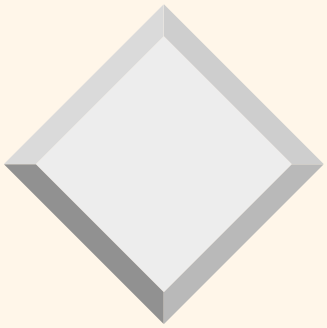


日誌與復原(回復)
Logging and Recovery

複習 ACID 性質

- 單元性(Atomicity)：DBMS應保證系統中每一個交易都具備單元性(an atomic unit of processing)，即必須保證每一交易必須完全執行完畢或完全未執行
- 一致性(Consistency)： DBMS應保證系統在執行交易前與交易後都在一致性的狀態下(不同的一致性的狀態)
- 隔離性(Isolation)： DBMS應保證系統所有交易在執行中，和其他交易是隔離的；即交易執行中應不被其他交易所干擾；即在執行過程中，變數值彼此獨立互不影響
- 永久性(Durability)： DBMS應保證系統所有交易一但確認(交付)(committed)更新後，此更新值一定會對資料庫更新，即不可因系統損壞等原因而未執行此確認(交付)的更新

- 復原(回復)管理程式(**Recovery Manager**)保證系統交易的單元性及永久性

各種交易狀態及運算

- 復原(回復)管理程式(recovery manager)必須記錄下列運算 (部分是日誌內資料)
  - BEGIN_TRANSACTION
  - READ 或 WRITE (Update)
  - COMMIT_TRANSACTION
  - ROLLBACK (或 ABORT)
  - END_TRANSACTION
  - 復原(回復)更新(Undoing an update)

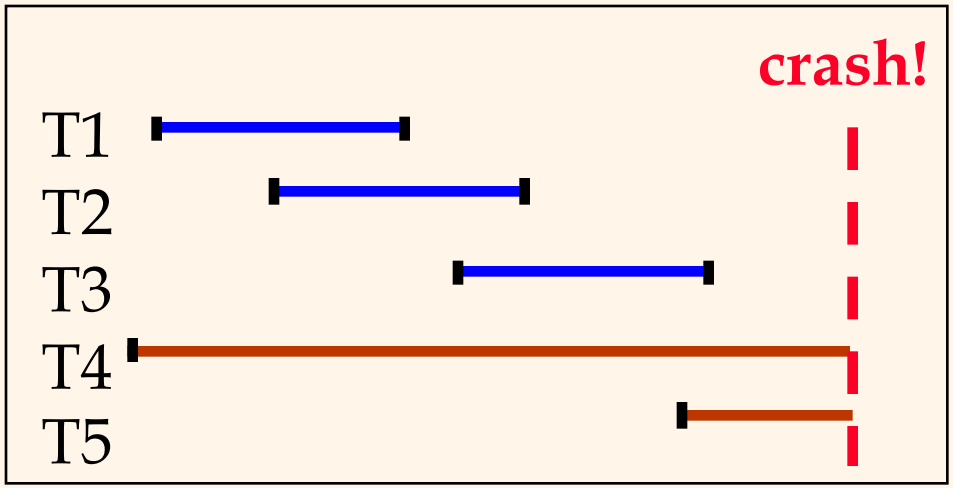
復原(回復)的概念

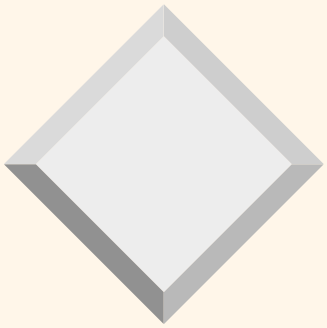
- 交易失敗復原(回復)通常表示將資料庫恢復到交易失敗前最新的資料狀態，要能達到此境界，系統必須保留所有交易過程中異動的資料，這些資料通常存在系統交易日誌中(***system log***)
- 一般的復原(回復)策略
  - 若發生影響整個資料庫系統的大災難(***extensive damage***)，如磁碟毀損(disk crash，***hard failure***)，復原(回復)的方法通常是將最近完整備份的資料庫系統重灌(restores)回來，再重新執行(***redoing***)記錄在日誌中已確認(交付)的交易
  - 若發生僅影響部份交易的小災難時 (soft failure)，復原(回復)方法通常是必須將某些未完成但對系統產生更新的交易，倒回執行(***undoing***)，有時也需重作(***redo***)某些交易，以保持資料一致性

動機

- 如何保持單元性(Atomicity)？
  - 某些交易被中斷 (“Rollback”)
- 如何保持永久性(Durability)？
  - 若DBMS系統當機？

