

# תרגול 1

---

מטה-דיון על הקורס "מערכות הפעלה"  
מהי מערכת הפעלה?  
עקרון הווירטואליזציה  
מערכת ההפעלה "לינוקס"

# מטה-דיון על הקורס "מערכות הפעלה"

---

# נהלי הקורס

- מפורט בסילבוס שבמודל

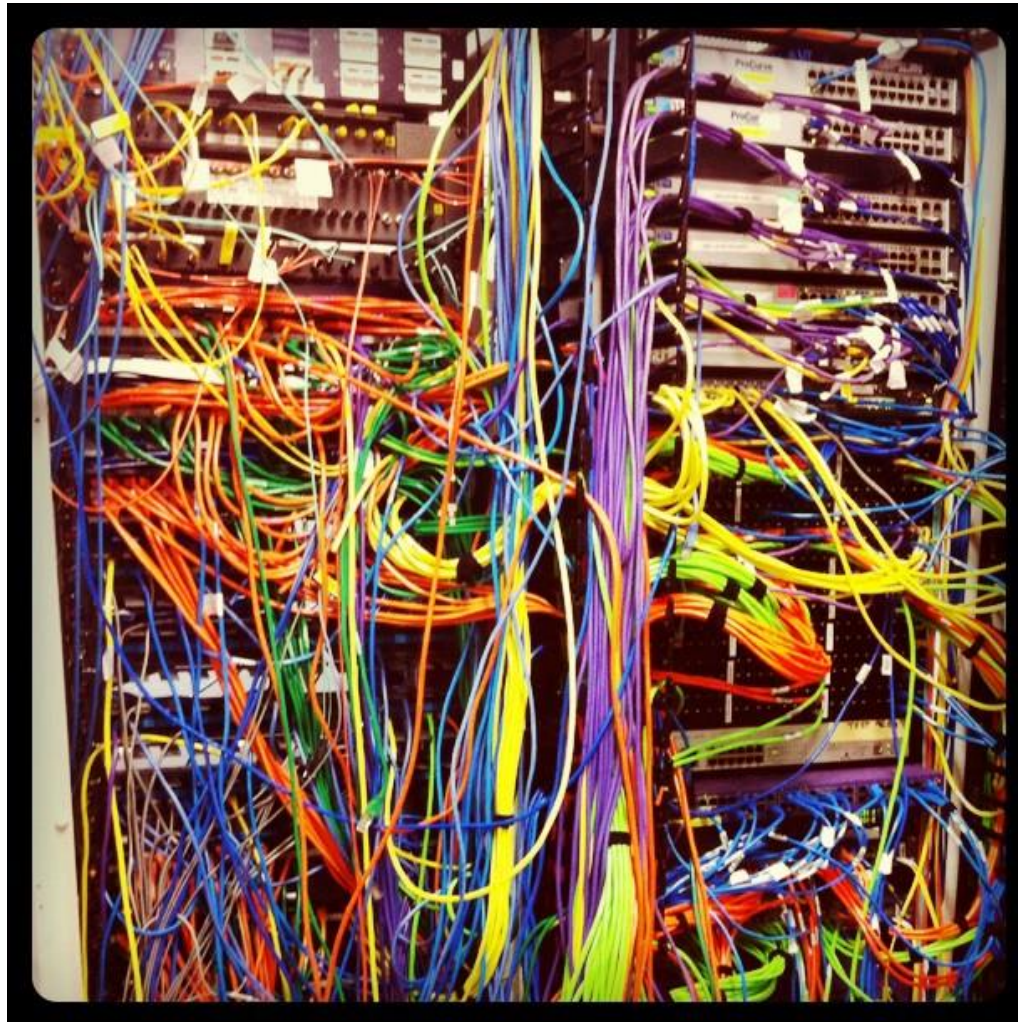
- תרגילי בית:

- 5 תרגילי בית, zero indexed

- הגשה בזוגות בלבד

- 25% תקף

# מערכות הפעלה ...be like



# למה הקורס קשה?

- המון חומר, המון קשרים בין הנושאים השונים.
- אין "גבולות גזרה" מוגדרים, לא ניתן לתפוס את כל המערכת בבת-אחת.
- תרגילי בית טכניים בסביבת עבודה "קשוחה".
- אין debugger, עבודה בטרמינל, מכונה וירטואלית, קוד ספגטי עם goto, ...

במערכות הפעלה	בחדו"א
שיפור של $\epsilon$ קטן לא מעניין	יהי $0 < \epsilon \dots$
תיאוריות המבוססות על תצפיות ועל היגיון	משפטים, אמת מוחלטת
מנסים למצוא פשרה בין מספר דרישות הנדסיות	מחפשים פתרונות אופטימליים

# למה בכל זאת כדאי?

- אין ברירה – קורס חובה...
- של 5 נ"ז!

- כי זה מעניין!
- לפחות אותנו ;)

- כדי לדעת איך עובדות מערכות מחשבים.

- מערכות מחשבים == המחשב הנייד שלכם, הפלאפון, וגם השרת של גוגל שעונה לשאילתות שלכם.

- כדי לעבור ראיונות עבודה באינטל, מיקרוסופט, מלאנוקס, וכו'.

- מצד שני, אם רוצים להיות מפתחי web, לא חייבים לדעת "מערכות הפעלה".

# אוקיי, השתכנעת... אז איך מצליחים בקורס?

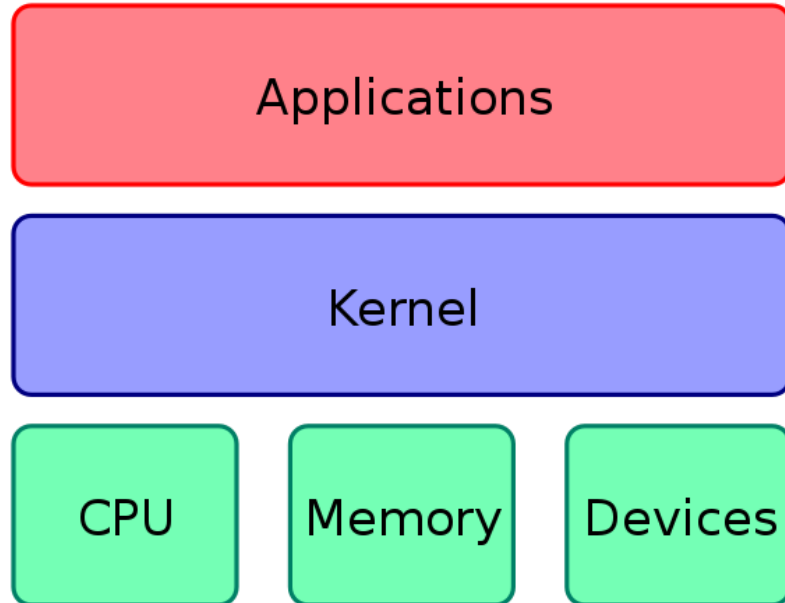
- אין חוכמות: משקיעים הרבה זמן ומאמצים.
- לומדים בצורה ספירלית.
- חוזרים לנושאים הקודמים בכל פעם שלומדים נושא חדש.
- לא מפחדים לשאול שאלות, לענות תשובות, ולטעות.
- חיוני כדי להתרגל לשפה החדשה.
- לא סובלים בגלל שיעורי הבית.
- נתקעתם? תשאלו חברים או אותנו.
- להצליח לבד זה טוב; ללמוד מאחרים זה עוד יותר טוב.

