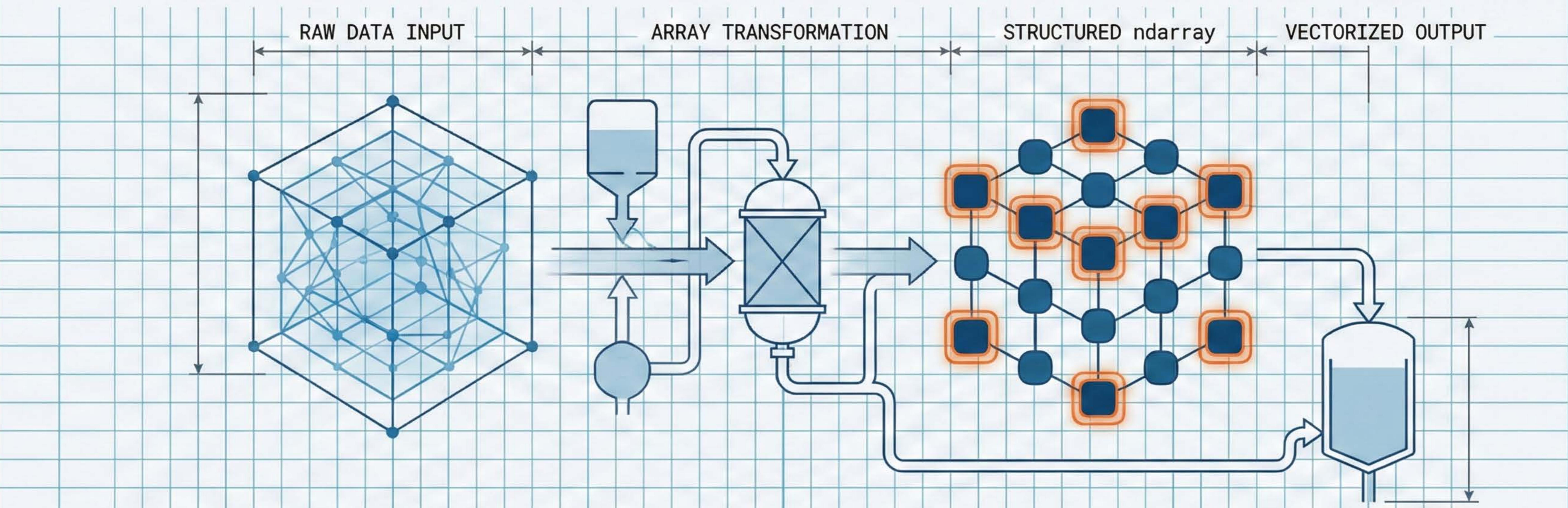


# Unit 03: NumPy 數值計算基礎

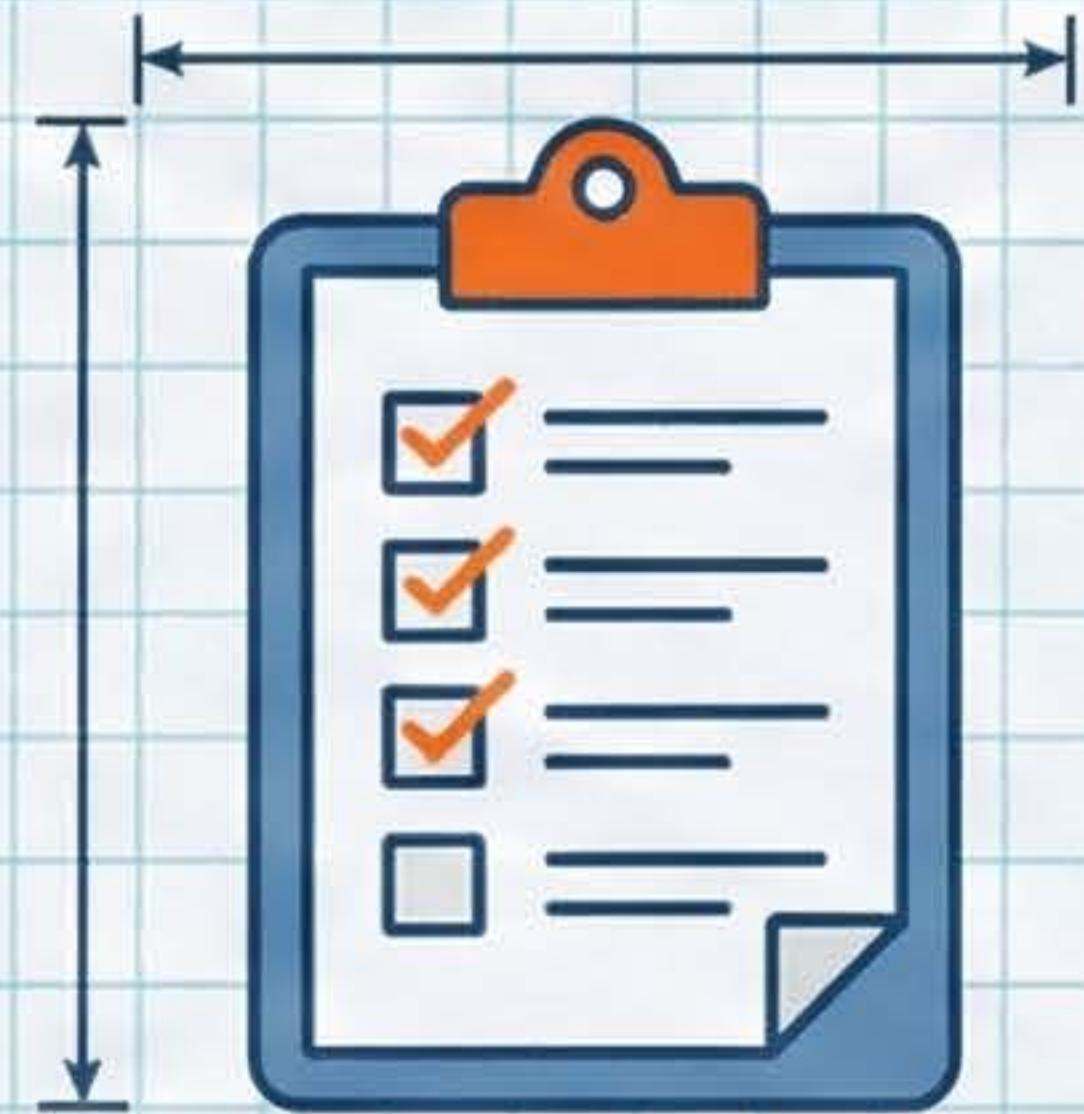
Python 科學計算的核心引擎：從陣列運算到化工模擬



- NumPy (Numerical Python) 是所有現代化工數據分析、機器學習 (Scikit-learn) 與深度學習 (TensorFlow) 的基石。
- 學習目標：理解 ndarray 結構、掌握向量化運算 (Vectorization) 、應用於反應動力學與質量平衡求解。

