



TEL AVIV UNIVERSITY

# מערכות הפעלה

מרצה: ערן טרומר  
סמסטר א' תשע"ב

## הרצאה 14:

# אתחול מערכת ההפעלה, מכונות וירטואליות

# אתחול מערכת ההפעלה

# מצב יציב של מערכת ההפעלה

## ❖ גרעין

(כולל מנהלי התקנים)

- קוד

- ניהול זיכרון וירטואלי

- תור תהליכים וחשבונאות

- מערכות הקבצים

(mounts, inodes)

- כתובות רשת, טבלאות

ניתוב, חוקי firewall

- שקעים וחיבורי רשת

- חוצצים ומטמונים

- מפתחות הצפנה לדיסק,

מצב RAID

## ❖ תהליכים

(כולל תהליכי שירות של מערכת ההפעלה)

- קוד

- נתונים

- קבצים פתוחים

- הרשאות

## ❖ חומרה

- טבלת שגרות הפסיקה

- טבלאות דפים

- אוגרים מיוחדים

- פסיקות שעון

- תצורת בקרים

- תצוגה גרפית (framebuffer, 3D)

(רשימה חלקית מאד)

# איך מגיעים למצב היציב

- ❖ כאשר מדליקים את המחשב, הזיכרון (DRAM) "ריק" וקוד מערכת ההפעלה נמצא על התקן אכסון או מחשב מרוחק
- ❖ טעינה לזיכרון של הקוד דורשת גישה להתקנים, אבל מנהלי ההתקן עוד לא נטענו...
- ❖ bootstrapping – ריחוף על ידי משיכה בשרוכי הנעליים
- ❖ פתרון:  
שרשרת של פעולות טעינה, שתחילתה בחומרה מזערית וסופה במערכת ההפעלה כולה

# טעינת הגרעין לזיכרון הפיזי

1. ספק הכוח מאותת למעבד על אירוע הדלקה

2. המעבד מאפס אוגרים ומצב פנימי, ומתחיל להריץ קוד בכתובת זיכרון קבועה (לדוגמה  $0\text{xFFFFFFFF}$  ב-x86)

3. על אחד השבבים ליד המעבד, מאוכסן קוד אתחול ראשוני (ROM / BIOS / UEFI). הקוד ממופה אל מרחב הזיכרון, לתחום כתובות הכולל את  $0\text{xFFFFFFFF}$ . קוד האתחול הראשוני ב-BIOS כולל:

- אתחול חומרה בסיסי ובדיקת תקינות (מעבד, בקר מסך, אכסון, מקלדת וכו') - POST
- לעיתים: תפריט תצורה אינטראקטיבי
- מציאה, קריאה והרצה של קטע האתחול הבא, מהתקן אחסון / רשת

# המשך טעינת הגרעין לזיכרון הפיזי

(תרחיש x86 מסורתי)

4. **קטע האתחול** שנטען ע"י ה-BIOS הוא קצר (מאות בתים) ונקרא מתחילת התקן האחסון (Master Boot Record).
- בתורו, הוא מוצא, קורא ומריץ את קטע אתחול נוסף, ארוך יותר.
  - לרוב על פי טבלת המחיצות (partition table) המאוחסנת ב-MBR.
5. **קטע האתחול הנוסף** מתוחכם יותר ומבין מערכות קבצים. הוא מחפש ומריץ קבצים המכילים תוכנית לטעינת הגרעין ותצורה עבורה
- חלונות: C:\NTLDR , C:\BOOT.INI
  - לינוקס: /boot/grub/stage2 , /boot/grub/grub.conf
6. **תוכנית טעינת הגרעין:**
- בוחרת קובץ המכיל קוד גרעין ופרמטרים עבורו, לפי התצורה ובחירת משתמש אינטראקטיבית
  - טוענת את קוד **הגרעין** לזיכרון ומריצה אותו

# אתחול הגרעין

❖ אתחול טבלת שגרות הפסיקה

❖ אתחול הזיכרון הווירטואלי

▪ לאחר הדלקה, המעבד משתמש במיפוי ברירת-מחדל בו כל כתובת לוגית ממופה ישירות לאותה כתובת פיזית (real mode)

▪ הליבה יוצרת טבלת דפים חדשה, כותבת את כתובתה לאוגר מיוחד (CR3) ומפעילה את מנגנון הדפדוף