- 當系統恢復時，期望下列交易
  - T1, T2 & T3 應具永久性
  - T4 & T5 應該中斷，不對資料庫產生影響





非大災難的復原(回復)技術
Techniques for recovery from non-catastrophic Xact failure

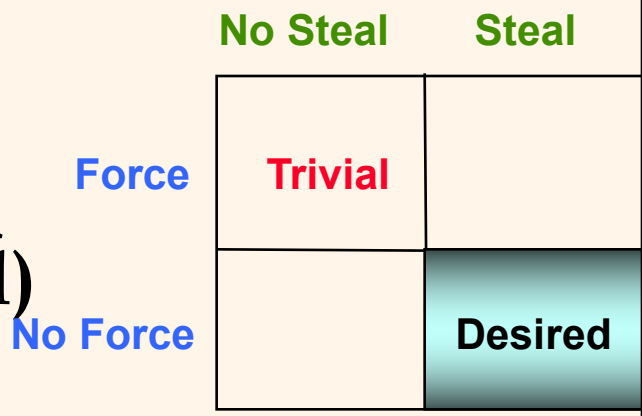
最早開始的技術

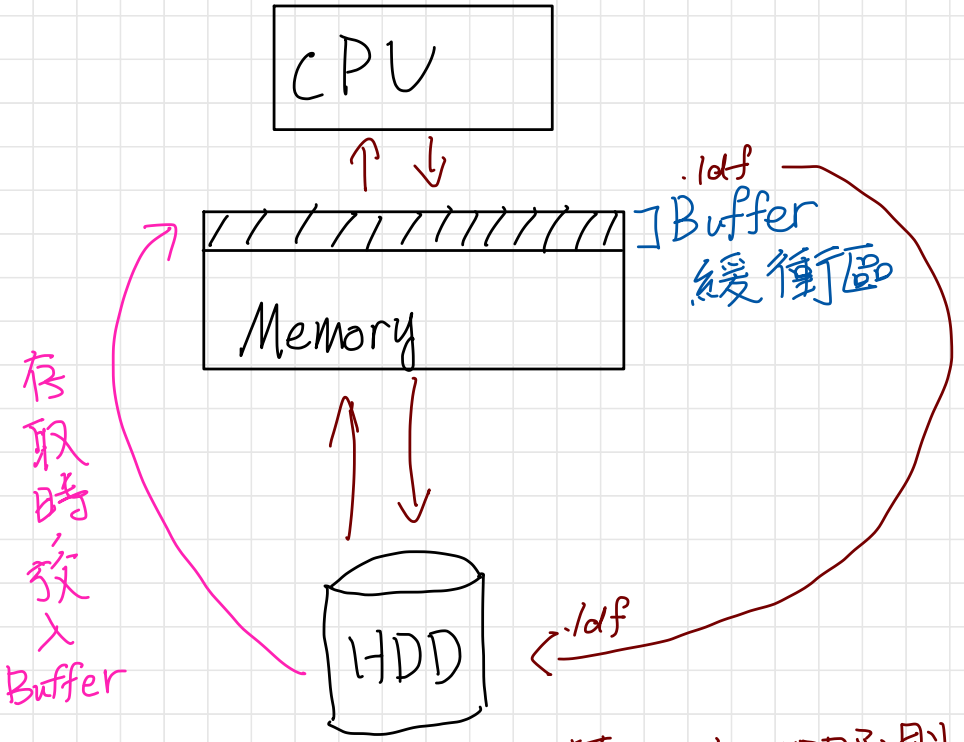
非大災難的復原(回復)技術

- **延遲更新(Deferred update)**
  - 延遲更新機制是直到交易達到確認點(交付點)(commit point)時，才對磁碟上的資料庫進行更新；而在確認(交付)前，更新通常僅發生在工作區間(local workspace， 或buffers)，一旦確認(交付)時，交易異動會記錄在日誌中，並更新磁碟上的資料庫，所以，若在未達確認點(交付點)時發生災難，則不會對資料庫產生任何異動，所以不必進行UNDO運算，但可能必須對已確認(交付)卻尚未對資料庫產生實際異動的交易，進行 REDO運算 ，所以此種機制又稱 NO-UNDO/REDO 演算法
- **立即更新(Immediate update)**
  - 立即更新是某些交易未達到確認點(交付點)時，即對磁碟上的資料庫進行更新；這種機制要求對資料庫更新時，必須先更新磁碟中的日誌檔。若發生災難時，尚未達到確認點(交付點)，卻已對資料庫產生異動，則必須對這些交易執行異動倒回(rolled back)，一般立即更新機制須進行 UNDO及 REDO ，所以此種機制又稱 UNDO/REDO 演算法

緩衝區間的處理(Buffer Pool)

