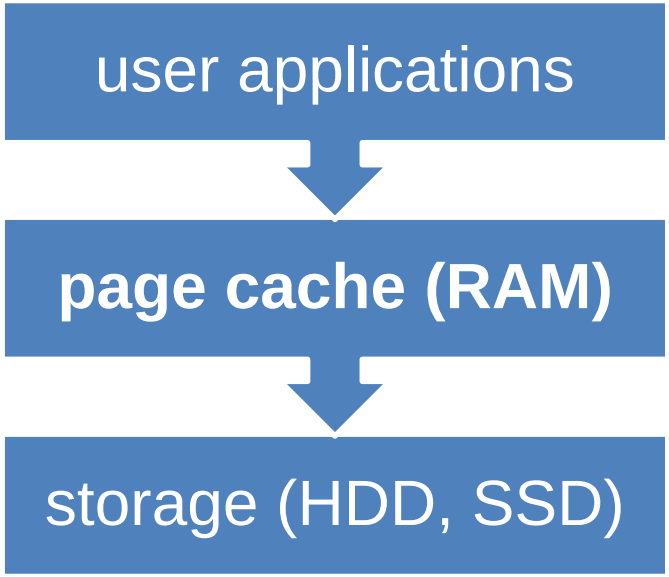
תרגול 10

מטמון הדפים (Page Cache)
מבני נתונים לניהול זיכרון פיזי
אופני גישה למטמון הדפים
אלגוריתם שחרור מסגרות (PFRA)

Special thanks to Gustavo Duartes who provided many of the figures and illustrations

TL;DR

- התקני איחסון (לדוגמה דיסק קשיח או SSD) הם איטיים בכמה סדרי גודל מהזיכרון (RAM).
- תוכניות אשר ניגשות הרבה לדיסק עלולות לסבול מזמני ההשהייה הארוכים.
- כדי להתגבר על הבעיה, מערכת ההפעלה שומרת מידע מהדיסק בזיכרון, במבנה הנקרא מטמון הדפים.
- מטמון הדפים מצמצם את מספר הגישות לדיסק בזכות עיקרון הלוקאליות (במרחב ובזמן).



מטמון הדפים

Page Cache

הבעיה: התקני איחסון איטיים

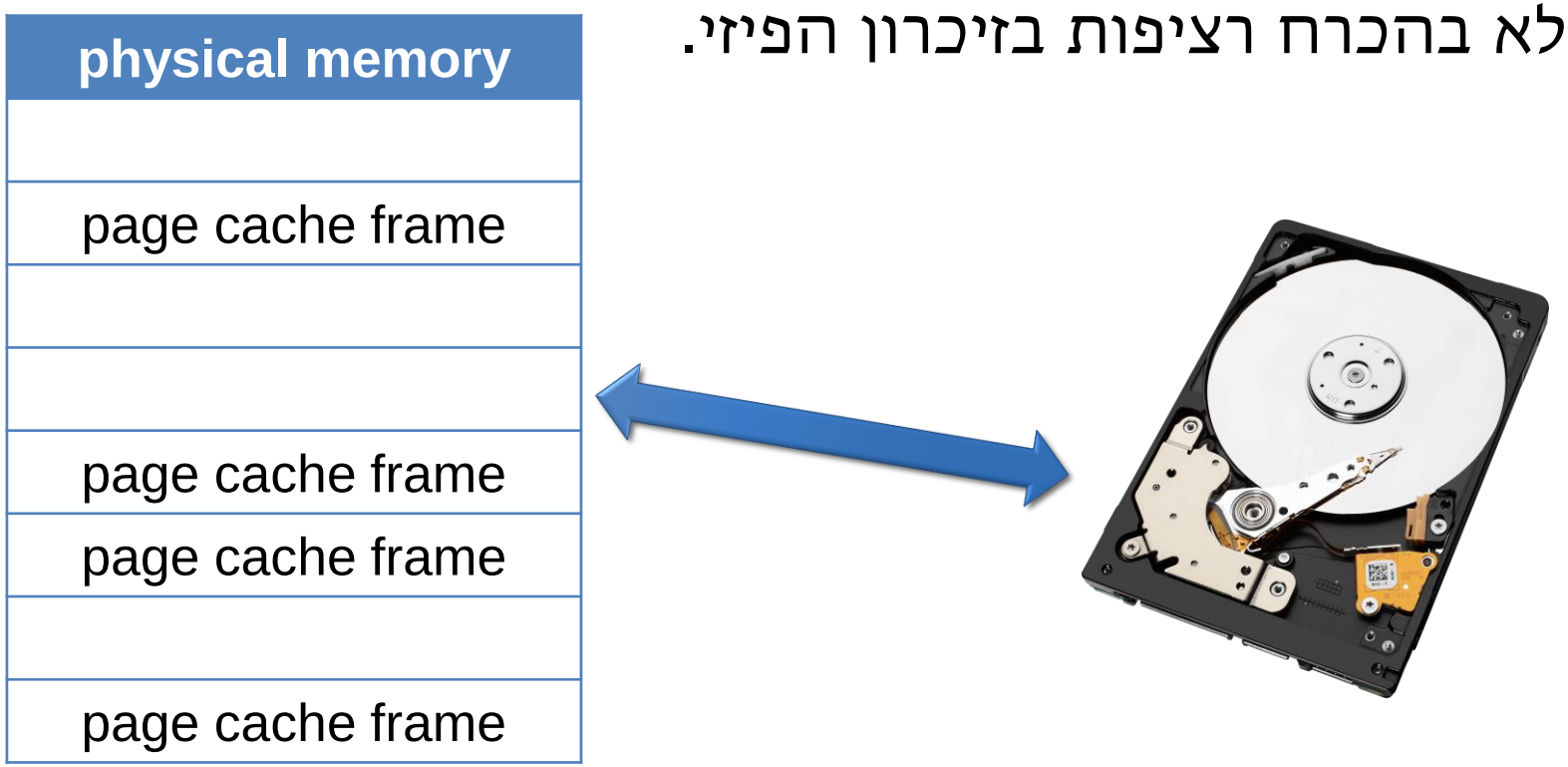
- התקני איחסון הם בעלי השהיה גבוהה יחסית לזיכרון.

- זמני ההשהיה האופיינים (נכון לשנת 2020) בגישה אקראית*:
- זיכרון (DRAM) – 100 ns.
- כונן SSD – 16 us.
- כונן דיסק קשיח (HDD) – 2 ms.
  - המרכיב הדומיננטי הוא זמן הזזת הראש הקורא (seek latency).

- *גישה אקראית מוגדרת כקריאה/כתיבה של 8B בכתובת כלשהי.

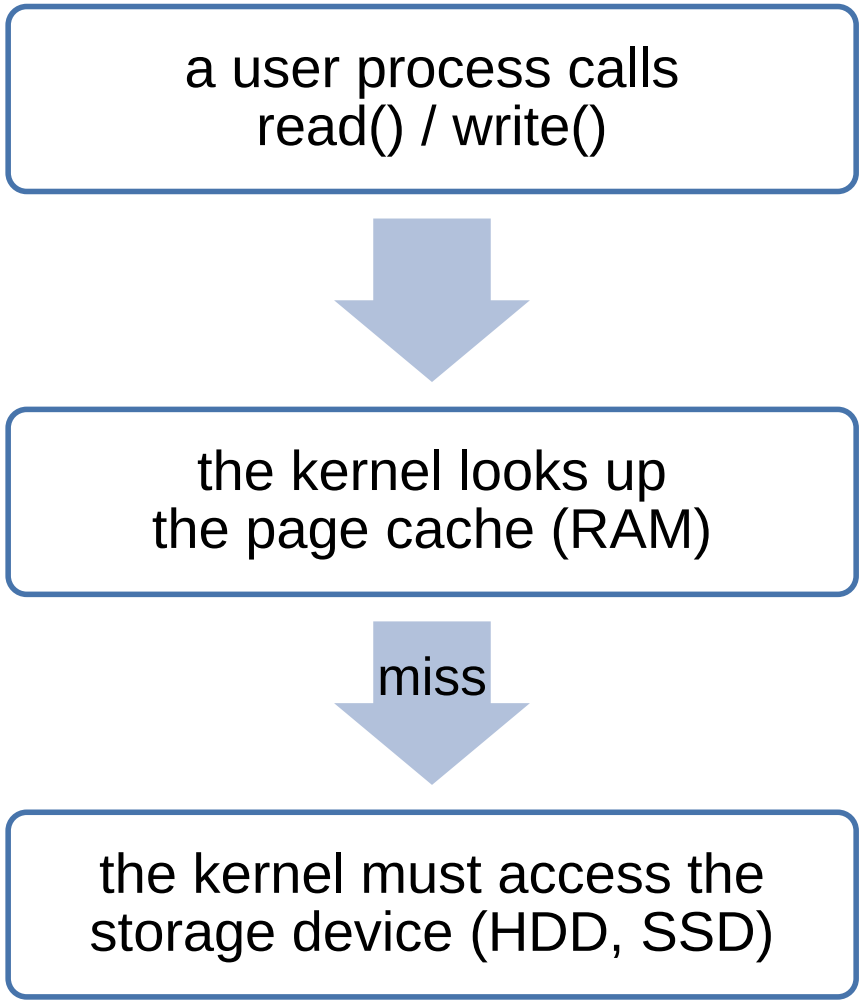
הפתרון: מטמון הדפים

- מטמון הדפים הוא אוסף מסגרות בזיכרון הפיזי השומרות עותק של קבצים שמקורם בהתקני אחסון.
- המסגרות של מטמון הדפים לא בהכרח רציפות בזיכרון הפיזי.



כל גישה לדיסק עוברת דרך מטמון הדפים

- כאשר תכנית מבקשת לקרוא/לכתוב לדיסק, הגרעין בודק קודם אם המידע המבוקש נמצא במטמון הדפים.
- אם כן – הגרעין משרת את הבקשה ממטמון הדפים.
- אם לא – הגרעין יקרא את המידע מהדיסק ואז יוסיף אותו למטמון הדפים.
  - בכל מקרה, הגרעין חייב להביא את המידע המבוקש לזיכרון כי המעבד לא יכול לגשת ישירות לדיסק.



