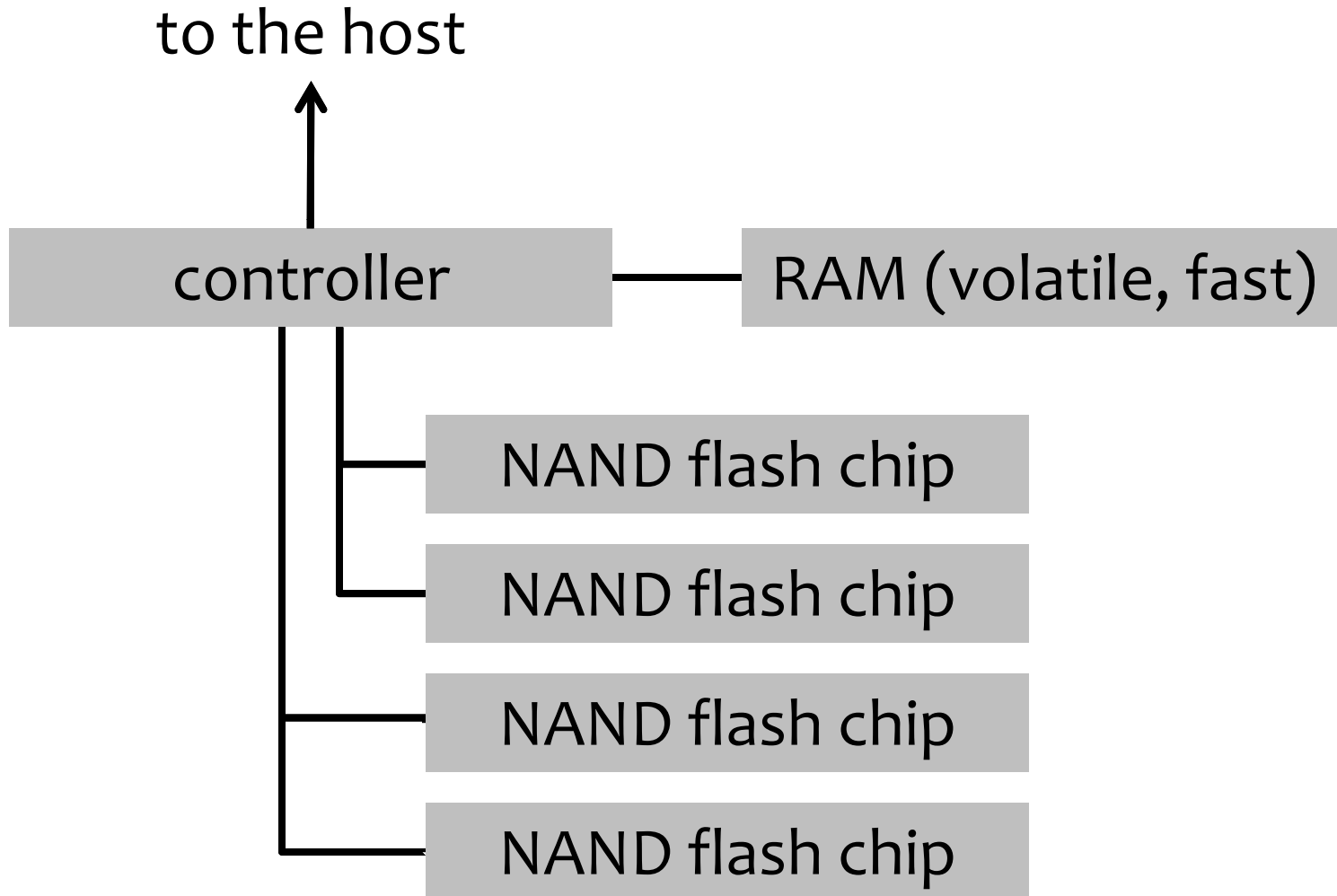
תזמון דיסקים קבועים

- ❖ בקשות גישה מצטברות בתור
- ❖ איזה בקשה לבצע כעת? שאלת מדיניות
- ❖ ביצוע לפי סדר קבלה (FCFS): הוגן לחלוטין אבל לא יעיל; הזרוע עלולה לנוע שוב ושוב מצד לצד על מנת לקרוא קטע (סקטור) בודד בכל פעם
- ❖ מזעור זמן השיוט (SSTF): הבקשה הבאה שתשורת היא הבקשה שניתן להגיע אליה תוך פרק הזמן המזערי; לא הוגן עד כדי הרעבה
- ❖ מדיניות מעלית: הזרוע נעה מהמסילה הפנימית לחיצונית וחזרה ומשרתת את כל בקשות הגישה לפי סדר תנועת הזרוע, בדומה לאוטובוס עם מסלול קבוע; הוגן ויעיל