▪ כעת המעבד ב-protected mode

❖ אתחול מבני נתונים לניהול מסגרות, מעבדים, חוטי גרעין, מנהלי התקן, תהליכים וכו'

❖ לפרטים נוספים: Gustavo Duarte

▪ How Computers Boot Up

<http://duartes.org/gustavo/blog/post/how-computers-boot-up>

▪ The Kernel Boot Process

<http://duartes.org/gustavo/blog/post/kernel-boot-process>

# אתחול מנהלי התקן

- ❖ ה-BIOS מספק מנהלי התקן בסיסיים
  - קוד האתחול ותוכנית טעינת הגרעין הסתמכו על כך
- ❖ כשהמערכת עוברת מ-real mode אל protected mode כבר לא ניתן להריץ קוד BIOS
- ❖ לכן קוד הגרעין הראשוני אשר נטען ע"י תוכנית טעינת הגרעין חייב לכלול את כל מנהלי ההתקן הנדרשים עבור החומרה
  - או לפחות את, כל הנדרשים כדי לקרוא, מהתקן מתאים, את שאר מנהלי ההתקנים
- ❖ אפשרויות:
  - הגרעין מהודר עם כל מנהלי ההתקן הנדרשים
  - `initrd`: יחד עם הגרעין טוענים קובץ נוסף המכיל מנהלי התקן לטעינה דינמית. קובץ זה נתפר עבור המכונה, על פי החומרה הנמצאת בה, כחלק מהתקנת מערכת ההפעלה.



# אתחול מערכת הקבצים

- ❖ אחד מהתקני האכסון (לדוגמה, מחיצה בדיסק) נבחר כראשי
  - עפ"י פרמטר שניתן לגרעין ע"י תוכנית העלאת הגרעין, או ברירת מחדל
- ❖ הגרעין ניגש להתקן, פותח את מערכת הקבצים שבו, ומציב אותה למרחב השמות
- ❖ כעת ניתן לקרוא ולהריץ מנהלי התקנים נוספים, ותהליכי משתמש.

# אתחול וכיבוי עולם תהליכי המשתמש

## (בלינוקס)

- ❖ הליבה מריצה את התוכנית המיוחדת `/sbin/init` כתהליך משתמש מספר 1, תחת הרשאות `.root`.
- ❖ התוכנית `init` קוראת קבצי תצורה מהספרייה `/etc` ומשלימה את האתחול, לדוגמה:
  - הצבה של מערכות קבצים נוספות
  - איכלוס הספרייה `/dev`
  - הרצת תהליכי שירות מערכת (יומן ארועים, שרותי רשת וכו')
  - הרצת תוכניות להתחברות משתמשים (טקסטואלי, גרפי, רשת וכו')
- ❖ בבוא היום, `init` גם מנהלת כיבוי מסודר של המערכת על ידי הפסקת תוכניות המשתמש וניתוק מערכות הקבצים בסדר הנדרש.

# מכונות וירטואליות

# בעיה בריבוי משתמשים מסורתי

❖ במערכות מרובות משתמשים, כל המשתמשים חולקים סביבה משותפת: סוג ותצורת הגרעין, שרותי מערכת, גרסאות תוכנה מותקנות, כתובות רשת, מרחב שמות קבצים ועוד.

❖ לפעמים הצרכים שונים ובלתי תואמים:

- אילו תוכנות וספריות צריך להתקין, ואיזה שרותים להפעיל?
- איזו גרסה? מתי לשדרג?
- קבצי תצורה גלובליים (`httpd.conf`, תצורת DNS)
- קביעת הרשאות מערכת
- ניטור יומני אירועים

❖ בעיית אבטחה: יותר מדי הזדמנויות לטעויות תכנות או תצורה אשר יאפשרו למשתמש אחד להזיק למשתמש אחר