- **強迫式(Force)：當交易確認(交付)時(commit)，所有被更新的資料立即被寫入磁碟中的資料庫，稱為強迫式(*force approach*)，否則稱為不強迫式(*no-force*)**
  - 強迫式：較差的反應時間(Poor response time)，但可保證永久性(durability)
- **偷跑式(Steal)：若交易未達確認點(交付點)時，即可將緩衝區資料寫入磁碟中的資料庫，稱為偷跑式(steal)，否則稱為不可偷跑(no-steal)**
  - 若不可偷跑，則緩衝區使用率差(poor throughput)
  - 但若可偷跑，則如何保證單元性(atomicity)？

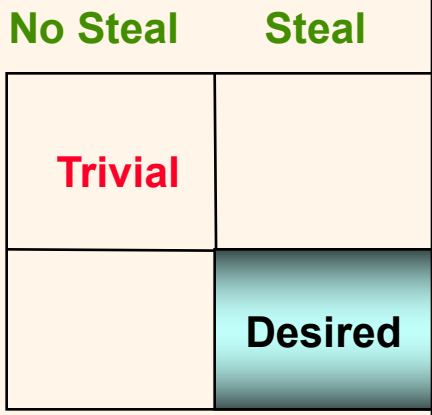




若 ldf 存入HDD了，則 commit 成功，
就算資料無真的傳入成功
⇒ 不強迫式 ⇒ 失敗用 ldf 做 REDO

緩衝區間的處理(Buffer Pool)－續

- 前述延遲更新(*deferred update*)的機制即採用不可偷跑的方法(no-steal)
- 但一般資料庫管理系統多採用偷跑/不強迫策略(*steal/no-force*)
  - 偷跑的好處是不需要在記憶體中空出一大塊的緩衝區存放暫時更新的資料
  - 不強迫的好處是當某更新資料達確認點(交付點)時，仍然可以留在緩衝區中，供其他交易執行更新，避免其他交易再從磁碟中(I/O)讀取資料的耗費
    ⇒減少 IO



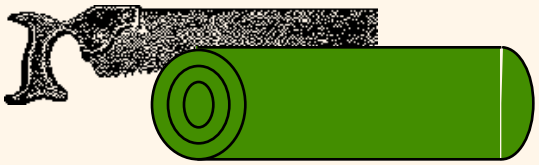
先寫入日誌檔協定 Write-Ahead Logging (WAL)

所有異動先寫入日誌檔　保證永久性

- 先寫入日誌檔協定(Write-Ahead Logging Protocol)
  - ①直到所有更新交易的UNDO-型態的**日誌資料**(UNDO-type log records)都已強迫寫入磁碟中，在磁碟中資料庫的舊資料(before image)才可被新資料(after image)更新
  - ，直到所有交易的REDO-型態 and UNDO-型態的**日誌資料**都已強迫寫入磁碟中，更新交易的確認(交付)運算(commit)才算完成
- 究竟如何完成紀錄日誌與復原(回復)？
  - 將介紹 ***ARIES*** 演算法

Return

寫入日誌的基本概念 (Logging)



- 對所有更新交易，要記錄所有REDO及 UNDO 的資訊於日誌檔中
  - 一般是順序式寫入日誌檔，且存放另一磁碟中
  - 僅將異動部份資料寫入日誌檔中(量較少)，所以，每頁日誌檔可存放多筆更新資料
- 日誌檔(Log)：依序存放一串(REDO/UNDO)動作
  - 日誌檔一般包含：

Read的日誌檔僅包含

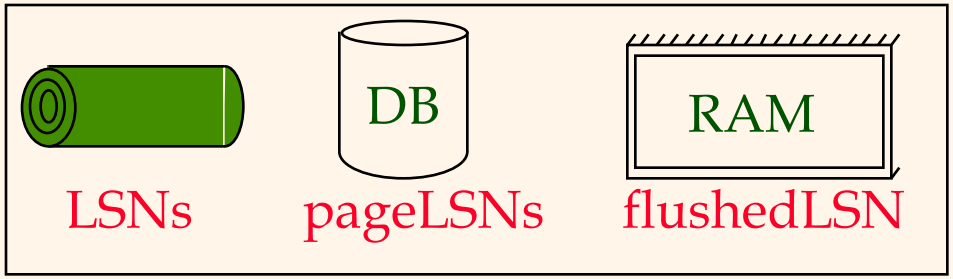
Update only

Log Sequence Num.