ואריאציות על מדיניות מעלית

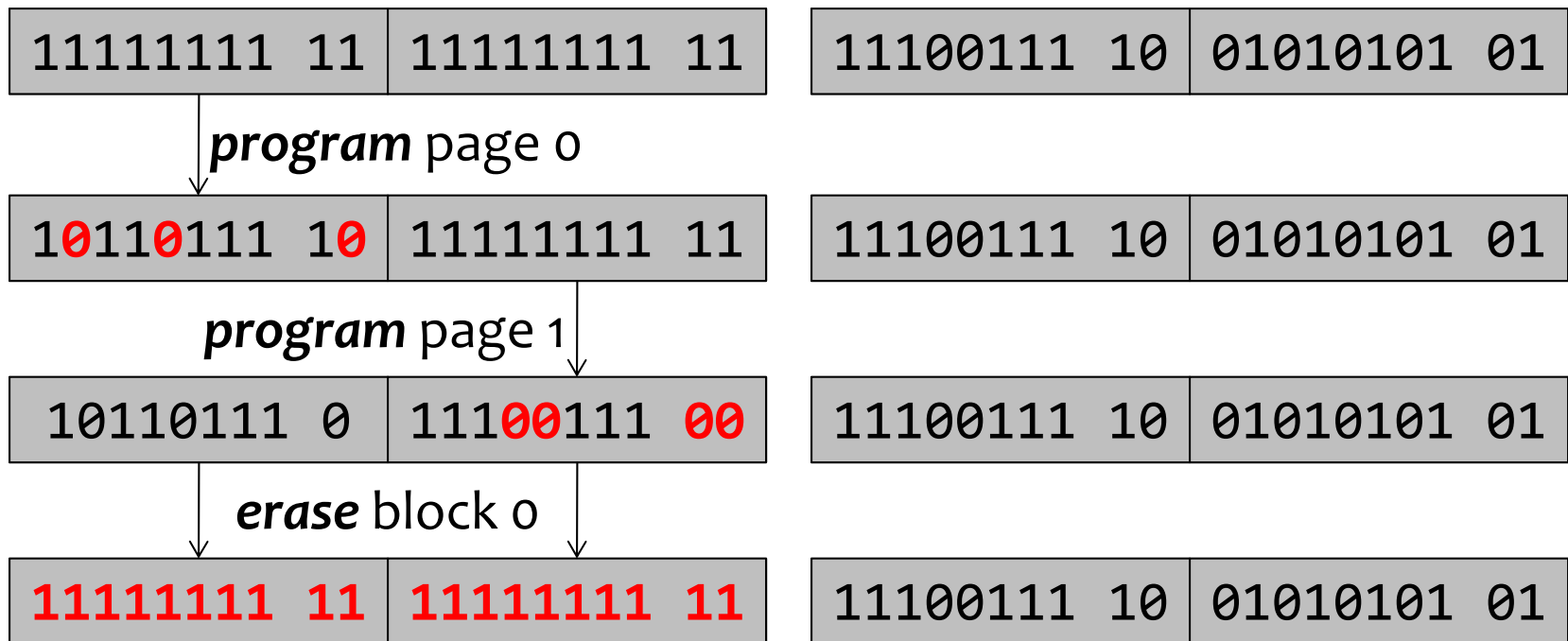
- ❖ scan: הזרוע נעה פנימה והחוצה בין שתי המסילות הקיצוניות ומשרתת בקשות שהיא פוגשת בדרך
- ❖ c-scan: כנ"ל, אבל משרתים בקשות רק בדרך פנימה, את הדרך החוצה עושים במהירות; יותר הוגן ואולי מעט פחות יעיל (מעט כי הזרוע מאיצה)
- ❖ look: כמו scan אבל הזרוע לא נעה מעבר למסילה הקיצונית שיש עבורה בקשה. יותר יעיל ופחות הוגן
- ❖ c-look: מובן מאליו
- ❖ אפשר להפעיל את מדיניות דומה גם לגבי קטעים בתוך מסילה
- ❖ האם כדאי לשרת בקשות למסילה שנמצאים בה שנכנסו לתור לאחר שהזרוע הגיע למסילה. מדוע? שקלו הגינות, יעילות

אחסון מבוסס flash (SSDs)



זיכרונות NAND Flash

❖ הזיכרון מחולק לבלוקים, שמחולקים ל-128 או 64 דפים, שמכילים נתונים ונתונים אודות הנתונים (metadata); דף מכיל כמה סקטורים (רצף שהמחשב קורא או כותב בבת אחת)



SSDs צריכים בקר חכם

- ❖ לא כדאי לשכתב סקטור במקום, כי צריך יהיה למחוק בלוק שלם
- ❖ תאי זיכרון נשחקים (נהרסים) במהלך כתיבות/מחיקות; מספר מחזורי הכתיבה הוא בין אלף ל-100 אלף (הולך ונהיה גרוע)
- ❖ הבקר כותב את סקטור i לדף j וזוכר את המיפוי
- ❖ מבנה הנתונים של המיפוי נשמר ב-RAM של הבקר ועל ה-flash
- ❖ הנתונים אודות הנתונים מכילים את זהות הסקטורים השמורים בדף, קוד לתיקון שגיאות, ומידע אחר שהבקר צריך

דוגמה לתכנון בקר של SSD

❖ SSD בגודל 32GB שיש לו 32K בלוקים בגודל 1MB, בכל אחד 128 דפים של 8K, ויש לו 1MB של RAM

❖ איך לנהל את המיפוי?

