

## הקדמה

2

## ניהול תנועות

### Transaction Management

1

### דוגמה: תוכנית להזמנת מקום בטיסה

- נניח שיש במסד הנתונים פריט A, ששומר את מספר המקומות הפנויים
- התוכנית קוראת את הערך של A ממסד הנתונים
- ▶ אם הערך גדול מאפס, התוכנית מקטינה את הערך של A באחד ומודיעה שההזמנה התבצעה
- ▶ אם הערך של A שווה לאפס, התוכנית מודיעה שאי אפשר לבצע הזמנה

4

### תוכנית הפועלת על מסד נתונים

- תוכנית (אפליקציה):
- ▶ קוראת נתונים ממסד הנתונים
- ▶ מבצעת חישובים ביזרון הפנימי המוקצה לה
- ▶ כותבת נתונים חדשים למסד הנתונים
- תוכנית נכתבת כך שתהיה נכונה בהנחה ש-
- ▶ התוכנית מסיימת את פעולתה בצורה תקינה (כלומר, בלי לעוף באמצע)
- ▶ התוכנית פועלת לבדה

3

### זוהי סידרת הפעולות המתבצעות כשאין מקום פנוי

```
• Read A (from the DB)
• Is A>0 ?
• Print "No available seat"
```

• כשאין מקום פנוי (קרי, ערכו של A הוא אפס), אז אין צורך לשנות את הערך של A במסד

6

### דוגמה לסידרת הפעולות של תוכנית המבצעת הזמנה

```
• Read A (from the DB)
• Is A>0 ?
• A := A - 1
• Write A (into the DB)
• Print "Reservation made"
```

• בצד שמאל מופיעות הפעולות של התוכנית שהתבצעו למעשה

• מכיוון שתוצאת הבדיקה הייתה ש-A גדול מאפס, התבצעו הפעולות של הקטנת ערכו של A והדפסת הודעה על ביצוע הזמנה

5

### ההפשטה (אבסטרקציה) של התוכנית כוללת רק פעולות מול מסד הנתונים

Read A  
Write A

זוהי ההפשטה כשמתבצעת הזמנה

Read A

זוהי ההפשטה כשאין מקום פנוי

8

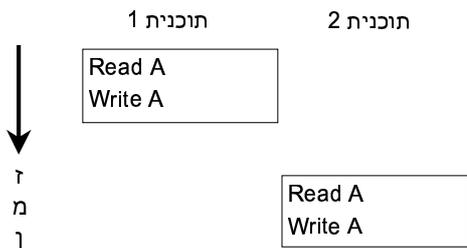
### פעולות מול מסד הנתונים לעומת פעולות בזיכרון הפנימי של התוכנית

Read A  
Is A > 0 ?  
A := A - 1  
Write A  
Print "Reservation made"

פעולות ה-Read וה-Write מתבצעות מול המסד שאר הפעולות מתבצעות בזיכרון הפרטי של התוכנית

7

### ביצוע סידרתי של שתי תוכניות להזמנת מקום



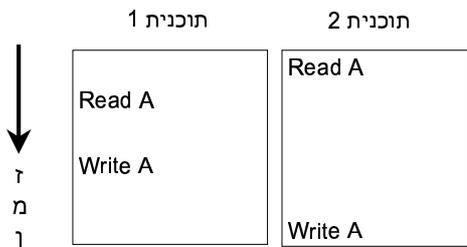
10

### ביצוע סידרתי לעומת ביצוע מקבילי

למעשה יש תוכניות רבות שרצות בו זמנית מאתרים שונים ומנסות לבצע הזמנות ביצוע סדרתי פירושו שהתוכניות רצות אחת אחרי השניה – כל פעם מתבצעת תוכנית אחת בלבד מתחילתה ועד סופה ביצוע מקבילי פירושו שהתוכניות משולבות זו בזו, כלומר בכל נקודת זמן מתבצעת פעולה בודדת של אחת התוכניות

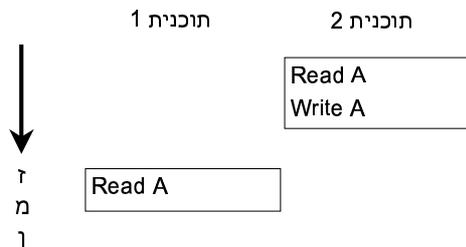
9

### ביצוע מקבילי של שתי תוכניות להזמנת מקום



12

### אפשרות נוספת של ביצוע סידרתי של שתי תוכניות להזמנת מקום



11

## יתרונות של ביצוע מקבילי

❁ סיבות לביצוע מספר תוכניות במקביל:

▶ אין סיבה להתעכב בגלל שתוכנית אחת מתעכבת (למשל, תוכנית מחכה לקלט מהשתמש)

▶ ניצול מקסימלי של משאבי המחשוב, במטרה לסיים את ביצוע התוכניות מהר ככל האפשר

← משיכת כסף מבנקומט - מי רוצה לחכות?

14

## ביצוע של מספר תוכניות במקביל

13

## דוגמה: העברת כסף משני חשבונות לחשבון שלישי

❁ חשבונות בנק A, B, C

❁ נעביר:

▶ 100 ש"ח מ-A ל-B

▶ 100 ש"ח מ-B ל-C

❁ מדובר למעשה בשתי תוכניות שמתבצעות במקביל (ראה דוגמה של ביצוע בשקף הבא) ▶ הביצוע בדוגמה אינו נכון, כי בסיום מוסיפים 100 ש"ח ל-B רק פעם אחת

למען הבהירות, הדוגמה בשקף הבא אינה הפשטה, אלא כוללת את כל הפעולות המתבצעות

16

## הבעיה שמעורר ביצוע מקבילי

❁ תוכנית המתבצעת ע"י מערכת מסד נתונים אינה מודעת לכך שהיא מתבצעת במקביל לתוכניות אחרות

▶ התוכנית נכתבה בהנחה שהיא מתבצעת לבדה

❁ מערכת מסד הנתונים חייבת לוודא שהביצוע המקבילי אינו גורם לשגיאות

15

## האם כל ביצוע מקבילי שגוי?

❁ כמובן שלא

❁ השקף הבא מראה ביצוע מקבילי של שתי התוכניות שרושם למסד את הנתונים הנכונים

18

ערכים התחלתיים במסד: A=900, B=500, C=100

Read(A)	A=900	
A:=A-100	A=800	C=100
Write(A)	<u>A=800</u>	C=0
Read(C)	C:=C-100	C=0
Write(A)		<u>C=0</u>
Read(B)	B=500	
Write(C)	B=600	
B:=B+100		B=500
Read(B)		B=600
B:=B+100		<u>B=600</u>
Write(B)		<u>B=600</u>
Write(B)	<u>B=600</u>	

17

ערכים מודגשים בקו נכתבים למסד

## מתי ביצוע מקבילי הוא נכון?

- אם הוא שקול לביצוע סדרתי
- שימו לב שמספיק שהביצוע המקבילי יהיה שקול לביצוע סדרתי אחד
- בדוגמה שנתנו, לכל הביצועים הסדרתיים אותה תוצאה, לכן
- אם ביצוע מקבילי שקול לביצוע סדרתי אחד, אז הוא שקול לכל הביצועים הסדרתיים
- אבל לא תמיד הדבר כך (דוגמה בשקף הבא)

20

ערכים התחלתיים במסד: A=900, B=500, C=100

Read(A)	A=900	
A:=A-100	A=800	C=100
Write(A)	A=800	C=0
Read(C)	B=500	C=0
C:=C-100	B=600	C=0
Write(C)	B=600	C=0
Read(B)	B=600	B=600
B:=B+100	B=600	B=700
Write(B)	B=600	B=700
Read(B)		B=600
B:=B+100		B=700
Write(B)		B=700

19

ערכים מודגשים בקו נכתבים למסד

## הגדרת המודל והמושגים הפורמליים

22

## נחזור לדוגמה של הזמנת מקום

- כשנשאר רק מקום אחד ורצות שתי תוכניות שמנסות לבצע הזמנה, אז התוכנית שרצה ראשונה תזכה במקום האחרון שנוותר
- מבחינתנו אין זה משנה איזה תוכנית רצה ראשונה – שתי האפשרויות נכונות כל עוד שתי התוכניות רצות באופן סדרתי
- כאמור, ביצוע מקבילי הנו נכון אם הוא שקול לאחד הביצועים הסדרתיים