# שני תפקידים למערכת ההפעלה

הכרחי לריבוי  
משתמשים

## ❖ הפרדה בין המשתמשים

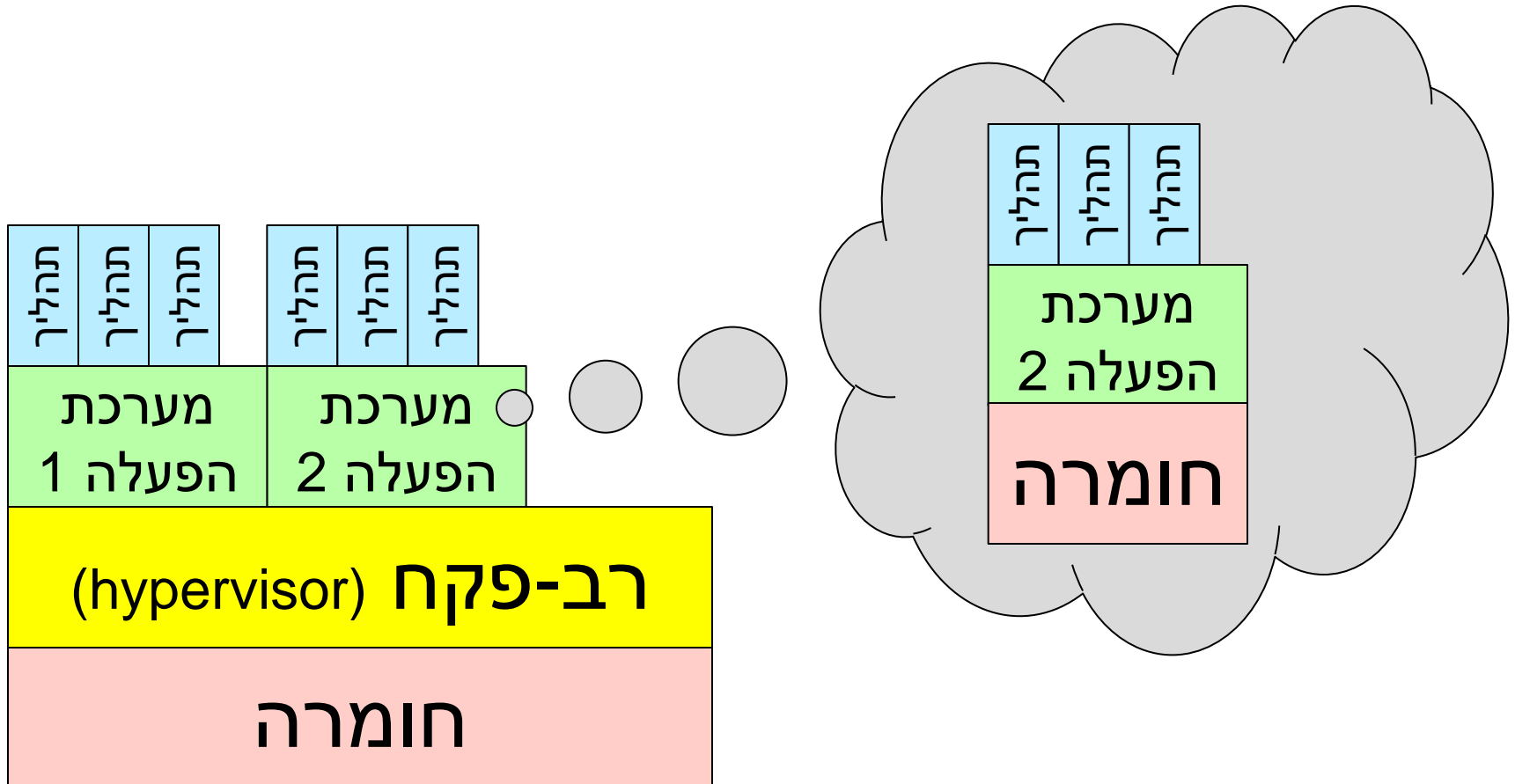
- זיכרון וירטואלי
- מניעת גישה ישירה לחומרה
- מיתוג זמן ריצה בין משתמשים

אולי שכל משתמש  
יסתדר בעצמו, על פי  
צרכיו?

## ❖ העלאת רמת ההפשטה

- מנהלי התקן
- קבצים, מערכות קבצים
- שקעים, פרוטוקולי רשת
- תהליכים וחוטים
- אמצעי סנכרון
- ספריות קוד

