- בין השימוש בין **buffering** (מכלאים) ו-**spooling** (מעקפים) ניתן אולי למנות:

1. מכלאים עושים שימוש בטכניקת DMA אבל מעקפים לא.

לא נכון: גם מעקפים משתמשים ב-DMA.

2. מכלאים מאפשרים חפיפה בין העיבוד של תכנית (תהליך) למטלות הק/פ שלה, אבל מעקפים לא.

לא נכון: גם מעקפים יכולים לטפל במטלות ק/פ בחפיפה לעיבוד התוכנית.

3. מעקפים מאפשרים חפיפה בין העיבוד של תכנית למטלות הק/פ של תוכניות אחרות, אבל מכלאים לא.

נכון: מכלאים מאפשרים רק את (2).

4. כמו במכלאים, אפשר להשתמש ברעיון של מעקפים רק בין הדיסק לזיכרון העיקרי.

לא נכון: מעקפים יכולים לעקוף כל טכנולוגית זיכרון איטית עם טכנולוגית זיכרון מהירה יותר.

- אם יש קיפאון אז ברור שאין התקדמות, אבל לא בהכרח ההיפך. ז"א, חוסר התקדמות לא נובע בהכרח ממצב קיפאון. הם אינם הפכים או משלימים אחד של השני.
המתנה חסומה והרעבה הם באמת הפכים אחד של השני כי הרעבה זה מצב של המתנה אינסופית (ז"א: אין חסם להמתנה) שזה בדיוק ההיפך מצב של המתנה חסומה.

- לגבי סמפורים:

1. שתי פעולות wait המופעלות על סמפורים שונים חייבים להיות מנועים הדדית.

לא נכון: אין קשר ביניהם.

2. הערך התחילי של סמפור לא יכול להיות שלילי.

לא נכון: כן יכול להיות. אם הוא מחכה לסדרה של סיגנלים.

3. מכניזם התור של סמפור חייב להיות בשיטת FIFO.

לא נכון.

4. אין טעם לספק פעולות initialize וקריאת ערך של סמפור כפעולות בסיסיות של סמפור.

לא נכון: יש טעם, פעולות חשובות לשימוש מושכל בסמפורים.

5. הערך ההתחלתי של סמפור לא יכול להיות 0.

לא נכון.

6. אחרי אתחול סמפור אין לשנות את ערכו אלא ע"י פעולות wait ו-signal.

נכון.

- שינוי העדכון של **q** ב-**RR**:

שינוי העדכון מאפשר קביעה דינמית של פלח הזמן לכל תהליך לפי סוגו ושיקולי המערכת. זה מאפשר מתן עדיפויות, aging וצמצום מספר מיתוגי ההקשר.

1. הגדלת q בצורה מתונה מטרתה להפלות פחות את התהליכים הוותיקים שקרוב לוודאי הם הארוכים. הם יצטרכו פחות פלח זמן ויוכלו לגמור מהר יותר.
2. בשיטה (1) פלחי הזמן לתהליכים מסוימים יכולים לגדול בלי הגבלה ולעכב מאוד את התהליכים הקצרים, בניגוד למדיניות RR, לכן אם נגביל את ה-Q עד Qmax אזי הוא יהווה חסם עליון כדי שלא ייווצר אפקט שיירה והידרדרות של האלגוריתם ל-FIFO. יש בעיה בקביעת ה-Qmax המתאים.
3. הגדלה הנדסית של פלח הזמן מגדילה את הסיכוי לגמור תהליכים בינוניים. אם ההכפלה אבל לא הספיקה אז כנראה תהליך ארוך שיישאר הרבה זמן במערכת ולכן אפשר להחזיר אותו לפלח הזמן ההתחלתי של q.

- בין הטיעונים לגבי מודל תהליכים ולגבי מודל המשימה פלוס תהליכונים ניתן למנות:

1. פלח הזמן הכדאי בין מיתוגי הקשר תהליכונים יכול להיות קצר יותר מאשר פלח הזמן הכדאי בין מיתוגי הקשר תהליכים.

נכון: מיתוגי הקשר תהליכונים זולים יותר.

2. תקלת דף הקוראת לתהליכון כלשהו במשימה בהכרח משביתה את כל התהליכונים במשימה עד להתגברות תקלת הדף.

לא נכון: לא אם יש ויסות ברמת תהליכונים.

