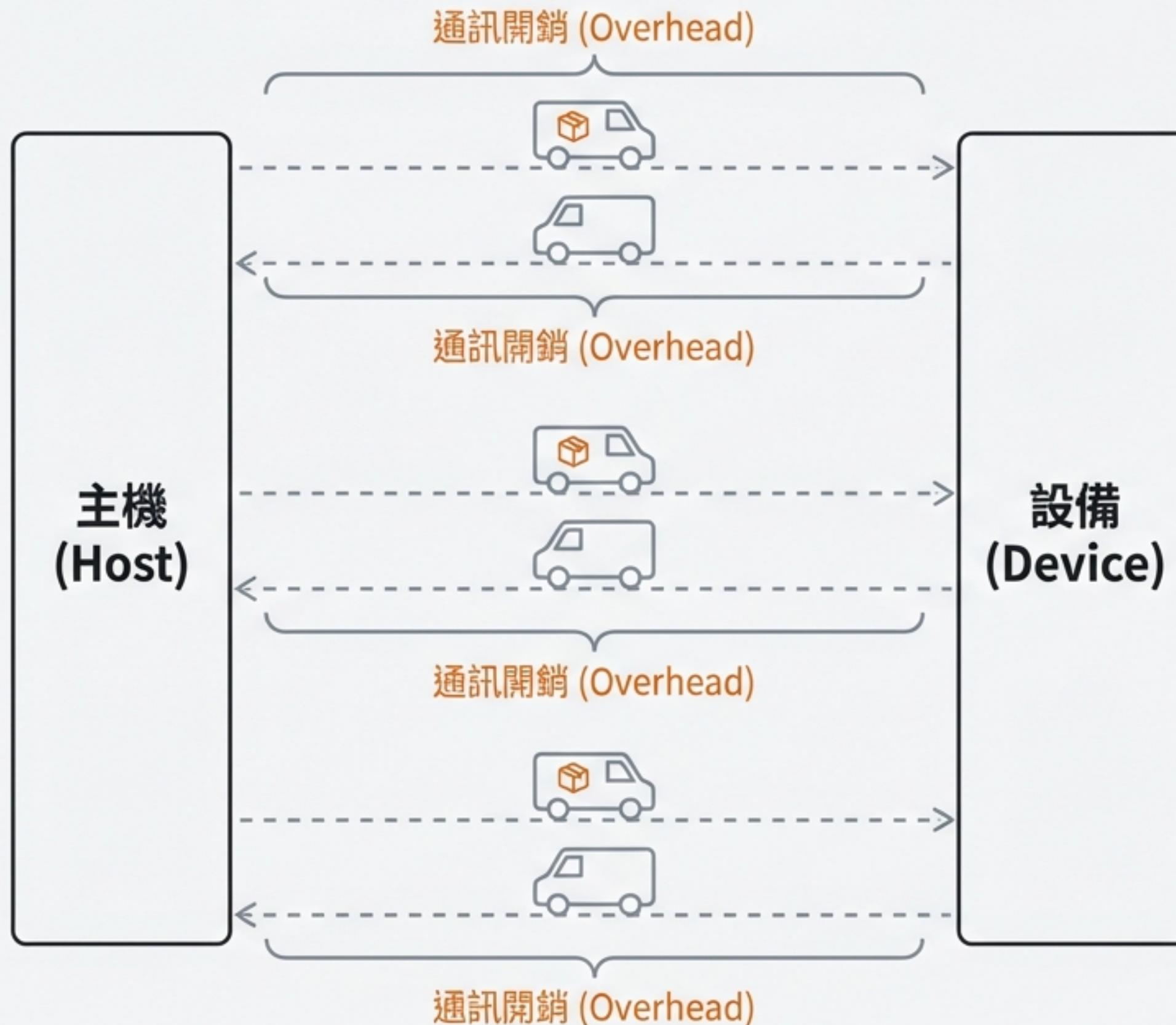


# e·MMC 5.1 封裝指令 (Packed Commands) 技術實務指南

---

從逐單派送到集運服務：徹底釋放零散 I/O 效能

# 效能瓶頸：逐單派送的隱形成本



## 為什麼傳統 I/O 在處理大量、零散請求時效率低落？

傳統的單次讀寫操作，就像快遞員「每送一個包裹，就要回總部一趟」。這種模式在處理大量小包裹時，時間都浪費在往返的路程上。

- **指令開銷 (Command Overhead)**：每次操作都需要獨立的指令週期 (e.g., `CMD23` + `CMD25`/18)，導致匯流排頻繁被命令/響應佔用。
- **匯流排效率低落**：有效數據的傳輸時間佔比極低，大部分時間消耗在通訊協議的握手手上。
- **延遲累積 (Latency Accumulation)**：總執行時間遠大於實際數據傳輸所需時間，尤其在文件系統元數據更新、日誌記錄等場景下問題更為嚴重。