# מהי מערכת הפעלה?

---


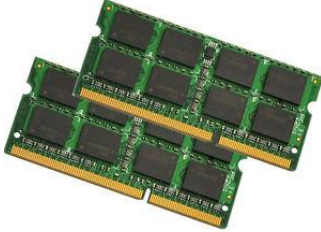



# מהי מערכת הפעלה?

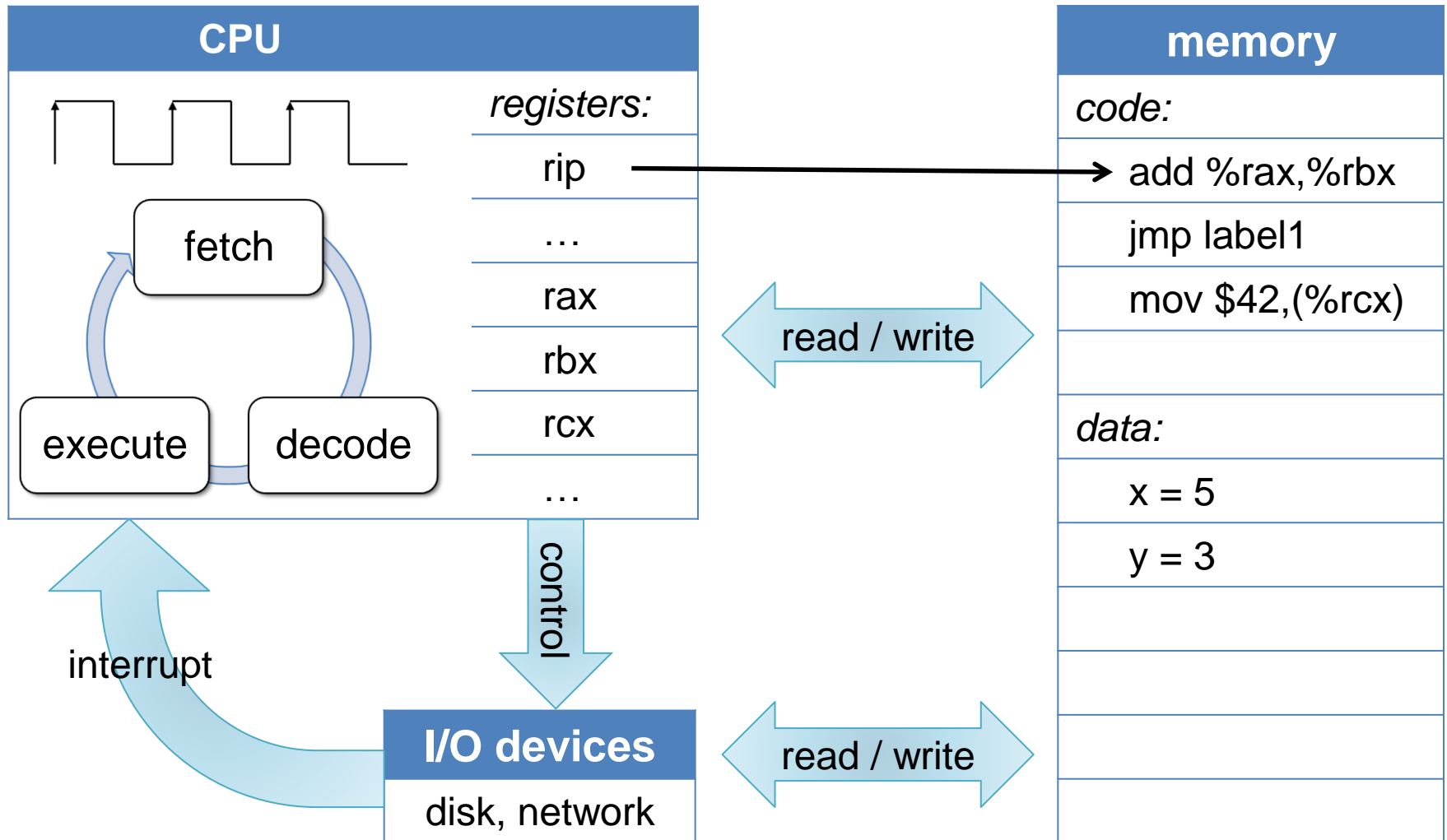


- מערכת תוכנה אשר אחראית על ניהול החומרה עבור המשתמשים.
- ליתר דיוק, זו ההגדרה של "גרעין" מערכת ההפעלה.
- שלושת רכיבי החומרה המרכזיים הם:
  - (1) המעבד,
  - (2) הזיכרון,
  - (3) והדיסק.
- כל תרגול בקורס עוסק בניהול של אחד הרכיבים הללו.

# יישור קו

		
<p><b>דיסק</b></p> <p>אמצעי אחסון איטי ועמיד (המידע נשמר גם אם מכבים את המחשב)</p>	<p><b>זיכרון</b></p> <p>אמצעי אחסון מהיר ונדיף (המידע נמחק כאשר מכבים את המחשב)</p>	<p><b>מעבד</b></p> <p>מקבל זרם של פקודות מכונה (אסמבלי) ומבצע אותן בצורה סדרתית</p>
<p>המעבד לא יכול לעבד מידע ישירות על הדיסק, הוא צריך קודם להעביר את המידע לזיכרון</p>	<p>הזיכרון נגיש לכל הליבות של אותו מחשב</p>	<p>כל מעבד מורכב ממספר ליבות חישוב עצמאיות שרצות במקביל</p>
<p>נפח אופייני – 1TB זמן גישה טיפוסי: 1 ms</p>	<p>נפח אופייני – 8GB זמן גישה טיפוסי: 100 ns</p>	<p>תדר אופייני – 3GHz זמן גישה טיפוסי: 1 ns (רגיסטרים + מטמונים)</p>

# יחסי הגומלין בין רכיבי החומרה



# תפקידי מערכת ההפעלה

- לחלק את משאבי החומרה בצורה יעילה והוגנת בין המשתמשים.
- האתגר של מערכת ההפעלה הוא לצרוך משאבים מועטים ככל הניתן.
- להציג למשתמשים אבסטרקציות וממשקים (API) כדי להקל את הפיתוח של אפליקציות.
- אבסטרקציות לדוגמה: תהליך, מרחב זיכרון, קובץ, socket, ...
- אוסף השירותים (== הממשק) שמערכת הפעלה מציגה נקרא קריאות מערכת.
- להגן על המידע של המשתמשים מפני משתמשים ו/או תוכניות אחרות זדוניות.
- לשמור על המידע של המשתמשים מפני נפילות חומרה.

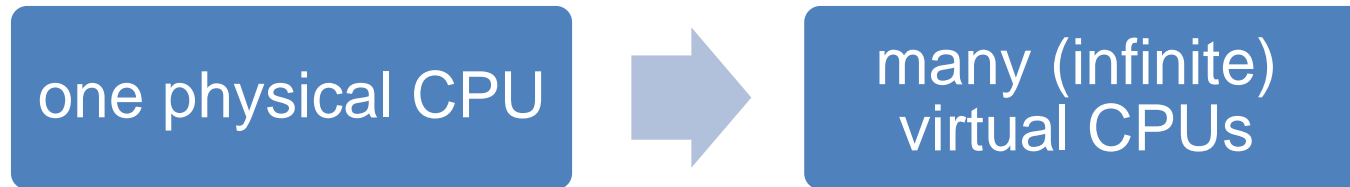
# עקרון הווירטואליזציה

---

# הבעיה: חלוקת משאבים

- ליוסי יש מעבד יחיד וזיכרון יחיד.
- אבל יוסי רוצה להריץ הרבה אפליקציות בו-זמנית.
  - לגלוש באינטרנט, לשמוע מוזיקה, ולעבוד על שיעורי הבית במערכות הפעלה.
- הפתרון: וירטואליזציה של משאבים פיזיים.
  - מערכת ההפעלה מציגה למשתמש גרסאות וירטואליות של המעבד והזיכרון.
  - משאבים וירטואליים מספקים **אבסטרקציה** שמתעלמת מפרטי המימוש של החומרה הספציפית, ולכן הם פשוטים יותר לשימוש.
  - משאבים וירטואליים גם מספקים **הגנה** כי הם מסתירים את המשאב הפיזי, וכך אף משתמש או אפליקציה לא יכולים להשתלט עליו.
  - אבל וירטואליזציה – כמו כל אבסטרקציה – גם מוסיפה **תקורה** (overhead) שפוגעת בביצועים.

