以下用 LRU/Clock 演算法，4 frames，逐步演示 trace1 的每一步，並標註 disk read/write：

**計算 page number（每頁 4KB，PAGE\_OFFSET=12）**

* 0000000 → 0
* 0001000 → 1
* 0002000 → 2
* 0003000 → 3
* 0002000 → 2
* 0004000 → 4
* 0003008 → 3
* 0003008 → 3
* 0000004 → 0
* 0001004 → 1
* 0005008 → 5
* 0002004 → 2

| **步驟** | **指令** | **page** | **frame狀態** | **disk read** | **disk write** | **說明** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | W 0000000 | 0 | [0] | 1 | 0 | page fault, dirty |
| 2 | R 0001000 | 1 | [0,1] | 2 | 0 | page fault |
| 3 | R 0002000 | 2 | [0,1,2] | 3 | 0 | page fault |
| 4 | R 0003000 | 3 | [0,1,2,3] | 4 | 0 | page fault |
| 5 | W 0002000 | 2 | [0,1,2,3] | 4 | 0 | hit, dirty |
| 6 | R 0004000 | 4 | [1,2,3,4] | 5 | 1 | page fault, evict 0 (dirty) |
| 7 | R 0003008 | 3 | [1,2,4,3] | 5 | 1 | hit |
| 8 | W 0003008 | 3 | [1,2,4,3] | 5 | 1 | hit, dirty |
| 9 | R 0000004 | 0 | [2,4,3,0] | 6 | 2 | page fault, evict 1 (clean) |
| 10 | R 0001004 | 1 | [4,3,0,1] | 7 | 2 | page fault, evict 2 (dirty) |
| 11 | R 0005008 | 5 | [3,0,1,5] | 8 | 2 | page fault, evict 4 (clean) |
| 12 | R 0002004 | 2 | [0,1,5,2] | 9 | 3 | page fault, evict 3 (dirty) |

**統計**

* **disk reads**：每次 page fault 都要讀新頁面，共 9 次。
* **disk writes**：每次換出 dirty page，要寫回磁碟，共 3 次（step 6, 10, 12）。
* **page fault rate**：9/12 = 0.75

**說明**：  
每次存取時，如果頁面不在 frame 內（page fault），就 disk read；如果被換出的頁面 dirty，就 disk write。  
LRU/Clock 在這個 trace 下 victim selection 一樣，所以結果完全一致。

**1. dirty, clean 是什麼意思？**

* **dirty page**：這個頁面在 memory 裡有被「寫入」過（W 指令），內容和磁碟上的版本不同。換出時必須寫回磁碟（disk write）。
* **clean page**：這個頁面在 memory 裡只被「讀取」過（R 指令），內容和磁碟一致。換出時不用寫回磁碟。

**2. W 跟 R 操作有什麼不同？為什麼 R 也會 disk write？**

* **disk read**：只有 page fault（miss）時才會發生，不管是 R 還是 W，只要 memory 沒有這個 page，就要從磁碟讀進來。
* **disk write**：只有「換出 dirty page」時才會發生。
  + 不是每次 W 都會 disk write，只有 dirty page 被 evict（換出）時才會。
  + 步驟 9~12 雖然都是 R，但換出的是 dirty page（之前有 W 過），所以要 disk write。

**3. page 是怎麼算的？**

* 以 4KB 為一頁（PAGE\_OFFSET=12），指令的第1個欄位是 16進位位址。
* page number = 位址右移 12 位元（logical\_address >> 12）。
* 不是看第4個 bit，而是整個位址右移 12 位元。

**4. hit 的判斷方式？W 跟 R 有什麼差別？**

* **hit**：page 已經在 frame 裡，不會 page fault。
* **miss**：page 不在 frame 裡，要 page fault。
* W 跟 R 都會判斷有沒有 hit，差別是：
  + R 只更新 ref bit/last used。
  + W 會把該 page 標記成 dirty（dirty bit = True）。
* 判斷有沒有 hit，就是看 page 有沒有在 frame 狀態裡。

**總結**：

* dirty/clean 只影響換出時要不要寫回磁碟。
* disk read 只在 miss 時發生，disk write 只在換出 dirty page 時發生。
* page number 是位址右移 12 位元算出來。
* hit 就是 page 在 frame 裡，W 會多標 dirty。

**📊 現象分析：**

**發生的情況：**

* 所有trace在到達最大memory frames時，disk writes都變成 **0**
* 這個轉變點很明顯：從有數值突然降到0，形成垂直下降

**🧠 背後的作業系統原理：**

1. **記憶體飽和點 (Memory Saturation Point)**：
   * 當memory frames數量達到或超過程式實際需要的unique pages數量時
   * 就不再需要進行page replacement了
   * 所有需要的pages都能常駐記憶體中
2. **Working Set概念**：
   * 每個程式都有一個**Working Set** (工作集)
   * 這是程式在某個時間窗口內實際使用的page集合
   * 當記憶體大於Working Set時，就不會有page faults了
3. **實驗數據證實**：
   * BZIP: 在317 frames後disk writes = 0
   * SWIM: 在2543 frames後disk writes = 0
   * GCC: 在2852 frames後disk writes = 0
   * SIXPACK: 在3890 frames後disk writes = 0

**💡 這個觀察的意義：**

1. **程式記憶體需求特徵**：
   * 可以從轉變點推算每個程式的實際Working Set大小
   * SIXPACK需要最多記憶體(~3890 frames = ~15MB)
   * BZIP需要最少記憶體(~317 frames = ~1.2MB)
2. **演算法效能邊界**：
   * 在記憶體充足時，所有演算法表現相同
   * 演算法差異只在記憶體不足時才顯現