3. במודל המשימה פלוס תהליכונים יש טעם לווסת ברמת המשימות מעבר לוויסות הנדרש ברמת התהליכונים.

נכון: לדוגמה, שלא תהיה בעיה (2).

4. קל יותר לבצע הדמיה (simulation) למודל התהליכים במסגרת מודל המשימה פלוס תהליכונים מאשר להיפך.

נכון: משימה + תהליכון = תהליך. להיפך מסובך יותר לדמות n תהליכונים במשימה ע"י n תהליכים עצמאיים או לדמות הכל בתהליך אחד.

- במערכת זיכרון וירטואלי עם דפדוף יש 4 מסגרות המכילות כעת 4 דפים פיסיים. נתונים לנו הזמן שבו נטען הדף למסגרת, זמן הגישה האחרון לדף, מצב סיבית ההתייחסות R ומצב סיבית הלכלוך D לכל דף. יש צורך כעת לבצע החלפת דף. לפי אלגוריתמי ההחלפה: FIFO, LRU, קירוב LRU ע"י סיבית R בלבד וקירוב LRU ע"י צירוף הסיביות (R, D), ייתכן מצב שבו כל אחד מ-4 אלגוריתמי ההחלפה בוחר בדף אחר כדף קורבן!

-

	SCAN	LOOK	C-SCAN	C-LOOK
כיוון השירות	2 הכיוונים	2 הכיוונים	בכיוון אחד	בכיוון אחד
שירות עד לאן	גליל קיצוני	בקשה קיצוני	גליל קיצוני	בקשה קיצונית
אפלייה נגד בקשות קיצוניות	פחות מ-LOOK יותר מאחרים	הכי מפלה לרעה	הכי פחות מפלה	יותר מ-SCAN פחות מהאחרים
מרחק חיפוש כולל	קצת יותר גרוע מ-LOOK	לרוב הכי קצר	הכי גרוע	די גרוע

- עקרון 0, 1, אינסוף, במקביליות:
0 - אין תהליכים במע' או אין מקביליות (=תהליך אחד).
1 - תהליך אחד או מקביליות מינימלית (= שני תהליכים).
אינסוף - מקביליות מלאה - multiprogramming.
ההשלכות במקביליות הם שצריך להרשות אינסוף תהליכים, אינסוף משאבים ועומק בלתי מוגבל של מדרג התהליכים וכו'.
לגבי סמפורים צריך להרשות אינסוף סמפורים ובכל סמפור יכולים להשתמש אינספור תהליכים.
מבחינת מצבי הסמפורים:
סמפור בינארי: 0 - אף אחד (יותר) לא נכנס.
1 - רק 1 יכול להיכנס.
אינסוף - כל השאר נחסמים על הסמפור.
סמפור כללי: החסם n ← אינסוף, ויכול להיקבע ע"י המשתמש בזמן אתחול הסמפור.

- בעיית היצרן צרכן:

producer:
```
produce item;
wait(space);
wait(mutex);
-- deposit item
signal(mutex);
signal(items);
```

consumer:
```
wait(items);
wait(mutex);
--- get item
signal(mutex);
signal(space);
--- consume item
```

1. הפתרון נכון רק ליצרן אחד ולצרכן אחד ולא למספר יצרנים/צרכנים במקביל.
לא נכון.
2. סכום הסמפורים space ו-items בכל שלב התחלתי של הלולאה הוא N.
נכון.
3. הפעולות על הסמפורים space ו-items מנועים הדדית.
לא נכון.
4. ייתכן מצב שבו יש פריט במאגר אבל הצרכן עדיין ממתין על הסמפור items.
נכון.

- מוצע אלגוריתם ויסות preemptive לפי עדיפות שנקבעת ע"י:
עדיפות = (זמן המתנה + פרץ עיבוד) / פרץ עיבוד
הנוסחה היא בעצם עדיפות (קבוע) = 1 + זמן המתנה / פרץ עיבוד. ז"א: יש התחשבות בזמן שהמתין מאז שהגיע (הזקנה) וגם התחשבות בקוצר של פרץ העיבוד.