- כל 1MB של סקטורים רצופים ימופו לבלוק אחד ב-flash
- את המיפוי הזה ה-SSD ישמור ב-RAM שלו
- אלגוריתם (פשוט) למצוא סקטור נתון בזמן קריאה ולשכתב סקטור
- איזון שחיקה: מחיקת כל הבלוקים באותו קצב בערך
- המיפוי נכתב ל-flash לפני שמכבים את ה-SSD; לא הנחה כל כך טובה כי לפעמים אספקת החשמל נפסקת בפתאומיות
- שיפורים מבוססים על עקרונות של מערכות קבצים מתאוששות ועל log-structured file systems שנדון בהם בהמשך

אמינות במערכות גדולות

- ❖ נניח שלדיסק בודד יש הסתברות של 0.1% להתקלקל במהלך 24 שעות
- ❖ (בפועל ההסתברות ליחידת זמן גדולה יותר עבור תינוקות וזקנים)
- ❖ תוחלת חיי הדיסק בערך 3 שנים (התפלגות גיאומטרית)
- ❖ אם כמות הנתונים דורשת שימוש ב-10 דיסקים (אחד לא מספיק), אז תוחלת הזמן לתקלה הראשונה היא _____

אמינות במערכות גדולות

- ❖ נניח שלדיסק בודד יש הסתברות של 0.1% להתקלקל במהלך 24 שעות
- ❖ (בפועל ההסתברות ליחידת זמן גדולה יותר עבור תינוקות וזקנים)
- ❖ תוחלת חיי הדיסק בערך 3 שנים (התפלגות גיאומטרית)
- ❖ אם כמות הנתונים דורשת שימוש ב-10 דיסקים (אחד לא מספיק), אז תוחלת הזמן לתקלה הראשונה היא 100 ימים; מעט מדי
- ❖ אין מה לחשוב אפילו על מערך של 100 או 1000 דיסקים...

גילוי ותיקון שגיאות

❖ אריתמטיקה בינרית: $0=0+0$, $1=0+1$, $1=1+0$, $0=1+1$ (xor)

❖ נגן על 4 סיביות נתונים x_1, x_2, x_3, x_4 על ידי שימוש בסיבית

נוספת x_5 ואילוץ אלגברי $x_1 + x_2 + x_3 + x_4 + x_5 = 0$

❖ חשב את x_5 $1 + 1 + 1 + 0 + _ = 0$

❖ גילוי שגיאה $1 + 1 + 0 + 1 + 1 = 0$

❖ תיקון (השלמת חוסר) $1 + ? + 1 + 1 + 1 = 0$

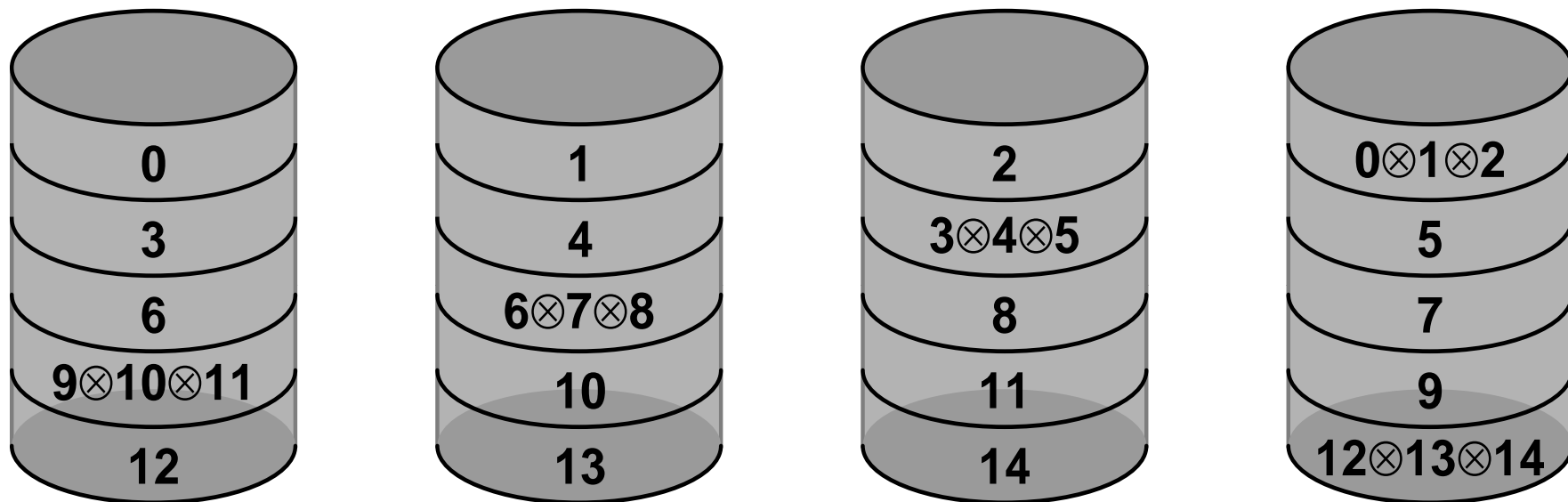
❖ האם אפשר לגלות ככה שגיאה בשתי סיביות? או לגלות וגם לתקן

