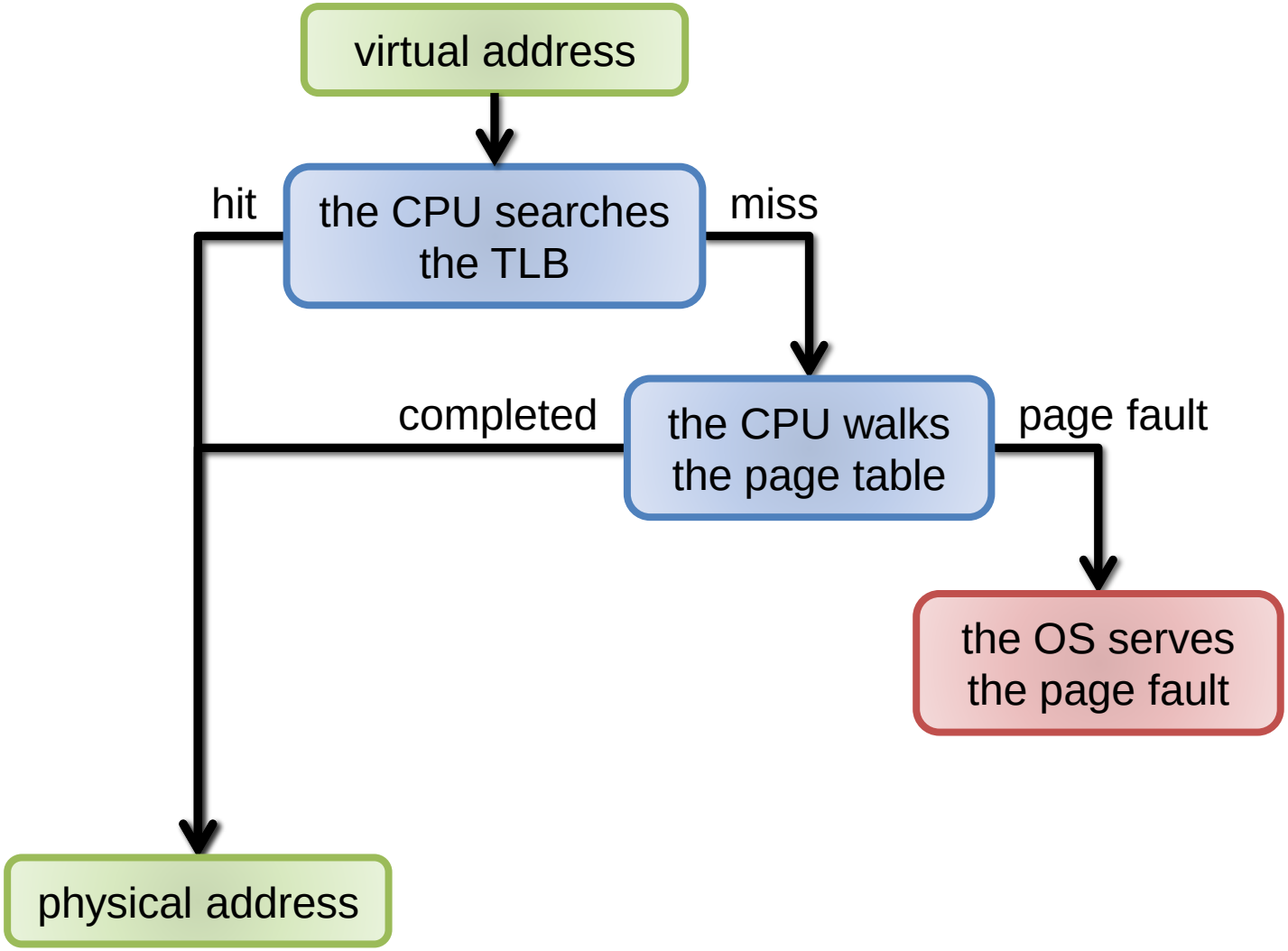
תרגול 11

מבני נתונים בגרעין לניהול זיכרון

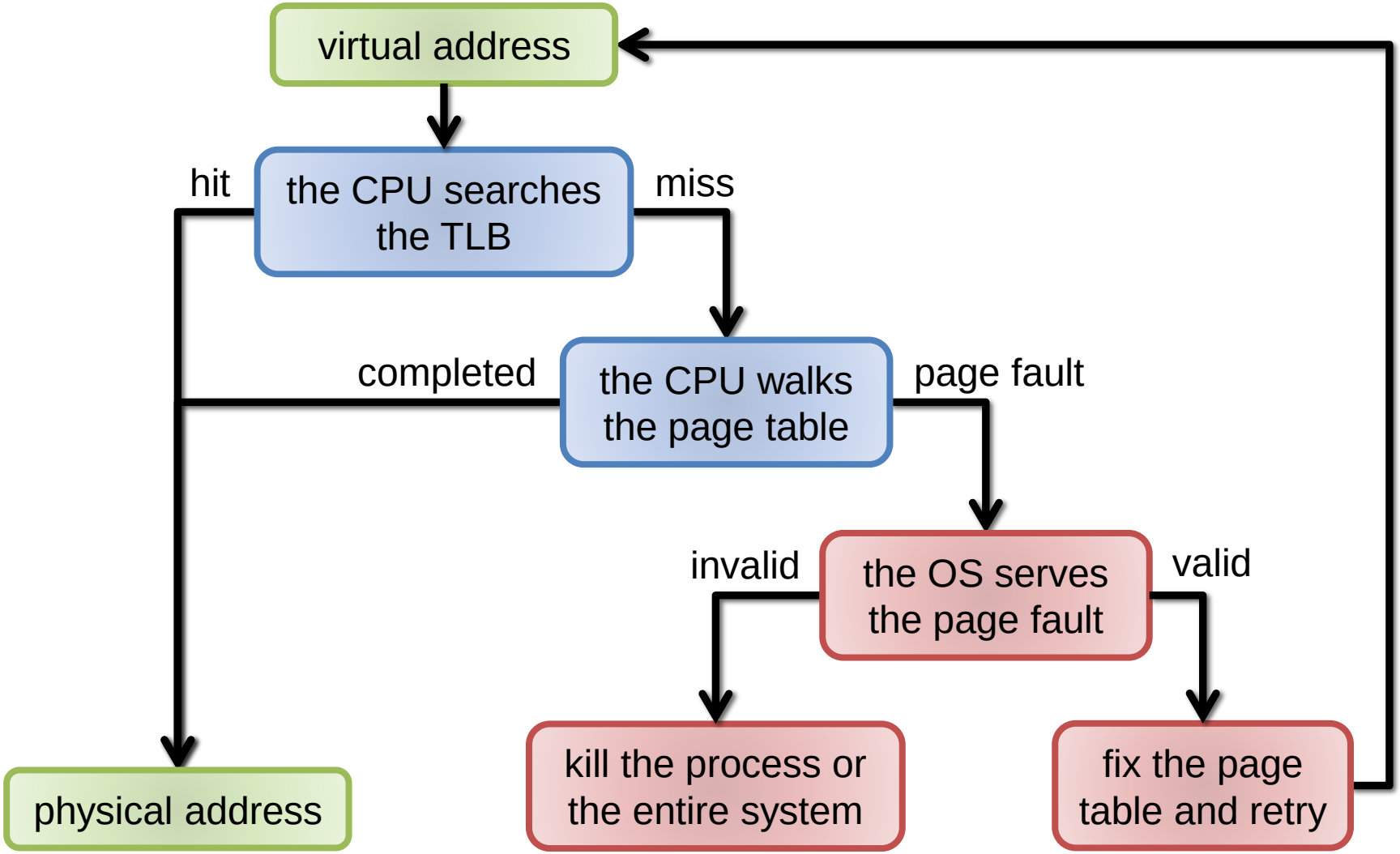
מנגנון copy-on-write

מנגנון demand paging

סיכום השיעור שעבר



מה נלמד היום?


virtual address
the CPU searches the TLB
hit
miss
completed
the CPU walks the page table
page fault
invalid
the OS serves the page fault
valid
physical address
kill the process or the entire system
fix the page table and retry

TL;DR

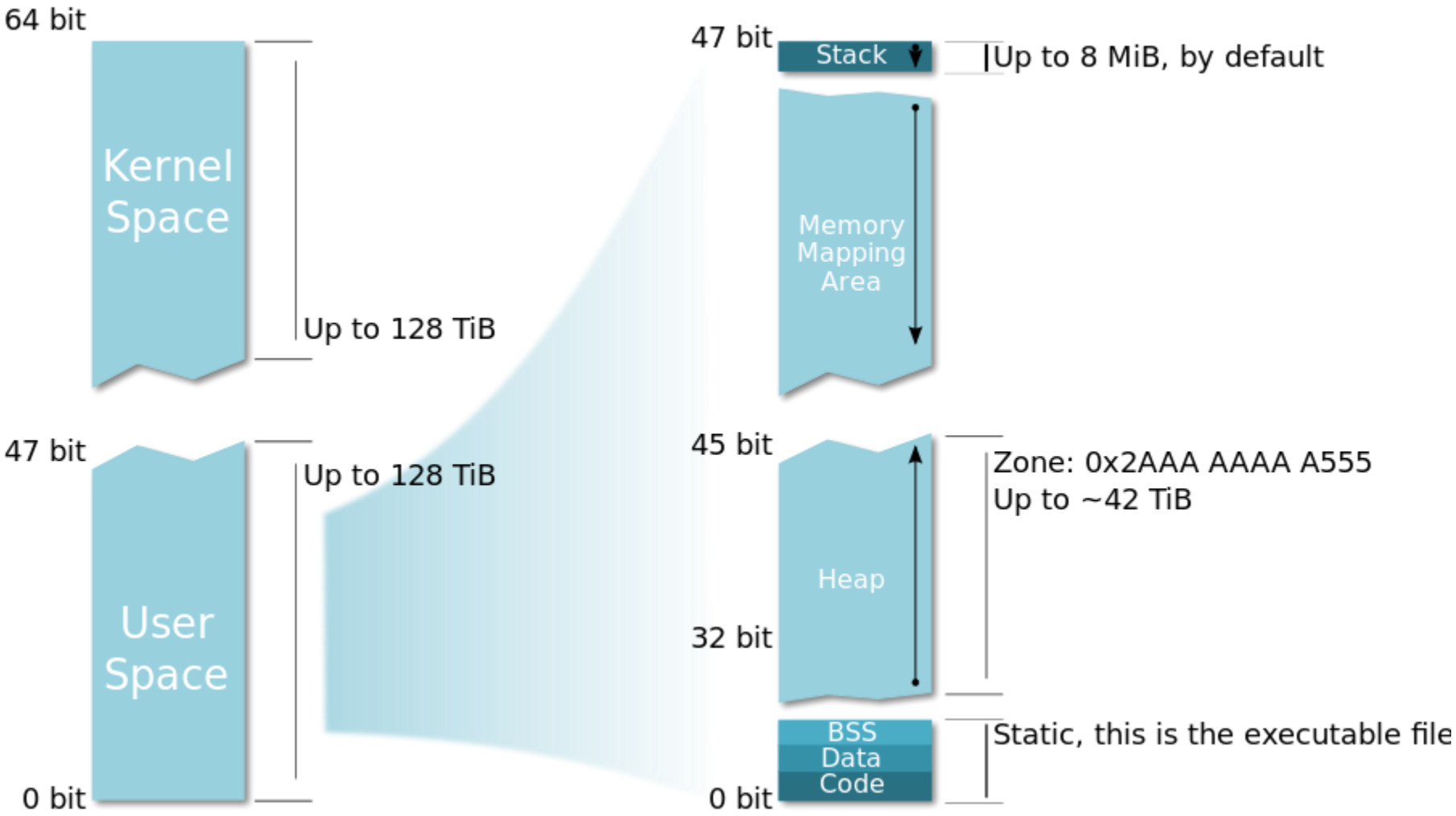
- זיכרון וירטואלי הוא **מנגנון משולב חומרה-תוכנה**:
  - **המעבד מתרגם** את הכתובת הווירטואלית לכתובת פיזית בעת גישה לזיכרון באמצעות הליכה בטבלת הדפים (page table walk) - ראינו בתרגול הקודם.
  - **מערכת ההפעלה מגדירה** את טבלת הדפים, וכך מגדירה את המיפוי בין כתובות וירטואליות לפיזיות - את זה נראה היום.