עקרון הלוקאליות

לוקאליות בזמן

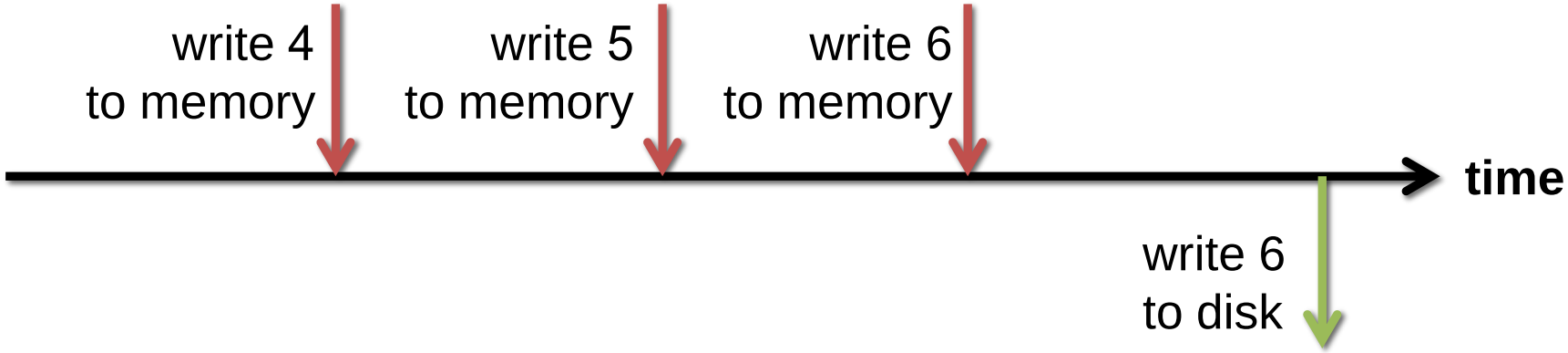
- אם המשתמש ניגש לבית מסוים בדיסק, יש סיכוי גבוה שהוא ייגש שוב לאותו בית בעתיד הקרוב.
  - לדוגמה: תהליך שעורך קוד ותהליך אחריו שמהדר אותו. התהליך השני יחסוך גישה לדיסק אם הקובץ כבר קיים במטמון הדפים.

לוקאליות במרחב

- אם המשתמש ניגש לבית מסוים בדיסק, יש סיכוי גבוה שהוא ייגש לבתים סמוכים בעתיד הקרוב.
  - לדוגמה: תוכנית הקוראת קובץ טקסט שורה אחר שורה.
  - בשורה הראשונה – הגרעין יקרא את הדף הראשון (4KB) של הקובץ מהדיסק.
  - בשורה השניה – המידע כבר קיים במטמון הדפים, ונקרא אותו ללא גישה לדיסק.

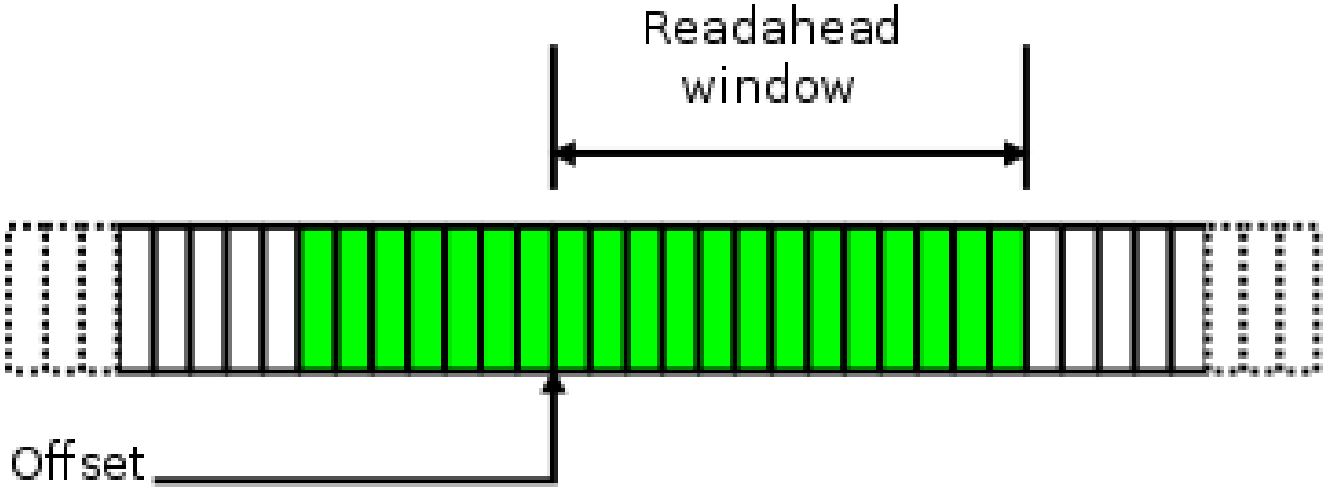
כתיבה מושהית (write-back)

- כתיבות לדיסק נדחות כדי לחסוך (אולי) כתיבות של ערכי ביניים.
- **יתרון:** צמצום של מספר הכתיבות לדיסק.
  - שיפור ביצועים לא רק לתוכנית הכותבת אלא לכלל המערכת: במידה והדיסק עמוס בבקשות קריאה/כתיבה, אז כתיבה מושהית תפחית את מספר הכתיבות הכולל (לעומת כתיבה מיידית, write-through).
- **חיסרון:** אובדן אמינות – המידע בזיכרון עלול ללכת לאיבוד אם יש נפילת מתח.



קריאה מראש (read-ahead)

- אם הגרעין חוזה גישה סדרתית לקובץ, הוא קורא למטמון הדפים את המסגרות הבאות עוד לפני שהתהליך ניגש אליהן.
  - הקריאה מתרחשת ברקע, לא עוצרת את התקדמות התהליך, ולא מבזבזת כמעט זמן מעבד.
- **יתרון:** פחות החטאות במטמון הדפים במידה והחיזוי נכון.
- **חיסרון:** בזבוז זיכרון במידה והחיזוי שגוי.



מבני נתונים לניהול זיכרון פיזי

סיווג מסגרות בזיכרון הפיזי

מסגרות של מטמון הדפים

- מכילות מידע שמקורו בקובץ.
- לדוגמה: איזור הזיכרון של הקוד.
- במידה וחסר זיכרון במערכת, הגרעין יפנה מסגרות אלו לקבצים בדיסק שמהם הן הגיעו.

מסגרות אנונימיות

- מכילות מידע שאינו קשור לשום קובץ, אלא לזיכרון הדינמי של התהליך.
- לדוגמה: איזור הזיכרון של המחסנית והערימה.
- במידה וחסר זיכרון במערכת, הגרעין יפנה מסגרות אלו למחיצה מיוחדת בדיסק – swap area.

טבלת המסגרות

- מערך עם כניסה לכל מסגרת בזיכרון הפיזי.
- כל כניסה במערך היא מסוג **struct page** ומכילה מספר שדות:

1. **refcount** – כמה מרחבי זיכרון מצביעים אל המסגרת?
   - אם ערך המונה הוא 0, אפשר לפנות את המס

המנגנון COW שלמדנו בשיעור שעבר משתמש בשדה זה

2. **mapping** –
   - מצביע ל-inode של הקובץ אם המסגרת שייכת למטמון הדפים. (inode נלמד בתרגול על מערכות קבצים).
   - מצביע ריק (NULL) במקרה של מסגרת אנונימית.
3. **index** –
   - ההיסט מתחילת הקובץ (offset) עבור מסגרת של מטמון הדפים.
   - שדה ריק עבור מסגרת אנונימית.

טבלת המסגרות – שדות נוספים

4. **flags** – דגלים המתארים את מצב המסגרת, כגון:

- PG_dirty – מציין שתוכן המסגרת "מלוכלך", כלומר כתבו למסגרת בעבר.

איך יודעים אילו מסגרות מלוכלכות?

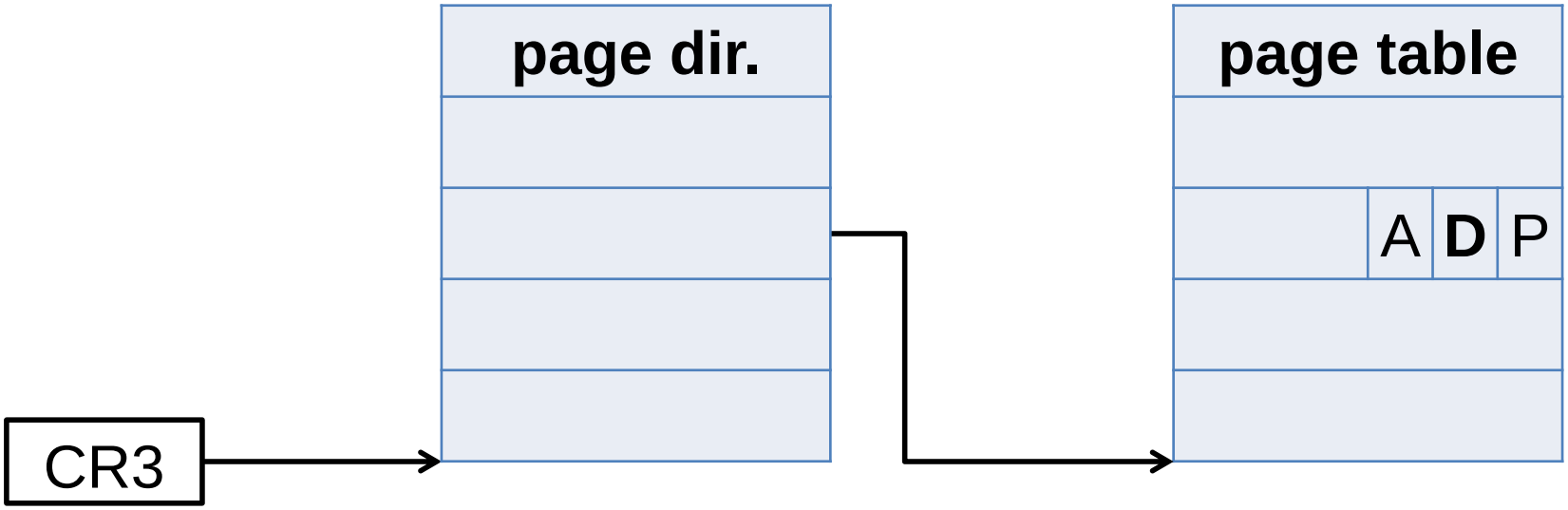
- PG_referenced, PG_active – שומרים את רמת הפעילות (נראה בהמשך).
אלו למעשה שני ביטים ולכן הערכים האפשריים הם 0, 1, 2, או 3.