# מכונות וירטואליות

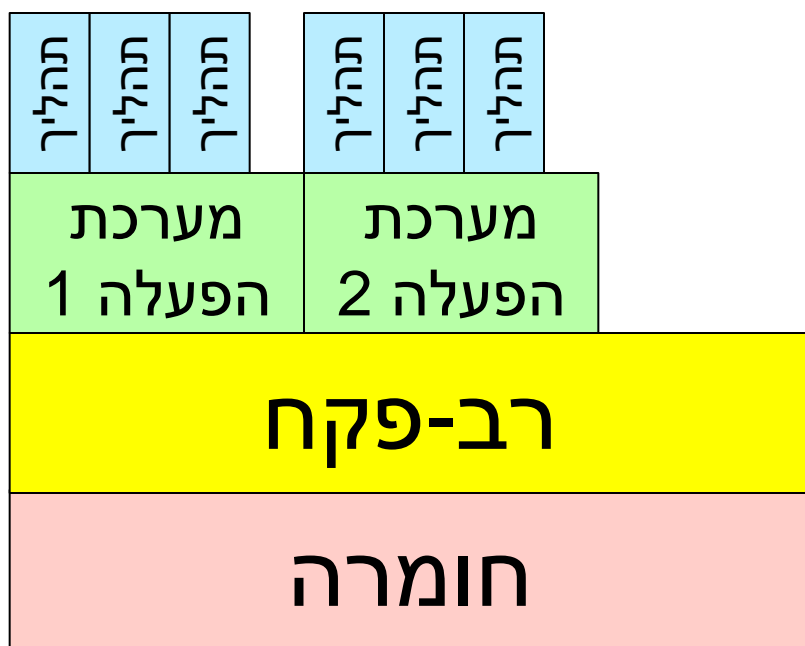


# מכונות וירטואליות: שימושים נפוצים

- ❖ הרצת תוכנות שנכתבו למערכת הפעלה אחרת
  - דוגמה, אופיס לחלונות במכונה וירטואלית תחת לינוקס
- ❖ שרתי אינטרנט משותפים (virtual hosting)
- ❖ איחוד שרתים אירגוניים
  - שרת הקבצים, שרת הדואר, שרת ההדפסה וכו' רצו על מכונות פיזיות נפרדות עם תצורה שונה; אפשר לאחדם
- ❖ שרותי מחשוב ענן מסוג Infrastructure as a Service: משתמשים יכולים ליצור בקלות מכונה וירטואלית חדשה "אי שם" בדיוק בתצורה הרצויה, ולשלם לפי שעה

# מכונה וירטואלית מסוג 1

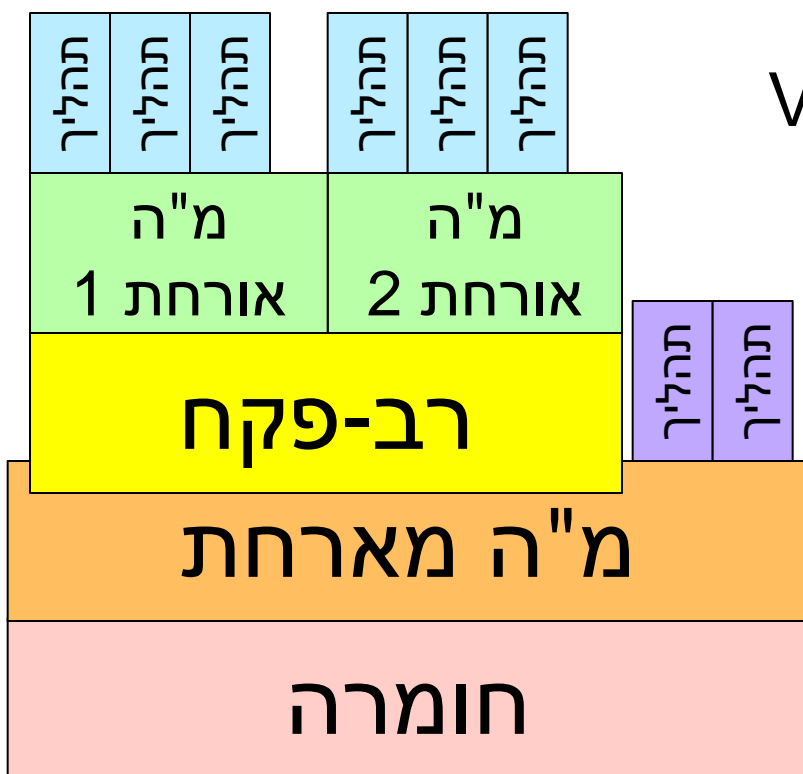
- ❖ הרב-פקח הוא שכבת תאום מינימלית המממשת רק הפרדה בין משתמשים. נקרא גם Virtual Machine Manager.
- ❖ דוגמאות: Vmware ESX , IBM VM/370





## מכונה וירטואלית מסוג 2

❖ הרב-פקח הוא מודול במערכת הפעלה כללית, ומערכות הפעלה האורחות רצות כתהליכים במערכת ההפעלה המארכת