- לינוקס מנהלת את טבלאות הדפים של תהליכים בצורה **"עצלה/דחיינית"** (lazy) כדי לחסוך זיכרון וזמן מעבד.
  - לינוקס דוחה ככל הניתן העתקת זיכרון מהאב לבן באמעות copy-on-write.
  - לינוקס מקצה מסגרות פיזיות לתהליך בצורה עצלה ע"י demand paging.

מבני נתונים בגרעין לניהול זיכרון

טבלאות דפים, מרחבי זיכרון, אזורי זיכרון, ...

מרחב הזיכרון של תהליך



מרחב הזיכרון של תהליך

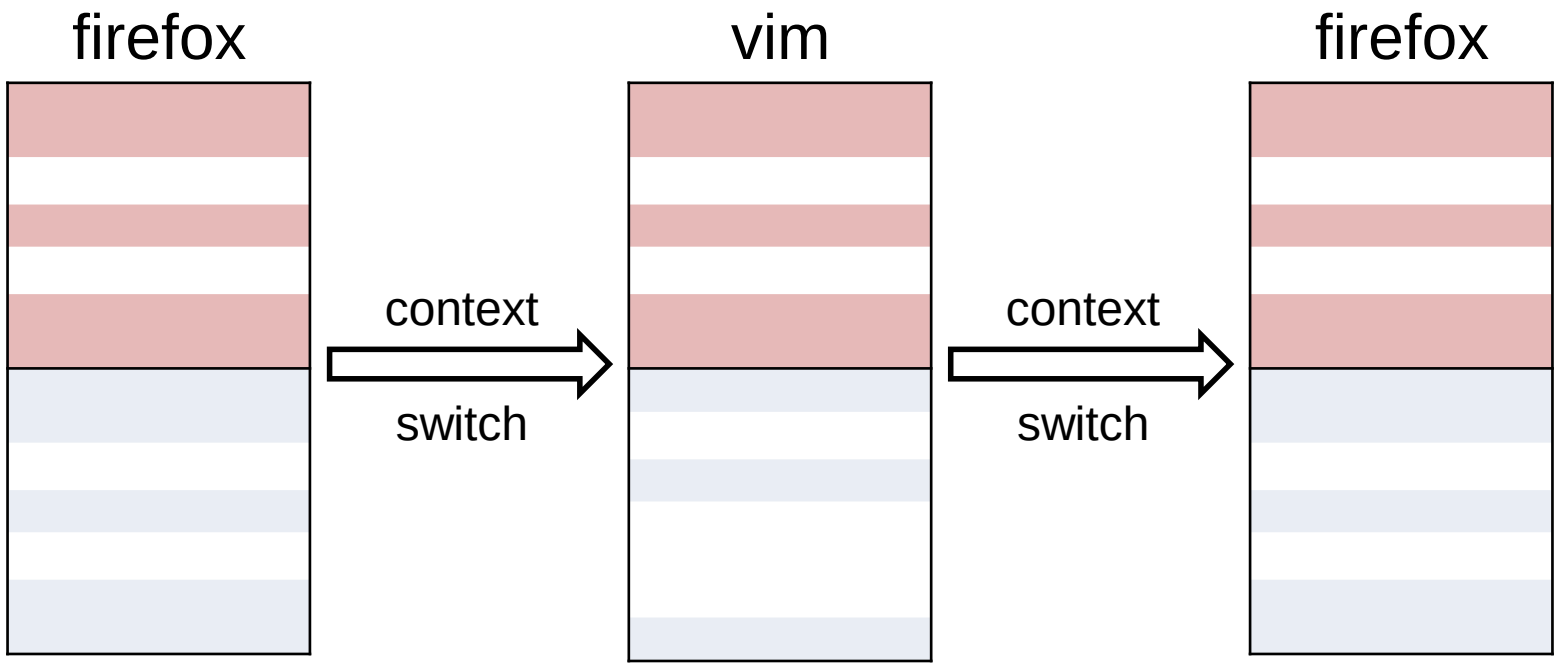
- במעבדי x64 רוחב כתובת וירטואלית הוא 48 ביט.
  - ← גודל מרחב הזיכרון הווירטואלי של כל תהליך הוא: $2^{48}\text{B} = 256\ \text{TB}$.

- בלינוקס, מרחב הזיכרון הנ"ל מחולק לשניים:
- **128TB העליונים** – מרחב הזיכרון של הגרעין.
  - במרחב זה נשמרים כל מבני הנתונים והקוד של מערכת ההפעלה, ובפרט: תורי הריצה (runqueues), מחסניות הגרעין, כל טבלאות הדפים כל התהליכים השונים, וכולי.
  - מרחב הגרעין לעולם אינו מפונה לדיסק (לעולם אינו swapped).

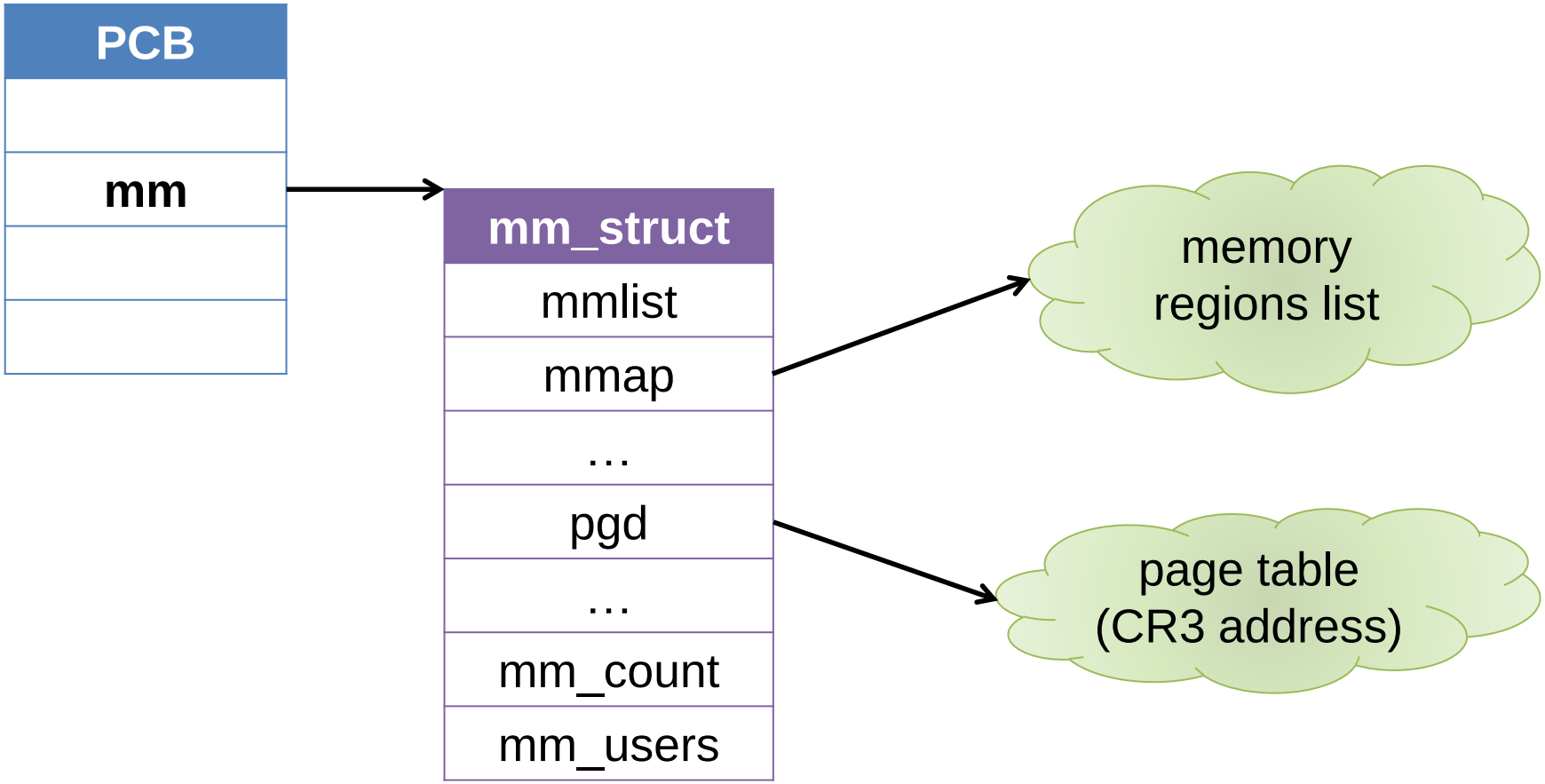
- **128TB התחתונים** – מרחב הזיכרון של המשתמש.
  - במרחב זה נשמרים קוד התוכנית, המחסנית, הערימה, ואזורי זיכרון נוספים.

