

**פרק 10 וחצי**  
**וירטואליזציה**

# מה יותר חד?



או המציאות שרואים דרך החלון?

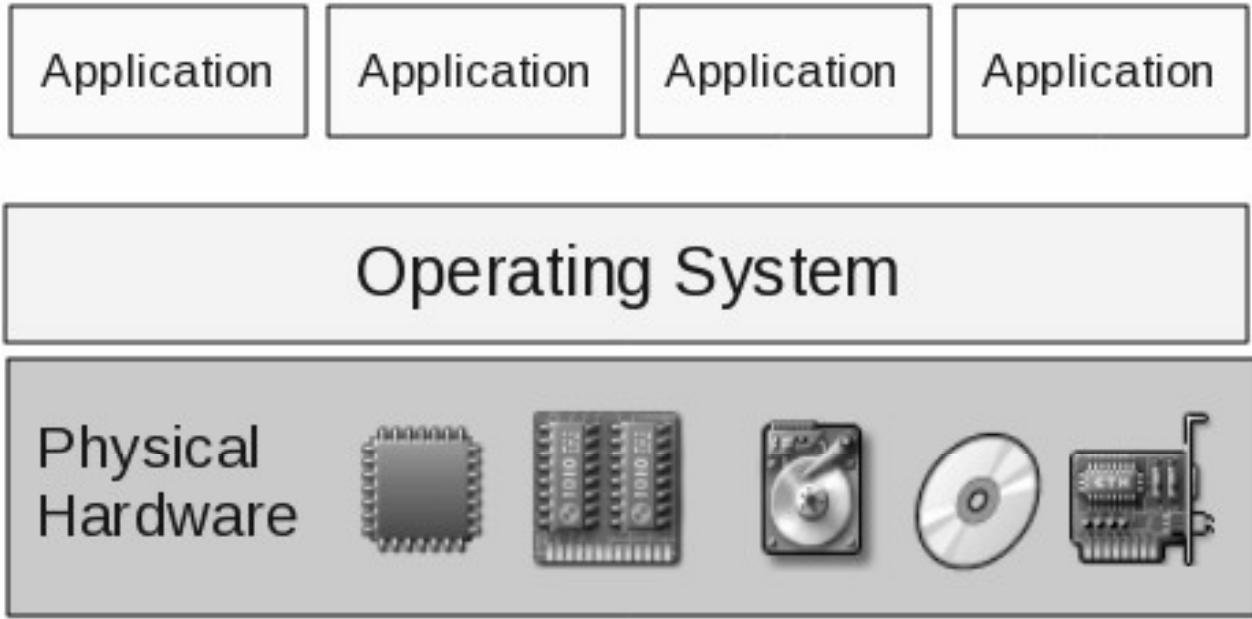
# מה יותר חד?



או המציאות שרואים דרך החלון?