5. **next_hash, prev_hash** –

- מצביעים למסגרת הבאה/הקודמת בשרשרת ההתנגשות של טבלת ערבול דפים (נראה בהמשך).

ביט dirty בטבלת הדפים

- כאשר המעבד כותב לדף מסוים הוא הולך בטבלת הדפים ואז מדליק את הביט dirty.
  - אם התרגום של הדף קיים ב–TLB אבל הביט dirty כבוי, המעבד ילך בטבלת הדפים כדי להדליק את הביט. הפעולה הזו מתבצעת ברקע ולא מעכבת את המעבד מלהמשיך לפקודה הבאה.



דוגמה: טבלת המסגרות

	refcount	flags	mapping	index
…				
#10	0	0	NULL	---
#11	3	AR	“/usr/lib/libc.so”	0
#12	2	A	NULL	---
#13	1	DR	“/home/assaf/file.txt”	5
#14	0	DA	“/home/dan/main.c”	0
…				

D = dirty
A = active
R = referenced

דוגמה: טבלת המסגרות

- בדוגמה מהשקף הקודם:
- מסגרת 10 היא ריקה.
- מסגרת 11 מצביעה לבלוק הראשון בקובץ /usr/lib/libc.so.
  - שלושה תהליכים שונים משתמשים כרגע במסגרת הזו.
- מסגרת 12 היא אנונימית.
  - שני תהליכים משתפים כרגע את המסגרת הזו (למשל אבא ובן אחרי fork()).
- מסגרת 13 מצביעה לבלוק השישי בקובץ /home/assaf/file.txt.
  - תהליך אחד בלבד משתמש כרגע במסגרת הזו.
- מסגרת 14 מצביעה לבלוק הראשון בקובץ /home/dan/main.c.
  - אף תהליך לא משתמש כרגע במסגרת הזו

למה לינוקס לא מפנה מיד את המסגרת הזו?

סיבוכיות גישה לטבלת המסגרות

- נסמן ב–N את מספר המסגרות הכולל.
- הכנסה ומחיקה של מסגרת באינדקס ספציפי בטבלה – O(1).
- מציאת מסגרת ריקה – O(N).
- כדי להקטין את הסיבוכיות, לינוקס משתמשת במבני נתונים נוספים.
  - לדוגמה ה–buddy allocator – לא נלמד עליו בקורס.
- חיפוש מסגרת ספציפית של קובץ X בהיסט a – O(N).
- כדי להקטין את הסיבוכיות, לינוקס משתמשת בטבלת ערבול דפים (page hash table).

טבלת ערבול דפים

- טבלת ערבול דפים מספקת מיפוי מהזוג (mapping, index) לכתובת מסגרת (אם יש כזו) המכילה את הדף במיקום index של האובייקט mapping.
- כמו בכל טבלת ערבול, יכולות להיות "התנגשויות": זוגות שונים של (mapping, index) יכולים להתמפות לאותו אינדקס בטבלת המסגרות.
- לינוקס מחברת את כל הזוגות המתנגשים לרשימה מקושרת כפולה מעגלית דרך השדות next_hash, prev_hash בטבלת המסגרות.
- כדי לחפש בטבלת ערבול דפים, לינוקס עוברת על כל המסגרות ברשימה ובודקת אם יש מסגרת עם (mapping, index) המבוקש.
- סיבוכיות החיפוש – O(1) בממוצע.

דוגמה: טבלת ערבול דפים

טבלת

	struct page *
#0	
...	
#50	
...	
#60	
...	
#70	
...	
#80	

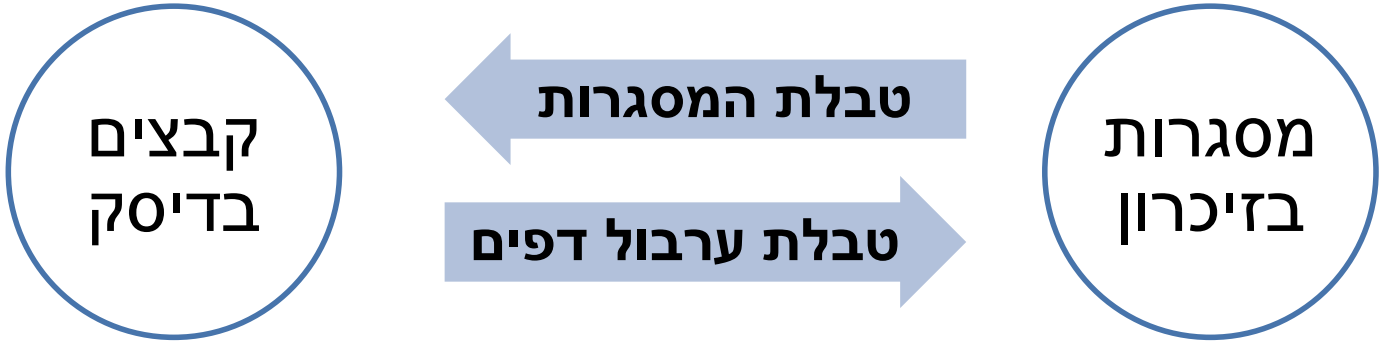
טבלת המסגרות

ערבול דפים

	next hash	prev hash	mapping	index
#0				
...				
#100			X	a
...				
#200			Y	b
...				
#300			Z	c
...				
#400			NULL	---

סיכום ביניים

- לינוקס שומרת מיפוי דו–כיווני:



- שני מבני הנתונים ממומשים בתוכנה בלבד ללא תמיכת חומרה.

- איך מטמון הדפים משתמש במבני הנתונים הללו?
  - כאשר לינוקס צריכה לקרוא דף מהדיסק, היא בודקת קודם בטבלת ערבול דפים אם המידע המבוקש כבר נמצא במטמון הדפים.
  - כאשר לינוקס צריכה לפנות דף "מלוכלך" לדיסק, היא בודקת בטבלת המסגרות את המיקום של הדף בדיסק.

הפסקה

אופני גישה למטמון הדפים

בלינוקס יש שני אופני גישה לדיסק

קריאה/כתיבה באמצעות קריאות המערכת **read()/write()**

- קריאת המערכת מביאה את המידע מהדיסק למטמון הדפים, ואז מעתיקה את הנתונים הרלוונטים אל או מהחוצצים (buffer) של המשתמש.

מיפוי קובץ לזיכרון באמצעות קריאת המערכת **mmap()**

- קריאת המערכת יוצרת איזור זיכרון חדש ולא קוראת/כותבת מידע מהדיסק.
  - ניהול זיכרון עצל/דחייני.
- נסיון גישה לזיכרון יגרום לחריגת דף שתקרא את המידע מהדיסק למטמון הדפים, ואז תעדכן את טבלת הדפים להצביע ישירות למסגרות של מטמון הדפים.

תזכורת: שני אופני פעולה של mmap()

מגובה קובץ (file-backed)

```
int fd=open("file",…);
mmap(…, fd, …);
```

- אזור הזיכרון מכיל מידע שמקורו בקובץ פתוח.
  - לדוגמה: אזור הקוד, אשר מגבה את הקובץ הבינארי של התוכנית.
- קריאה/כתיבה לאזור זיכרון מגובה קובץ מתורגמת לקריאה/כתיבה למקום המתאים בתוך הקובץ.

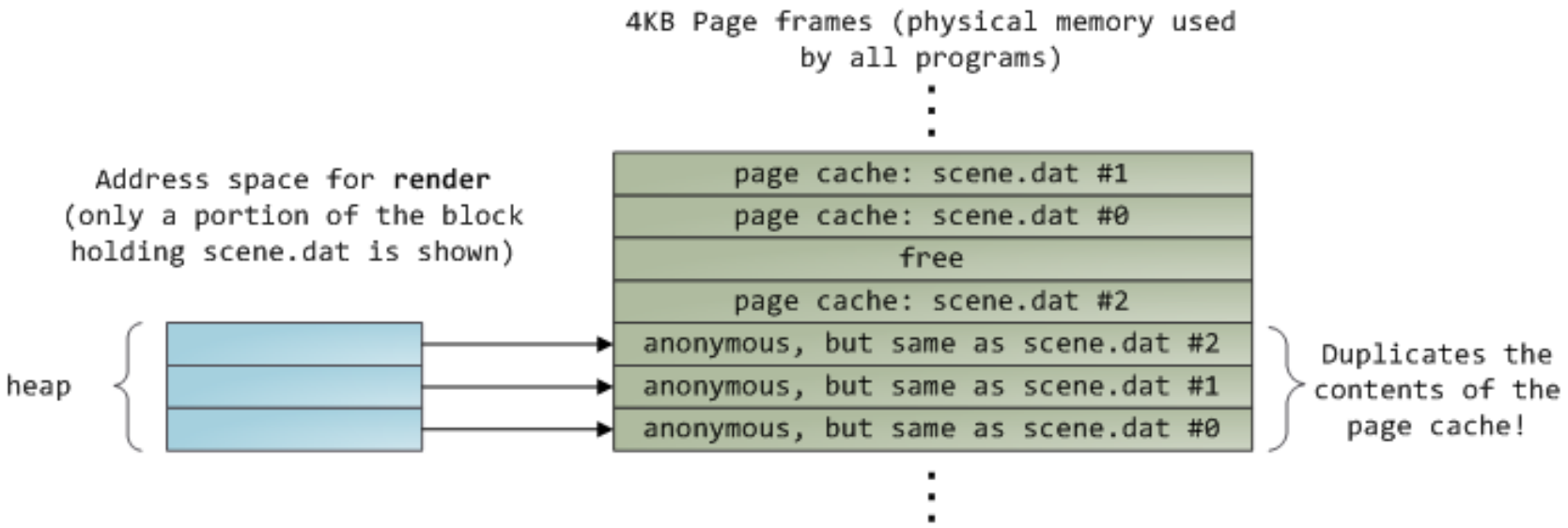
אנונימי (anonymous)

```
mmap(…, fd=-1, …);
```

- אזור הזיכרון מכיל מידע שאינו קשור לשום קובץ, אלא לזיכרון הדינמי של התהליך.
- לדוגמה: אזורי הזיכרון של המחסנית והערימה.