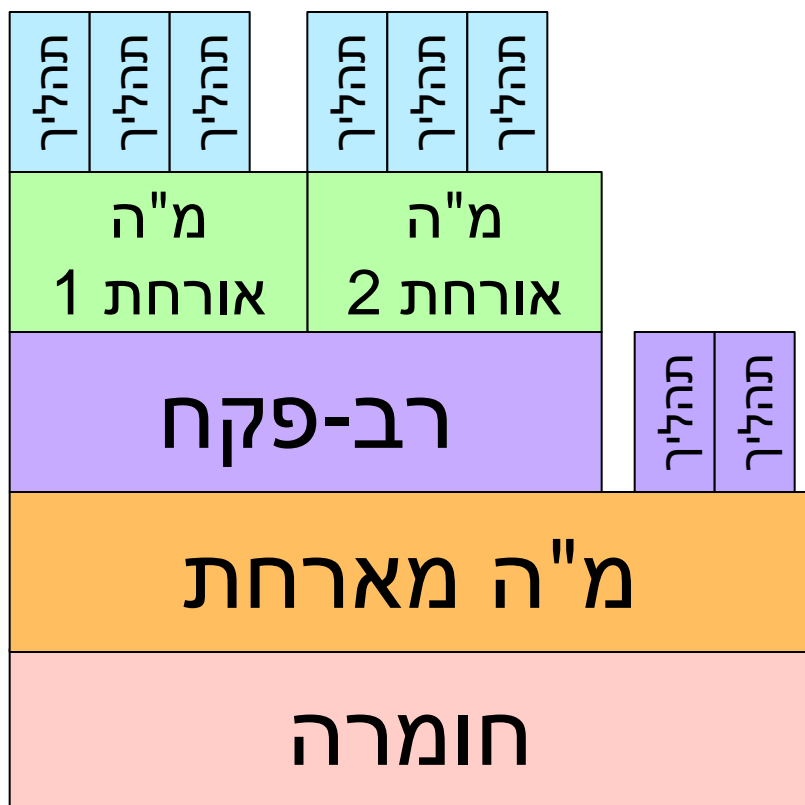
❖ דוגמאות: VMware Workstation, VirtualBox, KVM

❖ יוצרים הפשטה ואז עוטפים אותה מחדש בממשק פרימיטיבי

- הדיסק הווירטואלי שהאורחת רואה הוא מעטפת סביב קובץ במערכת הקבצים של המארכת, עם DMA ופסיקות מזוייפים

# אמולטור

- ❖ מקרה קיצוני של סוג 2:  
המכונה הוירטואלית רצה  
כתהליך רגיל ללא סיוע הגרעין  
המארח, לדוגמה ע"י אמולציה  
פקודה-אחר-פקודה של  
החומרה הווירטואלית.
- ❖ מאפשר להריץ אורחות  
בארכיטקטורת מעבד שונה  
מהמארכת
- ❖ חסרון: ביצועים
- ❖ דוגמאות: Bochs, QEMU



# מינוח

בהקשרנו, מכונה וירטואלית היא

❖ לא "מחשב וירטואלי" במובן של תהליך עם קריאות מערכת

❖ לא "מכונה וירטואלית" במובן של מפרש לשפת תכנות

▪ Java Virtual Machine, למרות שמו, כלל אינו דומה למכונה פיזית,

אלא מספק אבסטרקציה גבוהה מאד

# וירטואליזציה

## ❖ המערכת הווירטואלית מריצה קוד

- במצב בלתי-מיוחס וירטואלי: אפשר פשוט להריץ במצב בלתי-מיוחס פיזי (עם מיפוי זיכרון נכון וטיפול בקריאות מערכת וחריגים)
- במצב מיוחס וירטואלי: חייב לרוץ פיזית במצב בלתי-מיוחס פיזי

## ❖ הקוד המיוחס הווירטואלי ינסה לבצע פעולות רגישות, לדוגמה:

- קלט/פלט
- שינוי טבלאות דפים
- שינוי אוגרים מיוחדים
- שינוי טבלאות שגרות הפסיקה או כיבוי מנגנון הפסיקות
- כתיבה לדפים השייכים לגרעין המארח (מסומנים מצב-מיוחס-בלבד)
- ❖ בכל אירוע כזה, הקוד האורח יעצר ע"י חריג. הגרעין המארח יקבל פיקוד וישלים את הפעולה באמצעות רמאות מתאימה.