# 解決方案：集運服務的革命

**封裝指令 (Packed Commands)：**  
一次整合，整批運送

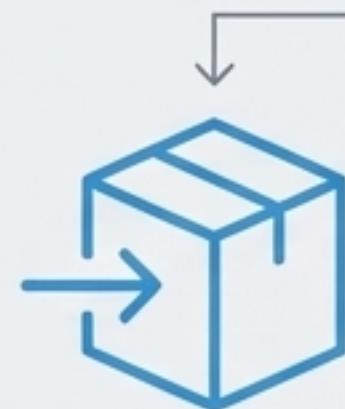
將多個讀取任務或多個寫入任務組合成一個群組，並在一次匯流排傳輸中完成。

- 根本性地降低通訊開銷**：將多次指令往返合併為一次。
- 顯著提升傳輸效率**：特別是在零散讀寫場景下，大幅提高性能與吞吐量。



# 行前準備：確認設備的集運能力

## 檢查項目: 主機 (Host) 必須檢查設備的 `EXT\_CSD` 暫存器



MAX\_PACKED\_READS

設備可封裝的最大讀取指令數量。

JEDEC 規範強制最小值為 5 ↗<sup>5</sup>\



MAX\_PACKED\_WRITES

設備可封裝的最大寫入指令數量。

JEDEC 規範強制最小值為 3 ↗<sup>3</sup>\

## 關鍵運作限制

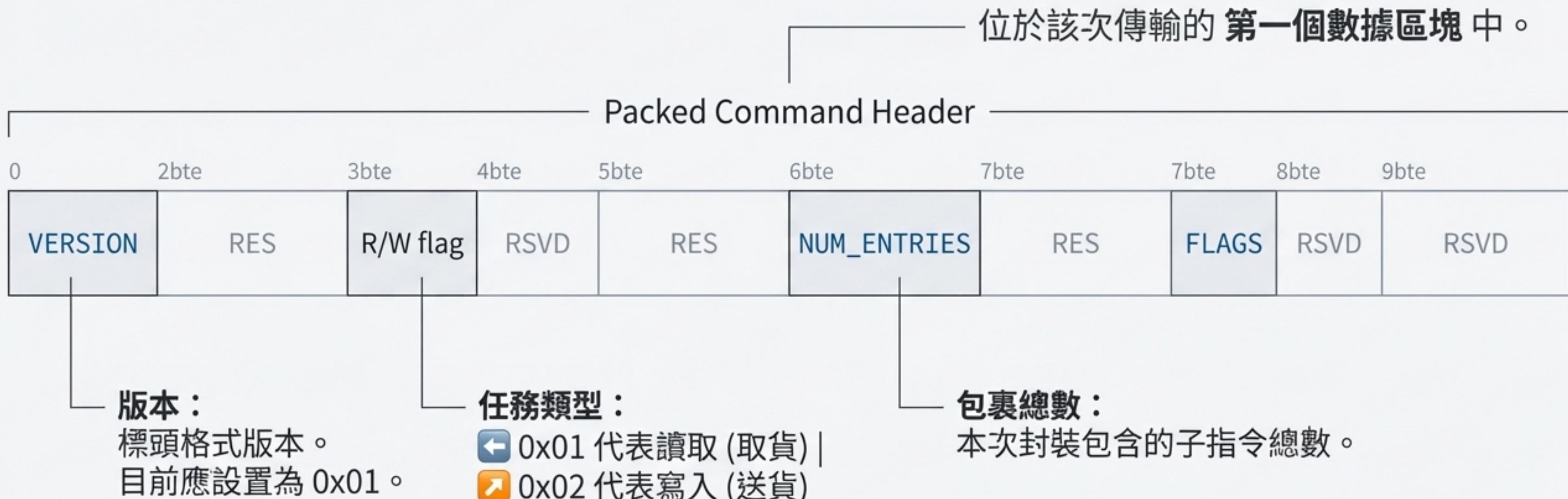


**同性質封裝：**一個封裝群組內必須 全部為讀取 或 全部為寫入 指令，不可混合。

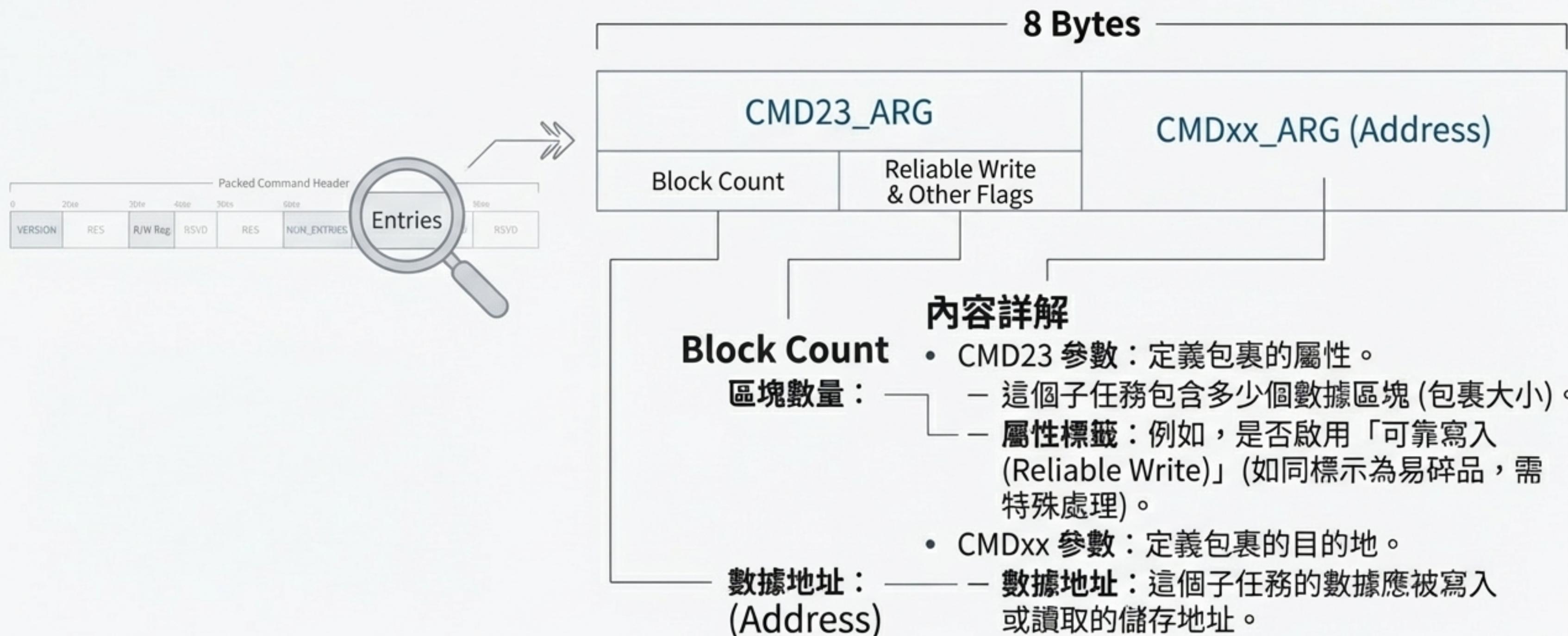


**地址一致性：**主指令 (CMD18/25) 的地址參數，必須與封裝群組中 第一個 指令的目標地址一致。