לפעמים נדמה שהגרסה הוירטואלית עולה על המקור

# בתיאוריה,



Ring 3 = **user mode**

Ring 1 & 2

Ring 0 = **kernel mode**

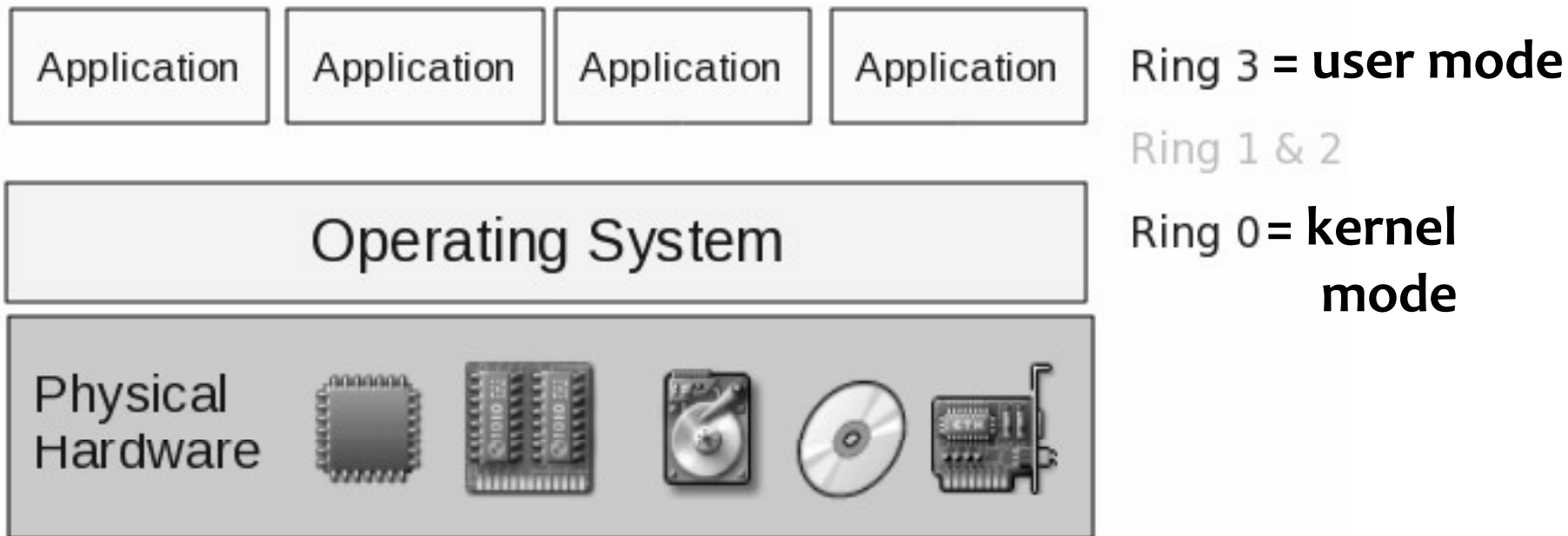
❖ מערכת ההפעלה מגינה על תהליכים ומפרידה אותם זה מזה (protection, isolation)

❖ יש מערכת הפעלה אחת שתחתה אפשר להריץ את כל היישומים

# אבל בפועל,

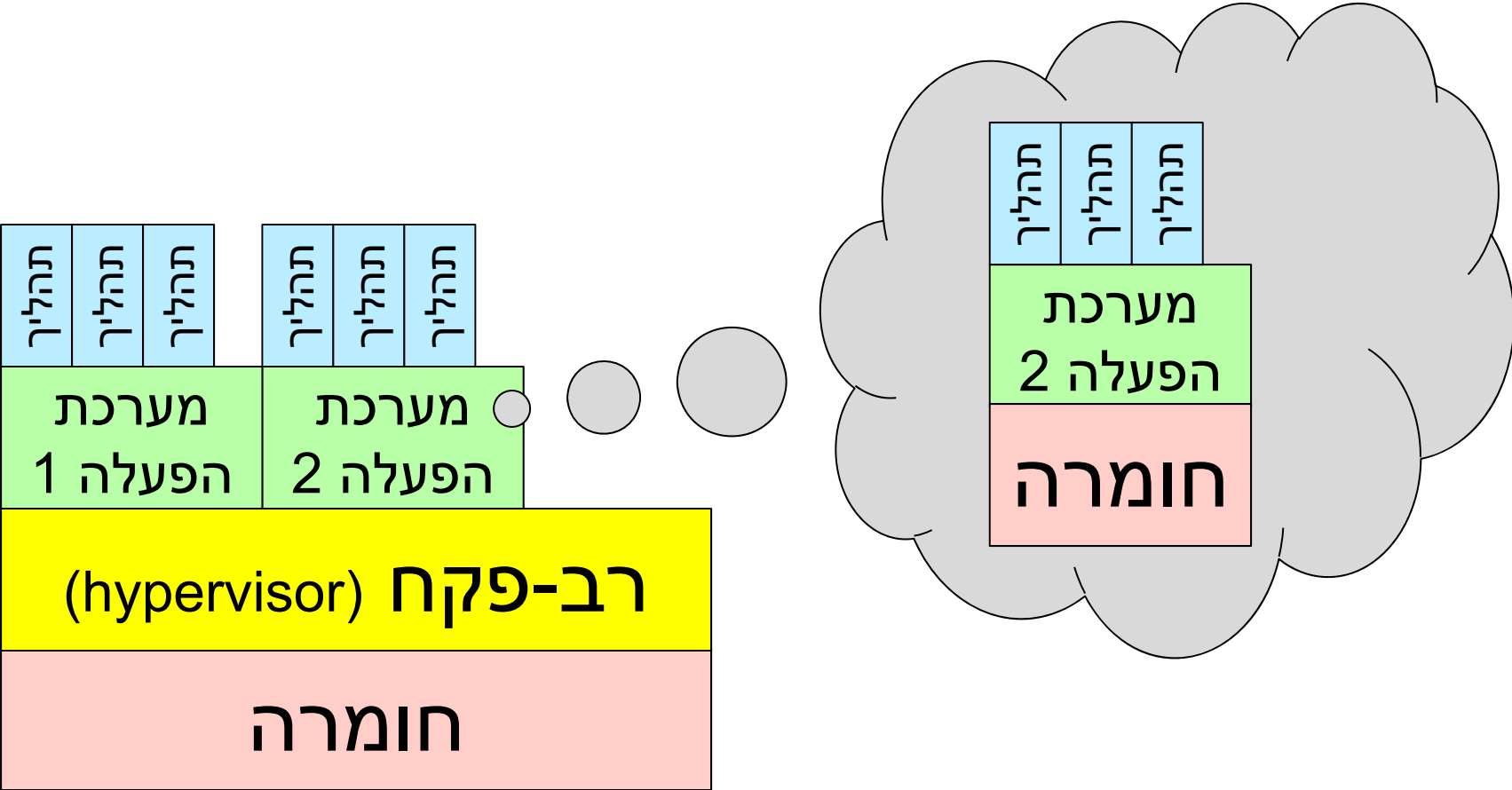
- ❖ הפרדה בין תהליכים לא מושלמת ולא מספיקה במצבים מסויימים, למשל כאשר צריך להריץ תוכנות של ארגונים שונים על אותו מחשב
- ❖ יש תוכניות שמניחות שרק עותק אחד שלהן רץ במחשב אחד (שרתי HTTP, מסדי נתונים)
- ❖ יש יישומים שרצים רק בחלונות, או רק בלינוקס, או רק ב-MacOS, או אפילו רק בגרסה מסויימת של אחת מהן, למרות שכולן רצות על אותה חומרה בדיוק