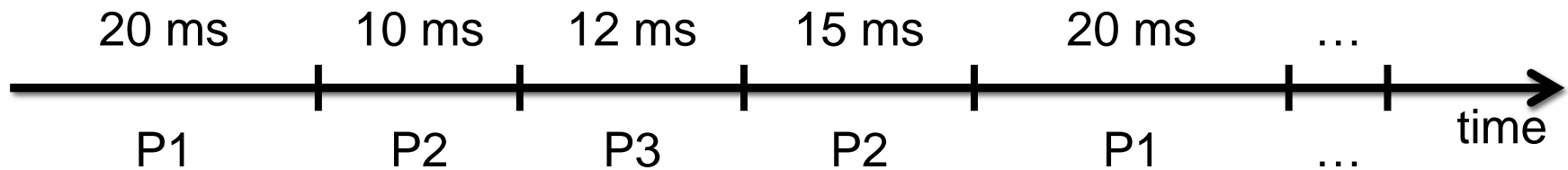
# וירטואליזציה של המעבד



- אפליקציה רצה של יוסי תיקרא תהליך.
- מערכת ההפעלה מעניקה לכל תהליך את האשליה שיש לו מעבד וירטואלי נפרד משלו.
- בצורה זו מערכת ההפעלה מקלה על התהליך וגם מגנה עליו.
- למשל, המעבד הווירטואלי מכיל את כל הרגיסטרים של המעבד הפיזי.
- התהליך לא צריך "לחשוב" באילו רגיסטרים מותר לו להשתמש כי המעבד הווירטואלי כולו שלו!
- כמו כן, תהליכים אחרים לא יכולים לקרוא/לכתוב לרגיסטרים של התהליך.

# איך מממשים וירטואליזציה של המעבד?

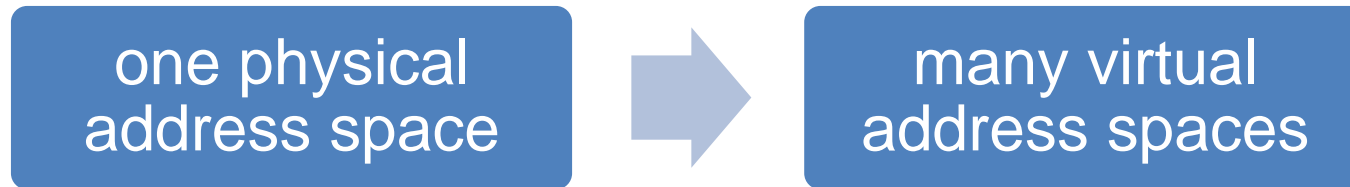
- מערכת ההפעלה מחלקת את הזמן של המעבד הפיזי בין התהליכים השונים (time sharing).
- הגרעין מחליף במהירות בין התהליכים: מריץ תהליך אחד לפרק זמן קצר (מילי-שניות) ואז משהה את ביצועו ועובר להריץ תהליך אחר, וכן הלאה.
- המשתמש מקבל אשליה של בו-זמניות ואינטראקטיביות.



- פרטים נוספים בתרגול בנושא החלפת הקשר.

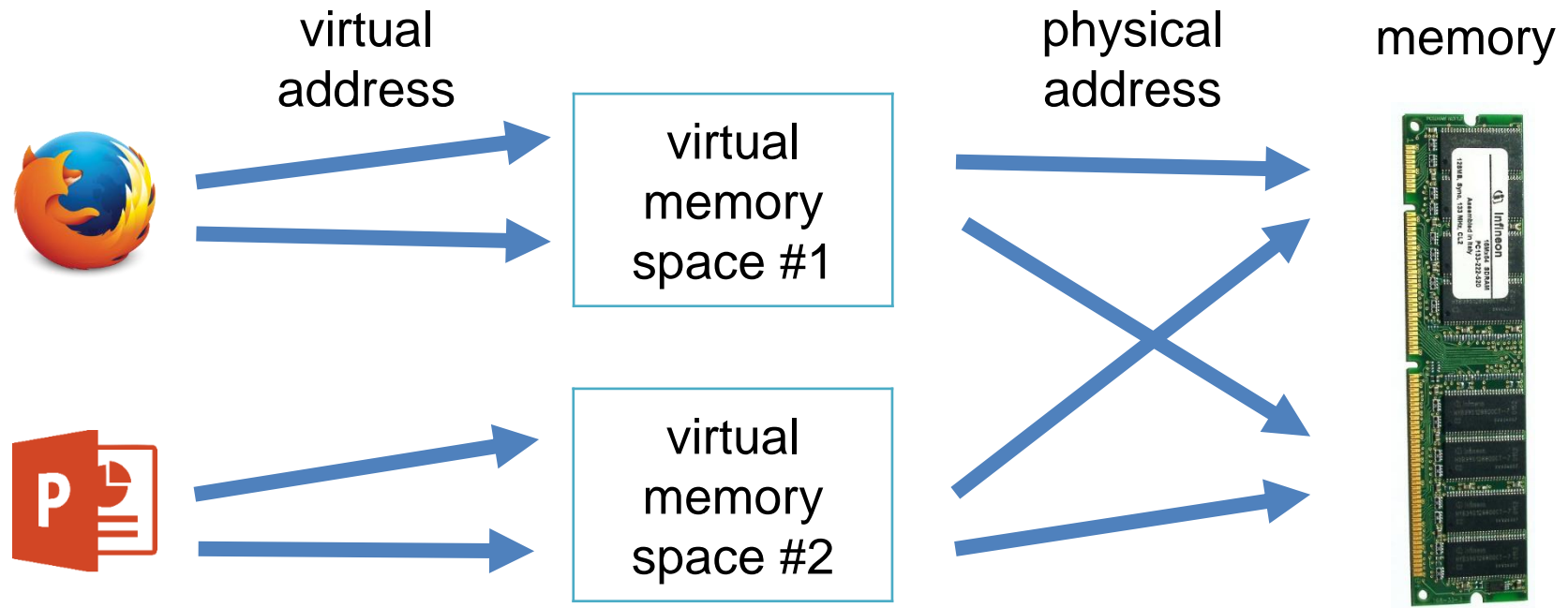


# וירטואליזציה של הזיכרון



- מערכת ההפעלה מעניקה לכל תהליך את האשליה שיש לו מרחב זיכרון נפרד משלו.
- בצורה זו מערכת ההפעלה מקלה על התהליך וגם מגנה עליו.
- התהליך לא צריך "לחשוב" באיזה זיכרון מותר לו להשתמש כי הזיכרון הווירטואלי כולו שלו!
- כמו כן, תהליכים אחרים לא יכולים לקרוא/לכתוב לזיכרון של התהליך.

- מערכת ההפעלה והמעבד מתרגמים את הכתובות הווירטואליות לכתובות פיזיות.
- פרטים נוספים בתרגול בנושא זיכרון וירטואלי.

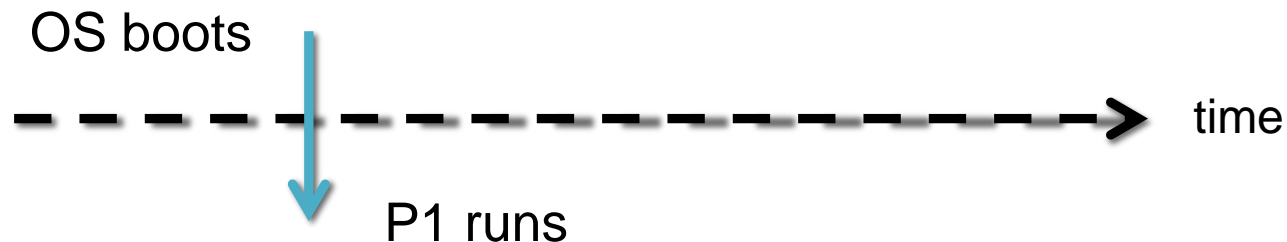


# הפסקה



# אתגרים במימוש עיקרון הווירטואליזציה

- ניסיון ראשון: הרצה ישירה (direct execution).
- נניח שיוסי לחץ על כפתור ההפעלה והדליק את המחשב.
- מערכת ההפעלה היא הראשונה שרצה.
- נטענת לזיכרון, מזהה את רכיבי החומרה, מאתחלת את מבני הנתונים שלה, ...
- לאחר שלב האתחול, מערכת ההפעלה מעבירה את השליטה על המעבד הפיזי לתהליך P1 כדי שירוך ישירות על המעבד.