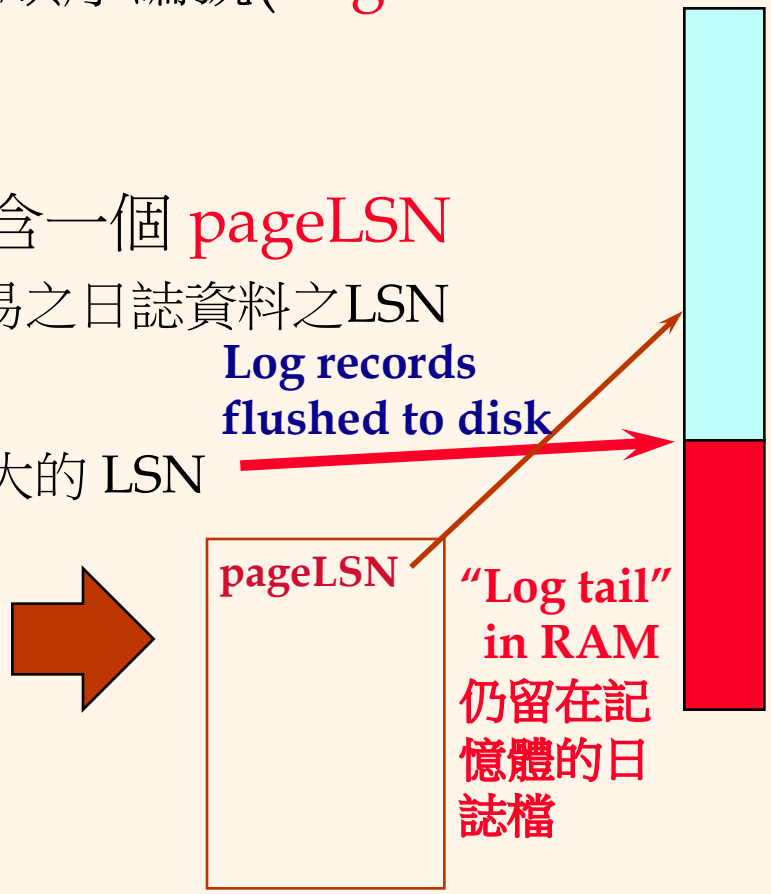
<LSN, PrevLSN, XID, Type, pageID, offset, length, old data, new data>

  - 及其他控制資料(後面介紹)

WAL協定及日誌檔內容



- 每一筆日誌資料都有具唯一性的日誌順序編號(Log Sequence Number, LSN)
  - LSNs通常是遞增，紀錄日誌資料的位置
- 資料庫中每一資料頁(*data page*)都包含一個 pageLSN
  - pageLSN是紀錄最近更新此資料頁的交易之日誌資料之LSN
- 系統紀錄 flushedLSN
  - flushedLSN是目前日誌檔寫入磁碟中最大的 LSN
- WAL: 更新資料寫入磁碟前，先將日誌檔寫入磁碟
  - pageLSN ≤ flushedLSN



日誌檔資料錄(Log Records)

日誌檔資料欄位

LSN
prevLSN(LAST_LSN)
XID (交易編號)
type
pageID
length
offset
before-image
after-image

僅更新才有

日誌資料型態種類

- **Update**
- **Commit**
- **Abort**
- **Undo(補償日誌記錄 (Compensation Log Records , CLRs))**
  - 以進行UNDO運算
- **End** (交易確認(交付)或中斷的標記)

❖ **與日誌檔相關的其他資料表**

日誌記錄 – 更新資料(UPDATE)

- 當更新完某資料頁，將會加入一筆更新型態的紀錄到日誌檔，該資料頁的 pageLSN將被設定為此筆日誌檔的LSN

return

日誌記錄 – 交易確認(Commit)

- 寫一筆確認(交付)紀錄於日誌檔中
- 該交易所有到lastLSN(某交易最新異動的LSN)的資料全部強迫寫入磁碟的日誌檔中
  - 保證 flushedLSN ≥ lastLSN
  - 寫入磁碟中的日誌檔是採順序式、同步寫入
  - 每一日誌頁含有多筆日誌資料錄
- 傳回確認(交付)成功Commit()

*只要ldf寫入磁碟，即commit成功

return

日誌記錄 – 交易中斷(Abort)