# בעצם אנחנו רוצים יותר: נדידה



- ❖ אם תוכנית רצה יכולה לנדוד בין מחשבים, זה מאפשר לנהל צבירי מחשבים בצורה טובה (ביצועים, צריכת אנרגיה), לשדרג, וכו'
- ❖ מערכות הפעלה שגרתיות לא תומכות בנדידה (migration), למרות שפרוייקטי מחקר (MOSIX) הראו איך לעשות את זה

# וירטואליזציה תציל את המצב



# שימושים נפוצים לוירטואליזציה

❖ 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:

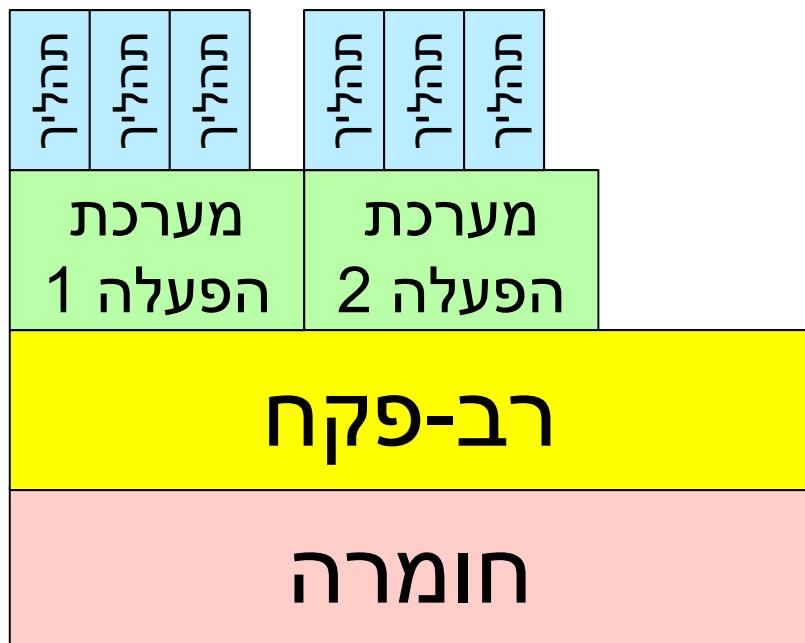
משתמשים יכולים ליצור בקלות מכונה וירטואלית חדשה "אי שם"