# 為什麼化工工程師需要 NumPy ?

## Python List (一般列表)



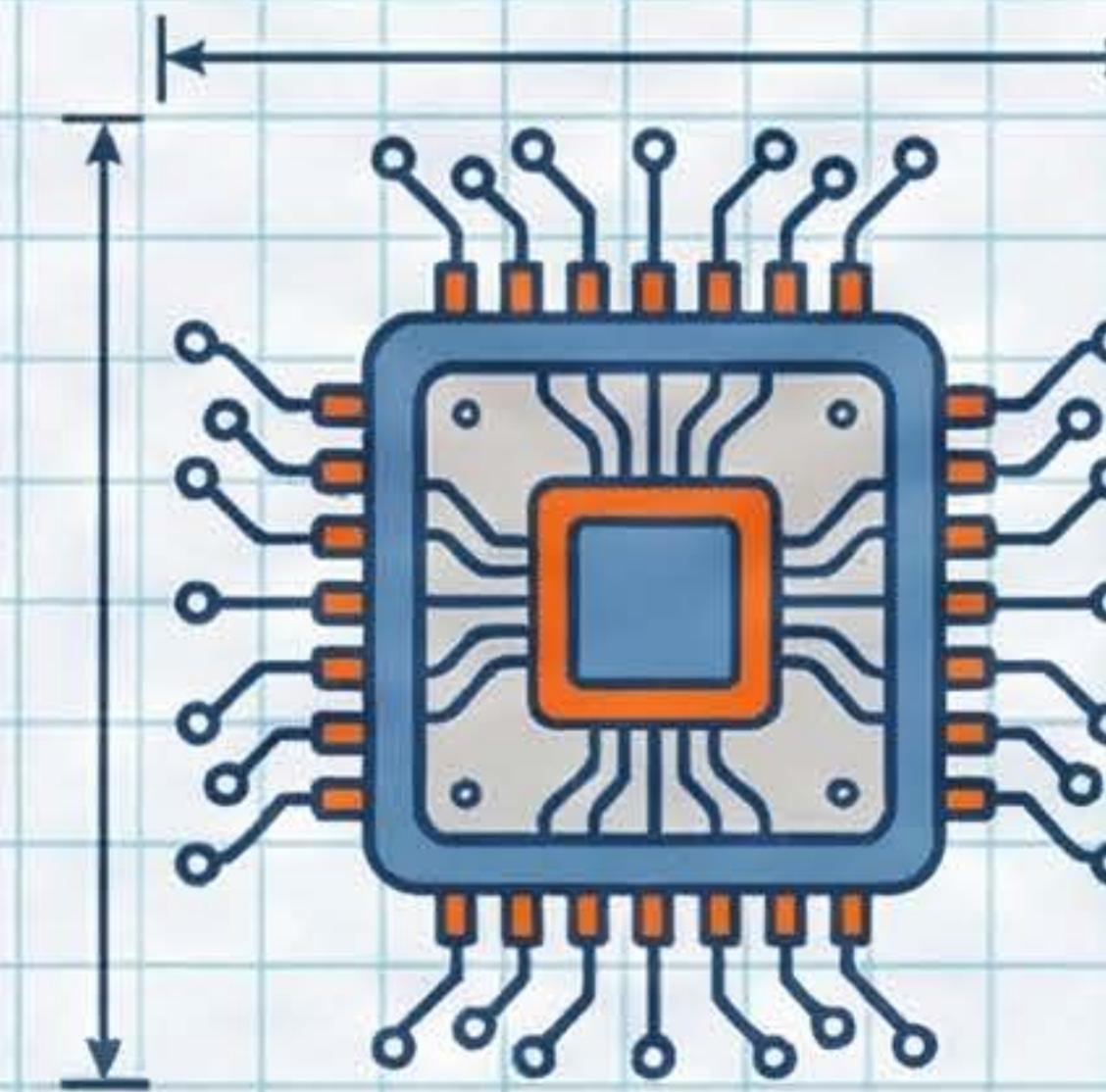
### 傳統方法 (Traditional)

- 慢速 (Slow)
- 需寫迴圈 (Explicit Loops)
- 記憶體分散

PARADIGM SHIFT  
(典範轉移)