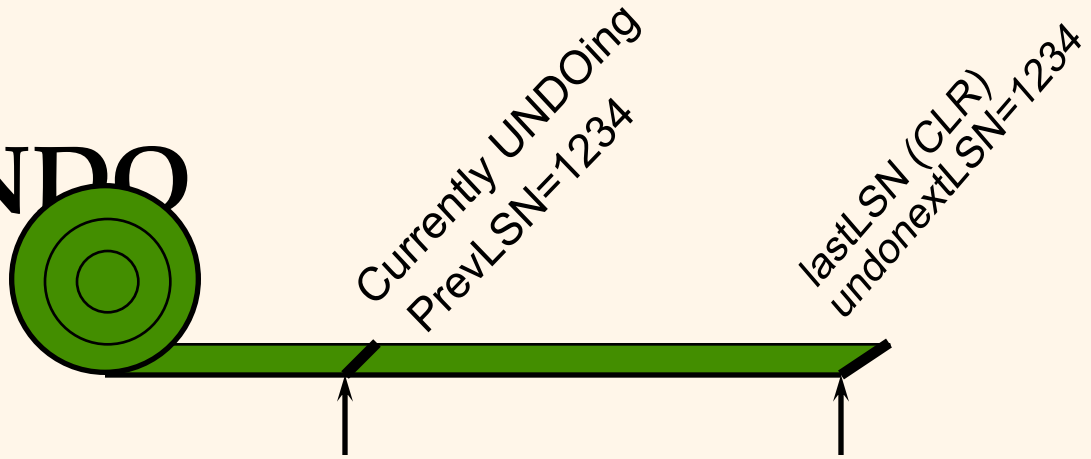
因為倒跑

- 假設由系統中斷某交易(不是發生當機)
- 必須將日誌檔的交易倒回執行，復原(回復)更新 (UNDOing updates)
  - 從交易表(Xact table)中取得lastLSN(該交易最新異動的LSN)
  - 利用prevLSN 將此交易之運算連結起來，進行復原(回復)更新(UNDOing updates)

Transaction table

return

日誌記錄 –UNDO



- 要執行 UNDO，必須先鎖定資料
  - 沒問題，由系統自動鎖定
- 在回復資料頁的舊資料前，先寫入一筆補償資料錄(CLR)：
  - 進行UNDO時，仍同時進行寫入日誌!!
  - CLR 多一欄位：undonextLSN (prevLSN⇒undonextLSN)
    - 記錄下一筆要進行undo的LSN (即正在進行UNDO的資料錄的prevLSN)
  - CLRs 永遠不必 Undone (但在重複舊曆史運作時也許需重作REDO，如此可保證單元性
  - 執行完 UNDO後，寫一筆 “end”到日誌檔

return

其他與日誌檔相關的表格

Xact Table

XID	LastLSN(最新)	status C/U committed Undone

- 交易表(Transaction Table)
  - 每一個執行中的交易，在交易表中紀錄一筆資料
  - 每一筆資料錄包含交易編號(***XID***)、***此交易最新的(最後的)LSN編號(lastLSN***)、此交易目前狀態(***status***)(有running(執行中)、commited、aborted)
- 緩衝區資料頁表(Dirty Page Table)
  - 存在緩衝區中的每一資料頁， 在緩衝區資料頁表中紀錄一筆資料
  - 每一筆資料錄包含資料頁編號(***PAGE ID***)－被異動的資料頁編號；***最早***對此資料頁異動的LSN編號 ***LSN(earliest)***

Dirty Page Table (DPT)

Page ID	LSN (earliest 最早對 Page 做異動的 LSN)

查核點(Checkpoint)

- **DBMS會定期產生查核點，在查核點時完成下列工作，避免當系統發生當機而耗費許多時間做復原(回復)處理：**
  - 暫時停止進行所有交易
  - 強迫將存放在緩衝區異動過的資料存入磁碟中
  - 在日誌檔寫入 [checkpoint]資料，且強迫將緩衝區的日誌檔寫入磁碟中
  - 系統恢復執行交易
- **寫入日誌檔的內容：**
  - begin_checkpoint ：表示查核點起始點
  - end_checkpoint ：包含目前交易表(**Xact table**)及緩衝區資料頁表(**dirty page table**)：
    - 寫入begin_checkpoint後，其他交易繼續執行，所以，上述表格僅在 begin_checkpoint 階段正確，稱為模糊檢查點(fuzzy checkpoint)
  - 系統會存放begin_checkpoint 的LSN在安全的地方(稱為*master* record)，以供進行復原(回復)時使用

一般交易的動作與機制

- 一般交易會進行一連串的讀、寫動作，最後以確認(交付)(commit)、或中斷(abort)結束
- 限制式2PL(Strict 2PL)：交易必須直到確認(交付)交易(commit)或被中斷(abort)才釋放排他性鎖定 ；所以，其他交易在此交易確認(交付)交易或被中斷前，無法取存(讀、寫)被此交易更新的資料，同時可以確保此排程具可復原(回復)性 (recoverability)
- **STEAL, NO-FORCE** 緩衝區管理機制、WAL(Write-Ahead Logging)機制(先寫入日誌檔)
- 查核點機制(Checkpoint)

各資料存放在何處的整體觀

LogRecords

LSN
prevLSN(LAST_LSN)
XID
type
pageID
length
offset
before-image
after-image

Data pages
Each with a
pageLSN(PAGE_ID)

master record
(begin_check_point)