## אבל מה אם...

- בעיה #1: תהליך  $P1$  ינסה לגשת לקבצים שאסור לו לגשת אליהם (למשל, קבצים של משתמש אחר במערכת)?
- בעיה #2: תהליך  $P1$  יריץ לולאה אינסופית וכך ימשיך להחזיק במעבד לנצח?
- איך מערכת ההפעלה תוכל להחזיר לעצמה את השליטה על המעבד?
- למשל, כדי לעצור את תהליך  $P1$  ולהריץ במקומו תהליך אחר  $P2$ .
- בעיה #3: תהליך  $P1$  יבצע פעולה איטית מאוד (כמו קריאה/כתיבה מהדיסק) שלא רצה על המעבד?
- הפתרונות לבעיות הללו מבוססים על מנגנוני חומרה שנלמד כעת.

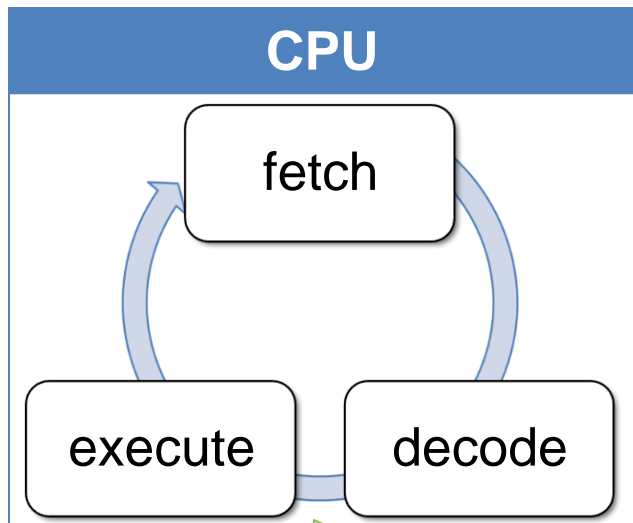
# פתרון לבעיה #1: רמות הרשאה

- המעבד יגדיר שתי רמות הרשאה:
  - **user mode** – רמת הרשאות נמוכה לשימוש תכניות רגילות (קוד משתמש).
  - **kernel mode** – רמת הרשאות גבוהה לשימוש מערכת ההפעלה.
- בכל רגע נתון, המעבד יהיה ברמת הרשאה אחת ויחידה.
- רמת ההרשאות הנוכחית (CPL = current privilege level) נשמרת ברגיסטר כלשהו של המעבד.

CPU			
		CPL	

0  $\Leftrightarrow$  kernel mode  
3  $\Leftrightarrow$  user mode

# פקודות מיוחדות ( privileged instructions )



- המעבד יחלק את פקודות המכונה לשתי קבוצות:

1. **פקודות מיוחדות (privileged)**

2. **פקודות לא מיוחדות (non-privileged)**

- פקודות מכונה שניגשות לדיסק, למשל, יהיו מיוחדות.

המעבד יבדוק בשלב זה (לפני ביצוע הפקודה) האם היא מיוחדת. אם הפקודה מיוחדת, אבל המעבד במצב משתמש – תיווצר חריגה. החריגה תעביר את השליטה לגרעין, והוא יהרוג את התהליך.

# איך משנים את רמת ההרשאה?

- במילים פשוטות: איך משתמש ייגש לדיסק כדי לקרוא קובץ?

- ניסיון ראשון: המשתמש יעלה את רמת ההרשאה, ואז יקרא לפונקציה של מערכת ההפעלה שתיגש לדיסק.

מה הבעיה  
בהצעה הזו?

- הצעה טובה יותר: המשתמש יקרא לפקודת מכונה מיוחדת אשר תבצע שתי פעולות בעת ובעונה אחת:

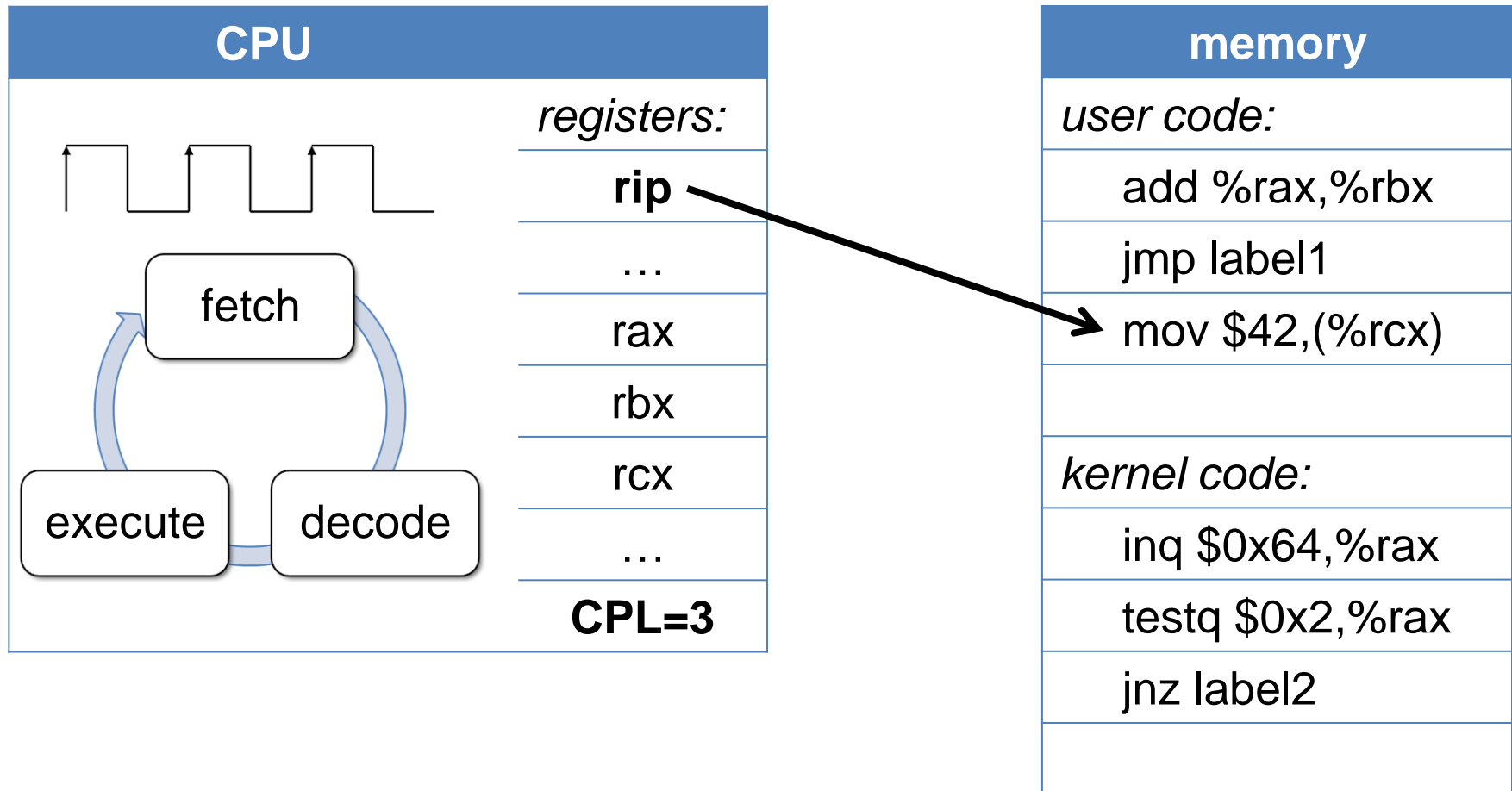
1. תעלה את רמת ההרשאה.

2. תעבור לבצע פונקציה של מערכת ההפעלה.