	לעומת SJF	לעומת FCFS
יתרונות של עדיפות	אין הרעבה כי יש פה מכניזם של הזקנה - יותר הוגן לוותיקים.	יש עדיפות לקצרים אבל גם התחשבות באחרים לפי זמן ההגעה שלהם. אין ממש אפקט שיירה. זמן סבב טוב יותר.
חסרונות של עדיפות	אין עדיפות מוחלטת לקצרים. קצר אחרי ארוך יכול לסבול קצת. ארוך ייכנס לפני קצר שרק הגיע. זמן סבב גרוע יותר.	יש זמן חישוב של עדיפויות. יש אפליה נגד ארוכים. אלגוריתם מורכב יותר.

- שיקום להגדלת גודל דף במע' זיכרון וירטואלית עם דפדפוף:

1. גודל טבלת הדפים (מס' הכניסות).

נכון: פחות כניסות ← טבלה קטנה יותר.

2. עלות ק/פ לדף.

נכון: באותו מחיר עדיף להביא יותר.

3. פיצול פנימי.

לא נכון: כמעט שאין (רק בדף האחרון) ובכל מקרה זה שיקול להקטנה.

4. פיצול חיצוני.

לא נכון: אין בכלל.

5. עקרון המקומיות.

לא נכון: שיקול להקטנה, חבל להביא קוד שלא נדרש.

- לגבי סיגנלים:

1. system calls של sockets מיושמים ע"י שימוש באותות.

לא נכון.

2. traps ב-unix מיושמים ע"י שימוש בסיגנלים.

נכון: זו אחת ממטרות הסיגנלים.

3. יש הבדל בין איתות SIGSTOP לבין איתות SIGKILL.

נכון: SIGSTOP גורם לתהליך להפסיק לקבל זמן מעבד ואילו SIGKILL מבטל לגמרי את התהליך.

4. אין בכלל קשר בין signaling context של תהליך הילד ושל תהליך ההורה שלו.

לא נכון: הבן יורש את ה-signaling context של ההורה.

- swapping מאפשר להעביר תהליכי משתמש בשלמותם מהזיכרון הפנימי לזיכרון החיצוני (דיסק) כאשר התהליך ממתין לק/פ ואינו יכול להמשיך להתבצע (או כאשר הוא ממתין לאירוע אחר). התהליך יוחזר לזיכרון הפנימי לאחר שהאירוע לו המתין התרחש והתהליך יכול להמשיך להתבצע. בצורה כזו שחלוף מאפשר ניצולת טובה של הזיכרון הפנימי והמעבד מאפשר להריץ יותר תהליכי משתמש ולתמוך ברמה גבוהה יותר של multiprogramming.

- הצורך ב-spooling (מעקפים) הוא כדי להתגבר על האיטיות יחסית של התקני הק/פ לעומת מהירות המעבד ע"י חפיפה בין עיבוד של תוכנית מסוימת לק/פ של תוכניות אחרות. לדוגמה, מעקף קלט של העברת עבודות לביצוע מקורא כרטיסים או סרט לדיסק במאגר עבודות או מעקף פלט של כתיבת תוצאות לדיסק לצורך הדפסה מאוחרת יותר. החפיפה מושגת ע"י התקני ק/פ חכמים (DMA) שעובדים במקביל למעבד וע"י ניצול זמנים מתים של המעבד לקידום מצב ה-spooling. השימוש במעקפים מונע בעיות בשיתוף משאבים כי הוא גורם לסיריליזציה בגישה למשאב מסוים, דבר המונע גם מצבי קיפאון.

- הוראת test & set הינה אחת ההוראות המסופקת ע"י המחשב. ניתן לבצע בעזרתה בדיקה של ערך משתנה ומתן ערך למשתנה זה בפקודה אחת אטומית. לאטומיות חשיבות מרובה לצורך מניעה משני תהליכים המשתמשים באותו משתנה או משאב מלשנות את ערכו בעת הפעולה. החסרון העיקרי של test & set הינו שהוראה זו מתבצעת בלולאה עד שהמשתנה מגיע לערך המאפשר לתת לו את הערך החדש. הלולאה הינה המתנה חסומה הגוזלת זמן מחשב רב וזה חסרונה. לכן נשתמש בהוראה זו רק לצורך פעולות מאוד קצרות, לדוגמה: גישה למשתנה הנועל עדכון מצביע לתורים. כאשר יש לנו פעולות ארוכות יותר ועדיין אנו רוצים אטומיות בגישה ניתן להשתמש בסמפורים. היתרון של סמפורים הוא שמימוש טוב שלהם, למשל, בעזרת תור FIFO ומונה, לא יחייב המתנה חסומה. כאשר המשאב עליו מתחרים מועסק ע"י