# 核心工具：運送總清單 (Packed Command Header)



# 清單細項：定義每個包裹的屬性



# 寫入流程：裝貨、出發、精準派送 (Packed Write)



## 1. 申報總量 (CMD23)

發送 CMD23 指令，其中：

- PACKED bit (bit 30) 設置為 1。
- 區塊總數 = 所有子任務的數據區塊總和 + 1 (標頭區塊)。

\*類比: 提前告知倉庫，這批貨物（包含清單本身）總共會佔用多少空間。

## 2. 開始傳輸 (CMD25)

發送

WRITE\_MULTIPLE\_BLOCK  
指令，開始數據傳輸。

\*類比: 貨車正式出發。

## 3. 數據封裝佈局

主機依序發送數據。

\*類比: 貨車的裝載順序：總清單放在最前面，後面緊跟著所有包裹。

## 讀取流程：發送訂單、集中取貨 (Packed Read)

### 操作時序 (兩階段)

#### 階段一：發送取貨清單 (Header)



- 發送 CMD23 (將 `PACKED bit` 設為 1，`count` 設為 1)。
- 緊接發送 CMD25，將 **僅包含 Header** 的數據區塊 傳送給設備。
- 類比：先用快遞把詳細的取貨清單送給倉庫，讓倉庫提前備貨。

#### 階段二：派車取貨 (Data)

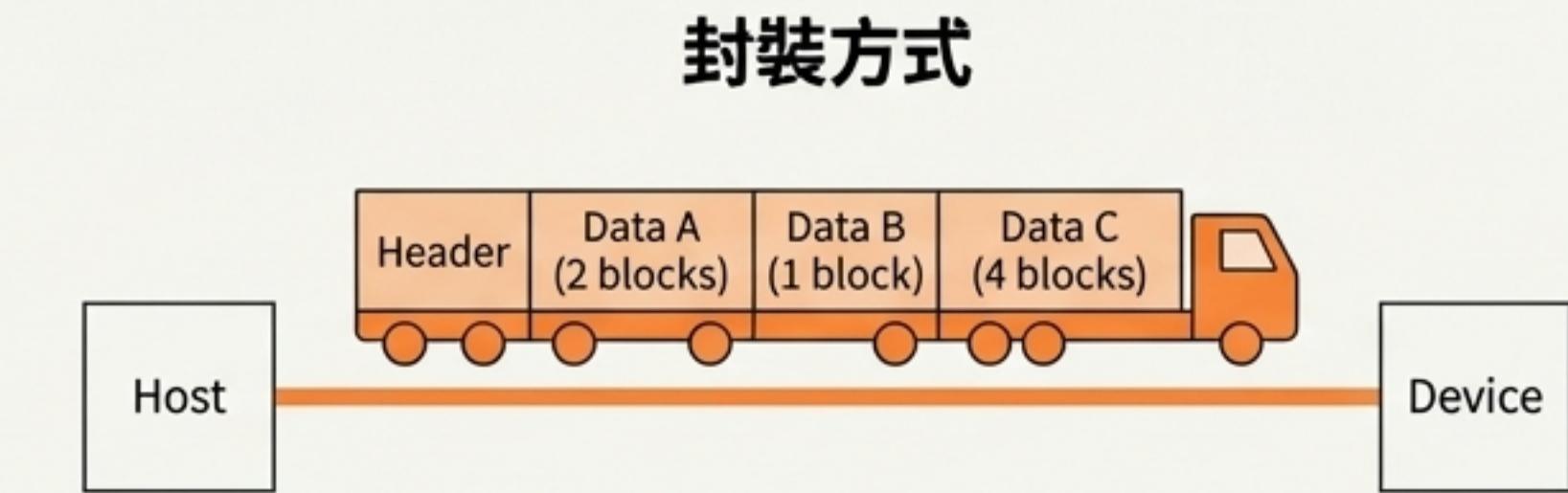
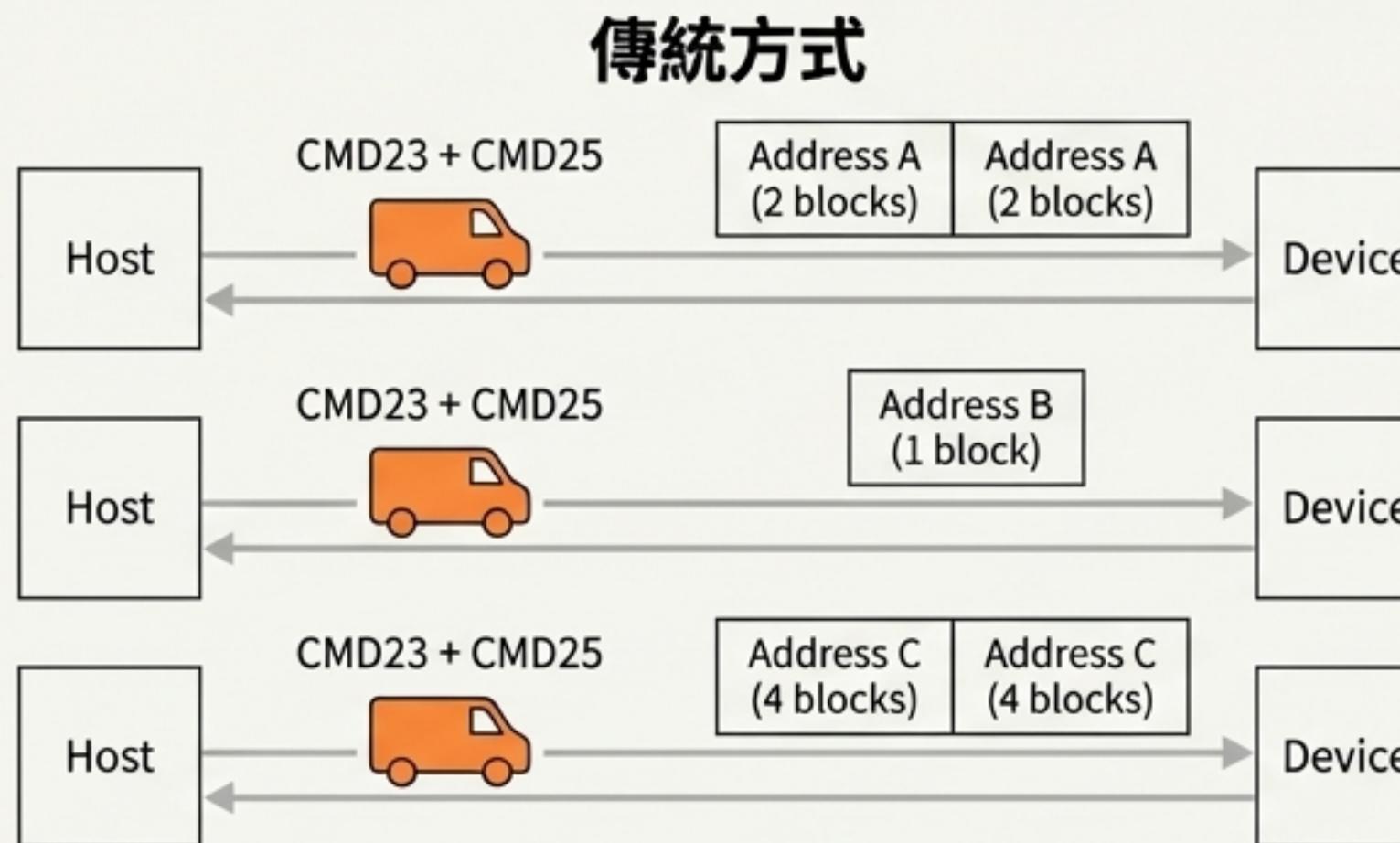