מרחב הגרעין

- מרחב הגרעין **משותף לכל התהליכים** כי הוא ממופה לאותו מקטע בזיכרון הווירטואלי של כל התהליכים.
  - באופן זה, הכתובת (הווירטואלית) של כל אובייקט בגרעין נשארת קבועה בכל מרחבי הזיכרון של כל התהליכים.



מתאר מרחב הזיכרון של תהליך



מתאר הזיכרון של תהליך

- השדה mm של כל PCB מצביע אל **מתאר מרחב הזיכרון** (memory descriptor) של אותו תהליך.
- מתאר מרחב הזיכרון מיוצג ע"י מבנה מסוג **mm_struct**.
- תהליכים אשר חולקים אותו מרחב זיכרון, כמו חוטים למשל, מצביעים על אותו מתאר מרחב זיכרון.

- ערך שדה mm של חוט גרעין הוא NULL.
- **חוט גרעין** הוא תהליך שנוצר ע"י מערכת ההפעלה כדי להריץ משימה כלשהי של מערכת ההפעלה.
  - לכל חוט גרעין יש PCB משלו, בדומה לתהליך רגיל.
  - חוטי גרעין לדוגמה: ksoftirqd, kswapd, khugepaged .

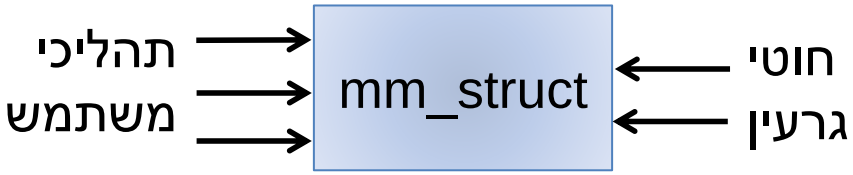
חוטי גרעין (kernel threads)

- מבחינת זיכרון, חוט גרעין ניגש אך ורק למרחב הגרעין (128TB העליונים אשר משותפים לכל התהליכים במערכת).
  - לחוט גרעין אין שום רכיב במרחב המשתמש, למשל, אין לו ערימה.
  - חוט גרעין רץ רק בהרשאות גרעין (CPL == 0) על מחסנית הגרעין שלו.

- לכן בלינוקס חוט גרעין לא מקבל מרחב זיכרון משלו, אלא פשוט פועל במרחב הזיכרון של תהליך המשתמש שרץ לפניו.
- כאשר מערכת ההפעלה מחליפה הקשר מתהליך רגיל לחוט גרעין, היא לא מחליפה את מרחב הזיכרון.
  - במילים פשוטות: רגיסטר CR3 נותר ללא שינוי כדי שיצביע על טבלת הדפים של התהליך הרגיל.

שדות במתאר מרחב הזיכרון של תהליך

- **mm_users** – כמה תהליכי משתמש חולקים את מרחב הזיכרון?
- **mm_count** – כמה תהליכי משתמש + חוטי גרעין חולקים את מרחב הזיכרון? כל תהליכי המשתמש יחד נחשבים כאחד, אבל כל חוט גרעין נספר בנפרד.



mm_users = 3
mm_count = 3

- כאשר mm_users == 0, מערכת ההפעלה מפנה את כל אזורי הזיכרון של המשתמש (מחסנית, ערימה, קוד, וכולי).
- כאשר mm_count == 0, מערכת ההפעלה מפנה את מתאר מרחב הזיכרון כולו (וטבלת הדפים כולה).
  - mm_count מונע פינוי מרחב זיכרון כאשר הוא בשימוש ע"י תהליך גרעין.

שדות במתאר מרחב הזיכרון של תהליך

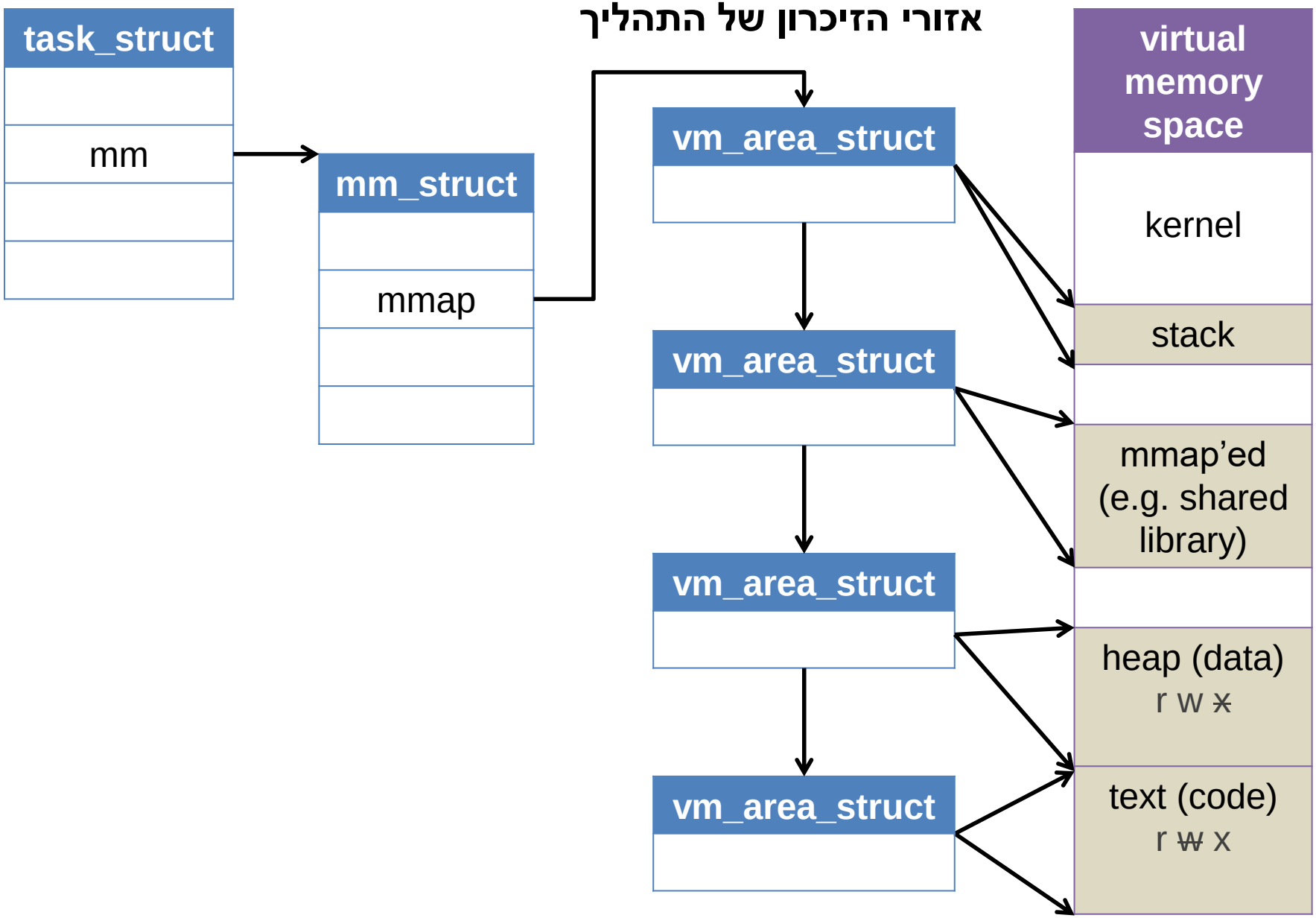
- **mmlist**: קישור לרשימה הגלובלית של מתארי מרחבי הזיכרון (מטיפוס list_head), הנחלקת ע"י כל התהליכים.
- **pgd**: הכתובת של שורש טבלת הדפים.
  - זה הערך שייטען ל-CR3 כאשר התהליך יזומן לריצה.

מכיל כתובת פיזית או וירטואלית?

- **mmap**: רשימה ממוינת של אזורי הזיכרון – נראה בהמשך.
- **rss**: מספר המסגרות שבשימוש (דפים מגובים בזיכרון).
- **total_vm**: מספר דפים כולל באזורי הזיכרון.

האם rss, total_vm יכולים להיות שונים?

אזורי הזיכרון של התהליך



אזורי זיכרון

- הגרעין מנהל את מרחב הזיכרון של תהליך ע"י חלוקה **לאזורי זיכרון** (virtual memory regions).
- אזור זיכרון הוא רצף כתובות במרחב הזיכרון של התהליך אשר שייך לתחום של 128TB התחתונים (כלומר לא לגרעין).

1. אזורי הזיכרון אינם חופפים.
2. לכל אזור הרשאות קריאה/כתיבה/ביצוע משלו.
3. תהליך יכול לגשת רק לכתובת שנמצאת באזור זיכרון כלשהו.
4. כתובת התחלתית וגודל של אזור זיכרון הם כפולות של גודל הדף.
   - זכרו שיחידת ההקצאה הבסיסית של מנגנון הזיכרון הווירטואלי היא דף.
5. ניתן להוסיף, להסיר, להגדיל ולהקטין אזורי זיכרון.

מתאר אזור זיכרון

- אזור זיכרון מאופיין ע"י **מתאר אזור זיכרון**, שהוא רשומה מטיפוס **vm_area_struct**.

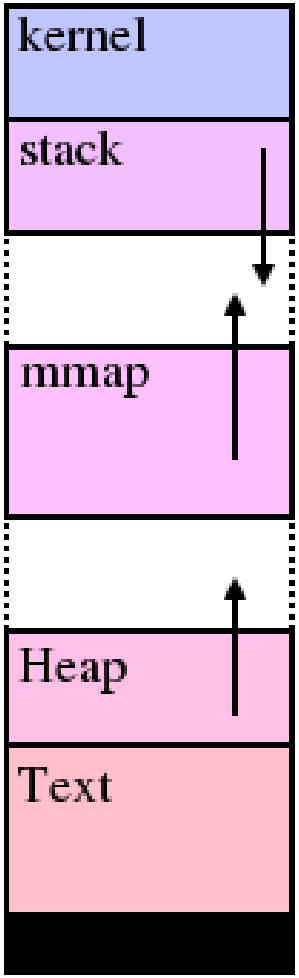
- שדות במתאר אזור זיכרון:
  - vm_start: כתובת התחלה של אזור הזיכרון.
  - vm_end: כתובת אחת אחרי האחרונה של אזור הזיכרון.
  - vm_next: מצביע למתאר אזור הזיכרון הבא ברשימה המקושרת של האזורים.
  - vm_mm: מצביע חזרה למתאר מרחב הזיכרון המכיל את האזור.
  - vm_flags: דגלים המציינים תכונות של האזור.
  - vm_page_prot: ערכי ביטים שונים שיוצבו לכל הכניסות של טבלת הדפים עבור הדפים באזור.