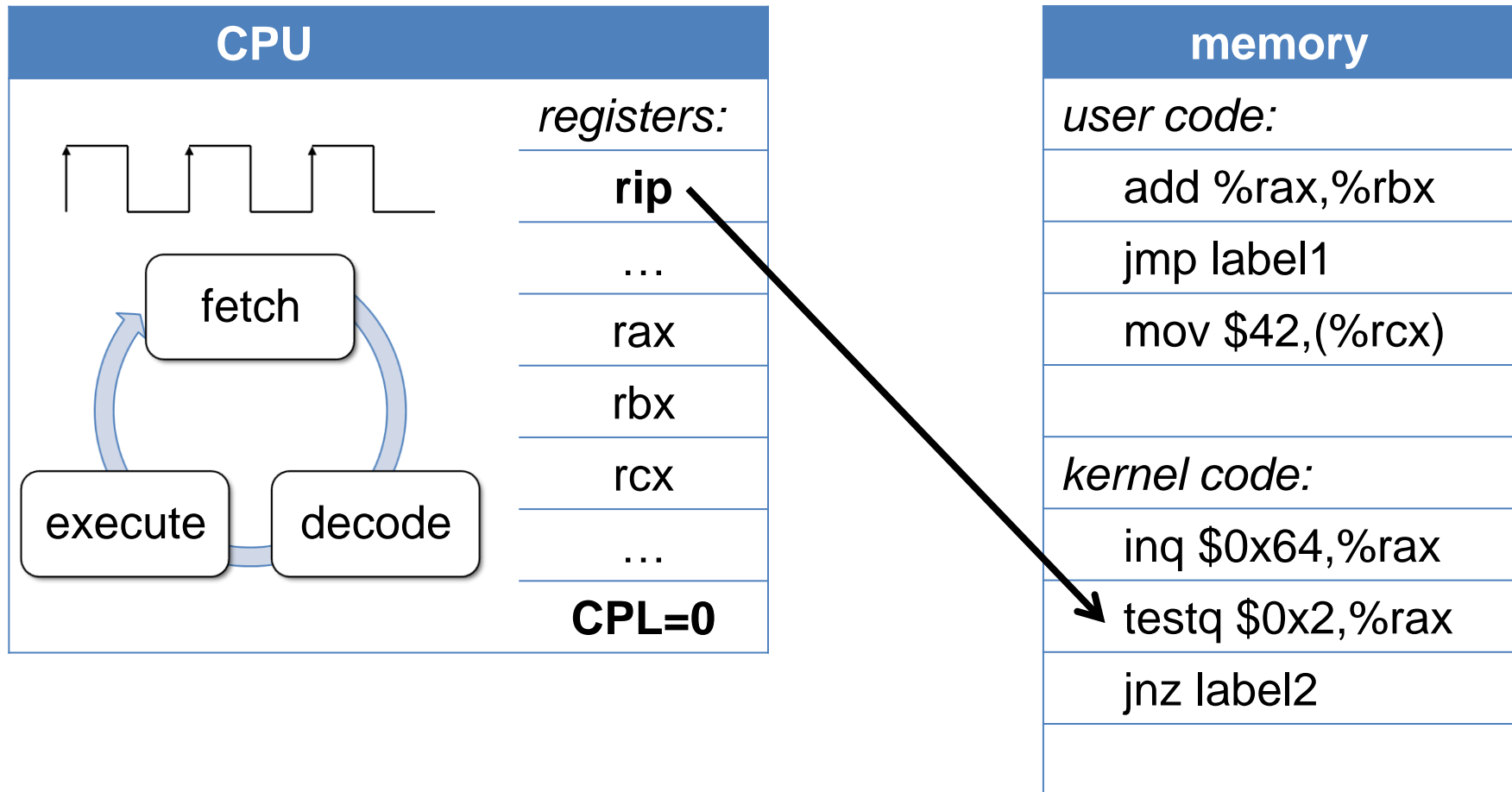
- הפקודה המיוחדת הזו מכונה **קריאת מערכת**, או באנגלית **System Call** (syscall בקיצור).



# מצב המערכת לפני פקודת `syscall`



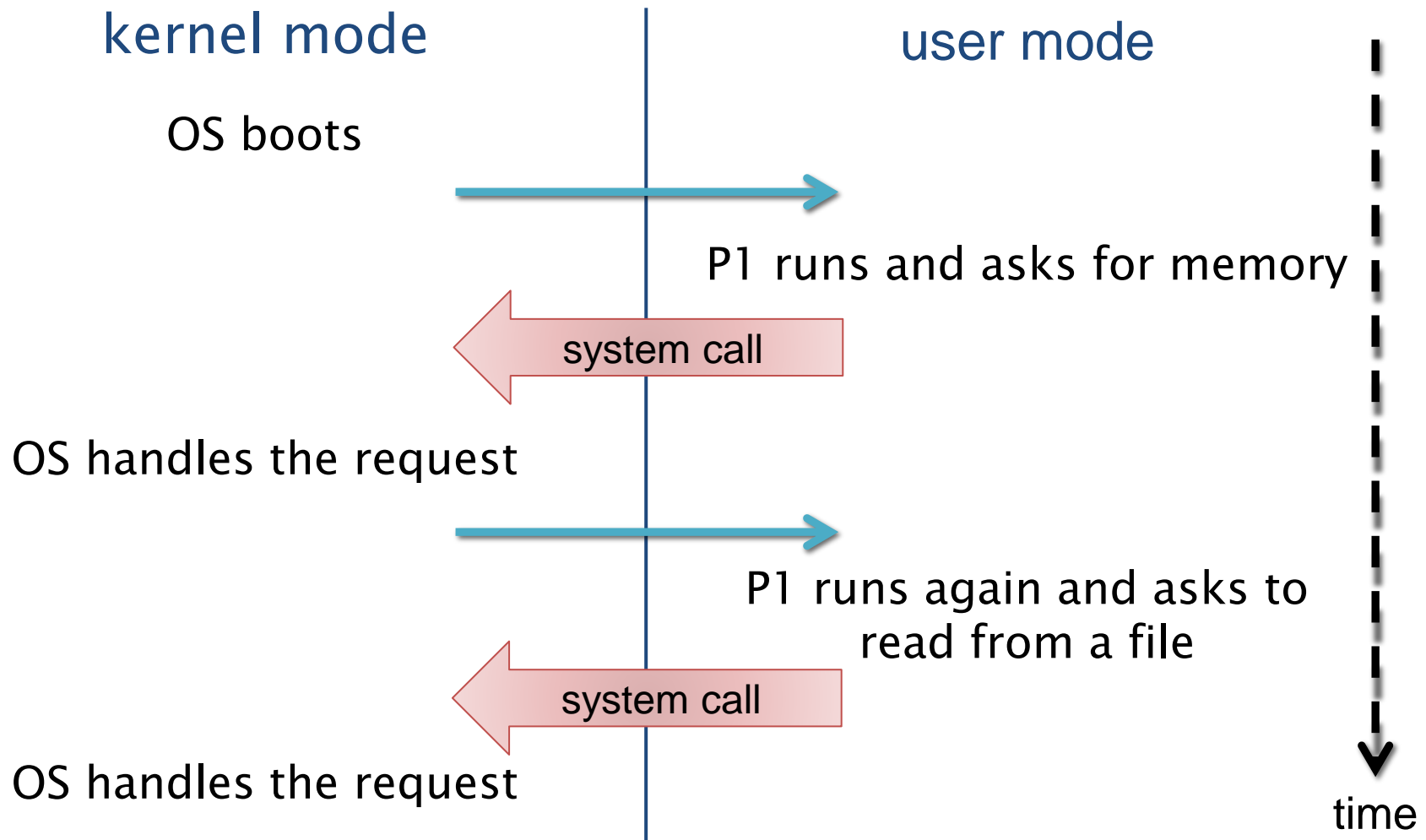
# מצב המערכת אחרי פקודת syscall



## קריאות מערכת

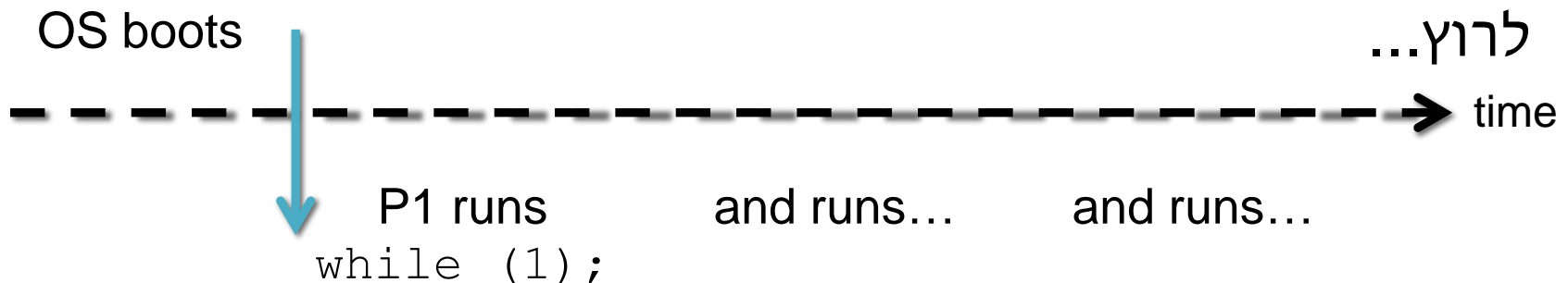
- קריאות מערכת מאפשרות העברה בטוחה ומבוקרת של השליטה על המעבד מפני שהן שער הכניסה היחיד של תוכניות משתמש לפקודות מיוחדות.
- קריאות מערכת מגדירות למעשה את הממשק של מערכת ההפעלה לתוכניות משתמש.
- תהליך משתמש קורא לקריאות מערכת כדי לבקש ממערכת ההפעלה שירות כלשהו כמו: לגשת להתקני קלט/פלט, ליצור תהליכים חדשים, לבקש עוד זיכרון ועוד.