תהליך אחד, יוכל תהליך שני להעביר את המעבד לרשות תהליכים אחרים עד שערך הסמפור יגיע לערך הרצוי. הזמן בו מדובר יכול להיות גדול בהרבה.
לכן test & set וסמפורים משלימים האחד את השני כאשר האחד מתאים לפעולות קצרות מאוד והשני לפעולות העשויות לדרוש יותר זמן.
סמפורים מבטיחים התקדמות שכן ברגע שתהליך יוצא שני יכול להיכנס ומניעה הדדית, אולם לא מובטחת המתנה חסומה. אם נוסיף מנגנון לניהול התור (למשל FIFO) נדע שהתהליכים יטופלו לפי סדר הגעתם וההמתנה תהיה חסומה.

- פסיקה חיצונית עשויה לשנות מצב של תהליך ממתין למוכן, לדוגמה: אירוע סיום פעולת ק/פ. אם התהליך הוא בעדיפות גבוהה יותר מהתהליך הנמצא עכשיו בריצה, והתזמון הינו לפי עדיפויות, יתבצע מיתוג הקשר והמעבד יעביר לרשות התהליך בעדיפות הגבוהה כאשר התהליך שהיה עד עכשיו בריצה יעבור לתור המכונים. התקורה או המחיר שמשלמת מע' ההפעלה לצורך מעבר זה נובעת משמירת הנתונים ב-PCB, העבר מתאר התהליך לתור המוכנים, עדכון הזיכרון האסוציאטיבי, חזרה מהפסיקה.

- במע' זיכרון וירטואלי המיושמת ע"י מדרג של כמה מקטעים ורמת דפים, האם הכרחי שטבלאות הדפים לכל מקטע יהיו תמיד בזיכרון?

הרעיון המרכזי מאחורי ניהול זיכרון ע"י מקטעים שמחולקים לדפים הינו שאין צורך להחזיק בכל הדפים השייכים למקטע מסוים בזיכרון. הדפים מובאים לזיכרון בהתאם לביקוש. בצורה כזו אנחנו יכולים לשמור על טבלאות דפים קטנות יותר שכן החלקים שאינם בשימוש לא יהיו בזיכרון. אם נתבקש לגשת לדף של טבלת הדפים שאיננו בזיכרון יהיה page fault ומע' ההפעלה תטפל בהבאתו פנימה באופן אוטומטי.

-

	SSDF	LOOK
יתרונות	מנצל את עקרון המקומיות. לא עושה זיגזגים גלובליים. טוב לעומס בינוני.	עקרון המעלית. פחות זיגזוגים. טוב לשירות דינמי. טוב לעומס כבד.
חסרונות	מפלה לרעה גלילים קיצוניים. סובל מזיגזגים מקומיים. אם נתקע בגליל מסוים, יכול לגרום להרעבה.	מפלה קיצונים, אבל פחות. תנועה בכיוון מסוים. אם נתקע בגליל מסוים, יכול לגרום להרעבה.

- בעיית הפיצול החיצוני גורמת לצורך בכיווץ. יש שיטות שונות להקצאת זיכרון דינמי אך, במוקדם או במאוחר, הזיכרון יהיה שבור. עבודות יסתיימו ויישארו חורים שעלולים לא להספיק לעבודה חדשה. עלול להיווצר מצב שסה"כ יש מספיק מקום, אך אף חור בודד אינו גדול מספיק. אפשר לעשות כיווץ חלקי או מלא, כיווץ סטטי (הכל מפסיק עד שמסתיים) או בצורה דינמית. לפעמים צריך לעשות כיווץ (לפחות חלקי) כדי לפנות מקום מסוימת לתהליך שצריך לשבת בדיוק שם. בעיות:
  1. הכיווץ הוא יקר. משתמשים במשאבי המע' שיכלו להיות מנוצלים לדברים אחרים.
  2. בכיווץ סטטי הכל מתעכב. כיווץ דינמי דורש הרבה תיאומים.
  3. המחיצות חייבות להיות relocatable.

- במע' התומכת ב-demand paging נמדד: ניצולת מעבד 20%, ניצולת דיסק הדפדוף 99.7%, ניצולת התקני ק/פ אחרים 5%. אילו מהשינויים המוצעים ישפרו את ניצול המעבד במע'?