- 再次發送 CMD23 (將 `PACKED bit` 設為 1，`count` 設為 **所有欲讀取數據的總區塊數**)。
- 緊接發送 CMD18 (READ\_MULTIPLE\_BLOCK)，從設備**一次性讀取所有數據**。
- 從次倫大貨車取行為所有貨物。
- 類比：倉庫備好貨後，派一輛大貨車，依清單一次性收回所有貨物。

# 應用場景：一次寫入三個不連續地址

任務描述：主機需要同時寫入三塊不連續的數據：

- Address A: 2 blocks
- Address B: 1 block
- Address C: 4 blocks



**1 次 指令往返**

CMD23 : Count 參數設為  $2 + 1 + 4 + 1$  (Header) = 8。

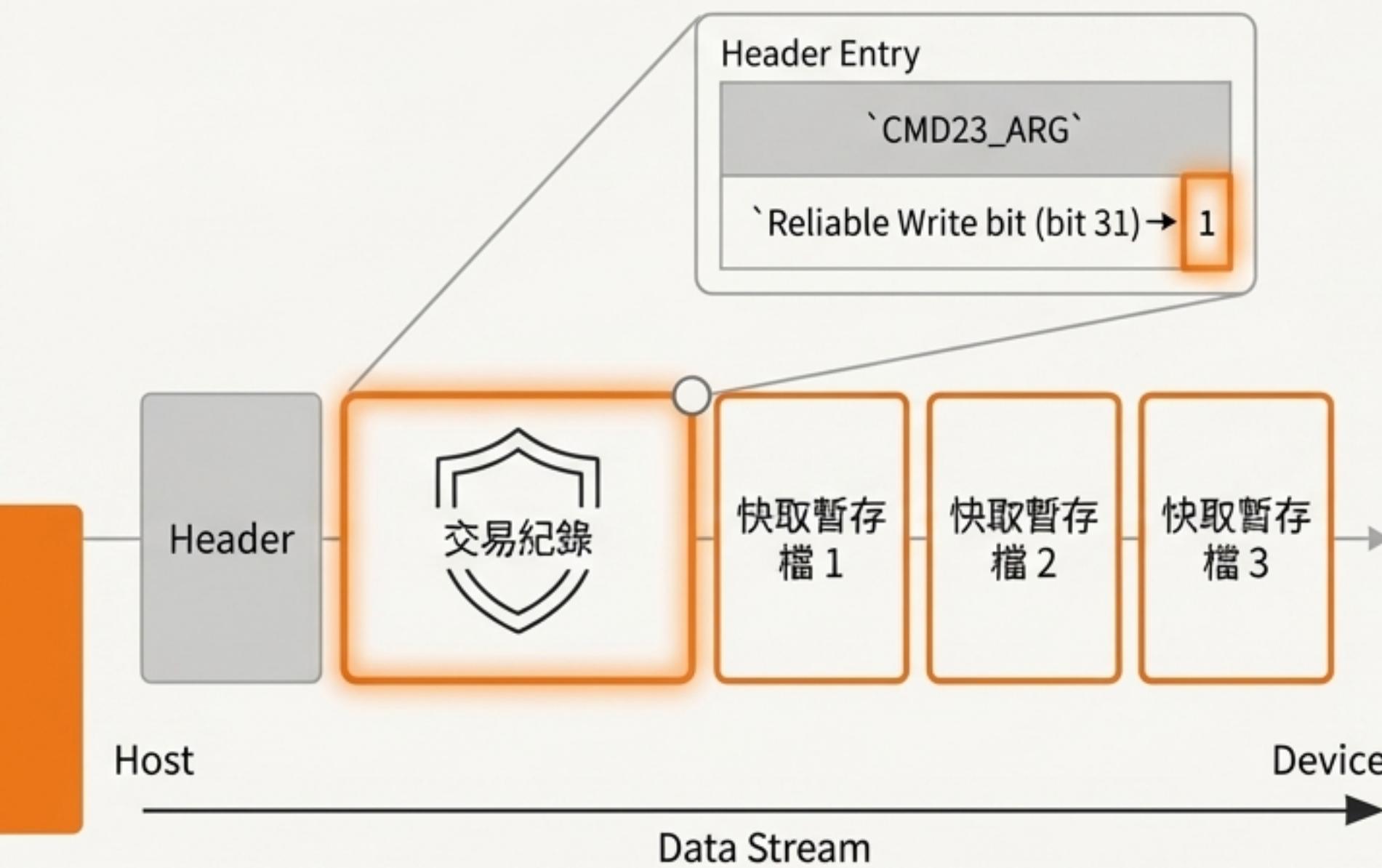
CMD25 : Address 參數設為第一個任務的地址，即 Address A。

實際數據流：[Header] + [Data A (2 blocks)] + [Data B (1 block)] + [Data C (4 blocks)]。

**3 次 獨立的指令往返**

結論：指令開銷從 3 次降為 1 次，大幅減少匯流排佔用。

# 進階應用：在封裝中混合使用可靠寫入 (Reliable Write)



**核心價值：**在同一次高速傳輸任務中，兼顧效能與關鍵數據的完整性。

## 實現機制：

- 在 **Packed Command Header** 中，針對需要保護的特定子任務條目 (Entry)。
- 將其 **CMD23\_ARG** 欄位中的 **Reliable Write bit (bit 31)** 設置為 **1**。