# תרחיש לדוגמה



## תזכורת: בעיה #2

- מערכת ההפעלה היא הראשונה שרצה לאחר הדלקת המחשב.
- בשלב מסוים, מערכת ההפעלה מעבירה את השליטה על המעבד הפיזי לתהליך P1 כדי שירץ.
- אם P1 יקרא לקריאת מערכת, אז מערכת ההפעלה תרוץ – כלומר תקבל לידיה את השליטה על המעבד שוב.
- אבל אם P1 יריץ לולאה אינסופית (בטעות או בזדון)?
- הוא ימשיך להחזיק במעבד לנצח ולא ייתן לאף תהליך אחר לרוץ...



## פתרון לבעיה #2: פסיקות שעון

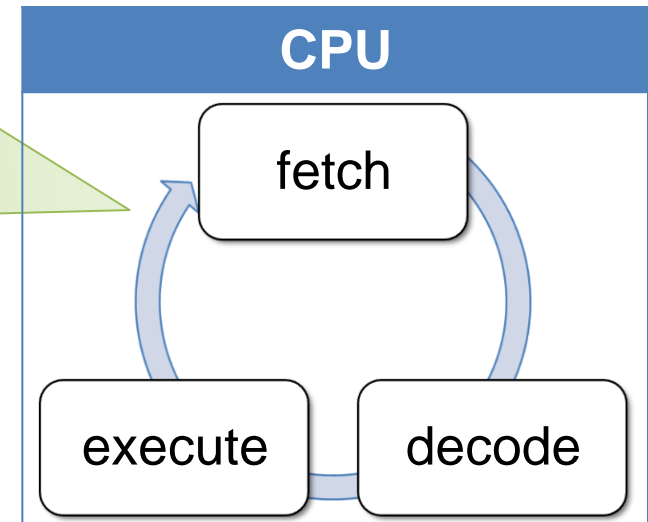
- לינוקס מפקיעה (preempt) את המעבד מתהליך אחד לטובת תהליך אחר, בעזרת התקן חומרה מיוחד – **השעון** (timer).

שימו לב: השעון הוא רכיב חיצוני למעבד ואינו קשור לתדר השעון הפנימי של המעבד.  
(זו טעות נפוצה של סטודנטים.)

- מערכת ההפעלה מבקשת מהשעון לשלוח פסיקה במרווחי זמן קבועים כדי להעביר את השליטה למערכת ההפעלה.
- כל הפסיקות, בפרט פסיקת שעון, מטופלות ב-kernel mode.
- במהלך הטיפול בפסיקה, מערכת ההפעלה יכולה להחליט שהיא מחליפה את התהליך הנוכחי שרץ כרגע על המעבד.
- הפעולה הזו נקראת "**החלפת הקשר**" – נחזור אליה בהמשך הקורס.

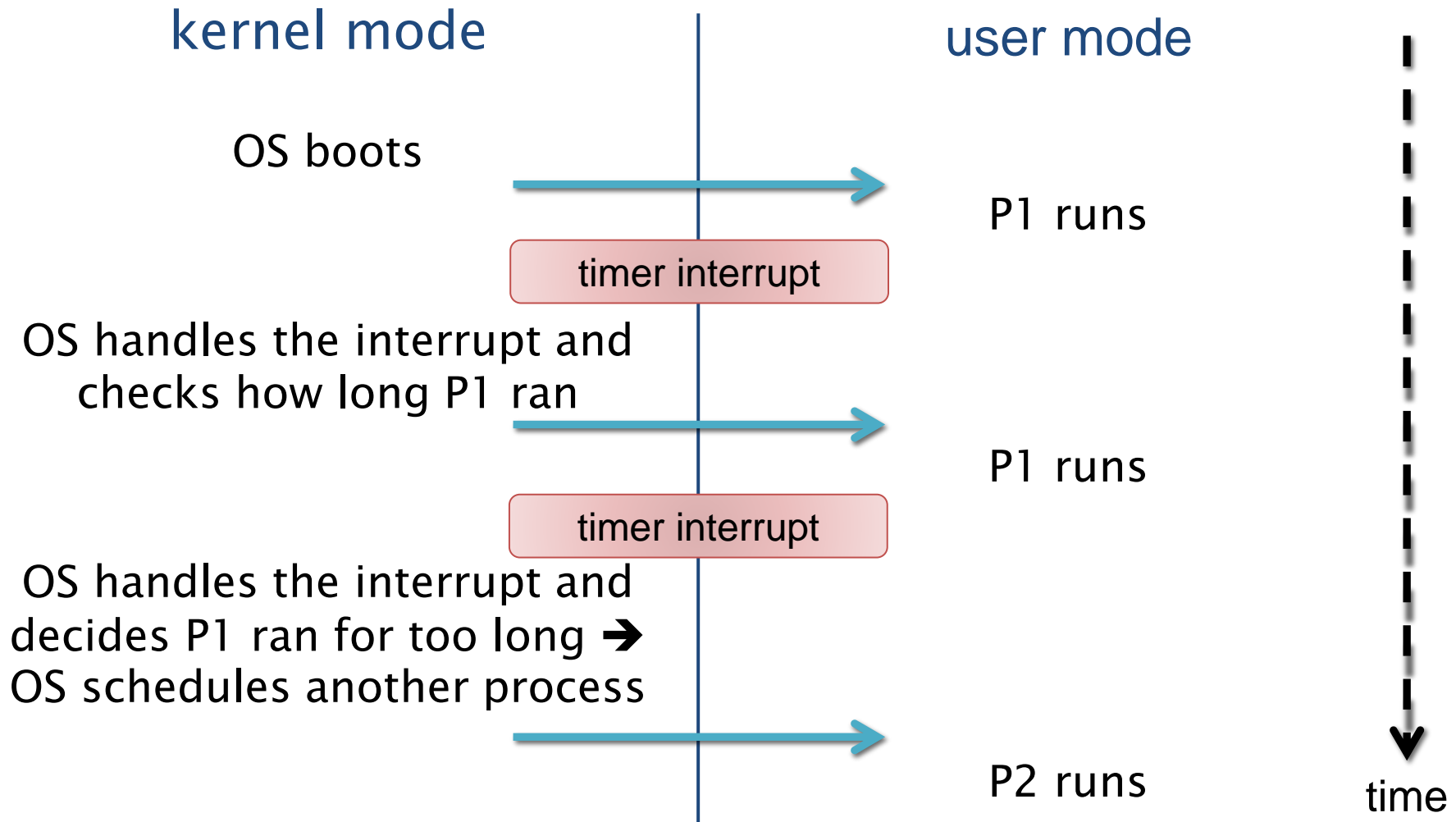
# הטיפול בפסיקות חומרה

המעבד בודק אם יש פסיקות ממתניות לאחר סיום של כל פקודת מכונה. אם יש פסיקה, המעבד מפסיק לבצע את הקוד הנוכחי ועובר לבצע את **שגרת הטיפול בפסיקה** (interrupt handler) במצב גרעין.