21

צד ימין הוא ההפשטה (אבסטרקציה) של הביצוע

Read(A)	Read(A)
A:=A-100	Read(C)
Write(A)	Write(A)
Read(C)	Read(B)
C:=C-100	Write(C)
Write(C)	Read(B)
Read(B)	Write(B)
B:=B+100	Read(B)
Read(B)	Write(B)
B:=B+100	Write(B)
Write(B)	Write(B)

24

## המודל

- כיצד נבדוק האם ביצוע מקבילי הנו נכון?
  - החשובה עשויה להיות תלויה בחישוב המדויק שכל תוכנית מבצעת
  - מסובך מדי (למעשה בלתי אפשרי) לנתח את החישוב המדויק שמבצעת כל תוכנית
  - לכן, מתייחסים רק להפשטה של התוכניות, כלומר מתייחסים רק לדברים הבאים:
    - פעולות הקריאה מהמסד והכתיבה לתוך המסד
    - הסדר, על ציר הזמן, שבו מתבצעות פעולות אלה

23

## תזמון Schedule

- תזמון הוא ביצוע של מספר תנועות במקביל בכל נקודת זמן מתבצעת פעולה אחת בלבד
- פעולת קריאה/כתיבה מתבצעת בצורה אטומית
- התזמון מייצג את הסדר על ציר הזמן (הציר האנכי) בו מתבצעות הפעולות

Read(A)	
	Read(C)
Write(A)	
Read(B)	
	Write(C)
	Read(B)
	Write(B)
Write(B)	

26

## תנועה Transaction

- תנועה (טרנסקציה) היא סדרה של פעולות קריאה וכתיבה שמתבצעות על פריטים (items) מהמסד
- תנועה היא הפשטה של תוכנית
- הפריטים יכולים להיות רשומות, שדות של רשומות, בלוקים וכד'

Read(A)
Write(A)
Read(B)
Write(B)

25

צד ימין הוא התזמון המופשט עבור הביצוע בצד שמאל

Read(A) A:=A-100	Read(C)	Read(A)	
			Read(C)
	Read(C)	Write(A)	
התזמון המקבילי המופשט של שתי התנועות צריך להיות נכון לכל חישוב אפשרי, שמתבצע עבור כ"א מהפריטים בין הקריאה לכתיבה של הפריט		Read(B)	Write(C)
			Read(B)
	Write(B)		Write(B)
Write(B)		Write(B)	

צד ימין הוא התזמון המופשט עבור הביצוע בצד שמאל

Read(A) A:=A-100	Read(C)	Read(A)	
			Read(C)
	Read(C)	Write(A)	
Write(A)			Write(C)
	C:=C-100	Read(B)	
			Write(C)
	Write(C)		Read(B)
Read(B)			Read(B)
	B:=B+100	Read(B)	
			Write(B)
	Write(B)	Write(B)	
Write(B)		Write(B)	

## שקילות של תזמונים

30

## תזמון סדרתי Serial Schedule

- תזמון סדרתי הוא תזמון שבו התנועות מתבצעות זו אחר זו באופן סדרתי
- כל תזמון סדרתי הוא נכון

Read(A)	
Write(A)	
Read(B)	
Write(B)	
	Read(C)
	Write(C)
	Read(B)
	Write(B)

29

## תזמון בר-סדרתיות Serializable Schedule

- תזמון הנו בר-סדרתיות אם הוא שקול לתזמון סדרתי כלשהו
- תזמון בר-סדרתיות הוא תזמון נכון
- צריך דרך לקבוע שתזמון הוא בר-סדרתיות מבלי לדעת מהם החישובים שמתבצעים
- כלומר, לכל חישוב אפשרי, התזמון צריך להיות שקול לתזמון סדרתי

32

## תזמונים שקולים Equivalent Schedules

- שני תזמונים הנם שקולים אם
  - הם מורכבים מאותן תנועות
  - לשני התזמונים אותה השפעה על המסד, כלומר למסד יש אותם ערכים עם סיומו של כ"א מהתזמונים
  - שני התזמונים מייצרים אותו פלט למשתמש, כלומר מציגים בפני המשתמש אותם ערכים לכל פריט שהם קוראים
- בהמשך ניתן הגדרה יותר פורמלית