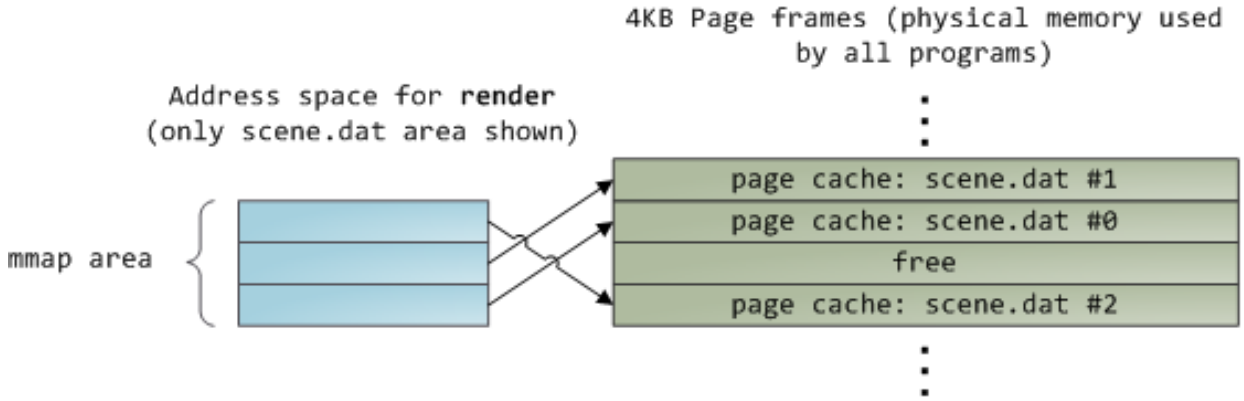
גישה באמצעות write()/read()

```
int fd = open("scene.dat", O_RDONLY);
char* buffer = (char*)malloc(3*PAGE_SIZE);
read(fd, buffer, 3*PAGE_SIZE);
```



גישה באמצעות mmap()

```
int fd = open("scene.dat", O_RDONLY);
char *array = (char*)mmap(NULL, 3*PAGE_SIZE,
    PROT_READ, MAP_SHARED, fd, 0);
x = array[0]; // 1st page fault
x = array[0]; // no page fault
x = array[PAGE_SIZE-1]; // no page fault
y = array[PAGE_SIZE]; // 2nd page fault
z = array[2*PAGE_SIZE]; // 3rd page fault
```



יתרונות השימוש ב-mmap() (לעומת read/write())

- **חיסכון בזמן** – אין צורך להעתיק את המידע ממטמון הדפים לחוצצים של התהליך.
- **חיסכון בזיכרון** – אין חוצצים במרחב המשתמש ולכן אין שכפול מידע שכבר קיים במטמון הדפים.
  - בנוסף, אם המיפוי משותף (נראה בשקפים הבאים), אז כל התהליכים משתפים ביניהם את אותה מסגרת בזיכרון הפיזי.
- **ממשק פשוט ונוח למתכנת** – ניגשים לקובץ כפי שניגשים לזיכרון.
  - קריאת מערכת אחת במקום הרבה קריאות מערכת read(),write().
- **קריאה דחיינית** – העתקת המידע נעשית רק בעקבות ניסיון גישה שיוצר חריגת דף.
  - שימו לב: זה יכול להיות חיסרון של mmap(), כפי שמוסבר בשקף הבא.

חסרונות השימוש ב-mmap() (לעומת read/write())

- כדי למפות קובץ שלם יש למצוא אזור זיכרון פנוי ורציף במרחב הווירטואלי שהוא בגודל הקובץ.
- המגבלה הזו לא באמת משמעותית במעבדי 64 ביט.

מה המגבלה על גודל הקובץ?

- קריאת המערכת mmap() איטית יחסית ל-read().
- read() עדיפה כאשר קוראים רק כמה בתים מהקובץ כי אז המחיר של קריאת המערכת mmap() + חריגת דף עלול להיות גדול יותר מזמן הגישה לדיסק.

מיפוי משותף מול פרטי

מיפוי משותף
MAP_SHARED

- מטמון הדפים מחזיק עותק יחיד של המידע.
- כל המיפויים המשותפים מצביעים לעותק הזה.
- כתיבות מצד תהליך כלשהו ייראו גם אצל תהליכים אחרים הממפים את אותו הקובץ.
- כתיבות לאיזור הזיכרון יחלחלו בסופו של דבר לקובץ בדיסק.

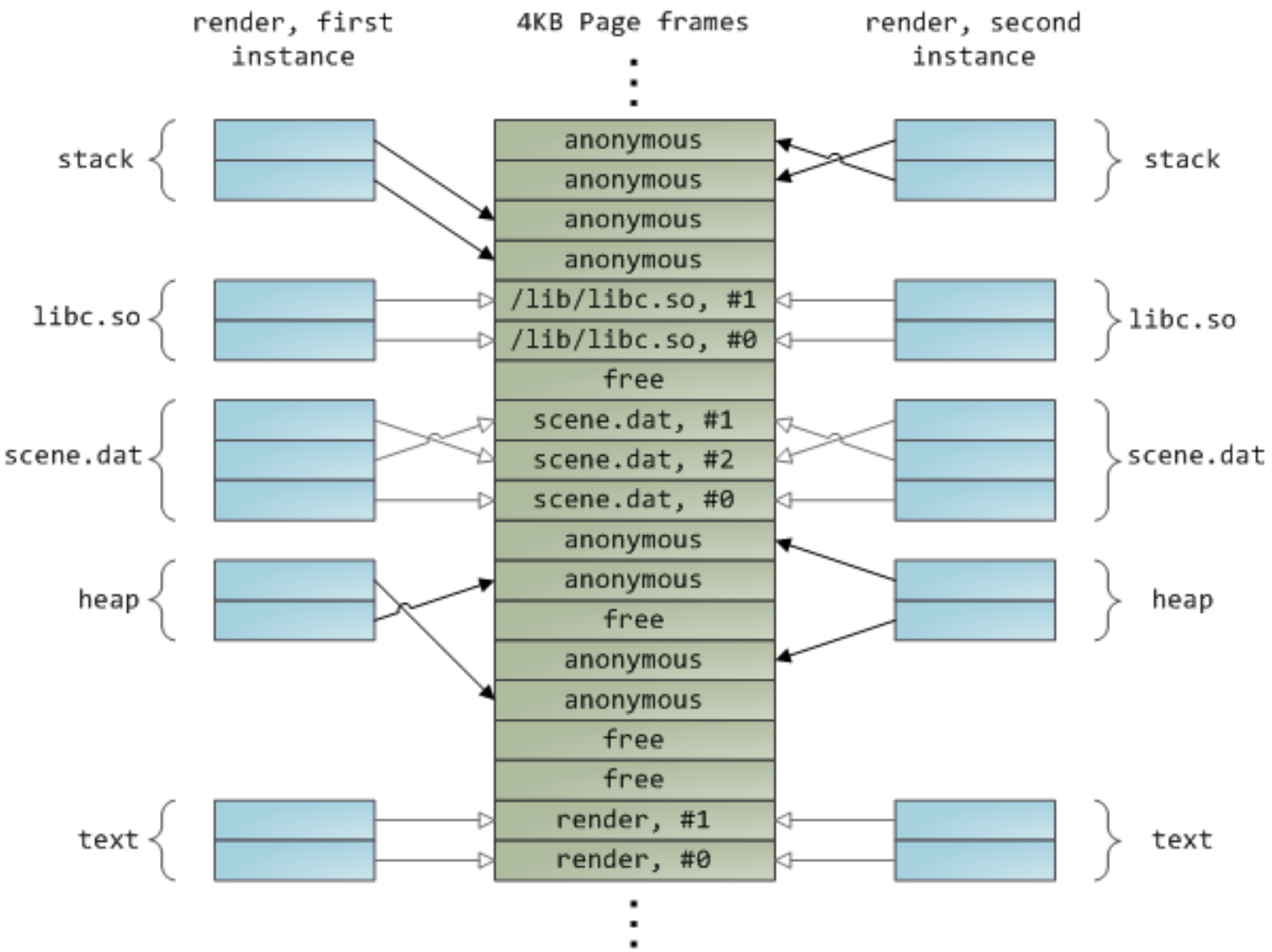
מיפוי פרטי
MAP_PRIVATE

- מטמון הדפים מחזיק עותק יחיד של המידע, אבל הוא מוגן באמצעות copy-on-write.
- בהתחלה, כל המיפויים מצביעים לעותק הזה.
- כתיבות מצד תהליך כלשהו יגרמו לחריגת דף והעתקת המידע למסגרת חדשה.
- כתיבות לא יגיעו חזרה לדיסק.

שיתוף זיכרון במטמון הדפים

- תכניות שונות משתמשות לפעמים בקבצים זהים.
  - לדוגמה: כל התכניות הכתובות בשפת C משתמשות בספריה הדינמית libc.
- כדי לא לטעון עותקים זהים ומיותרים של הקובץ בזיכרון, מטמון הדפים שומר עותק יחיד וכל התהליכים מצביעים לעותק זה.
  - כלומר, הכניסות המתאימות בטבלת הדפים מצביעות לאותה המסגרת.
- למשל בשרטוט המופיע בשקף הבא, שני תהליכים המריצים את אותה התוכנית מצביעים לאותו עותק של הספריה libc, לאותו עותק של התוכנית (render), ולאותו עותק של קובץ אחר (scene.dat).
  - כל המיפויים מוגדרים כפרטיים, כלומר הם מוגנים ע"י COW.

שיתוף זיכרון במטמון הדפים

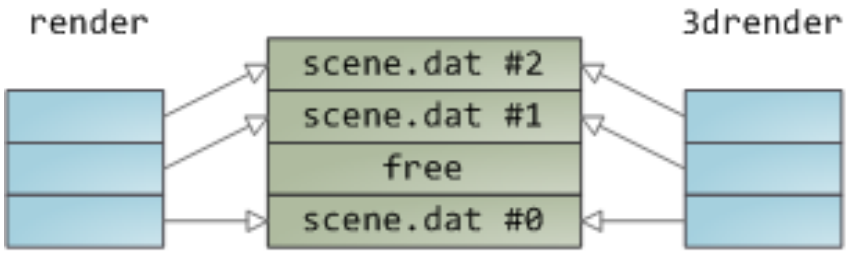


COW במטמון הדפים

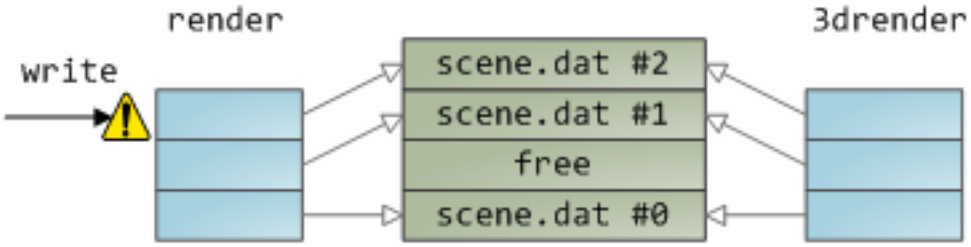
Page table entry marked read-only

Page table entry marked read/write

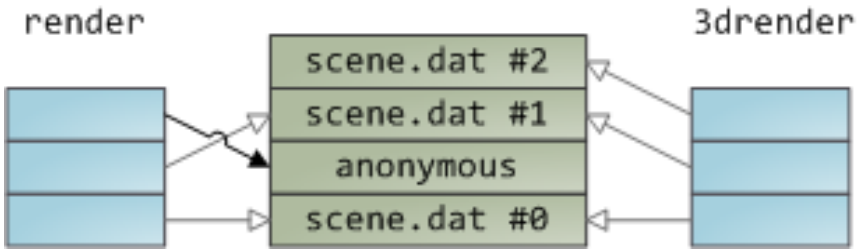
1. Two programs map scene.dat privately. Kernel deceives them and maps them both onto the page cache, but makes the PTEs read only.