בדיוק בתצורה הרצויה, ולשלם לפי שעה



# IBM עשתה את זה בשנות ה-70

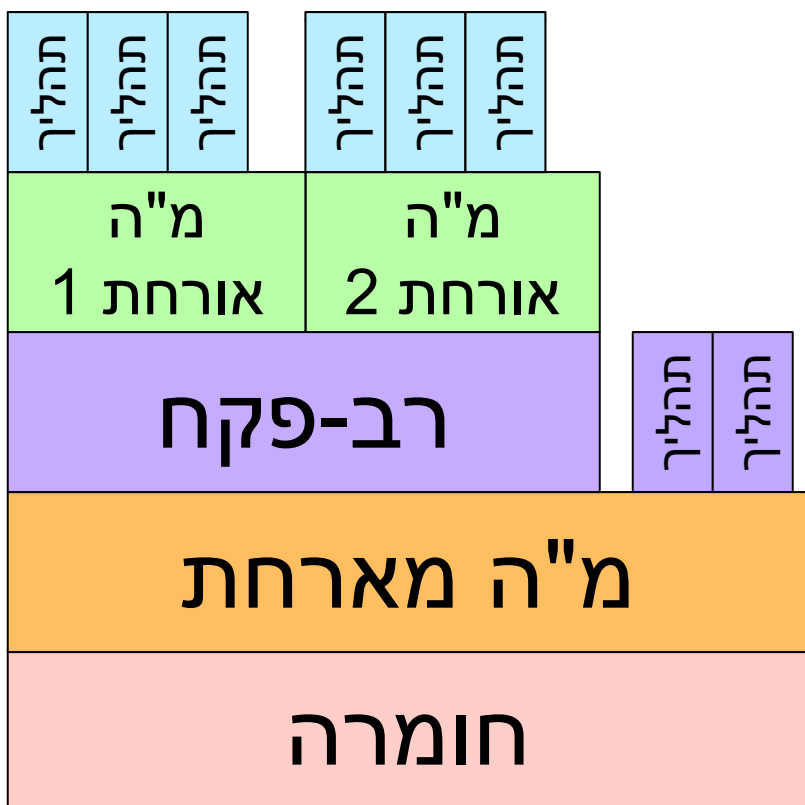
- ❖ נכתוב מערכת הפעלה שבדרך כלל רצה במצב מיוחד
- ❖ נריץ אותה במצב משתמש, על מעבד שזורק חריג בכל פעם שתוכנית מנסה להריץ פקודת מכונה שמותר להריץ רק במצב מיוחד (full virtualization)
- ❖ הטיפול בחריג מתבצע בשכבת הווירטואליזציה (רב-פקח, hypervisor) שמפענח מה מערכת ההפעלה ניסתה לעשות, בודק אם מותר לה, ואם כן עושה את מה שצריך לעשות