## NumPy Array (陣列)



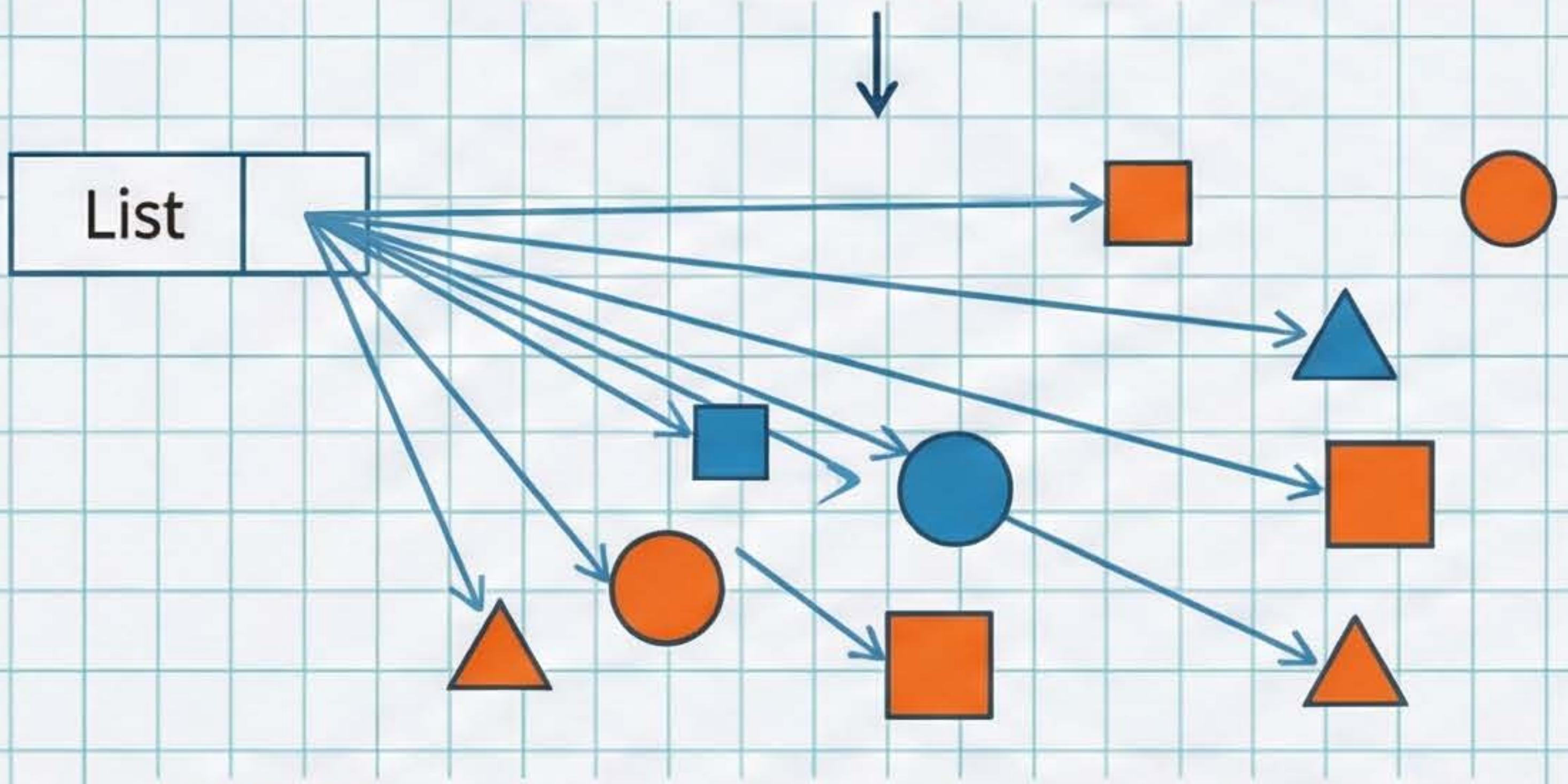
### AI/ML 方法 (AI/ML)