הרשאות של אזור זיכרון

- הדגלים המציינים את הרשאות האזור נשמרים בשדה **vm_flags** והם מאפשרים לגרעין לסווג גישות חוקיות ולא חוקיות לדפים באזור.

- VM_READ, VM_WRITE, VM_EXEC – האם מותר לקרוא/לכתוב/לבצע נתונים בדפים באזור.
- VM_MAYREAD, VM_MAYWRITE, VM_MAYEXEC – "הרשאת הרשאה" לכל אחת מההרשאות הנ"ל.
  - לדוגמה VM_MAYWRITE קובע האם מותר להדליק את VM_WRITE.
  - הדגלים האלה קשורים לקריאת המערכת mprotect() – מעבר לחומר הקורס.
- VM_SHARED – האם צריך לשתף דפים באזור זה עם תהליכי בן.
- VM_LOCKED – אסור לפנות את הדפים באזור מהזיכרון לדיסק.

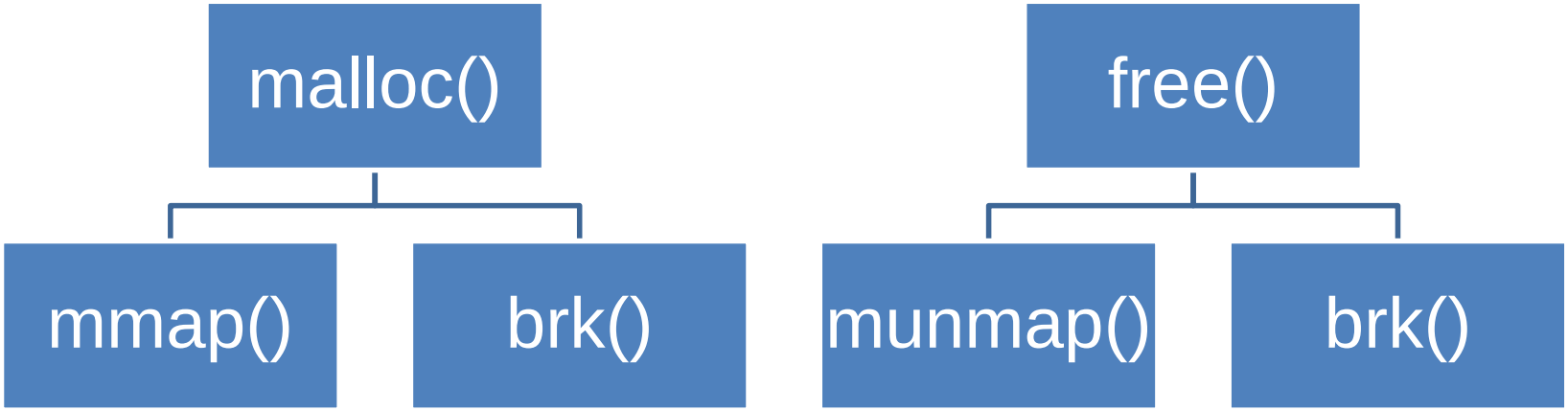
מתי נוצרים אזורי זיכרון?



- עם היווצרו, תהליך מקבל מרחב זיכרון עם אזורי הזיכרון הבאים:
  1. אזור לקוד (code או text).
  2. אזור לנתונים סטטיים (data).
  3. אזור לערימה של הזיכרון הדינמי (heap).
  4. אזור למחסנית user mode.
  5. אזורים נוספים: אחד לפרמטרים של שורת הפקודה, אחד למשתני מערכת.
- הוספה של אזורים נוספים מתאפשרת באמצעות קריאת המערכת **mmap()**.

ניהול זיכרון דינמי

- תהליך משתמש יכול להקצות או לשחרר זיכרון באמצעות פונקציות הספריה malloc() , free() .
- malloc() מחפשת בלוק זיכרון פנוי באזור הערימה.
- במידה ואין מספיק זיכרון פנוי בערימה, malloc() פונה לקריאת המערכת **brk()** כדי להגדיל את אזור זיכרון הערימה.



קריאת המערכת sbrk()

```
void *sbrk(intptr_t increment);
```

- פעולה: מגדילה או מקטינה את הקצה העליון (program break) של אזור הזיכרון של הערימה (heap).
  - אם increment > 0 , אז sbrk() מקצה זיכרון נוסף בערימה.
  - אם increment < 0 , אז sbrk() משחררת זיכרון מהערימה.
  - אם increment == 0 , אז sbrk() מחזירה את ראש הערימה הנוכחי.

קריאת המערכת mmap()

```
void *mmap(void *addr, size_t length,
     int prot, int flags,
     int fd, off_t offset);
```

- פעולה: יוצרת אזור זיכרון חדש ומוסיפה אותו לרשימת אזורי הזיכרון של התהליך.

1. אם fd הוא אינדקס של קובץ פתוח, אז האזור החדש ימפה את אותו קובץ.
2. אם fd==-1, אז האזור החדש הוא אנונימי.

סיווג אזורי זיכרון

מגובה קובץ (file-backed)

- אזור הזיכרון מכיל מידע שמקורו בקובץ.
  - זכרו כי "קובץ" אינו בהכרח קובץ "רגיל" המאוחסן בדיסק.
  - הקובץ נקרא "ממופה לזיכרון", או memory mapped file.
- קריאה/כתיבה לאזור הזיכרון מתורגמת לקריאה/כתיבה למקום המתאים בתוך הקובץ.

אנונימי (anonymous)

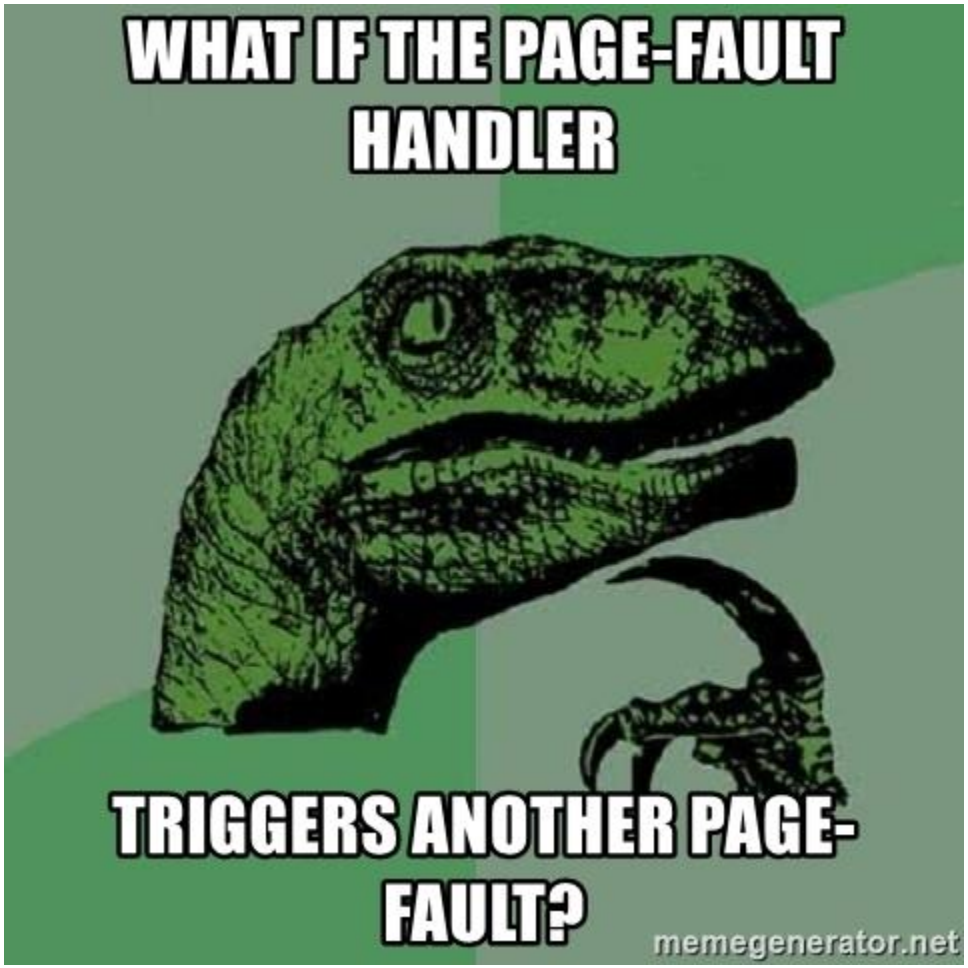
- אזור הזיכרון מכיל מידע שאינו קשור לשום קובץ, אלא לזיכרון הדינמי של התהליך.
- במידה וחסר זיכרון במערכת, הגרעין יכול לפנות דפים אנונימיים למחיצה מיוחדת בדיסק – swap area.

מה הסיווג של אזורי הזיכרון:
ערימה, מחסנית, קוד ?

מרחבי זיכרון וקריאות מערכת

- חוטים הנוצרים ע"י קריאת המערכת clone() משתפים את מרחב הזיכרון ע"י הצבעה לאותו מתאר מרחב הזיכרון של תהליך האב.
  - יש להגדיל את מונה השיתוף (mm_users) של מתאר מרחב הזיכרון של תהליך האב.
- קריאת המערכת execv() ודומותיה טוענות תהליך חדש ולכן הן משחררות את מרחב הזיכרון ומקצות אחד חדש.
- קריאת המערכת fork() מקצה לתהליך הבן מרחב זיכרון משלו.
  - במקרה שכזה צריך להעתיק את מרחב הזיכרון של האב לזה של הבן.
  - בפועל, בדרך-כלל אין באמת העתקה בזכות מנגנון **copy-on-write**.