31

## תזמון שאינו בר-סדרתיות

- אין תזמון סדרתי שקול
- שני התנועות קוראות עבור B את הערך המקורי שנמצא במסד לפני שהן מתחילות לפעול – דבר בלתי אפשרי בתזמון סדרתי

Read(A)	Read(C)
Write(A)	
Read(B)	Write(C)
	Read(B)
Write(B)	Write(B)

34

## תזמון בר-סדרתיות והתזמון הסדרתי השקול לו

Read(A)	Read(C)	Read(A)	
Write(A)		Write(A)	
Read(B)		Read(B)	
	Write(C)	Write(B)	Read(C)
Write(B)			Write(C)
	Read(B)		Read(B)
	Write(B)		Write(B)

33

## בדוגמה מהשקף הקודם

- התזמון המקורי שקול לתזמון הסדרתי למרות
  - שבתזמון המקורי  $T_2$  קוראת וכותבת את C לפני ש- $T_1$  קוראת וכותבת את A
  - ובתזמון הסדרתי  $T_2$  קוראת וכותבת את C אחרי ש- $T_1$  קוראת וכותבת את A
- אבל בשני התזמונים
  - $T_1$  קוראת וכותבת את B לפני ש- $T_2$  קוראת וכותבת את B

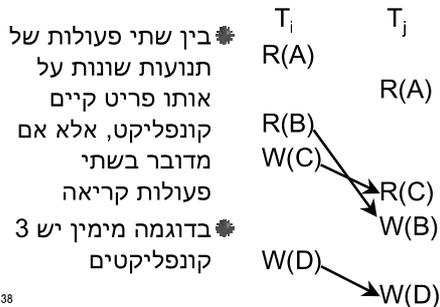
36

## עוד תזמון בר-סדרתיות והתזמון הסדרתי השקול לו

$T_1$	$T_2$	$T_1$	$T_2$
	Read(C)	Read(A)	
	Write(C)	Write(A)	
Read(A)		Read(B)	
Write(A)		Write(B)	
Read(B)			Read(C)
Write(B)			Write(C)
	Read(B)		Read(B)
	Write(B)		Write(B)

35

## קונפליקטים בין תנועות

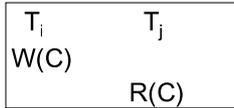


38

## קונפליקטים בין תנועות

37

## קונפליקט בין כתיבה לקריאה



\* אם  $T_j$  קוראת ערך של פריט  $C$ , שנכתב ע"י  $T_i$ , אז

- ▶ בכל תזמון שקול חייב להתקיים ש-  $T_i$  מבצעת  $W(C)$  לפני ש-  $T_j$  מבצעת  $R(C)$

הסבר: תמיד אפשר למצוא חישוב של  $T_i$  שנותן ל-  $C$  ערך ייחודי, שאינו יכול להינתן ע"י אף תנועה אחרת

40

## הרעיון הבסיסי

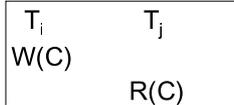
\* קונפליקט קובע סדר בין שתי פעולות

\* הסדר הזה חייב להישמר בכל תזמון שקול

- ▶ "חייב להישמר" זה לא לגמרי מדויק
- ▶ ליתר דיוק, הסדר הזה חייב להישמר אם אין כתיבות עיוורות (blind writes)
- ← הנושא של כתיבות עיוורות יוסבר מאוחר יותר
- ← עקרונית, כתיבה עיוורת פירושה כתיבה שלא מתחשבת בערך הקודם, למשל כותבים אפס ללא קשר מה היה הערך הקודם של הפריט

39

## המסקנה מקונפליקט בין כתיבה לקריאה



\* אם  $T_j$  קוראת ערך של פריט  $C$ , שנכתב ע"י  $T_i$ , אז

- ▶ בכל תזמון שקול חייב להתקיים ש-  $T_i$  מבצעת  $W(C)$  לפני ש-  $T_j$  מבצעת  $R(C)$
- \* מסקנה: בכל תזמון סדרתי שקול  $T_i$  מופיעה לפני  $T_j$

42

## הסבר יותר מפורט

\* בכל תזמון שקול פעולת ה-  $R(C)$  של  $T_j$  חייבת לקרוא אותו ערך כמו בתזמון המקורי