Xact Table

Xact ID
lastLSN
status

Dirty Page Table

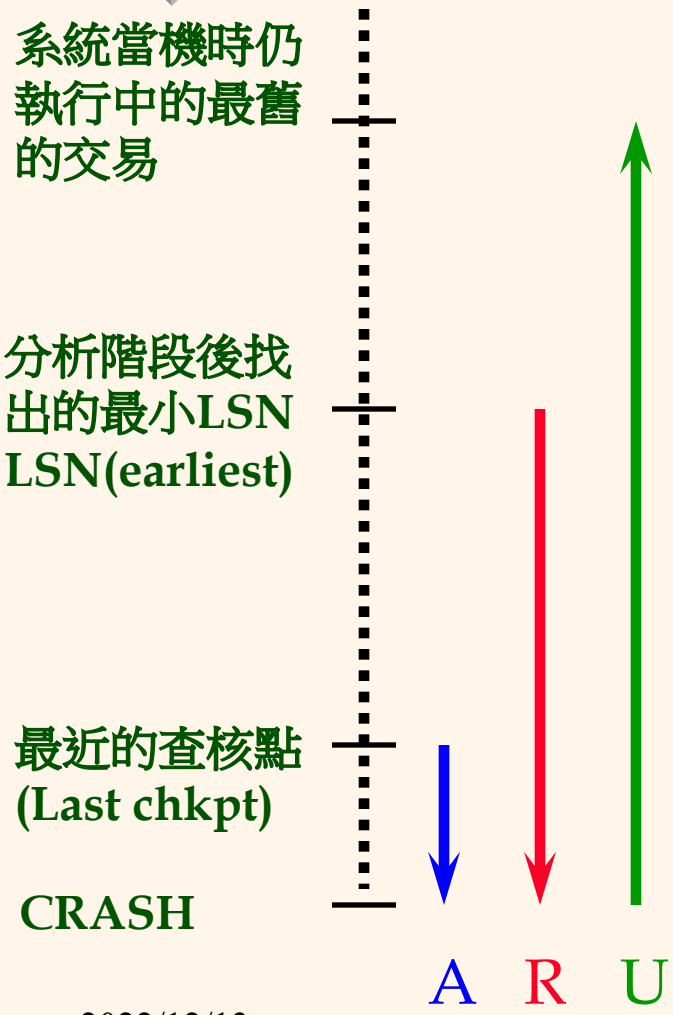
PAGE_ID
LSN(earliest)

flushedLSN

ARIES 復原(回復)演算法 -- 包含3 個復原(回復)概念

- **Aries**復原(回復)演算法採取 steal/no-force 機制，且根據 3 個復原(回復)概念：
  - **Write-ahead logging(先寫入日誌檔)**
  - **在重作(REDO) 階段重複執行過去的歷史運算 (Repeating history during redo)**
    - ARIES會追溯並恢復至系統當掉前的狀態 ，而在系統當掉時未完成的交易必須 undone
  - **在UNDO階段，也會記錄更新於日誌檔(Logging changes during undo)**
    - 如此可以避免ARIES重複執行已完成UNDO的運算(即若執行復原(回復)時發生當機，不必重作已完成UNDO的運算)

Aries 災難復原(回復)的三個階段



- 從最近的查核點開始 (最近的checkpoint記錄於 master record)
- 3個階段(phases)
  - 分析Analysis：找出從查核點開始，哪些交易已確認(交付)、哪些是未完成(被中斷)的交易 恢復 Xact Table 以及 DPT
  - **重作REDO 所有的運算**
    - (重作所有執行過的交易)
  - **UNDO** 那些被中斷的交易

下一頁-Example

復原(回復): 分析階段

- 此階段工作
  - 找出應從日誌檔中何處開始進行重作(REDO)的工作
  - 找出仍存在緩衝區中的資料頁(dirty (updated) pages)
  - 找出系統當機時仍在執行的交易
  - 重建在查核點時的資料表
    - **更新 Xact table及 Dirty page table**
- 從日誌檔中的begin_checkpoint 開始掃描整個日誌檔，以重建這些資料表
  - 若讀到某交易T的日誌檔資料為end型態 ，則將此交易T從交易表中刪除，因為他已經執行完畢(abort 或 committed)
  - 若不是end的日誌檔資料錄： 若此交易尚未在交易表中出現，則加入一筆到**交易表**中；若是舊交易，則更新**交易表**之lastLSN=LSN及更新交易狀態(Xact status)，若已經確認(交付)，則更新狀態維為C (commit)，否則為 U(未來必須 undone)
  - 若讀到需重作的日誌檔資料錄，若資料頁P尚未出現在**Dirty Page Table**，則：
    - 加P至D.P.T., 設定值 LSN(earliest)=LSN

return

復原(回復)：重作階段(REDO)