נשים לב שדיסק הדפדוף מהווה את צוואר הבקבוק במערכת וכל דרך להביא לפחות תקלות דף תוריד מהעומס הנוצר על המעבד ותביא לשיפור הניצולת. כיום המערכת במצב של דשדוש אשר נוצר כאשר המערכת עסוקה בדפדוף פנימה והחוצה במקום לשרת תהליכים הזקוקים לזמן עיבוד.
מצב הדשדוש מתרחש כאשר רמת ה-multiprogramming גבוהה.
1. מעבד מהיר יותר.

לא יעזור שכן הבעיה אינה מהירות המעבד (המעבד בניצולת נמוכה).
2. העלאת רמת ה-multiprogramming.

שינוי זה יחמיר את הבעיה שכן יהיו יותר תהליכים בעיבוד ויותר תקלות דף, כלומר: החרפת הדשדוש.
3. הורדת רמת ה-multiprogramming.

יביא לשיפור שכן ניתן יהיה לשמור יותר דפים השייכים ל-working set של כל תהליך בזיכרון והיו פחות תקלות דף ← ניצולת המעבד תעלה.

4. דיסק דפדפוף גדול יותר.
יביא לשיפור מזערי. השיפור צפוי מהקטנת העומס על דיסק הדפדוף.
5. התקני ק/פ אחרים מהירים יותר.
לא יעזור - גם עכשיו הם בקושי עמוסים.
6. הגדלת הזיכרון.
יביא לשיפור שכן נוכל לשמור יותר מדפי ה-working set בזיכרון ויהיו פחות תקלות דף ← ניצולת תשתפר.
7. הגדלת גודל הדף.
לא יעזור שכן עדיין דיסק הדפדוף יהיה עמוס.

- מוצע לממש סמפור בעזרת מונה ותור LIFO:
דרך נוחה למימוש סמפור הינה להצמיד לכל סמפור תור ומונה. כאשר תהליך ביצע wait לא מוצלח (wait על סמפור שערך המונה שלו הוא 0) התהליך ייכנס לתור הסמפור עם מצב waiting. באופן מקבל, signal יגרום שתהליך יועבר מתור הסמפור עם מצב ready.
ניהול התור במשטר "אחרון נכנס ראשון יוצא" הינו בעייתי, אינו הוגן ואינו מבטיח המתנה חסומה. כל זמן שמגיעים תהליכים חדשים, תהליך שכבר ממתין בתור זמן רב ימשיך להמתין (המתנה לא חסומה). תנאי מניעה הדדית והתקדמות מתקיימים.

- סמפור בינרי יכול לקבל ערך 0 או 1. סמפור כללי יכול לקבל ערך שלם כלשהו מ-0 ומעלה. נוח להשתמש בסמפור בינרי למניעה הדדית כאשר 0 משמעותו שהמשאב אינו ברשותי ו-1 שהוא ברשותי. סמפור כללי עם ערך מקסימלי כלשהו יכול לשמש למימוש מחסן ל-n עצמים (יצרן-צרכן), כאשר המחסן יתמלא לא נוכל להוסיף וכאשר יתרוקן (ערכו 0) לא נוכל להמשיך להוציא.
סמפור כללי יכול לשמש כסמפור בינרי אבל לא הפוך.

- קל יותר לתמוך בשיתוף קוד בעזרת מקטעים מאשר בעזרת דפדוף כי מקטעים הינם חלוקה לוגית של התוכנית לפרוצדורות, נתונים וכו', לעומת דפים שהם חלוקה פיזית.

- סמפור כללי אותחל לערך 9. לאחר מכן בוצעו עליו בסדר כלשהו 27 פעולות down ו-23 פעולות up. הערך הנותר של הסמפור הוא: 5 = 23 + 27 - 9.

- ויסות מעבר **preemptive** של עדיפות:
1. מכיוון שזה אלגוריתם עדיפות, התהליך תמיד יכול לרוץ עד שמחליט להשהות את עצמו.
לא נכון.
2. העדיפות של כל תהליך תמיד עולה עם התמשכות הריצה שלו.
לא נכון.
3. לתהליכים עתירי עיבוד עם עדיפות נמוכה יש השפעה על זמן סבב/תגובה של תהליכים עם עדיפות גבוהה.
לא נכון.
4. <u>תהליך בעדיפות נמוכה עלול לסבול מהרעבה.</u>
נכון.