\* לכן, פעולה זו חייבת להתבצע אחרי פעולת ה-  $W(C)$  של  $T_i$

- ▶ אחרת  $T_j$  לא תקרא את הערך הנכון, כי רק  $T_i$  יודעת לחשב אותו (כלומר, תמיד אפשר למצוא חישוב ספציפי שרק  $T_i$  ואף לא עסקה אחרת יודעת לבצע)

41

$T_i$	$T_j$
$R(B)$	$W(B)$

**קונפליקט בין קריאה לכתיבה**

- אם  $T_i$  מבצעת  $R(B)$  לפני ש- $T_j$  מבצעת  $W(B)$ , אז הערך ש- $T_i$  קוראת אינו תלוי בערך ש- $T_j$  כותבת
- בכל תזמון שקול חייב להתקיים ש- $T_i$  מבצעת  $R(B)$  לפני ש- $T_j$  מבצעת  $W(B)$  (בהנחה שאין כתיבות עיוורות)
- מסקנה: בכל תזמון סדרתי שקול  $T_i$  מופיעה לפני  $T_j$  (אם אין כתיבות עיוורות)

44

$T_i$	$T_j$
$W(D)$	$W(D)$

**קונפליקט בין כתיבה לכתיבה**

- נניח ש- $T_i$  מבצעת  $W(D)$  לפני ש- $T_j$  מבצעת  $W(D)$
- האם  $T_i$  חייבת להופיע לפני  $T_j$  בכל תזמון שקול?
- רק אם הכתיבה של  $T_j$  אינה עיוורת, כלומר
- הערך ש- $T_j$  כותבת תלוי בערך ש- $T_i$  כתבה קודם לכן
- מסקנה: בכל תזמון סדרתי שקול  $T_i$  מופיעה לפני  $T_j$  אם הכתיבה אינה עיוורת

43

**שקילות קונפליקטים**

46

**קונפליקטים בין תנועות (דוגמה קודמת)**

$T_i$	$T_j$
$R(A)$	$R(A)$
$R(B)$	$R(C)$
$W(C)$	$W(B)$
$W(D)$	$W(D)$

- בדוגמה מימין יש 3 קונפליקטים, ולפי כל אחד מהם  $T_i$  חייבת להופיע לפני  $T_j$  בכל תזמון שקול
- לכן, התזמון הסדרתי שבו  $T_i$  מתבצעת ראשונה ו- $T_j$  שניה הנו שקול לתזמון הנתון (אם אין תנועות נוספות)

45

**תזמון בר-סדרתיות קונפליקטית  
Conflict Serializable Schedule**

- תזמון  $S_1$  הנו בר-סדרתיות קונפליקטית
- אם קיים תזמון סדרתי  $S_2$  כך ששני התזמונים הנם שקולי קונפליקטים

48

**הגדרה של שקילות קונפליקטים  
Conflict Equivalence**

- שני תזמונים הנם שקולי קונפליקטים אם
- הם מורכבים מאותן תנועות
- כל צמד של פעולות שיש ביניהן קונפליקט מופיע בשני התזמונים באותו סדר
- הערה: הקונפליקטים הנם
- קריאה ולאחריה כתיבה על אותו פריט
- כתיבה ולאחריה קריאה על אותו פריט
- כתיבה ולאחריה כתיבה נוספת על אותו פריט

47

## הוכחת המשפט

- אם אין מעגל, נבצע מיון טופולוגי
- תזמון סדרתי, שבו העסקאות מבוצעות בזו אחר זו לפי הסדר של המיון הטופולוגי, שקול (לפי ההגדרה של שקילות קונפליקטים) לתזמון המקורי
- כי הסדר בין כל הקונפליקטים נשמר בשני התזמונים
- כדי להוכיח את הכיוון ההפוך נניח שיש מעגל

50

## גרף הקדימויות Precedence Graph

- גרף הקדימויות עבור תזמון נתון מכיל צומת לכל תנועה
- צלע מכוונת מ- $T_i$  ל- $T_j$  אם יש קונפליקט בין  $T_i$  לבין  $T_j$  שקובע ש- $T_i$  צריכה להופיע לפני  $T_j$
- משפט: תזמון הנו בר-סדרתיות קונפליקטית אם ורק אם גרף הקדימויות שלו הנו חסר מעגלים

49

לפיכך, אם יש מעגל  
 $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4 \rightarrow T_1$   
 אז נובע ש-

- $T_1$  חייבת להופיע לפני  $T_1$  בכל תזמון סדרתי שהוא שקול קונפליקטים לתזמון המקורי
- מכיוון שהדבר בלתי אפשרי, נובע שאין תזמון סדרתי שהוא שקול קונפליקטים לתזמון המקורי
- מ.ש.ל.