- 將過去執行過的交易重作(repeat History)，以重建系統當機時的狀態：
  - 重作所有更新交易(包括被中斷的交易)，及重作補償日誌資料錄(CLRs)
- 從D.P.T.表中最小的 LSN(earliest) 開始掃描整個日誌檔，重作(REDO)每一CLR或更新的LSN ，除了：
  - 此LSN異動的資料頁未出現在Dirty Page Table (表示異動已更新到磁碟的資料庫中，所以，不必重作) ，或
  - 此LSN異動的資料頁出現在Dirty Page Table中， 但 **LSN(earliest) > LSN(log record)** (表示異動已更新到磁碟的資料庫中，所以，不必重作) **，或**
  - pageLSN (在資料庫中) >LSN (日誌檔資料錄) (表示有更新的交易(LSN)，對DB異動已更新到磁碟的資料庫中，所以，不必重作)
- 重作的動作：
  - 重新執行日誌檔中的交易行為
  - 將pageLSN設為重作的日誌檔資料錄的 LSN ，不必再記錄到日誌檔中

復原(回復): UNDO 階段

- Undo是利用分析階段重建的交易表開始進行，此表記錄系統當機時尚在執行中的交易，及最新的日誌檔的LSN；每次進行UNDO均加入一筆CLR，UNDO是反方向進行，直到所有在預備UNDO 的資料集(ToUndo)的動作均UNDO完畢(ToUndo資料集清空)

ToUndo={ *l* | *l* 是未完成的交易之最新LSN (lastLSN)}

Repeat：重複

- 選擇 ToUndo中最大的 LSN
- 若此 LSN 為 CLR且undonextLSN==NULL
  - 為此交易寫入一筆 End 資料錄
- 若此 LSN 為 CLR且undonextLSN != NULL
  - 將此 undonextLSN內之值 (下次應執行UNDO之LSN)寫入 ToUndo資料集中
- 若此 LSN 為更新(update)，則 Undo此更新，寫入一筆CLR，將 prevLSN寫入 ToUndo資料集中

Until ToUndo is empty(直到ToUndo集合空了(全部處理完畢))

return

Example

crash

(a)

LSN	LAST_LSN	TRAN_ID	TYPE	PAGE_ID	OTHER INFORMATION
1	0	T1	update	C	...
2	0	T2	update	B	...
3	1	T1	commit		...
4	begin checkpoint				
5	end checkpoint				
6	0	T3	update	A	...
7	2	T2	update	C	...
8	7	T2	commit		...

(b)

TRANSACTION TABLE

TRANSACTION ID	LAST LSN	STATUS
T1	3	commit
T2	2	in progress

DIRTY PAGE TABLE

PAGE ID	LSN
C	1
B	2

考試：分析階段完成後的樣子？

(c)

TRANSACTION TABLE

TRANSACTION ID	LAST LSN	STATUS
T1	3	commit
T2	8	commit
T3	6	in progress

DIRTY PAGE TABLE

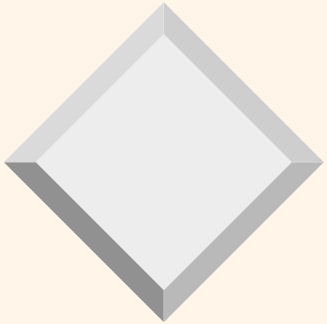
PAGE ID	LSN
C	1
B	2
A	6

Figure 21.6 An example of recovery in ARIES. (a) The log at point of crash. (b) The Transaction and Dirty Page Tables at time of checkpoint. (c) The Transaction and Dirty Page Tables after the analysis phase.