-

	דפדוף	קיטוע
מנגנון	החלוקה היא פיסית, למסגרות בגודל קבוע בזיכרון ודפים באותו גודל על הדיסק. כתובת לוגית מתורגמת דרך טבלת דפים לכתובת פיסית.	החלוקה כאן היא לוגית לקטעים מגודל משתנה. טבלת מקטעים מתרגמת כתובת לוגית לפיסית.
שיתוף והגנה	בעזרת דפדוף ניתן ליישם שיתוף של תוכניות כבדות (כמו מהדרים). טבלת הדפים של כל משתמש תצביע על אותם דפים בזיכרון. ניתן לייחס גם ביטים של הגנה לכל דף ולשמור אותם בטבלת הדפים. בכל פנייה לזיכרון יבדקו גם את ההרשאות המתאימות.	החלוקה לוגית ולכן יותר נוחה מבחינת הצורה של המשתמש הן לצורך שיתוף והן לצורך הגנה.

- סיבות להרעבה:

1. המע' מתעלמת כל הזמן לפחות מתהליך אחד שלא מצליח להיכנס לביצוע. נכון.
2. העדיפות של כל תהליך מועלית בהתאמה למשך הזמן שהוא שוהה במע'.
3. לפחות תהליך אחד מחכה למאורע במע' שכבר לא יקרה.
4. שניים או יותר תהליכים יכולים לגשת לנתונים קריטיים רק בצורה חלופית.

- מיתוג הקשר והחלפת מצב:

1. שיתוף הקשר הוא פעולה מורכבת יותר מאשר החלפת מצב במע'.
2. קריאת מערכת גורמת להחלפת מצב אבל לא בהכרח למיתוג הקשר.
3. אפשר לבצע מיתוג הקשר מבלי לבצע החלפת מצב בשלב כלשהו.
4. פסיקה חיצונית תמיד גורמת למיתוג הקשר.

- ypcat passwd | a.out &

הפקודה ypcat passwd מדפיסה את רשימת כל המשתמשים במע'. התוכנית a.out מיינת שורות לפי סדר א"ב אבל יש בה שגיאה של חלוקה ב-0 שקורית בזמן ריצה. איך יסתיימו 2 התהליכים? a.out מסתיים בהודעת שגיאה (signal) של נקודה צפה ו-ypcat מסתיים עם הודעת שגיאה של broken pipe. ז"א: שניהם מסתיימים בטעות.

- תהליך יתום לא יכול להפוך ל-zombie. ככלל: תהליך לא יכול להיות גם יתום וגם zombie באותה עת. אם מת אביו של תהליך zombie גם תהליך ה-zombie ימות מיד.

- כתיבה לתוך קובץ רגיל היא כתיבה לדיסק בעוד שכתיבה לתוך pipe היא כתיבה לזיכרון הפנימי.
כתיבה וקריאה מתוך pipe מסונכרנת כך שצד קורא מחכה לצד כותב וצד כותב מחכה לצד קורא. קריאה וכתיבה מתוך קובץ אינה מסונכרנת ← עדיף להשתמש ב-pipe עבור תקשורת.

- פסיקות חיצוניות – אירועים אסינכרוניים: נובעות מנפילת מתח, תקלה או סיום פעולת ק/פ.
- פסיקות פנימיות – אירועים סינכרוניים: ע"י המשתמש (system calls), למשל: התחלת פעולת ק/פ.
- אירועים סינכרוניים: מלכודות יזומות ע"י המערכת.

	Best Fit	Next Fit	Worst Fit
יתרונות:	1. מנצל מקום נתון בצורה טובה יותר. 2. משאיר שטחים קטנים לא מנוצלים. 3. סביר שיידרש כיווץ יחסית מאוחר.	1. סריקה מהירה.	
חסרונות:	1. דורש סריקה יקרה של כל המקומות הפנויים (או מיון). 2. קשה לנצל את המקומות הנותרים.	1. ניצול מקום לא תמיד טוב. 2. בממוצע משאיר חורים בינוניים. 3. סביר שיידרש כיווץ מוקדם יותר מאשר best fit.	1. דורש סריקה יקרה (או מיון). 2. ניצול מקום גרוע. 3. סביר שיידרש כיווץ מוקדם בהרבה מ-next fit ומב-best fit.