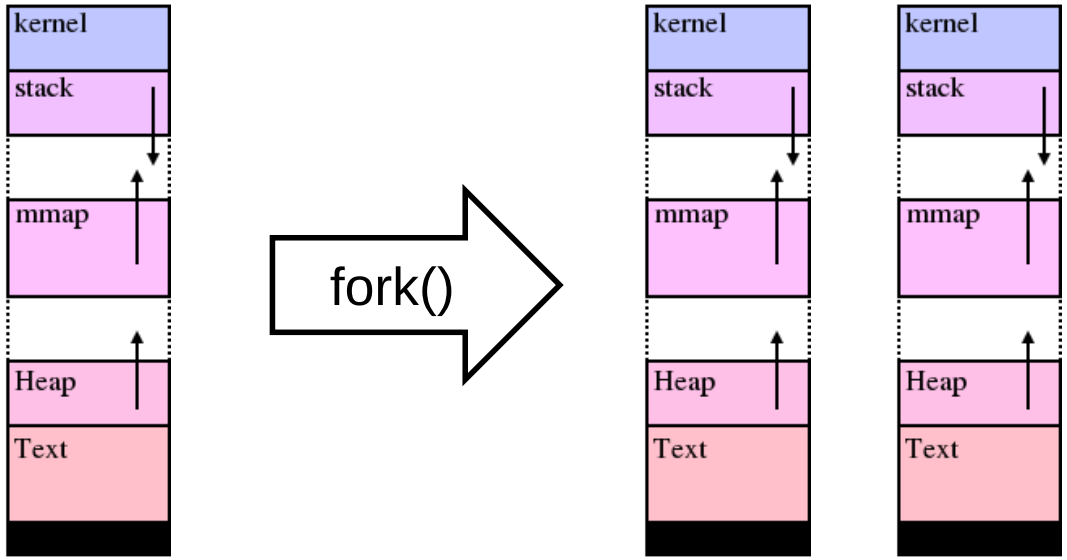
הפסקה


WHAT IF THE PAGE-FAULT HANDLER
TRIGGERS ANOTHER PAGE-FAULT?
memegenerator.net

מנגנון COPY-ON-WRITE

מוטיבציה למנגנון COW

- קריאת המערכת fork() דורשת להעתיק את מרחב הזיכרון של האב לזה של הבן. אבל העתקה פשוטה של מרחב זיכרון היא:
  1. **איטית**: הרבה זמן דרוש להעתקה של כל הדפים.
  2. **אולי מיותרת**: מרחב הזיכרון של תהליך הבן יימחק אם הבן יטען תוכנית חדשה ע"י קריאה ל–execv() מיד בתחילת ריצתו.

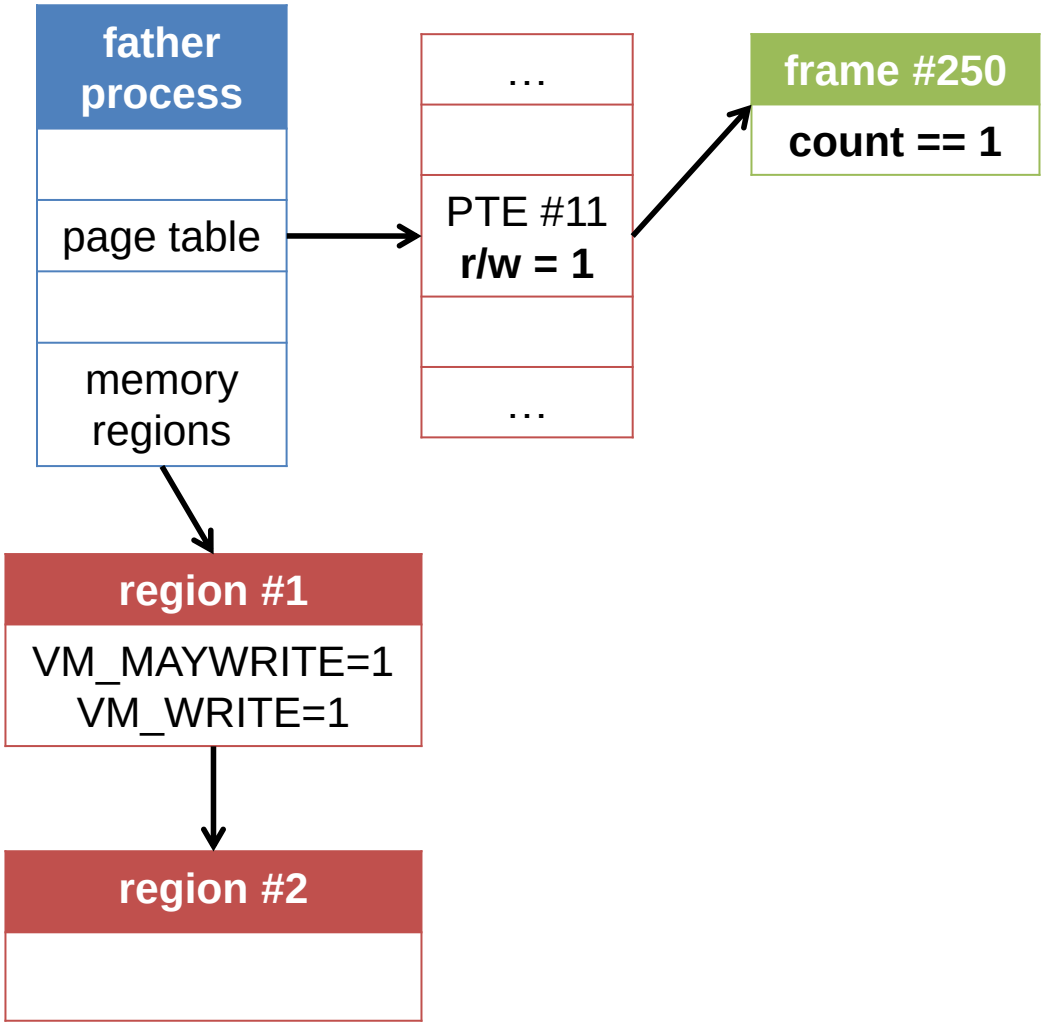


הפתרון: copy-on-write (COW)

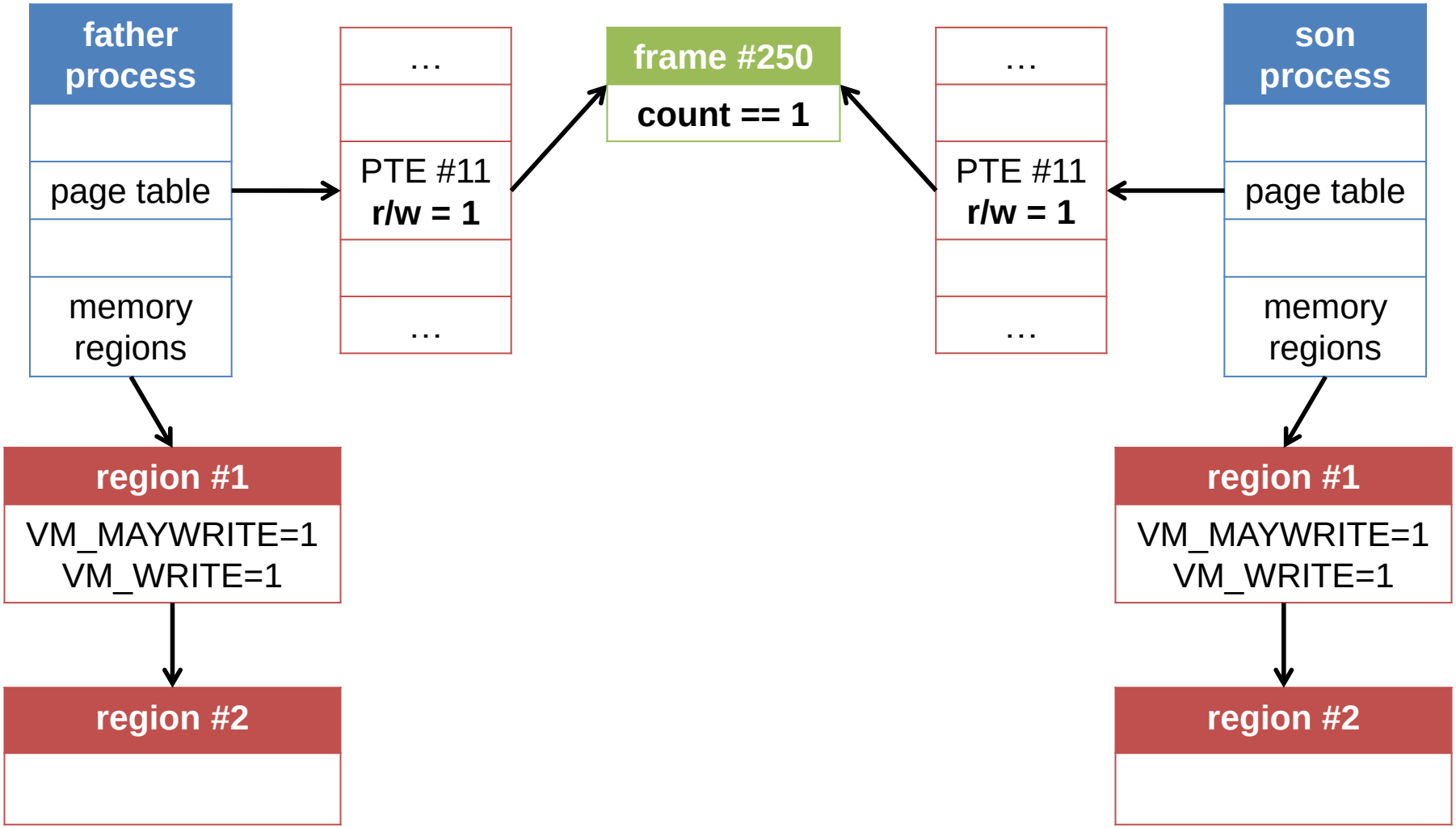
- הרעיון של מנגנון copy-on-write (COW) הוא:
  - דפים הניתנים לכתיבה שאינם יכולים להיות משותפים (לדוגמה, המחסנית), מוגדרים בתחילה כמשותפים אבל **מועתקים לעותק פרטי כאשר אחד התהליכים השותפים (האב או הבן) מנסה לכתוב אליהם לראשונה.**
  - שאר הדפים (כדוגמת דפי קוד או דפי נתונים לקריאה בלבד) הופכים למשותפים בין מרחבי הזיכרון של האב והבן.

- מנגנון COW פותר את שתי הבעיות שהוצגו קודם:
  1. COW מקטין את זמן הביצוע של fork() כי הוא "פורס לתשלומים" את ההעתקה של כל מרחב הזיכרון להרבה העתקות קטנות בגודל דף שיתבצעו בעתיד---בכל כתיבה ראשונה לדף שאינו משותף.
  2. במידה ותהליך הבן יבצע מיד execv(), מרחב הזיכרון שלו יימחק וכך תיחסך רוב פעולת ההעתקה.