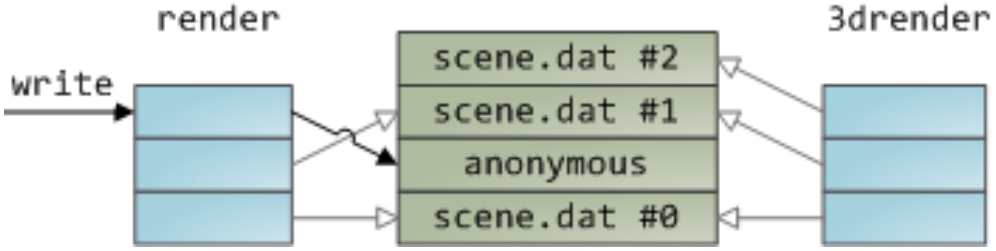
2. Render tries to write to a virtual page mapping scene.dat. Processor page faults.



3. Kernel allocates page frame, copies contents of scene.dat #2 into it, and maps the faulted page onto the new page frame.



4. Execution resumes. Neither program is aware anything happened.

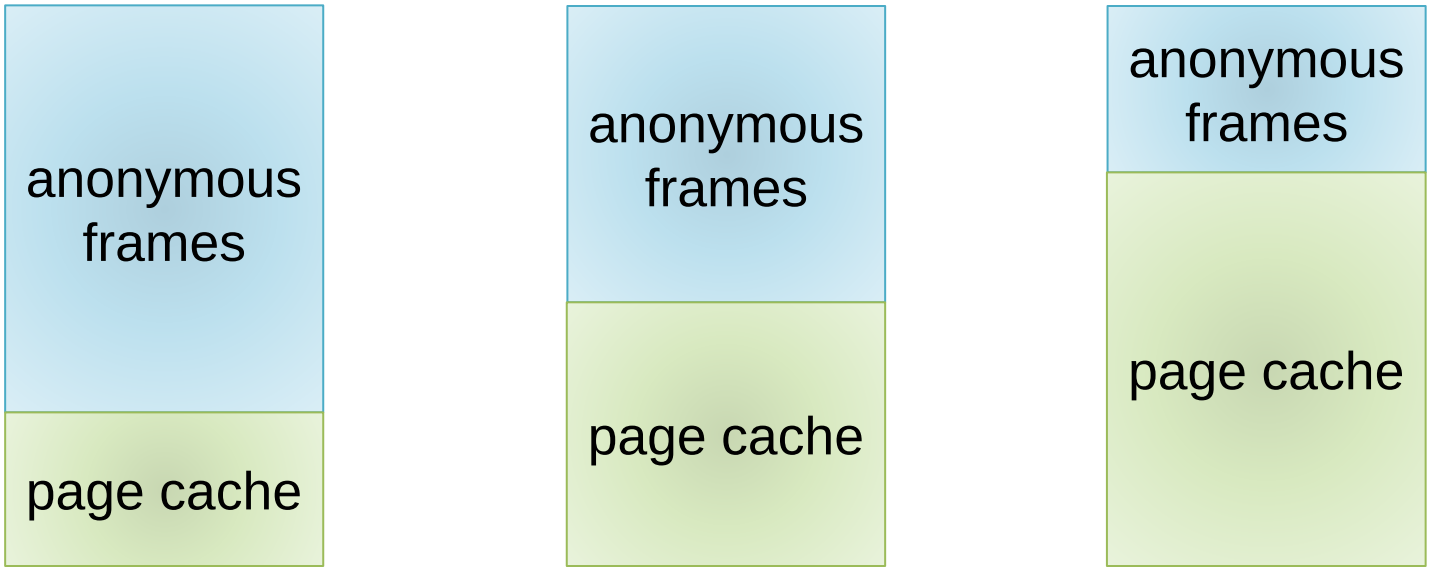


אלגוריתם שחרור מסגרות

Page Frame Reclamation Algorithm (PFRA)

מה גודל מטמון הדפים?

- מצד אחד, תהליכים שניגשים הרבה לדיסק יעדיפו מטמון דפים גדול כדי לחסוך גישות יקרות לדיסק.
- מצד שני, תהליכים חישוביים יעדיפו מטמון דפים קטן כדי שלא יבוא על חשבון זיכרון אנונימי לתהליכים.
  - מעט זיכרון לתהליכים ← יותר swapping ← יותר גישות לדיסק.



מה גודל מטמון הדפים?

- בלינוקס אין חסם על גודל מטמון הדפים.
- כל בקשה להוסיף מסגרת למטמון הדפים נענית בחיוב.
  - כאשר יש הרבה זיכרון פנוי ניתן לנצל אותו כדי להגדיל את מטמון הדפים וכך לשפר את הביצועים שלו.
- גם בקשות של תהליכים להקצאת מסגרת אנונימית נענות תמיד בחיוב.

- אבל מה קורה כאשר אין יותר מסגרות פנויות?

אלגוריתם שחרור מסגרות

- כאשר לא נותרו עוד מסגרות פנויות, הגרעין קורא לאלגוריתם שחרור מסגרות (**PFRA** = Page Frame Reclamation Algorithm).
- מסגרות של מטמון הדפים יפונו חזרה לקבצים המתאימים בדיסק רק אם הן מסומנות dirty, כלומר אם נכתב אליהן מידע חדש.
  - אם המסגרת נקיה אין צורך לכתוב אותה שוב לדיסק כי הקובץ המקורי ממנו הגיעה המסגרת כבר שומר אותה.
- מסגרות אנונימיות יפונו לאיזור מיוחד בדיסק הנקרא מאגר דפדוף. מסגרת אנונימית תמיד תפונה לדיסק, בין אם היא מלוכלכת או נקיה, בגלל שהיא נמצאת בשימוש של תהליך פעיל.

- שימו לב: PFRA מפנה רק מסגרות של תהליכי משתמש, ולעולם לא מפנה מסגרות בשימוש הגרעין.

מאגרי דפדוף בלינוקס

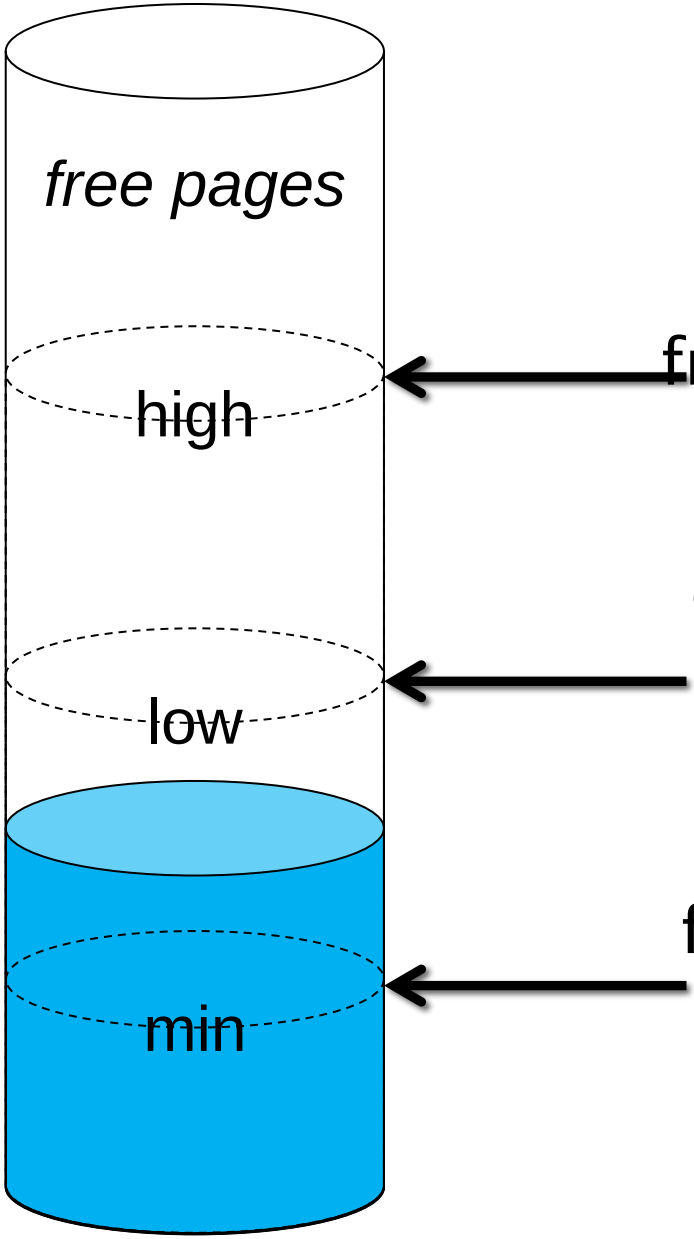
- מאגר דפדוף (swap area) הוא אזור מיוחד בדיסק אליו מפונים דפים מהזיכרון.
- לינוקס מאפשרת להגדיר מספר מאגרי דפדוף, ובנוסף ניתן להפעיל ולכבות מאגרי דפדוף באופן דינמי תוך כדי פעולת המערכת.

מה היתרון של מספר מאגרי דפדוף שונים?

- כל מאגר דפדוף הוא שטח דיסק המחולק למגירות (slots).
  - כל מגירה היא בדיוק בגודל דף / מסגרת (4KB).
  - המגירה הראשונה מכילה מידע ניהולי על המאגר: גודל, גרסה, וכו'.
- אלגוריתם הדפדוף משתדל להקצות מגירות ברצף לדפים מפונים.