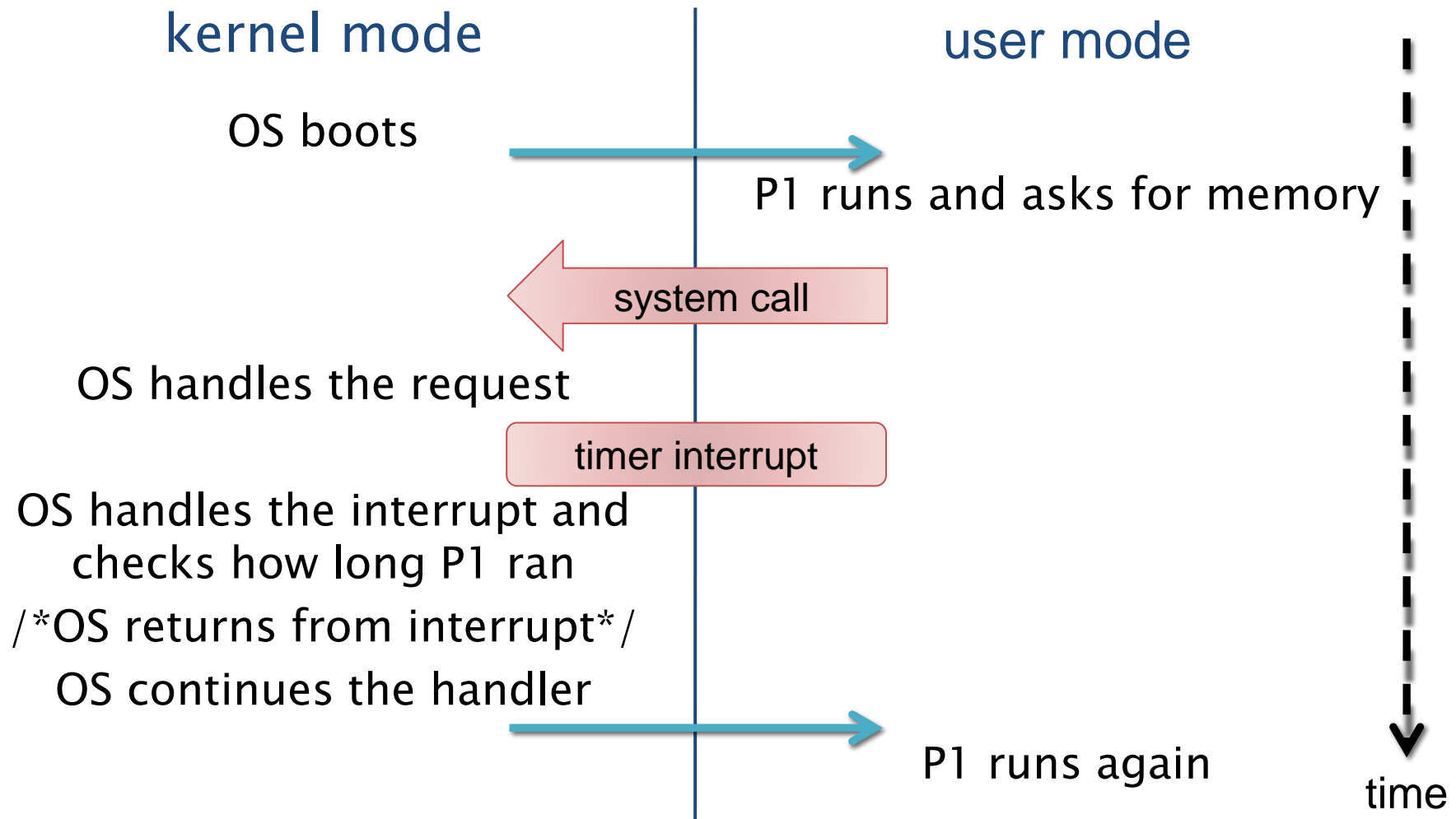
- לאחר סיום הטיפול בפסיקה, המעבד יחזור לבצע את הקוד הקודם.
- כדי לדעת לחזור, המעבד ומערכת ההפעלה צריכים לשמור את המצב של המעבד ברגע קבלת הפסיקה.
- שימו לב: פסיקה אינה קוטעת ביצוע של פקודת מכונה. פסיקות מטופלות "בין" פקודות מכונה.