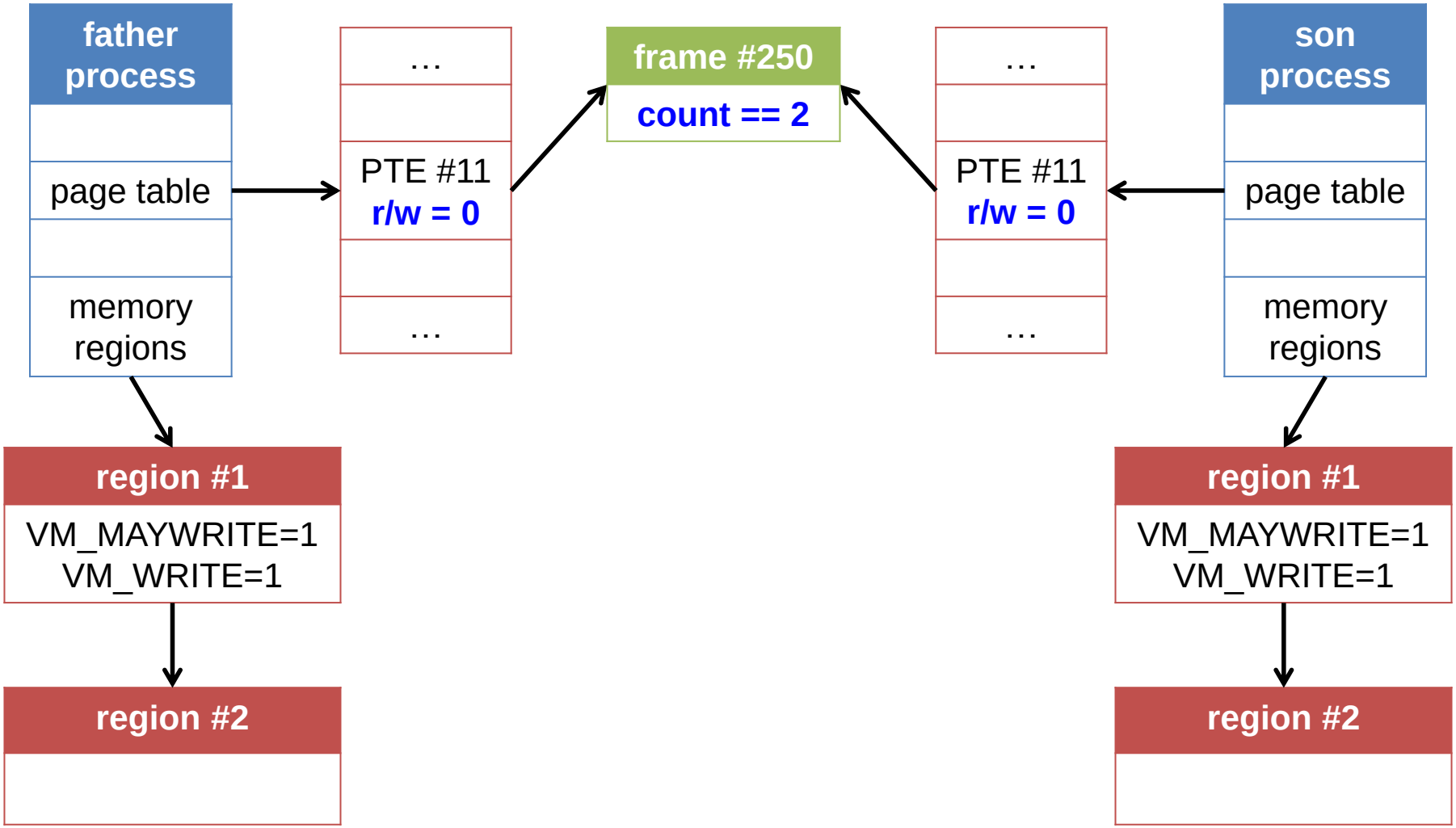
דוגמה: לפני קריאת מערכת fork()



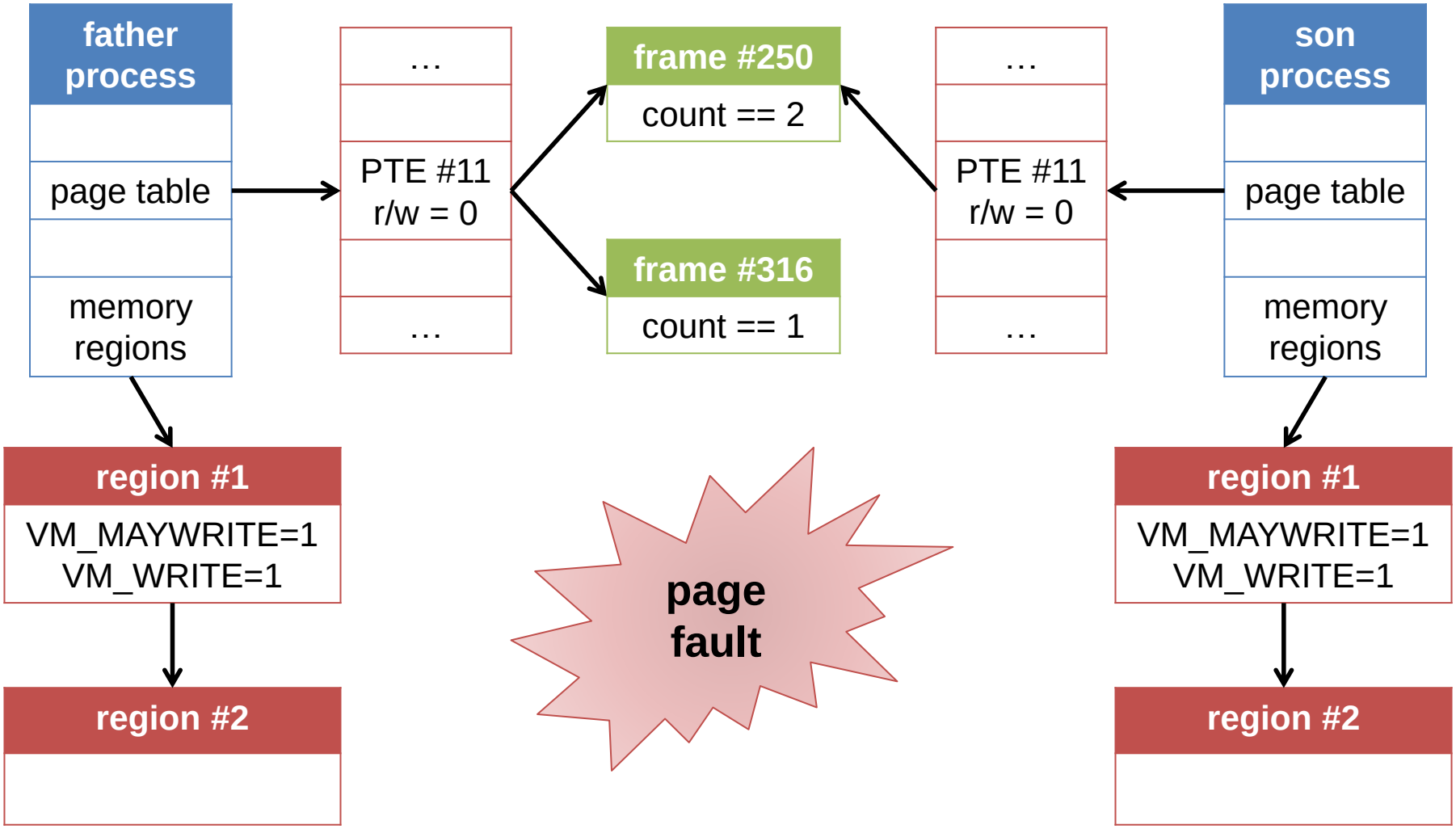
דוגמה: אחרי קריאת מערכת fork()



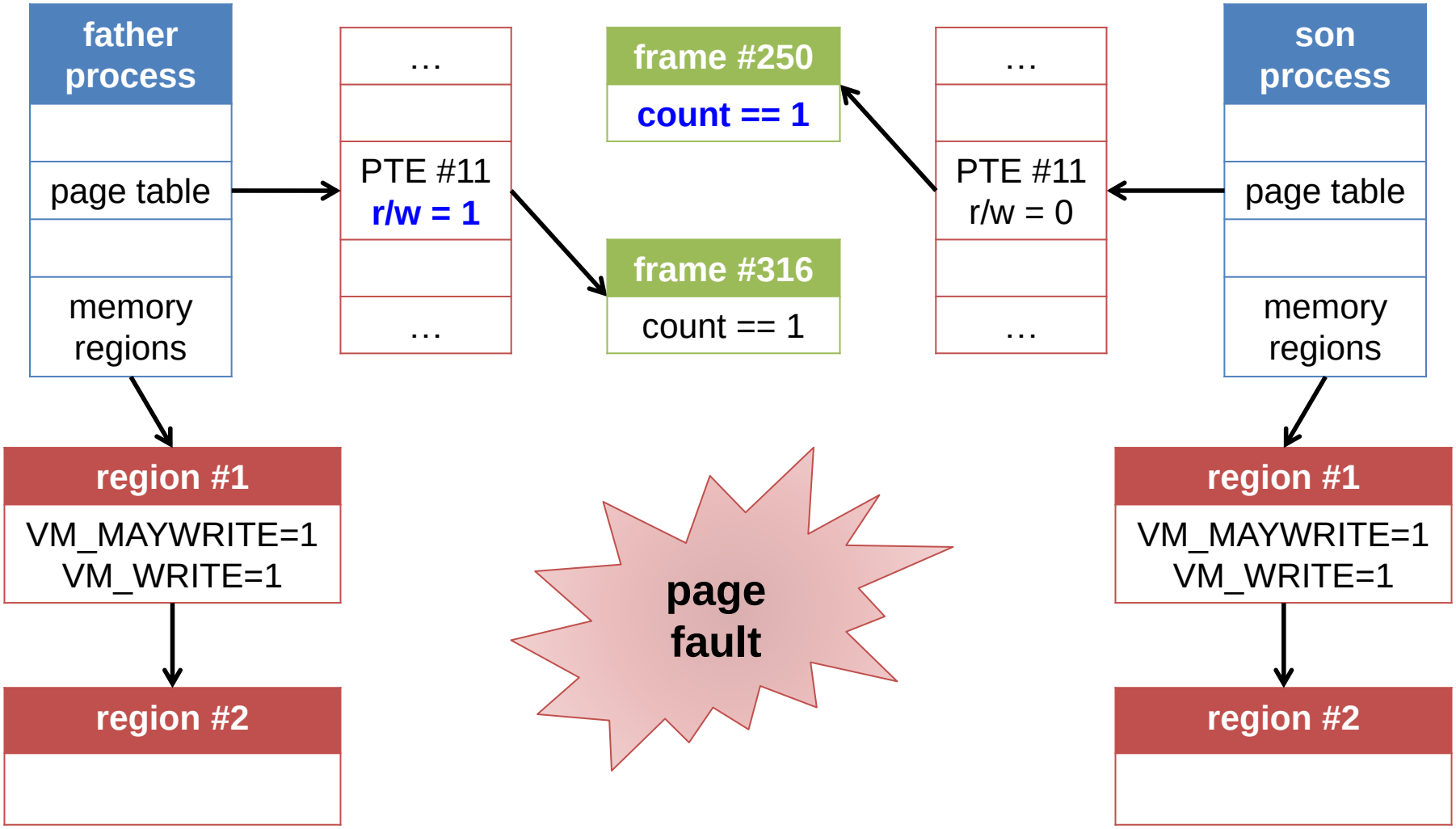
דוגמה: אחרי קריאת מערכת fork()