למה עדיף ברצף?

מתי מפנים זיכרון?



- כאשר free_pages > **pages_high** PFRA מפסיק לפעול.
- כאשר free_pages < **pages_low** PFRA מתחיל לפעול באופן אסינכרוני (ברקע).
- כאשר free_pages < **pages_min** PFRA מתחיל לפעול באופן סינכרוני (בחזית).

שני אופנים לפינוי זיכרון

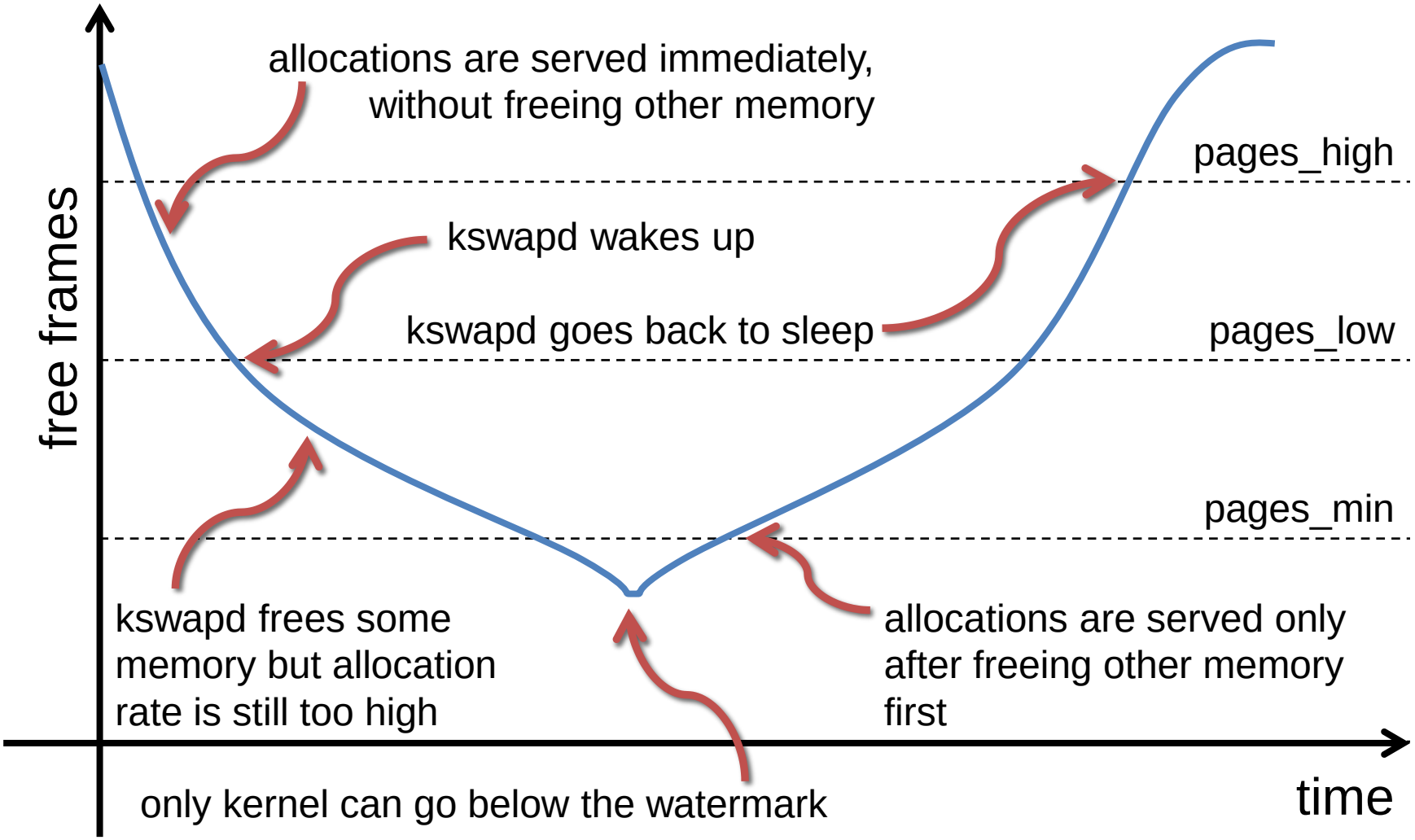
פינוי אסינכרוני

- אם במהלך הקצאת זיכרון הגרעין מגלה שמספר המסגרות הפנויות נמוך מדי, הוא מעיר חוט גרעין מיוחד – kswapd – שתפקידו להריץ את PFRA.
- kswapd רץ ברקע ומפנה זיכרון במקביל לתוכניות אחרות אשר מבקשות זיכרון.
  - יעיל במערכות מרובות מעבדים.

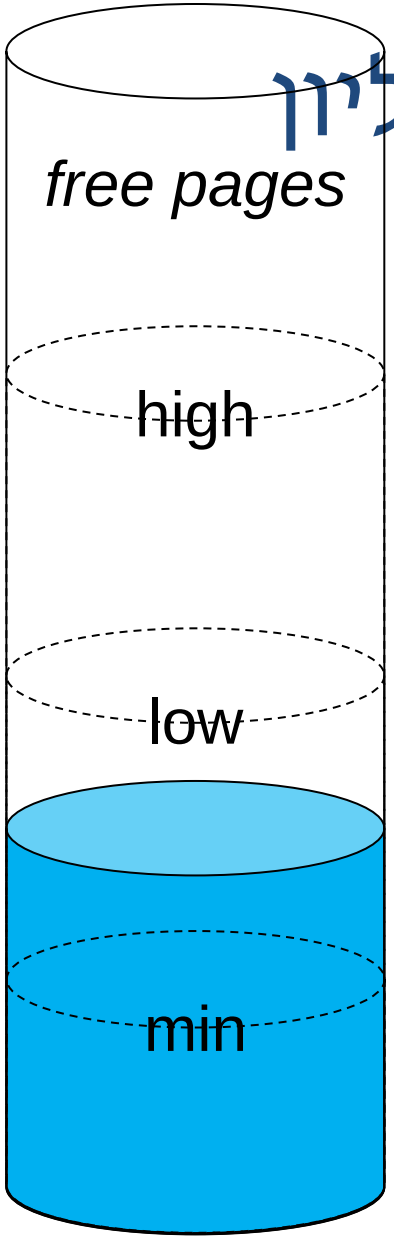
פינוי סינכרוני

- אבל kswapd לא בהכרח מפנה זיכרון מספיק מהר...
- אם במהלך הקצאת זיכרון הגרעין מגלה שמספר המסגרות הפנויות קטן מהסף הקריטי אז PFRA נקרא ישירות.
- במקרה זה התהליך שצריך את הזיכרון נאלץ להמתין.

תרחיש לדוגמה



סף מינימום, סף תחתון, וסף עליון



- מדוע מוגדר סף מינימום קריטי (min)?
  - PFRA דורש זיכרון פנוי כדי לרוץ, למשל כדי לתחזק מבני נתונים של הגרעין. אם מספר המסגרות הפנויות ירד מתחת לסף הקריטי, הגרעין עלול לקרוס.
- מדוע מוגדר סף תחתון (low)?
  - כדי להקטין את הזמן של הקצאה חדשה.
  - הגרעין פינה מראש בצורה אסינכרונית כדי שיהיה זיכרון מעבר לסף הקריטי וכך אין צורך לפנות מסגרות בצורה סינכרונית.
- מדוע מוגדר סף עליון (high)?
  - כדי לא לפנות יותר מדי דפים ולפגוע בביצועים.
  - כדי לא לבצע עבודה מיותרת.

עקרונות כלליים של PFRA

- העקרונות המרכזיים המשותפים לכל המימושים של PFRA:

1. בחינת כל המסגרות של כל התהליכים ועדיפות לפינוי מסגרות של מטמון הדפים שאף תהליך לא מצביע אליהן ו/או מסגרות נקיות.
2. מעקב דינמי אחר רמת הפעילות של כל המסגרות ועדיפות לפינוי המסגרות ה"קרות" ביותר.
   - תהליכי משתמש שיוצאים להמתנה ארוכה יאבדו בהדרגה את כל המסגרות שהם החזיקו ולא יפריעו לשאר התהליכים.
3. פינוי מסגרת משותפת מכל מרחבי הזיכרון המצביעים עליה בבת-אחת.

סיווג מסגרות של PFRA

מסגרות של מטמון הדפים

נקיות לא מוצבעות	מלוכלכות לא מוצבעות	נקיות מוצבעות	מלוכלכות מוצבעות

מסגרות אנונימיות

מוצבעות

- מסגרות לא מוצבעות – מסגרות שאף טבלת דפים (כלומר אף תהליך) לא מצביעה עליהן.
  - מדוע אין מסגרות אנונימיות לא מוצבעות?
  - כי מסגרות אנונימיות נמחקות מיד עם סיום התהליך המצביע עליהן, כלומר מיד כאשר הן לא מוצבעות.

אילו מסגרות עדיף לשחרר?

1. מסגרות לא מוצבעות – כי פינוי מסגרות מוצבעות דורש עדכון טבלאות הדפים של תהליכי המשתמש המצביעים עליהן.
2. מסגרות נקיות – כי פינוי מסגרות מלוכלכות דורש כתיבה חזרה לדיסק.

- האם תמיד עדיף לשחרר מסגרות לא מוצבעות על-פני מסגרות אנונימיות?
- לא. לדוגמה: תכנית אחת שעורכת קוד ותכנית שניה שמהדרת (מקמפלת) אותו.
- אחרי שהתכנית הראשונה מסתיימת ולפני שהתכנית השניה מתחילה, המסגרת של הקובץ לא מוצבעת, אבל כדאי לשמור אותה בזיכרון למרות זאת.

אלגוריתם פינוי מסגרות אופטימלי

- האלגוריתם האופטימלי (== מביא למספר ההחטאות הנמוך ביותר) יפנה את המסגרות שניגש אליהן בעתיד הרחוק ביותר.
  - Bélády's optimal page replacement policy.

- מכיוון שלא ניתן לדעת את העתיד, PFRA מסתכל על העבר ומנסה לחזות ממנו את העתיד:
- דפים שניגשו אליהם לאחרונה – הצפי הוא שייגשו אליהם שוב בעתיד הקרוב.
- דפים שניגשו אליהם לפני זמן רב – הצפי הוא שייגשו אליהם שוב רק בעתיד הרחוק.
- בדומה לאלגוריתם LRU: Least Recently Used.