範例說明

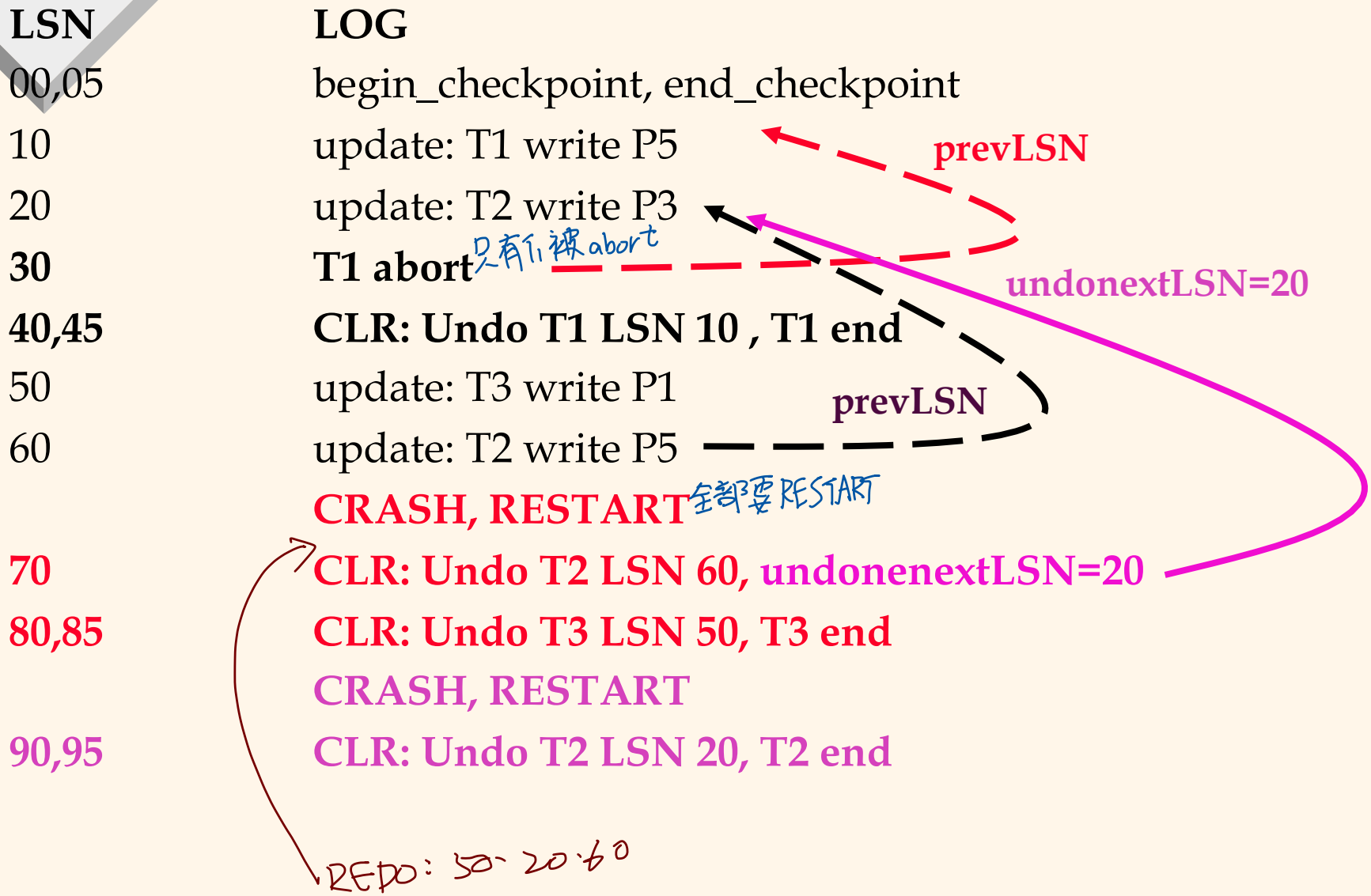
- The analysis phase starts from location 4 until it reaches the end. The end-checkpoint record would contain the T.T. and D.P.T. as in Fib (b), and the analysis phase will further reconstruct these tables.
  - Log 6, a new entry for T3 is made in the T.T and a new entry for page A is made in the D.P.T.
  - Log 8, T2's status in T.T is updated.
- REDO phase: the smallest LSN in the dirty page is 1. The LSN {1,2,6,7} corresponding to the updates for pages C, B, A, and C reapplied.
- The UNDO is applied only to the active Xact T3. UNDO phase starts at log 6 and proceeds backward in the log. The backward chain of updates for Xact T3 is followed and undone(only log 6 in this example)

TOUNDO{ 6 } For LSN=6 ∅commit當作已做完了,已經經過commit



本章節暫時告一段落
祝期末考順利

Example—crashes during restart



承LSN60下 CRASH, RESTART略

Xact Table

XID	LastLSN	status
~~T_1~~	(已被abort)	
T_2	60	U
T_3	50	U

*U = Uncommited

Dirty Page Table (DPT)

PAGE_ID	LSN(earliest)
P_1	50
P_3	20
P_5	60 Ø T_1:abort前不用理他，因為已經abort掉

Redo

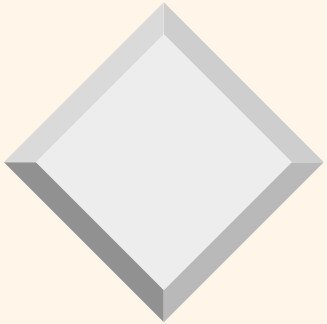
To UNDO: {60, 50} For LSN = 60

To UNDO: {50, 20} For LSN = 50

To UNDO = 20

Summary of Logging/Recovery

- Recovery Manager guarantees Atomicity & Durability.
- Use WAL to allow STEAL/NO-FORCE w/o sacrificing correctness.
- LSNs identify log records; linked into backwards chains per transaction (via prevLSN).
- pageLSN allows comparison of data page and log records.



Summary, Cont.

- Checkpointing: A quick way to limit the amount of log to scan on recovery.
- Recovery works in 3 phases:
  - Analysis: Forward from checkpoint.
  - Redo: Forward from oldest recLSN.
  - Undo: Backward from end to first LSN of oldest Xact alive at crash.
- Upon Undo, write CLRs.
- Redo "repeats history": Simplifies the logic!