## 應用範例：

- 一次寫入操作中，包含「交易紀錄」（設為可靠寫入）和「快取暫存檔」（普通寫入）。
- 即使發生意外斷電，也能確保交易紀錄的寫入是完整的，同時不犧牲整體傳輸效率。

# 錯誤追蹤：當包裹派送失敗時

## 核心規則: 失敗即中止 (Fail-and-Stop)

若封裝群組中的 任一 子指令執行失敗，其後續的所有子指令將 不會被執行。



## 事後追蹤機制

錯誤發生後，主機可檢查 `EXT_CSD` 暫存器來定位問題：



- `PACKED_COMMAND_STATUS`  
一個狀態標誌，用於判斷本次封裝任務是否發生了任何錯誤（成功 / 失敗）。

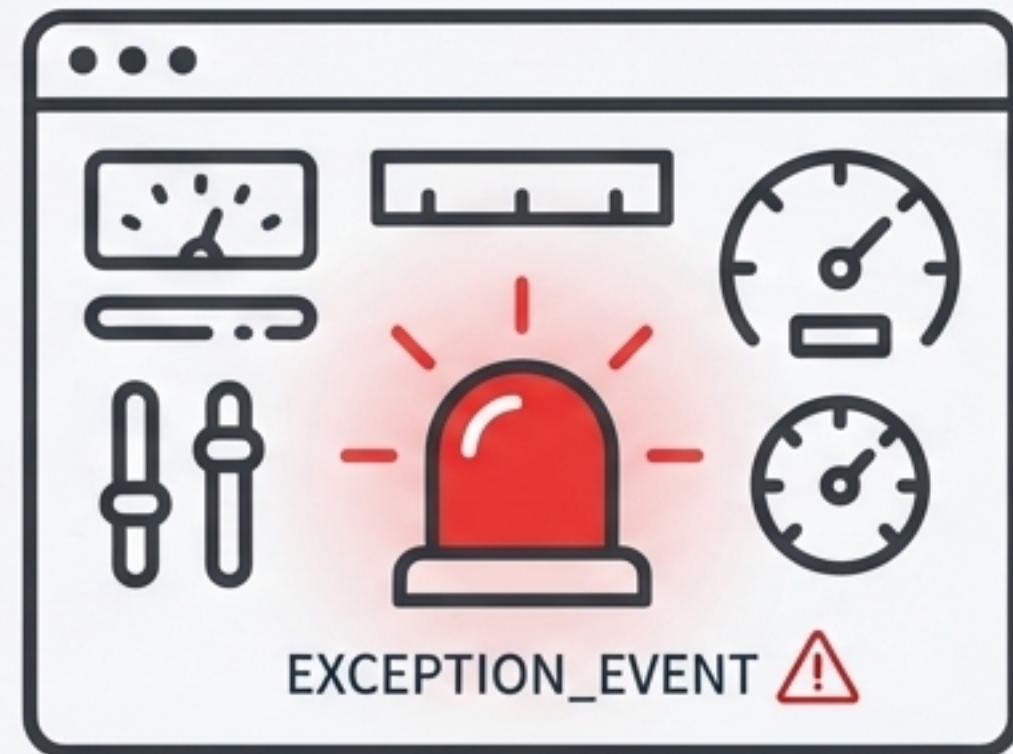


- `PACKED_FAILURE_INDEX`  
精確指出是 `Header` 清單中的第幾個指令（從 0 開始計數）發生了錯誤。

# 主動預警：啟用即時異常通知

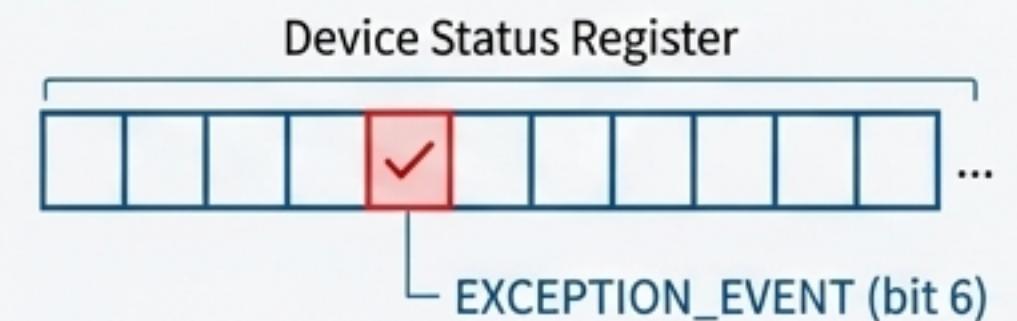
## 啟用步驟

在 EXCEPTION\_EVENTS\_CTRL 暫存器中，將 PACKED\_EVENT\_EN 位元啟用。