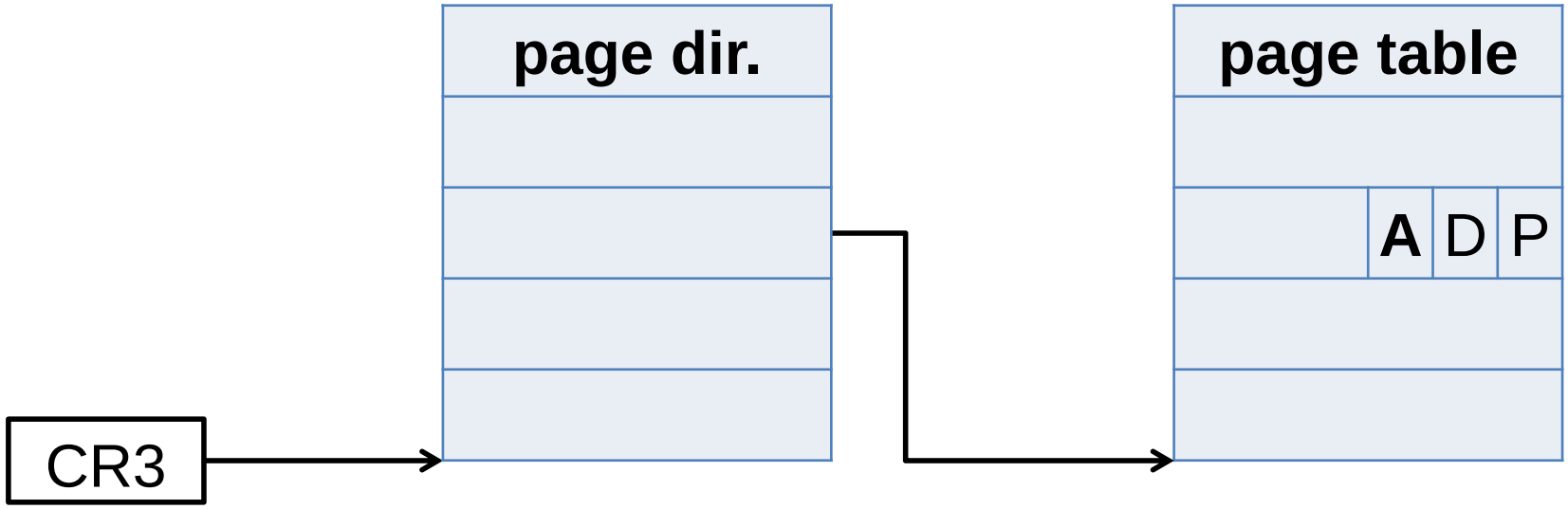
מעקב אחרי גישה לדפים

- כדי לממש את האלגוריתם שתואר בשקף הקודם, מערכת ההפעלה צריכה לדעת על כל גישה לכל דף בזיכרון.
- <u>אופציה #1:</u> המעבד יצור פסיקה בכל גישה לזיכרון כדי להעביר את השליטה למערכת ההפעלה.
  - זה פתרון לא סביר כי המעבד ניגש לזיכרון בכל פקודה לפחות פעם אחת (כדי לקרוא את הפקודה), ולכן התקורה תהיה בלתי נסבלת.
- <u>אופציה #2:</u> המעבד יתחזק בעצמו מונה לכל מסגרת המכיל את ה"גיל" שלה.
  - זה פתרון יקר בחומרה ולכן מעבדי אינטל/AMD לא מספקים תמיכה כזו.
- <u>אופציה #3:</u> מערכת ההפעלה תשתמש במנגנון חומרה אחר כדי להעריך/לקרב את רמת הפעילות של המסגרות בזיכרון: ביט accessed של הכניסות בטבלת הדפים.

ביט accessed בטבלת הדפים

- כאשר המעבד ניגש לדף מסוים לראשונה, הוא מתרגם את הכתובת הוירטואלית שלו לכתובת פיזית ע"י הליכה בטבלת הדפים ואז מדליק את הביט accessed.
  - גישות נוספות לאותו הדף לא ישנו את מצב הדגל accessed.

ומה אם התרגום של הדף נמצא ב-TLB?



רמת פעילות של מסגרת

- שני הביטים active, referenced מגדירים לכל מסגרת בטבלת המסגרות את רמת הפעילות שלה: ערך שלם בין 0 ל-3. למשל:

	refcount	flags	mapping	index
...				
#10		A,R=0,0 ➔ activity = 0		
#11		A,R=0,1 ➔ activity = 1		
#12		A,R=1,0 ➔ activity = 2		
#13		A,R=1,1 ➔ activity = 3		
...				

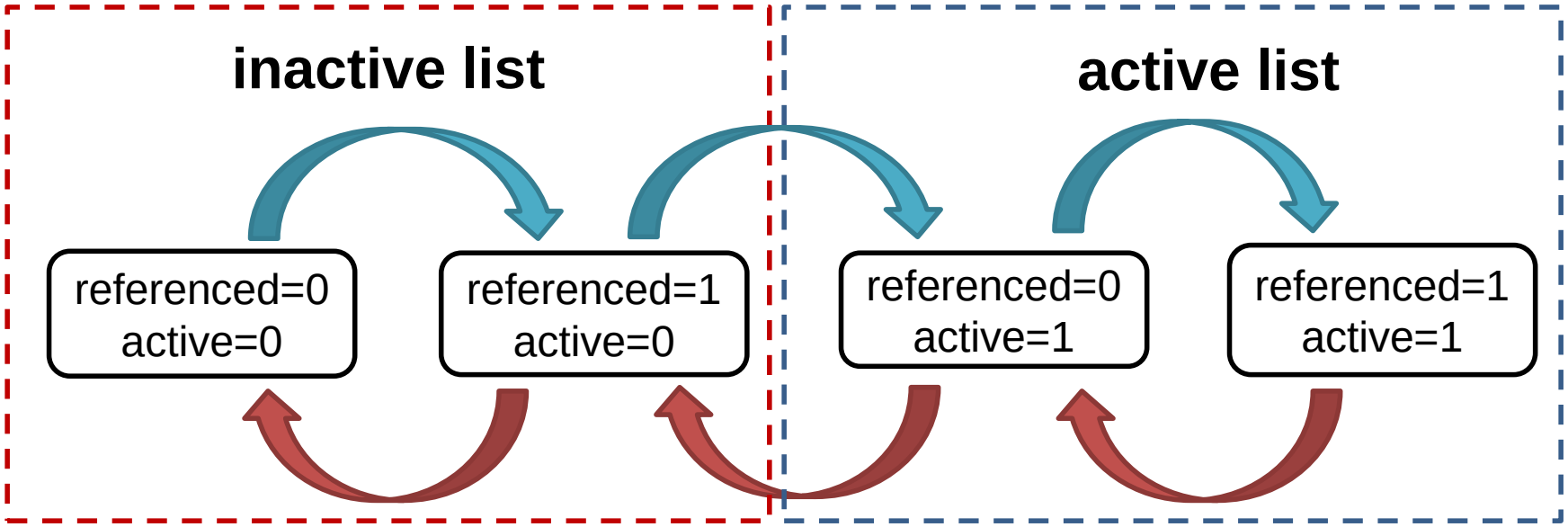
עדכון רמת הפעילות

- בכל פרק זמן מסוים, מערכת ההפעלה סורקת את טבלאות הדפים ובודקת את מצב הדגל accessed בכל הכניסות.
- בכל כניסה שבה הדגל accessed דלוק, מערכת ההפעלה מגדילה את רמת הפעילות של המסגרת (activity++) ומנקה את הדגל accessed.

- בנוסף, בכל פרק זמן מסוים, או כאשר מספר המסגרות הפנויות נמוך, מערכת ההפעלה מקטינה את מוני הפעילות של כל המסגרות (activity--).
- הרציונל: כדי לא להתחשב בהיסטוריה רחוקה מדי. אם מסגרת הייתה פעילה בעבר, זה לא אומר שהיא עדיין פעילה בהווה.

שחרור מסגרות שלא היו בשימוש לאחרונה

- כל מסגרת בזיכרון שייכת לאחת מבין שתי רשימות מקושרות (active, inactive) בהתאם לרמת הפעילות שלה.
- מסגרות יכולות לעבור בצורה דינמית בין הרשימות.
- PFRA סורק את המסגרות ברשימת ה-inactive ומפנה אותן קודם.

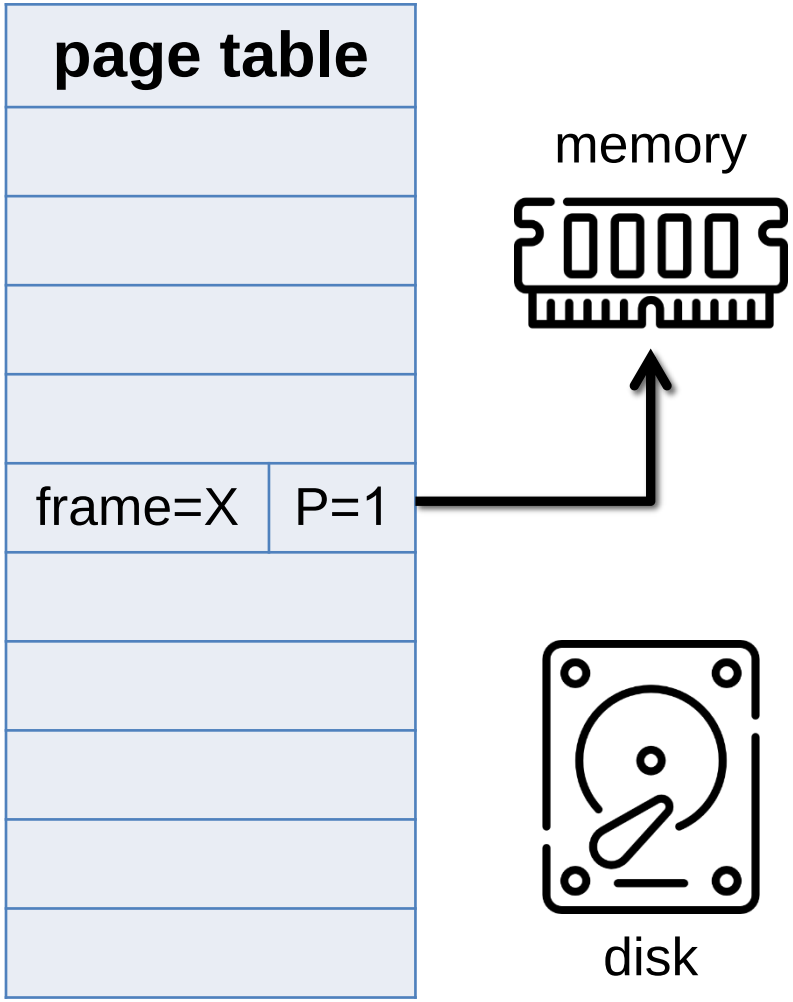


פינוי מסגרות משותפות

- מסגרות משותפות הן מסגרות המוצבעות ממספר מרחבי זיכרון בו-זמנית.
  - דוגמה #1: מסגרת במטמון הדפים אשר מוצבעות ממספר תהליכים שממפים את אותו הקובץ לזיכרון.
  - דוגמה #2: מסגרת אנונימית שמשותפת בגלל מנגנון COW (לפני שבוצע נסיון כתיבה למסגרת).
- כאשר PFRA נדרש לפנות מסגרת משותפת, הוא מנתק אותה מכל מרחבי הזיכרון המצביעים עליה בבת-אחת.
  - כדי לממש את האלגוריתם, לינוקס שומרת מיפוי הפוך (מבנה נתונים שלא נלמד עליו בקורס) מכל מסגרת פיזית אל מרחבי הזיכרון המצביעים עליה.
  - PFRA עובר על כל מרחבי הזיכרון הללו ומעדכן את הכניסות המתאימות בטבלאות הדפים.