# תרחיש לדוגמה



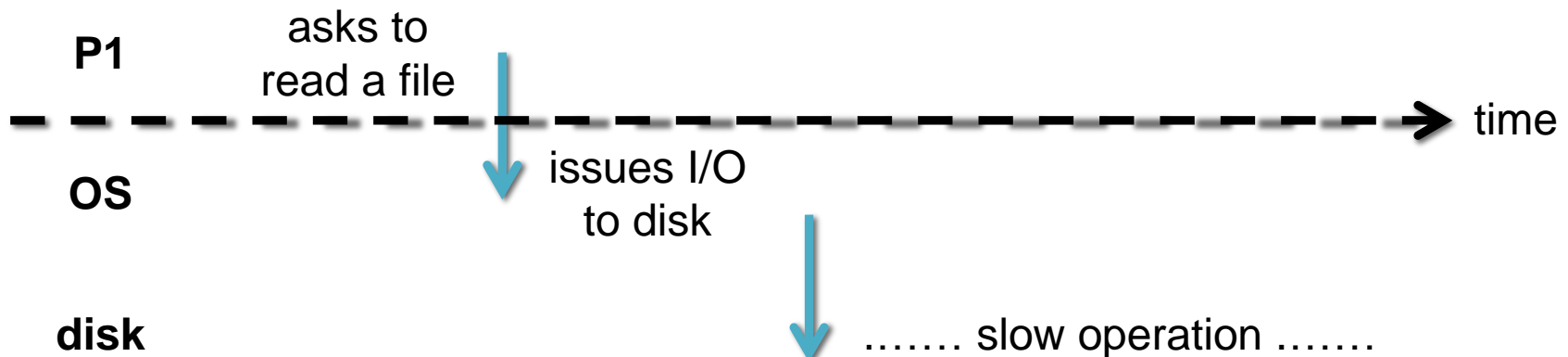


# פסיקות יכולות להגיע גם במצב גרעין !



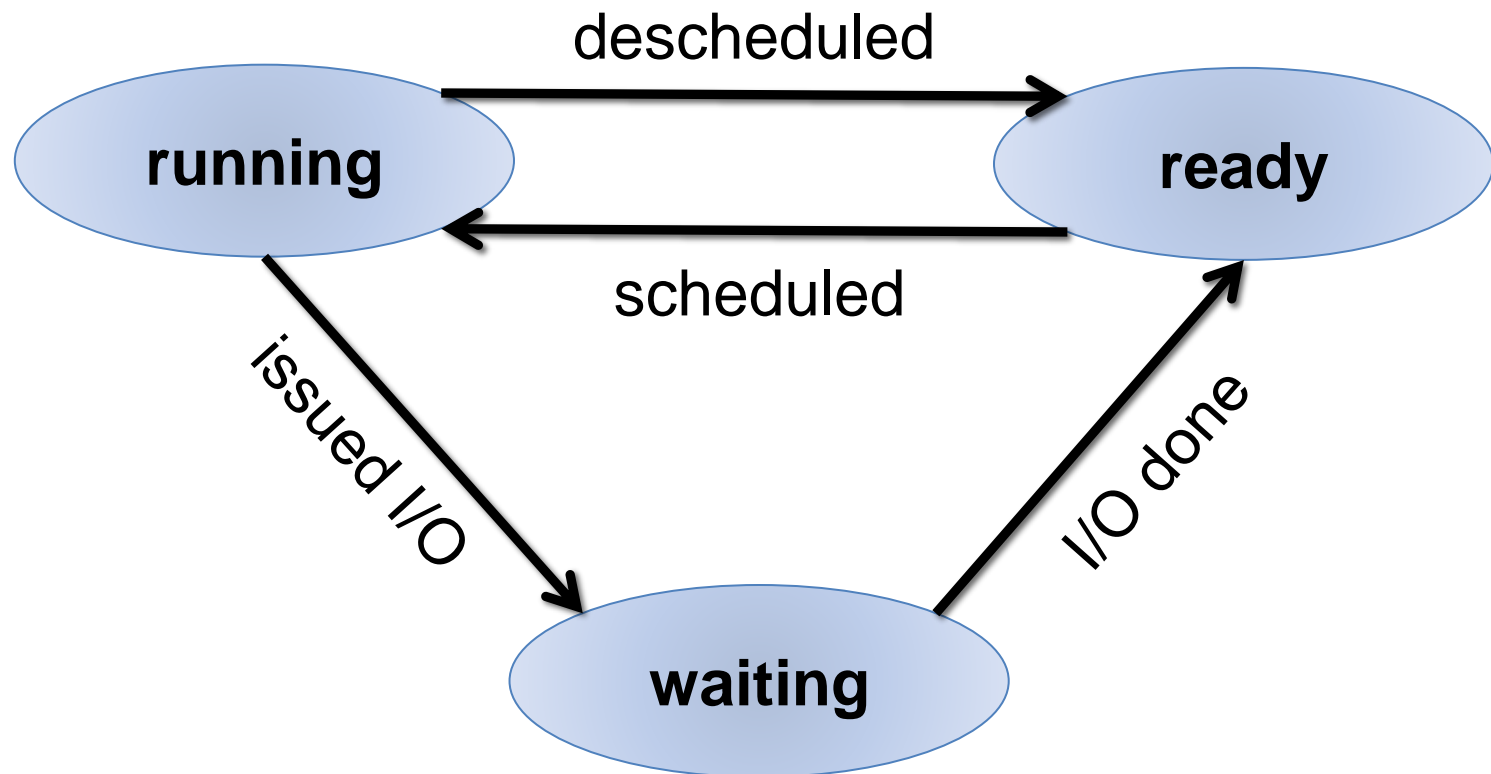
## תזכורת: בעיה #3

- תהליך יכול לבקש ממערכת ההפעלה שירותי I/O, לדוגמה: קריאה/כתיבה מהדיסק או מכרטיס הרשת.
- גישה להתקני I/O היא איטית מאוד: סדר גודל של מספר מילישניות == מיליוני פקודות מעבד.
- בזמן ההמתנה להתקני I/O התהליך לא רץ והמעבד חסר פעילות.
- איך נוכל לנצל טוב יותר את המשאבים של המערכת?

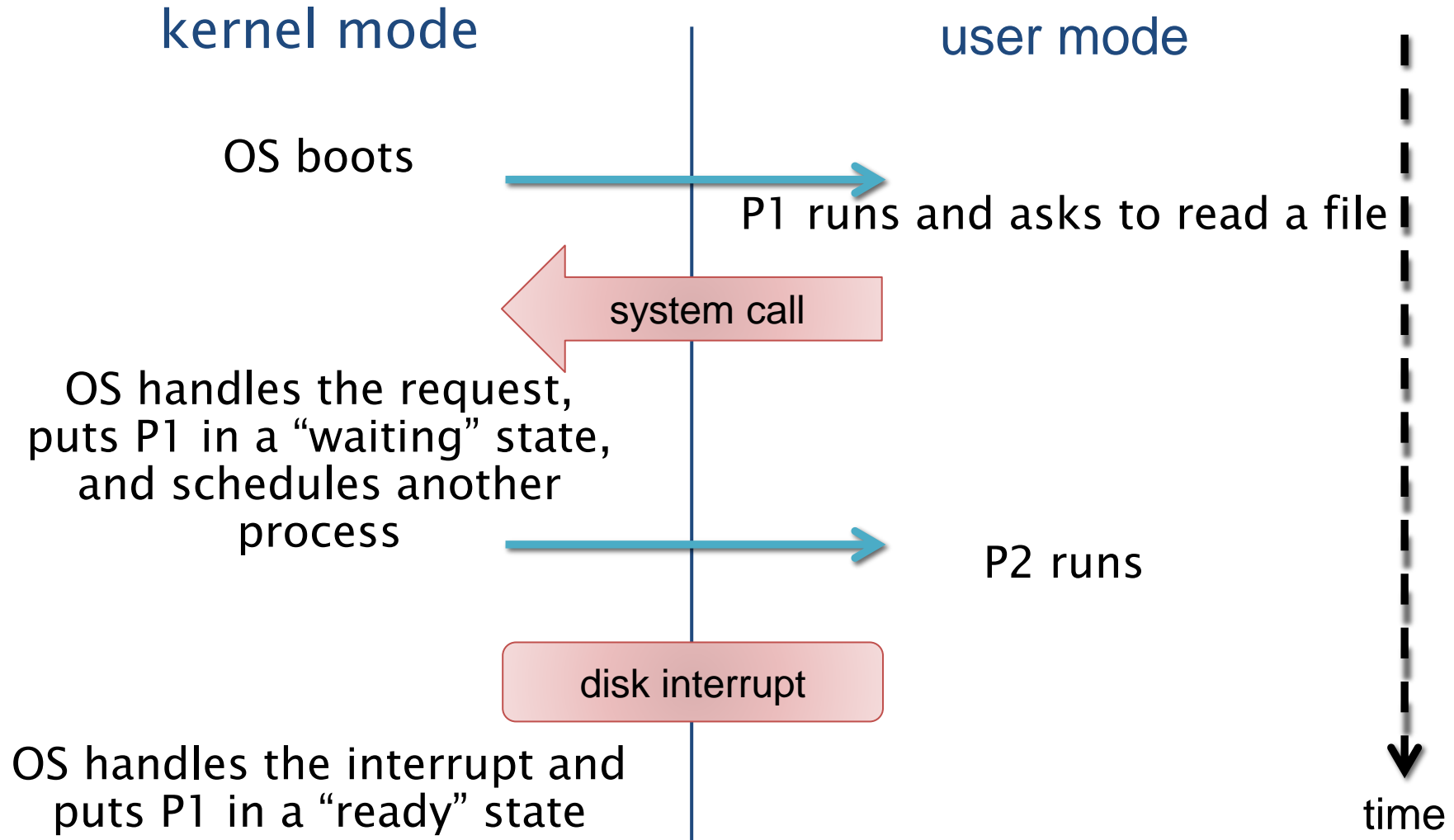


## פתרון לבעיה #3: מצבי המתנה

- מערכת ההפעלה תסווג את התהליכים במערכת לשלושה מצבים אפשריים (בפועל יש יותר, אלו המצבים העיקריים):



# תרחיש לדוגמה



# לסיכום

- בעיות #1 , #2 הן בעיות אבטחה (security).
- בעיה #3 היא בעיית יעילות (efficiency).
- מערכת ההפעלה שואפת להשיג גם יעילות וגם אבטחה, ולכן היא פותרת את שלושת הבעיות הללו באמצעות מנגנון:  
**הרצה ישירה מוגבלת (limited direct execution)**

אבל כדי לשמור על אבטחה, התהליכים לא יכולים להריץ כל פקודה.

כדי לקבל ביצועים גבוהים, התהליכים רצים ישירות על המעבד הפיזי.

# מערכת ההפעלה "לינוקס"

---

## קצת היסטוריה...

- דניס ריצ'י וקן תומפסון ממעבדות Bell מפתחים מערכת הפעלה קניינית בשם יוניקס (UNIX).

1973

- ריצ'רד סטולמן מכריז על מיזם GNU במטרה לפתח מערכת הפעלה חופשית תואמת יוניקס (וגם ספריות וכלים נוספים).

1983

- מיזם גנו מתחיל לפתח את גרעין מערכת ההפעלה, אך הפיתוח מתגלה כמסובך ומתקדם באטיות רבה.

1990

- לינוס טורבאלדס מתחיל לפתח את גרעין לינוקס במהלך לימודיו באוניברסיטת הלסינקי.

1991

- טורבאלדס משנה את רשיון לינוקס ל-GPL וכך לינוקס הופכת לגרעין מערכת ההפעלה של מיזם GNU.

1992

# בקורס נלמד ונשתמש בלינוקס

- הסיבה המרכזית לכך: לינוקס היא תוכנה חופשית וקוד פתוח.
- קוד המקור של גרעין לינוקס זמין לשימוש, לשינוי ולהפצה בחינם לכל אחד.
- היום השם "לינוקס" מתייחס למשפחה של מערכות הפעלה המבוססות על גרעין לינוקס ורכיבי התוכנה של מיזם GNU.
- הספציפית בה נשתמש בקורס היא Red Hat.
- גרסת הגרעין היא 2.4.18–14.
- כל הקוד כמובן חופשי לכולם באינטרנט.
- בתרגיל בית 0 תתקינו את מערכת ההפעלה על המחשב האישי שלכם באמצעות מכונה וירטואלית ( virtual machine).