# וירטואליזציה של קריאות מערכת וקלט/פלט

❖ תהליך אורח מבצע קריאת מערכת write

← פסיקת תוכנה

- הגרעין המארח מפעיל שגרת טיפול של הרב-פקח

- הרב-פקח מבין מה קרה, ומפעיל את שגרת פסיקות התוכנה של הגרעין האורח

❖ הגרעין האורח מתרגם את הבקשה (דרך מערכת הקבצים ומנהל ההתקן שלו) לפקודת פלט לבקר דיסק וירטואלי

← חריג

- הגרעין המארח מפעיל שגרת טיפול של הרב-פקח

- הרב-פקח מבין שנעשה ניסיון לכתוב לבקר דיסק וירטואלי העוטף קובץ במערכת הקבצים המארכת, ומתרגם את הפעולה לשגרה המתאימה של מודול מערכת הקבצים

- מודול מערכת הקבצים בגרעין האורח מעתיק את המידע לחוצץ, וממשיך את ריצת הגרעין האורח.

❖ אחרי שכתב את כל המידע אל ה-"דיסק", הגרעין האורח ממשיך את ריצת התהליך האורח.

# ארכיטקטורות פגומות

- ❖ בארכיטקטורות נפוצות רבות, יש בעיה של פעולות רגישות אשר לא יגרמו לחריג מתאים, אלא יכשלו בשקט או ישנו את משמעותן. לכן הליבה האורחת לא תפעל נכון.
  - לדוגמה, ב-x86: פקודות לכיבוי מנגנון פסיקות נכשלות בשקט במצב בלתי-מיוחס.
- ❖ פתרונות:
  - תרגום קוד בזמן ריצה. כל פעם שקוד חדש רץ, הוא "יתוקן" על ידי הרב-פקד כך שפקודות בעייתיות יוחלפו בפסיקות תוכנה.
  - גרסאות משופרות של הארכיטקטורה, כגון VT-x ו-AMD-V, אשר מוסיפות את כל החריגים המתאימים
  - מה יותר יעיל?

לפרטים נוספים: [en.wikipedia.org/wiki/X86\\_virtualization](http://en.wikipedia.org/wiki/X86_virtualization)

# paravirtualization

❖ גישה נוספת: האורחים יתנזרו מפעולות רגישות, ובמקום אלו יפנו בצורה מסודרת אל הרב-בקר בעזרת פסיקת תוכנה יזומה, על פי ממשק שמוצע על ידי הרב-בקר.

▪ אין בעיה של ארכיטקטורות פגומות

▪ אפשר לייצג בקשות ברמה גבוהה יותר, לכן תקורה נמוכה יותר

❖ חסרון: צריך להדר מחדש את הגרעין האורח

❖ דוגמה: Xen

כעת כל האורח רץ כולו, בIODעין, במצב משתמש. דומה עקרונית לתהליך רגיל במערכת הפעלה סטנדרטית, אבל ממשק קריאות המערכת שונה מאד.

# וירטואליזציה של זיכרון וירטואלי

❖ שלוש רמות של כתובות זיכרון:

- פיזי

- וירטואלי-במארח = פיזי-באורח

- וירטואלי-באורח

❖ מה עושים כאשר הגרעין האורח משנה את טבלאות הדפים שלו?

- פתרון תוכנה: חשבונאות כפולה. הרב-בקר יתפוס את הכתיבה ויעדכן "טבלת צללים" שממפה כתובת וירטואלית-באורח לפיזית. האוגר במעבד יצביע לטבלת הצללים.

- פתרון חומרה: תרגום כפול, דרך טבלת האורח ואז דרך טבלת המארח.

לפרטים נוספים: [Tanenbaum, Modern Operating Systems](#)



# פלאי המכונות הווירטואליות

❖ הקפאה, הפשרה ושמירת snapshots

❖ שכפול מכונות וירטואליות

❖ הגירה של מכונות וירטואליות בין מכונות פיזיות

▪ אמינות

▪ ניצול משאבים ע"י השמה מיטבית של מכונות וירטואליות לפיזיות

❖ הקלטת של ריצה ושחזור (לדוגמה לצורך ניתוח כשלים)

❖ הפצת "מכשירים וירטואליים" – מצב מכונה שלם המבצע משימה

כל אלו אפשריים גם בגישה מסורתית של תהליכים, אבל קשים הרבה יותר לביצוע בגלל הסמנטיקה המורכבת של ממשק קריאות המערכת.

**- דיון מסכם -**