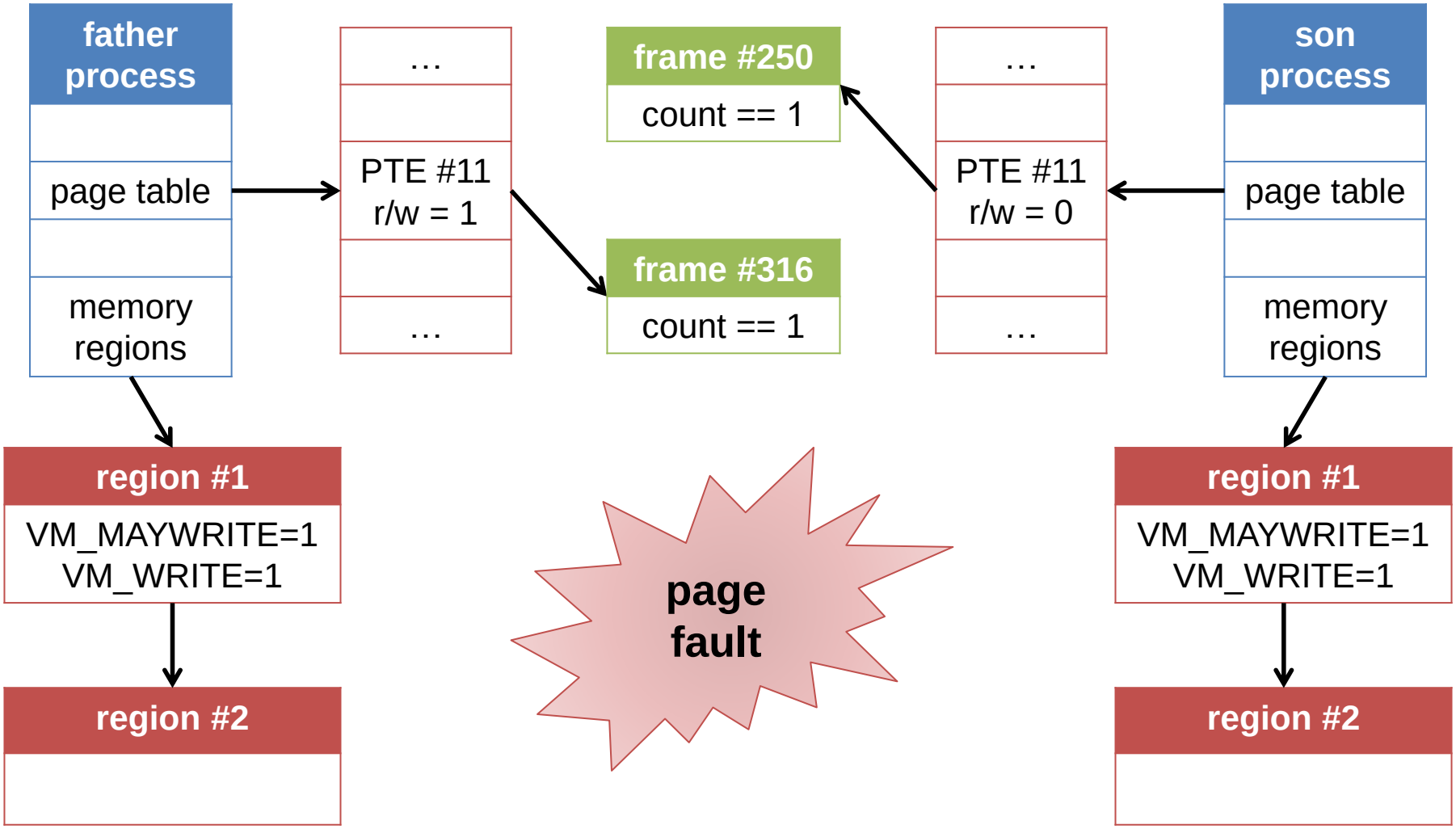
דוגמה: תהליך ראשון מנסה לכתוב



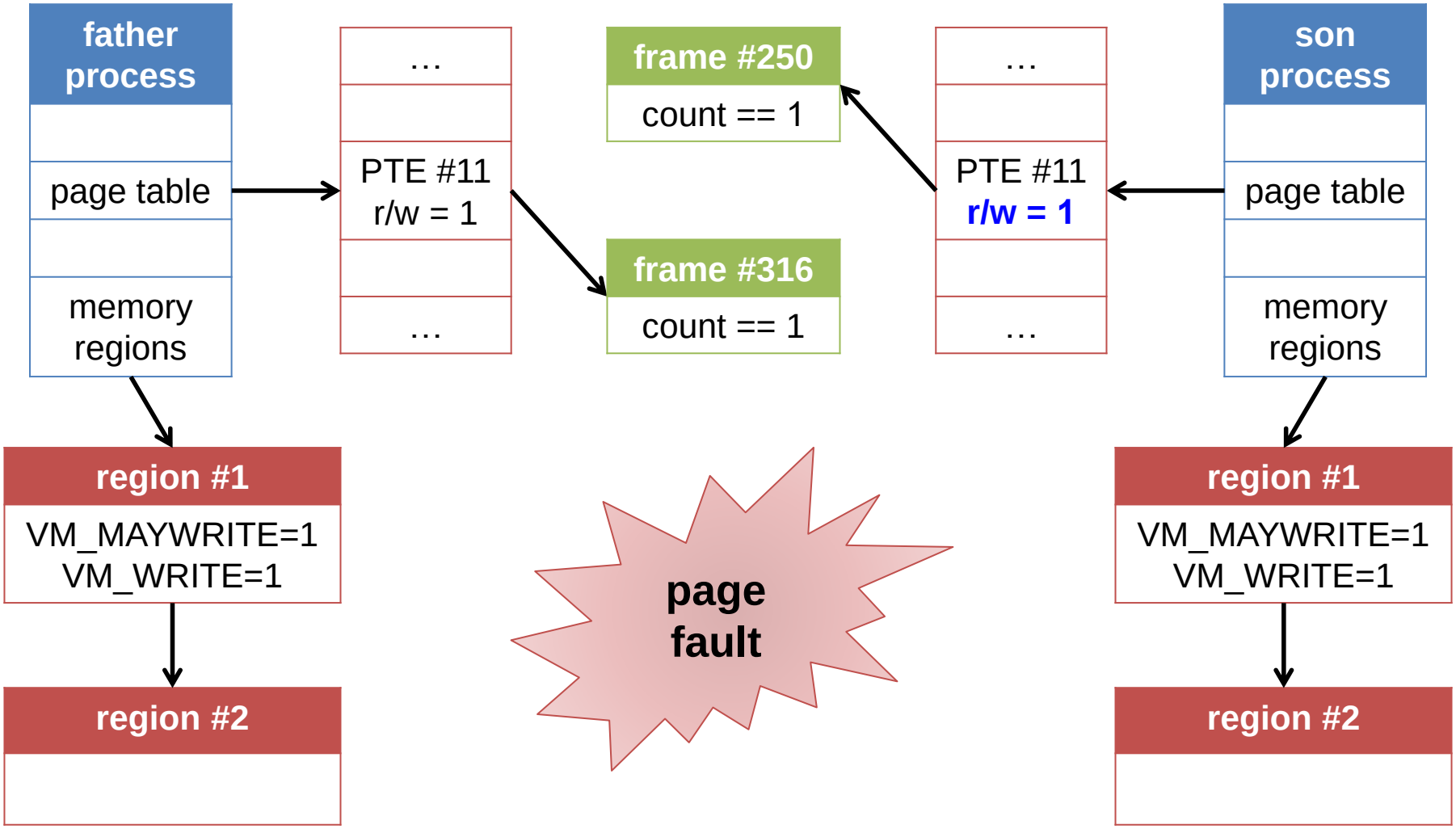
דוגמה: תהליך ראשון מנסה לכתוב



דוגמה: תהליך שני מנסה לכתוב



דוגמה: תהליך שני מנסה לכתוב



COW הוא דוגמה למנגנון "עצל"

- המנגנון מורכב משני שלבים:
1. הגנה על דפים במסגרת קריאת המערכת ()fork.
2. שכפול מסגרות לאחר נסיון כתיבה שגרם ל– page_fault.
- שלב 1 (הגנה על דפים) מאפשר לגרעין לדחות את שלב 2 (שכפול מסגרות) ככל הניתן.
- **<u>שאלה</u>**: האם בהכרח שיפרנו את הביצועים בעזרת COW?
- אם, בסופו של דבר, האב והבן יכתבו לכל הדפים שלהם, לא חסכנו שום עבודה והיינו יכולים להעתיק את כל מרחב הזיכרון מלכתחילה.
  - למעשה אפילו הוספנו עבודה מיותרת של עדכון טבלאות הדפים בשלב 1 + תקורה נוספת של חריגות דף בשלב 2.
- אבל באופן השימוש הנפוץ (fork+execv) חסכנו הרבה עבודה.
  - כי שילמנו רק על שלב 1 שהוא זול יותר משלב 2.

COW: העתקת מרחב זיכרון לתהליך בן

- הפונקציה copy_mm(), המופעלת מתוך do_fork(), "מעתיקה" את מרחב הזיכרון של תהליך האב לתהליך הבן.
- לכל אזור זיכרון של האב:
  - מעתיקה את מתאר אזור הזיכרון לתהליך הבן (עותק חדש ונפרד).
  - מעתיקה את הכניסות המתאימות מטבלת הדפים של האב לזו של הבן.
  - לכל דף באזור הזיכרון, מגדילה את מונה השיתוף של המסגרת המתאימה.