# אבל לא כל מעבד תומך בזה

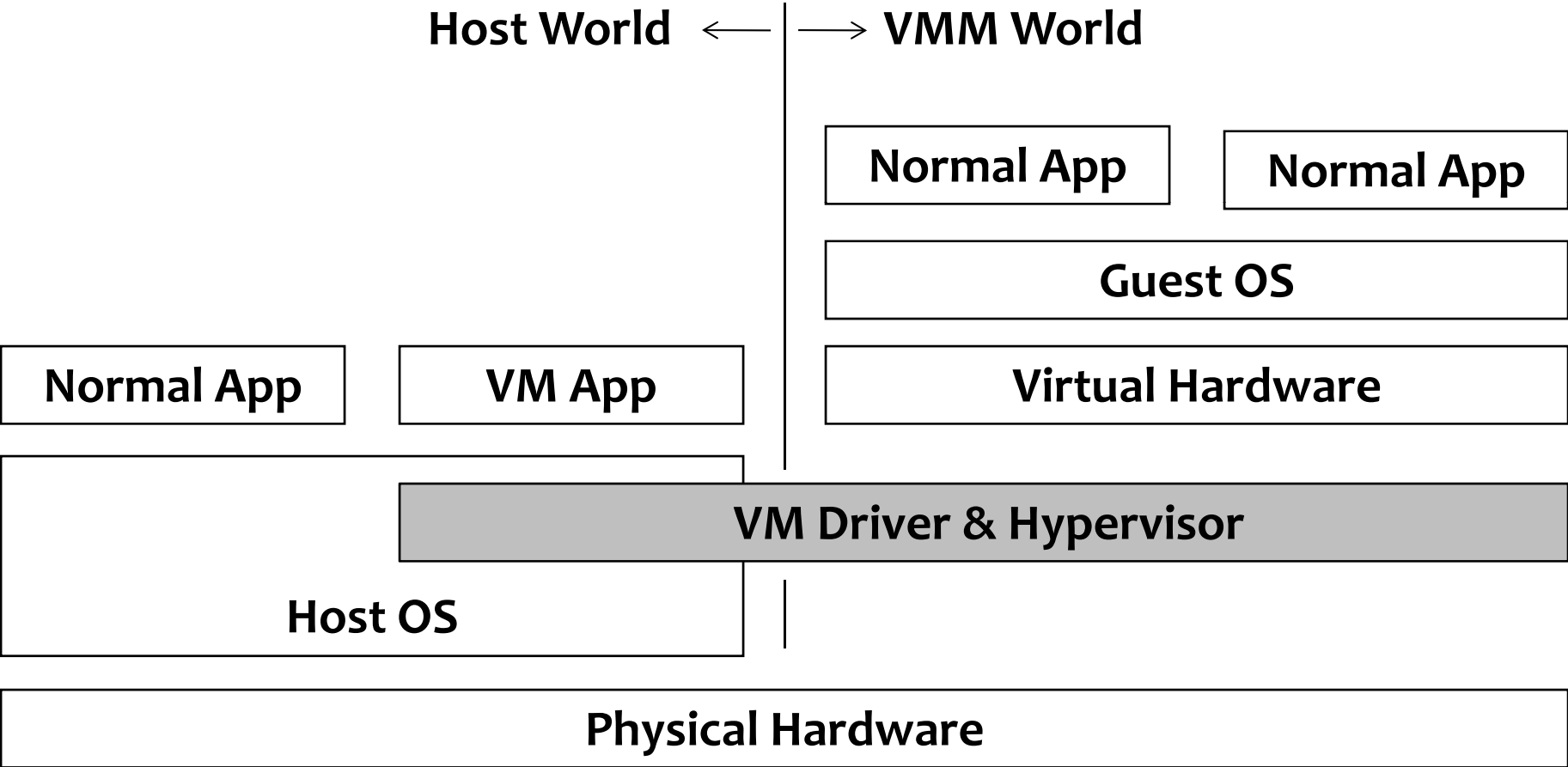
- ❖ מחשבים אישיים זולים גרמו לחוסר עניין בוירטואליזציה במשך הרבה שנים
- ❖ למה לחלוק אם אני יכול להרשות לעצמי מחשב פרטי?
- ❖ אבל יש לשאלה הזו תשובות טובות, ווירטואליזציה חזרה לאופנה
  
- ❖ אבל המעבדים הדומיננטיים בנקודת הזמן הזו (מעבדי x86) לא תמכו בחריג בכל נסיון להפעיל פקודה שמותר רק במצב משתמש; חלק מהפקודות פשוט לא התבצע (בשקט, בלי חריג)

# חיקוי (emulation)



- ❖ תוכנה (רגילה) מסמלצת מעבד ואת כל שאר ההתקנים
- ❖ כל פקודת מכונה מפוענחת בתוכנה הזו ומבוצעת מול מצב מעבד (משתנים שמייצגים אוגרים וכו')
- ❖ מאפשר וירטואליזציה על כל מעבד, אבל איטי
- ❖ דוגמאות: Bochs, QEMU

# הפתרון של VMWare (1998)



# הפתרון של VMWare (1998)

- ❖ **שכתוב קוד בינרי:** כאשר קוד בינרי נטען לזיכרון, הרב-פקח משכתב אותו כך שפקודות בעייתיות מוחלפות בקריאה לרב פקח
- ❖ פקודות כאלה הן נדירות, ולכן רוב פקודות המכונה מטופלות