## 監測方式

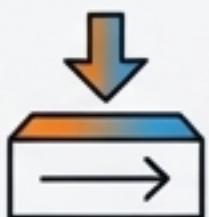
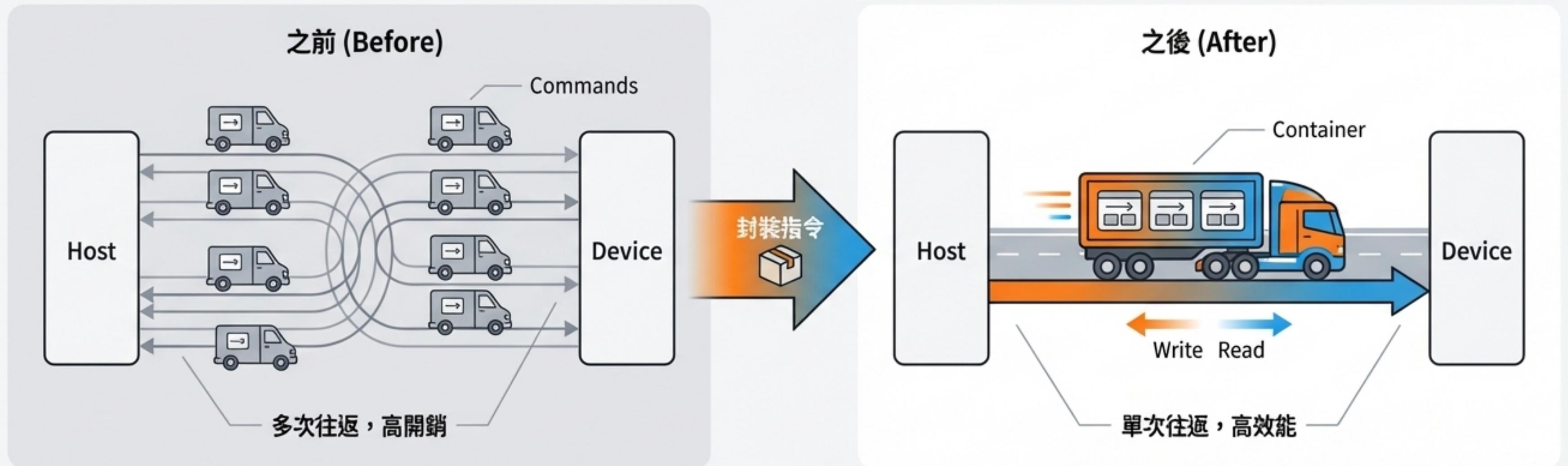
一旦封裝指令執行出錯，設備會在回報的設備狀態 (Device Status) 中，將 EXCEPTION\_EVENT (bit 6) 置位。



## 系統價值:

- 使主機系統無需在每次操作後都輪詢 PACKED\_COMMAND\_STATUS。
- 能夠更即時、主動地響應儲存錯誤，觸發相應的錯誤處理流程。

# 總結：更少的往返，更高的效率，更智慧的傳輸



**降低開銷 (Reduced Overhead)**  
從根本上解決了零散 I/O 的指令開銷與延遲問題。



**提升效能 (Increased Performance)**  
在真實世界的應用場景中，顯著提高吞吐量與系統響應速度。



**增加彈性 (Greater Flexibility)**  
允許在單一操作中混合使用可靠寫入等進階功能，實現精細化控制。



**健全機制 (Robust Mechanism)**  
清晰的「失敗即中止」規則與可追溯的錯誤回報機制，讓系統開發與除錯更為便捷。

# Q&A

感謝聆聽