שגיאה?

מערכי דיסקים (RAID)

- ❖ מספר דיסקים משמשים כהתקן אחסון אחוד
- ❖ ניתן למימוש ברמת מנהל ההתקן, כמו דיסקים לוגיים ומחיצות, או ברמת בקר הדיסקים שדרושה רמת ביצועים גבוהה
- ❖ מבוסס על מיפוי בלוקים של נתונים בהתקן הוירטואלי (מערך הדיסקים) לדיסק+היסט, כלומר למיקום של בלוק פיזי של נתונים
- ❖ מספר ואריאציות שנקראות רמות שמספקות קומבינציות שונות של שירותים; רמה גבוהה אינה בהכרח טובה יותר לכל שימוש

שחזור נתונים עם RAID



❖ המערך מחולק לרצועות

❖ רצועה שומרת $n-1$ בלוקים של נתונים ובלוק זוגיות

❖ אפשר לקרוא בלוק בודד; או לקרוא רצועה ולבדוק שגיאות

❖ אפשר לשחזר כל בלוק חסר, ואפשר גם לשחזר דיסק שלם שהתקלקל

❖ כתיבה של בלוק על ידי קריאה של הבלוק ובלוק הזוגיות ועדכון, או על ידי

כתיבה של רצועה שלמה בלי לקרוא קודם כלום

הפשטות נוספות

- ❖ בקר RAID או מנהל התקן RAID גורם לכמה של דיסקים להיראות כיחידת אחסון אחת (גדולה ואמינה יותר)
- ❖ מחיצות: חלוקה של דיסק פיזי לדיסקים לוגיים; טבלה בתחילת הדיסק מתארת את החלוקה, כל מערכות ההפעלה מבינות אותה
- ❖ logical volumes: מנגנון יותר מתוחכם ממחיצות (partitions) שמחלק את הדיסק לרצפים (extents) ומצרף כמה רצפים ליחידת אחסון לוגית אחת; למשל, אפשר לאחד את החלקים המהירים של 4 דיסקים שונים ליחידת אחסון אחת

דיסקים אופטיים

- ❖ תקליטורים מאחסנים 650-700 MB, תקליטי DVD עד 4.7 GB, Blu-ray עד 50 GB
- ❖ מידע מיוצג על ידי חורים במשטח
- ❖ תהליך דפוס מהיר להפצה של מידע ספרתי, אפשרות צריבה (איטית יחסית) בכוננים מתאימים
- ❖ אורך חיים גבוה מתאים לשמירת מידע ארכיונית
- ❖ שימשו לגיבויים אבל היום דיסקים מגנטיים משמשים לגיבויים (בעבר גם סרטים מגנטיים שימשו לגיבויים)

תצוגות גרפיות וכרטיסי קול

- ❖ הבקר מכיל יחידת זיכרון (frame buffer) שכל מילה בה ממופה לפיקסל (לפעמים נעשה שימוש בזיכרון הראשי של המחשב)
- ❖ בבקרים פשוטים המעבד צובע פיקסלים על ידי כתיבה לזיכרון הזה
- ❖ בבקרים מתוחכמים המעבד יכול להורות לבקר לצבוע משולש שלם, לחשב הסתרות של עצמים במרחב, וכולי
- ❖ המבנה נגזר מהצורך להעביר לתצוגה נתונים בקצב גבוה, בעיקר עבור אנימציה ווידאו
- ❖ למשל: $30 \text{ f/s} \times 1 \text{ Mpixel/frame} \times 3 \text{ B/pixel} = 90 \text{ MB/s}$
- ❖ בכרטיסי קול קצב הנתונים נמוך יותר, אבל יישומים מסויימים דורשים שהייה נמוכה (low latency)