ישירות על ידי החומרה; ביצועים טובים

- ❖ הרב-פקח יכול לרוץ ישירות על

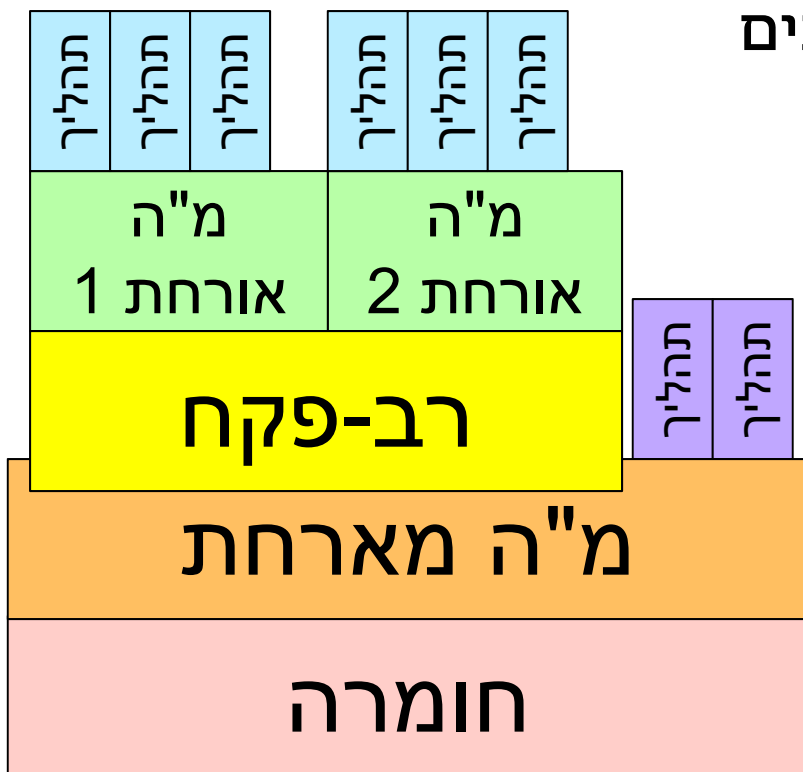
החומרה, או שהוא יכול לרוץ כחלק ממערכת הפעלה מארחת