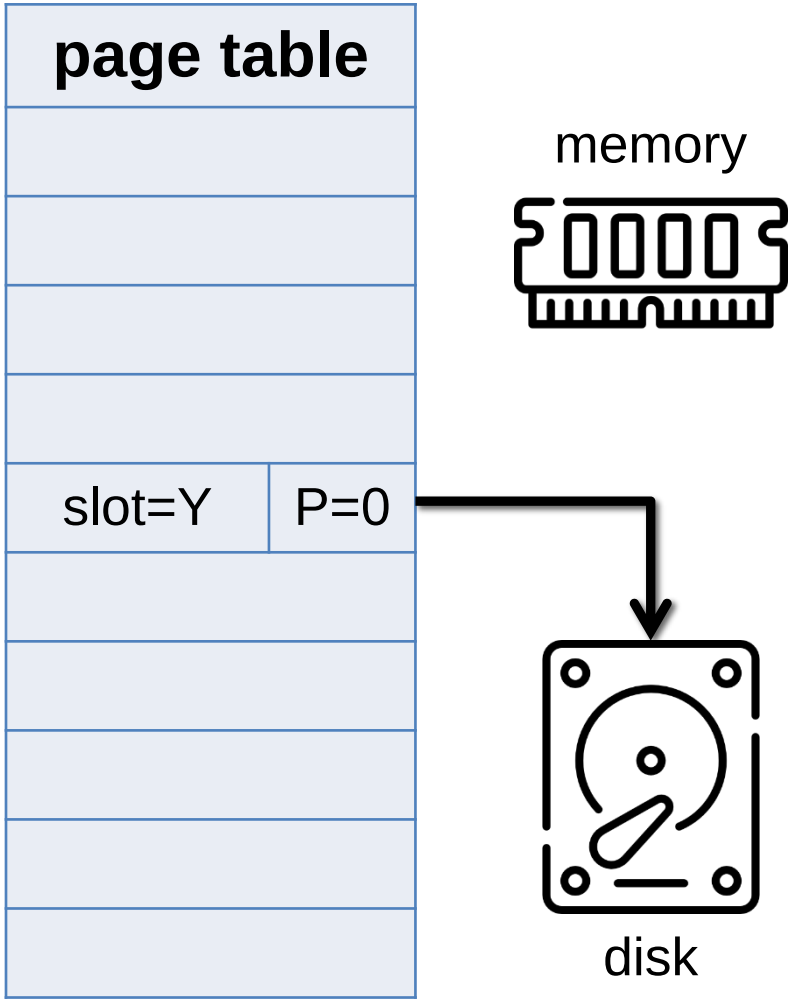
דוגמה מסכמת 1

- נתבונן על מסגרת אנונימית בזיכרון אשר מוצבעת ע"י תהליך אחד.
- הכניסה המתאימה בטבלת הדפים של התהליך מכילה ביט present == 1 וכן את מספר המסגרת בזיכרון הפיזי.
- הכניסה המתאימה ל-X בטבלת המסגרות מכילה refcount == 1.



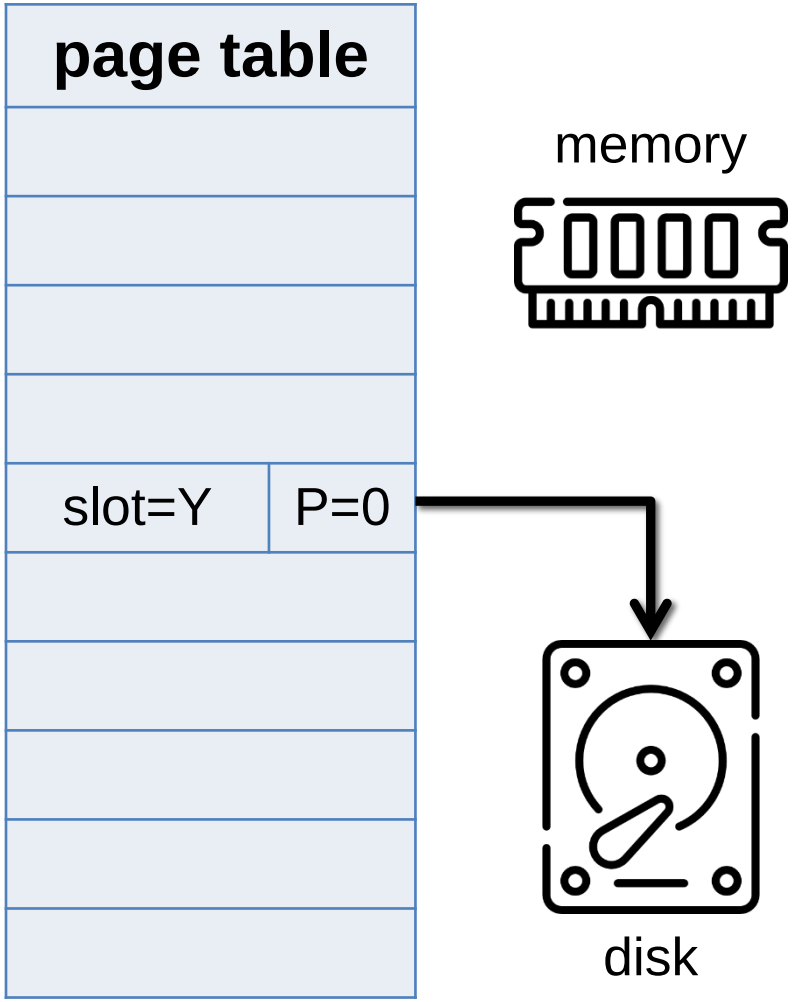
דוגמה מסכמת 2

- כעת נניח ש- PFRA מחליט לפנות את המסגרת לדיסק.
- הגרעין מחפש מגירה פנויה במאגר דפדוף כלשהו ומעתיק אליה את המסגרת.
- הכניסה המתאימה בטבלת הדפים של התהליך מכילה ביט present == 0 וכן את מיקום המסגרת בדיסק.
- הכניסה המתאימה ל-X בטבלת המסגרות מכילה refcount == 0.

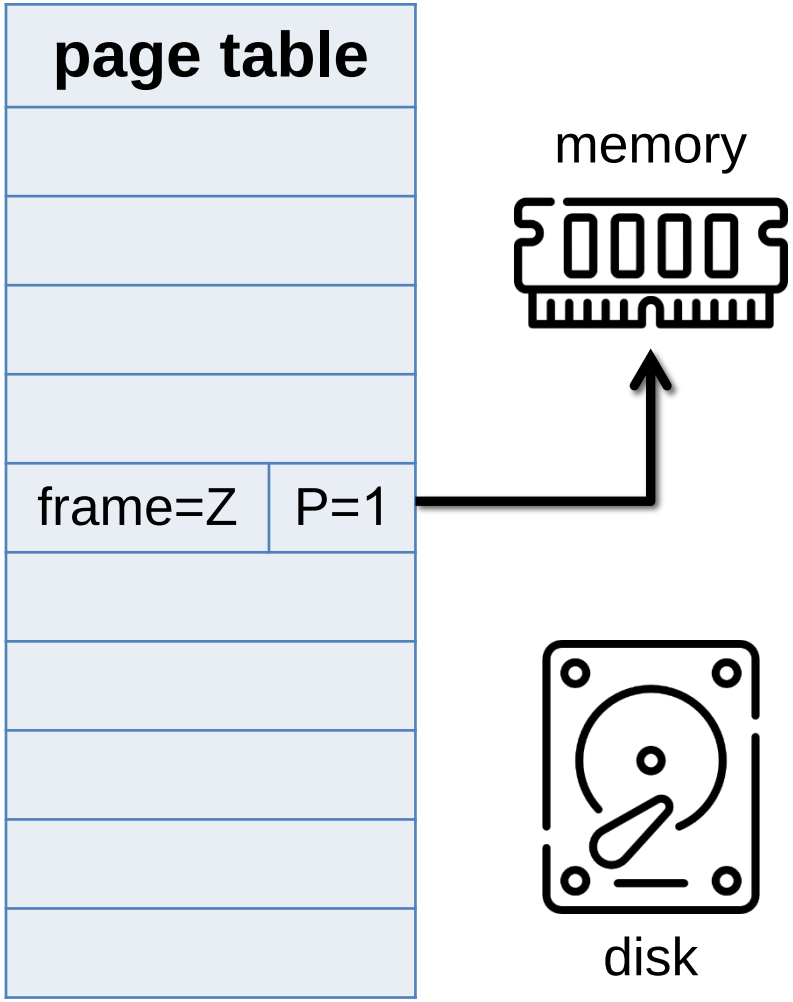


דוגמה מסכמת 3

- אם התהליך ינסה לגשת לדף המתאים, הוא יחטוף חריגת דף.



דוגמה מסכמת 4



- הטיפול יעבור לגרעין, שיקרא את המסגרת חזרה לזיכרון – לא בהכרח למיקום הקודם שלה.
- הכניסה המתאימה בטבלת הדפים של התהליך תצביע אל מיקום המסגרת בזיכרון.
- הכניסה המתאימה ל-Z בטבלת המסגרות מכילה refcount == 1.