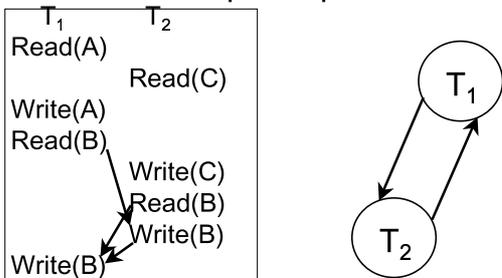
52

אם יש מסלול  $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4$   
 אז נובע ש-

- $T_1$  חייבת להופיע לפני  $T_2$  בכל תזמון סדרתי שהוא שקול קונפליקטים לתזמון המקורי
- $T_2$  חייבת להופיע לפני  $T_3$  בכל תזמון סדרתי שהוא שקול קונפליקטים לתזמון המקורי
- $T_3$  חייבת להופיע לפני  $T_4$  בכל תזמון סדרתי שהוא שקול קונפליקטים לתזמון המקורי
- ובאופן טרנזיטיבי:
- $T_1$  חייבת להופיע לפני  $T_4$  בכל תזמון סדרתי שהוא שקול קונפליקטים לתזמון המקורי

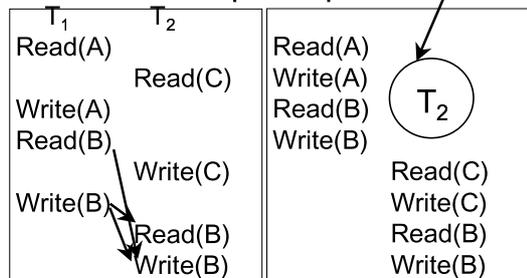
51

דוגמה לתזמון שאינו בר-סדרתיות  
 קונפליקטית



54

דוגמה לתזמון בר-סדרתיות  
 קונפליקטית



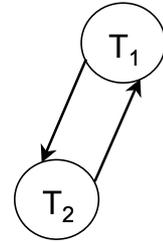
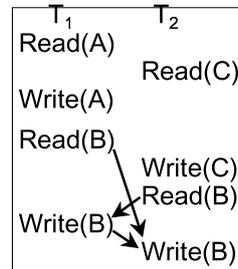
53

## התאוששות מפילות

זוהי רק הקדמה קצרה ואינטואיטיבית של המושגים הבסיסיים

56

## עוד דוגמה לתזמון שאינו בר-סדרתיות קונפליקטית



55

## ביצוע אטומי של תוכנית

- תוכנית הפועלת על מסד נתונים צריכה להתבצע באופן אטומי, כלומר
  - ▶ התוכנית מתבצעת בשלמותה, או שאינה מתבצעת כלל
- תוכנית מתחייבת (מבצעת *commit*) לאחר שביצעה את כל פעולותיה
  - ▶ אם התוכנית התחייבה, אז כל פעולותיה נשמרות
  - ▶ אחרת, שום פעולה שביצעה לא נשמרת במסד

58

## דוגמה: העברת כסף מחשבון לחשבון

- חשבוניות בנק A, B
- נעביר 100 ש"ח מ-A ל-B
- התוכנית:  $A:=A-100$   
 $B:=B+100$
- התוכנית נכונה, אבל מה קורה אם המחשב נופל אחרי ביצוע הפקודה הראשונה?
  - ▶ הורדנו כסף מ-A ולא הוספנו ל-B
  - ▶ התוכנית הסתיימה בצורה שגויה

57

## מנגנון התאוששות (Recovery)

- לכל מערכת מסד נתונים מנגנון התאוששות שאחראי על הדברים הבאים:
  - ▶ לבטל את כל הפעולות של התוכנית אם היא נפלה (*abort*) לפני שביצעה *commit*
  - ▶ לוודא שכל הפעולות של התוכנית נשמרות במסד אם התוכנית ביצעה *commit*
- מאוחר יותר נסביר איך פועל מנגנון ההתאוששות

59