- ❖ לא נדרש שינוי של האורחות או

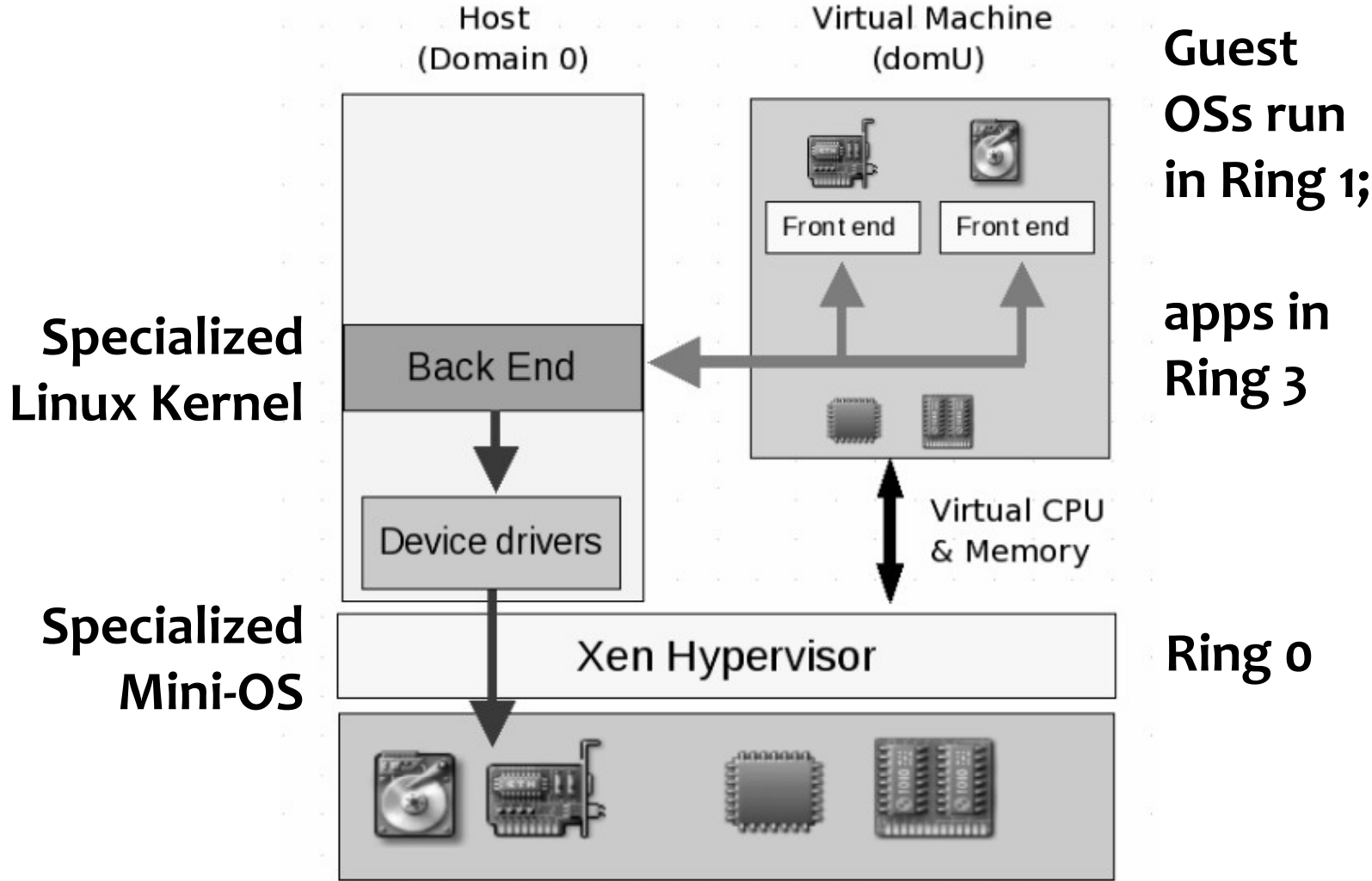
המארחת

- ❖ שכתוב מסובך, אבל תסבוכת חד

פעמית



# פאראוירטואליזציה: הפתרון של Xen



# פאראוירטואליזציה: הפתרון של Xen

- ❖ ניצול העובדה שמערכות הפעלה קיימות צריכות רק 2 טבעות הגנה מתוך ה-4 שיש במעבדי אינטל
- ❖ דורש שינוי קטן בקוד המקור של מערכות הפעלה אורחות (כך שירוצו בטבעת 1 ולא 0); בפרט, דורש גישה לקוד המקור
- ❖ דורש רב פקח ייעודי שאינו מערכת הפעלה קיימת אבל מספק שירותים דומים (ניהול זיכרון ומעבדים)
- ❖ דורש אורחת אחת מיוחדת (domain 0) שמנהלת קלט/פלט
- ❖ לא צריך שכתוב קוד

# אם כל התסבוכת מיועדת להתגבר על פגם בארכיטקטורה

❖ אז כדאי לתקן את הארכיטקטורה של המעבדים

❖ אינטל ו-AMD עשו את זה בסביבות 2003

❖ הוסיפו תמיכה בוירטואליזציה מלאה במעבדים שלהם, ויותר  
מאוחר גם ברכיב החומרה שמבצע תרגום כתובות (MMU)

❖ דגל נוסף במעבד (נוסף על משתמש/מיוחס) מציין האם המעבד  
מריץ כרגע מערכת הפעלה אורחת או מארחת, וזורק חריגים  
בהתאם

❖ מאפשר להריץ מערכות אורחות בלי שכתוב בינרי ובלי שכתוב של  
קוד המקור של מערכת ההפעלה האורחת

❖ מנוצל ב-KVM וגרסאות חדשות של סביבות וירטואליזציה אחרות



# וירטואליזציה של זיכרון וירטואלי

❖ שלוש רמות של כתובות זיכרון:

▪ וירטואלי-באורחת

▪ פיזי-באורחת = וירטואלי-במארחת

▪ פיזי

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

❖ פתרון תוכנה: חשבונאות כפולה. הרב-פקח יתפוס את הכתיבה ויעדכן "טבלת צללים" שממפה כתובת וירטואלית-באורחת לפיזית

❖ האוגר במעבד יצביע לטבלת הצללים

❖ פתרון חומרה: תרגום כפול, דרך טבלת האורח ואז דרך טבלת

המארח

# שימושים לא שגורתיים

- ❖ 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.

# הרהורים על תיאוריה ומעשה

- ❖ הטכנולוגיה הזו התפתחה בעיקר כי מערכות הפעלה קיימות לא הצליחו לעשות את כל מה שהמפתחים שלהם הבטיחו, בפרט לשמור על הפרדה טובה בין תהליכים ומשתמשים; אם היינו מצליחים לקיים את כל ההבטחות, לא היה צריך וירטואליזציה
- ❖ אבל אולי לא חשוב מה מערכות הפעלה היו צריכות לעשות; אפשר לגלות מה הן כן עושות טוב, להשתמש בהן כאבני בניין, ולהשיג את כל השאר בדרך אחרת (וירטואליזציה)
- ❖ אחרי הפיתוח של הדרך האחרת, מתברר שיש לה עוד שימושים