- תהליך ההגנה: קריאת המערכת fork ניגשת לדפים:
  - שאינם משותפים (VM_SHARED כבוי)
  - שניתן לאפשר בהם כתיבה (VM_MAYWRITE דלוק)
- ומכבה את הביט r/w ב-PTE של אותו דף.

COW: טיפול ב-page fault

תרחיש הטיפול:

- האב או הבן מנסים לכתוב לדף מוגן ע"י COW.
- **המעבד** ניגש לסיביות הבקרה ב-PTE של הדף, ומגלה כי r/w כבוי.
- המעבד יוצר חריגת דף (**page fault**).
- **הגרעין** מטפל בחריגה, ובודק שהדף **שייך** לאחד מאזורי הזיכרון ושהגישה בכלל חוקית (דגל VM_WRITE דלוק **במתאר האזור**).
- הגרעין בודק את ערך המונה השיתוף של המסגרת:
  - אם count > 1, מקצים מסגרת חדשה, מעתיקים אליה את המסגרת המקורית, ומצביעים את הדף למסגרת החדשה.
    - במסגרת הישנה מבוצע `count--` .
    - במסגרת החדשה מוצב `count = 1` .
    - בעותק החדש מאופשרת הכתיבה.
  - אחרת (count == 1), הגרעין פשוט מאפשר כתיבה בדף ע"י הדלקת הדגל r/w.

מנגנון DEMAND PAGING

דוגמת קוד

```
char *a = (char*)mmap(NULL, 4096,
      PROT_READ | PROT_WRITE, MAP_ANONYMOUS,
      -1, 0); // OS doesn't allocate memory,
      // only updates the memory region list

x = a[0]; // page fault
      // OS maps the page to the zero page

a[0] = 6; // another page fault
      // OS allocates a new frame
      // and copies the zero page into it
```

הקצאת מסגרות לפי דרישה (Demand Paging)

- כאשר תהליך מקצה זיכרון באמצעות קריאת המערכת mmap(), לינוקס מקצה מסגרות לפי דרישה (demand paging).
  1. בשלב הראשון, **רק רשימת אזורי הזיכרון מתעדכנת.**
  2. הכניסות המתאימות בטבלת הדפים עדיין לא מצביעות למסגרות (ע"י סימון ביט present == 0).
  3. המסגרת מוקצית או מועתקת מהדיסק רק בניסיון הגישה הראשון לדף, בעקבות page fault

Major page fault –
חריגת דף שניגשת לדיסק,
ולכן דורשת יציאה להמתנה.
רלוונטית למיפוי מגובה קובץ.

Minor page fault –
חריגת דף שאינה ניגשת לדיסק,
ולכן אינה חוסמת את התהליך.
רלוונטית למיפוי אנונימי.

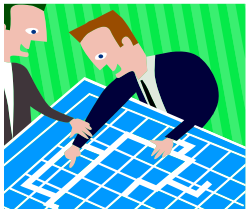
Demand Paging – טיפול בחריגת דף

תחילה, הגרעין בודק אם הכתובת שגרמה לחריגה היא **חוקית**.

במידה וכן:

1. אם אזור הזיכרון ממפה קובץ שנמצא בדיסק, יש לטעון את המסגרת מהדיסק (major page fault).
2. אם מעולם לא ניגשו לאזור הזיכרון (**זיכרון אנונימי קר**):
   1. אם הגישה לכתיבה, מוקצית מסגרת חדשה מלאה באפסים (minor page fault).
   2. אם הגישה לקריאה, הכניסה בטבלת הדפים מצביעה על מסגרת של דף קבוע מיוחד ממולא אפסים, הקרוי ZERO_PAGE.
      - דף זה מסומן read-only, כך שבכתיבה הראשונה לדף הוא ישוכפל לעותק פרטי לפי שיטת COW.

- בכל המקרים, הקצאת מסגרת חדשה עשויה לדרוש גם הוספת כניסות מתאימות בכל הרמות של טבלת הדפים.



חריגת דף (page fault)

- החומרה מתריעה באמצעות **חריגת דף** על:
  - גישה לדף שאינו נמצא בזיכרון, כלומר הביט present==0 בכניסה המתאימה בטבלת הדפים.
  - גישה לא חוקית (שלא לפי ההרשאות בטבלת הדפים) לדף שנמצא בזיכרון, למשל ניסיון כתיבה לדף שמותר לקריאה בלבד.
- חריגת דף מפעילה את שגרת הטיפול הממומשת בפונקציית הגרעין do_page_fault().
- בסיום הטיפול בחריגה **מבוצעת מחדש** ההוראה שגרמה לה.
  - אלא אם כן, כמובן, הטיפול בחריגה הורג את התהליך.

לא כל חריגת דף היא תקלה!

- לינוקס צריכה לנתח את נסיבות החריגה ולהחליט אם היא חוקית וכיצד לטפל בה.
  - כתיבה לדף שמותר לקריאה בלבד עשויה להיות חוקית, למשל ב-COW.
  - קריאה מדף שעבר ל-swap היא חוקית; הגרעין צריך להחזיר את הדף לזיכרון.

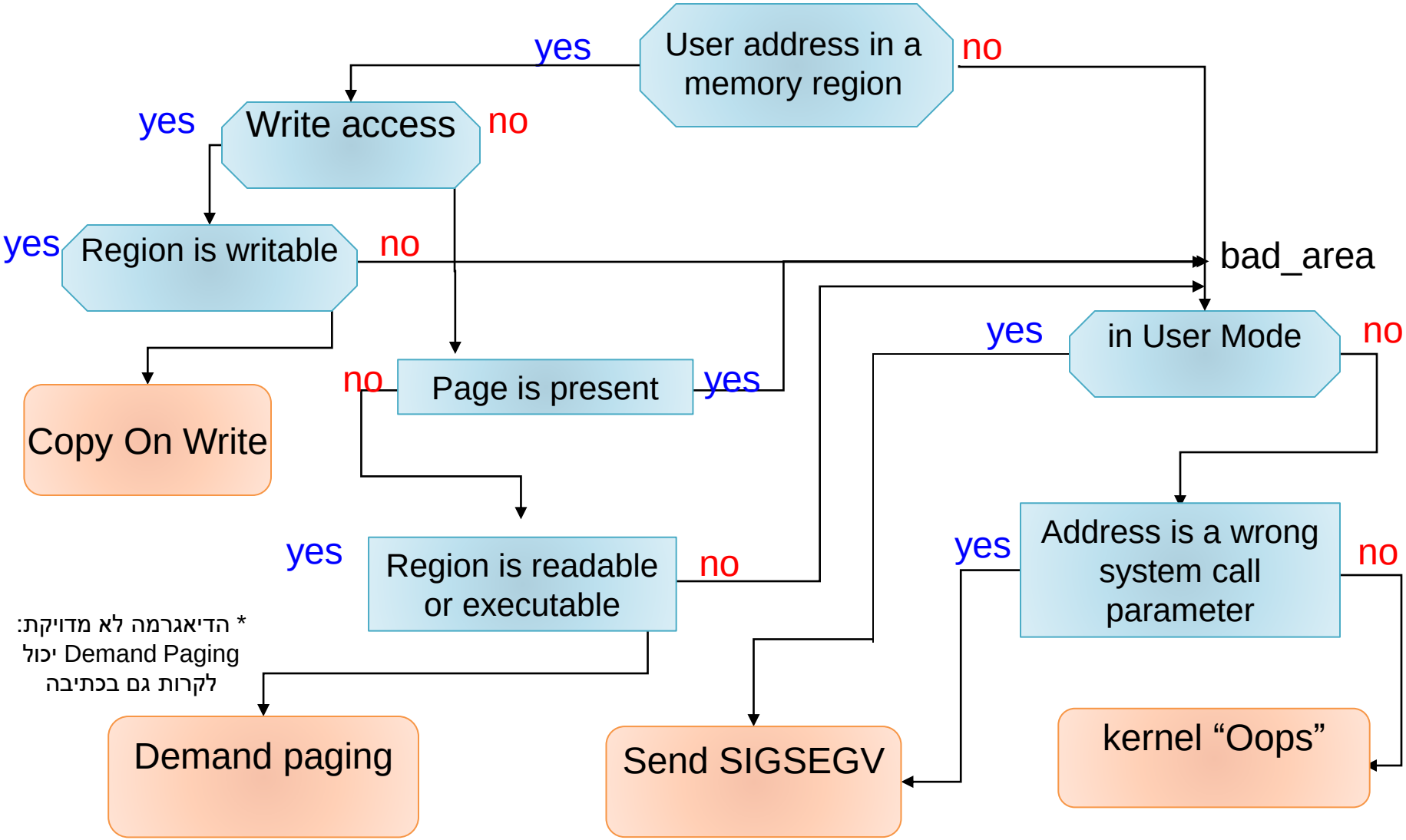
- כדי שמערכת ההפעלה תוכל לטפל בחריגת הדף, החומרה מעבירה לשגרת הטיפול קוד שגיאה של 3 ביטים הנשמר במחסנית:
  - **ביט 0 כבוי**: גישה לדף שאינו בזיכרון (present == 0). אחרת, גישה לא חוקית לדף בזיכרון.
  - **ביט 1 כבוי**: הגישה הייתה לקריאה או לביצוע קוד. אחרת, הגישה הייתה לכתיבה.
  - **ביט 2 כבוי**: הגישה כשהמעבד ב-kernel mode. אחרת, הגישה ב- user mode.
- כמו כן, החומרה מעבירה את הכתובת הווירטואלית שגרמה לחריגה ברגיסטר CR2.

טיפול בתקלות

- החריגה מסווגת כתקלה (גישה לא חוקית) אם:
  1. הפעולה (קריאה או כתיבה) לא מורשית לפי הרשאות האזור.
  2. גישה מקוד משתמש לדפי הגרעין.
  3. גישה לכתובת בתחום המשתמש שאיננה שייכת לשום אזור זיכרון.

- אם הגישה הייתה מקוד תהליך משתמש, נשלח לתהליך signal מסוג SIGSEGV, לציין "גישה לא חוקית לזיכרון".
- אם הגישה הייתה מקוד גרעין, מוכרזת תקלת מערכת - kernel oops.

סיכום: טיפול בחריגת דף במצב משתמש



* הדיאגרמה לא מדויקת:
יכול Demand Paging
לקרות גם בכתיבה