- 高效能 (10-100x Faster)
- C 語言核心
- 向量化並行運算

+ 核心價值：NumPy 是 Pandas 與 Scikit-learn  
+ 的底層架構，掌握它才能駕馭高階 AI 工具。

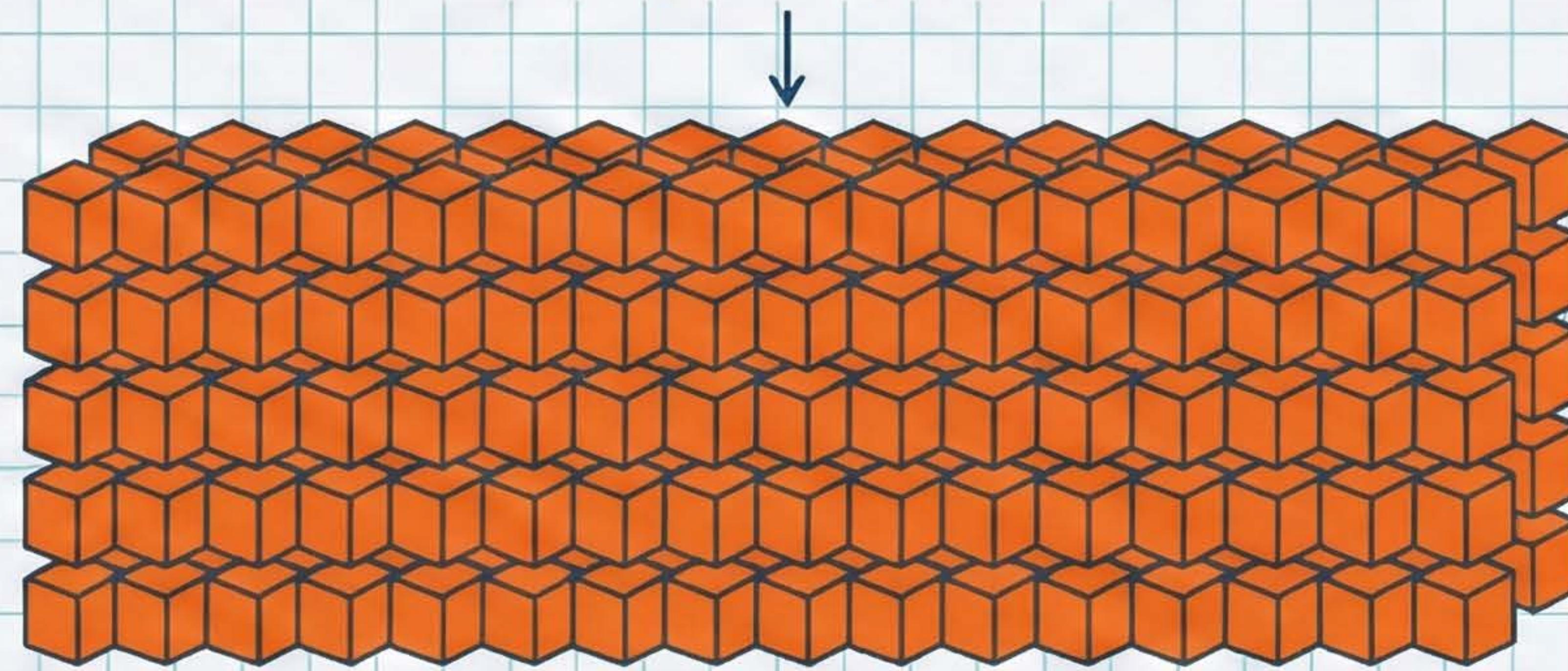
# 解構核心：ndarray vs. List

Python List (一般列表)



**分散儲存** (Scattered Pointers) - 像堆滿雜物的倉庫

NumPy Array (陣列)



**連續記憶體配置** (Contiguous Memory) - 像輸送相同流體的管線

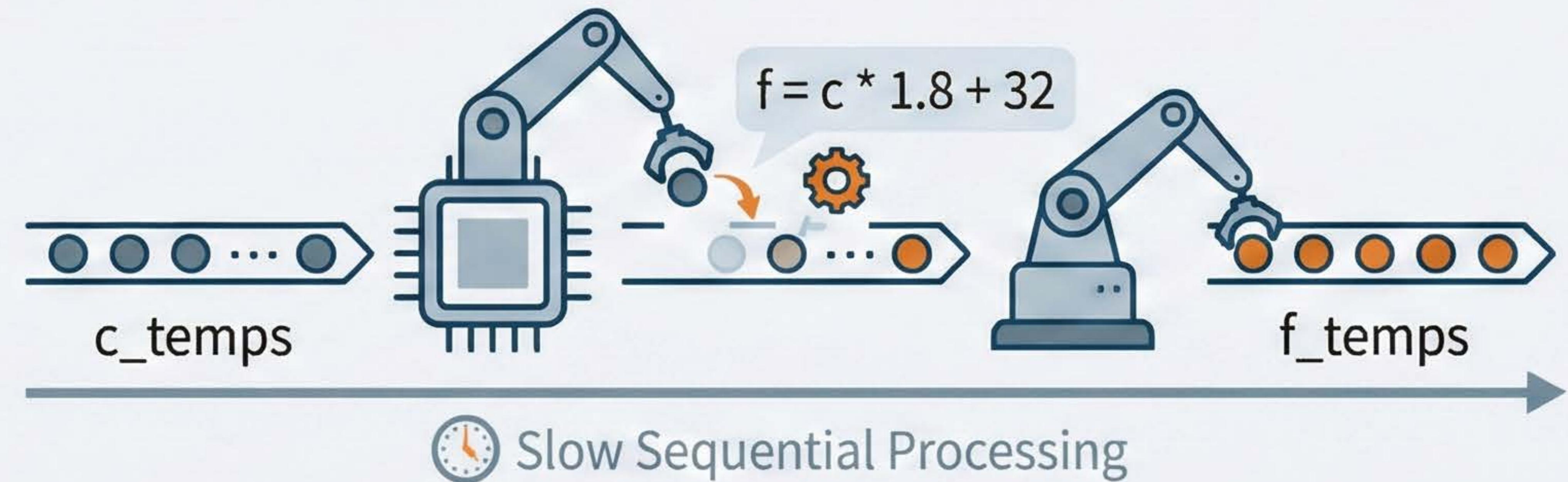
特性	Python List	NumPy Array
同質性 (Homogeneity)	可混雜不同型態 (Any Type)	所有元素型態必須相同 (e.g., float64)
記憶體 (Memory)	<b>指標分散</b> (Overhead high)	<b>連續配置</b> (CPU Cache Friendly)

# 向量化運算 (Vectorization)：告別迴圈

情境：溫度轉換 (攝氏 → 華氏)

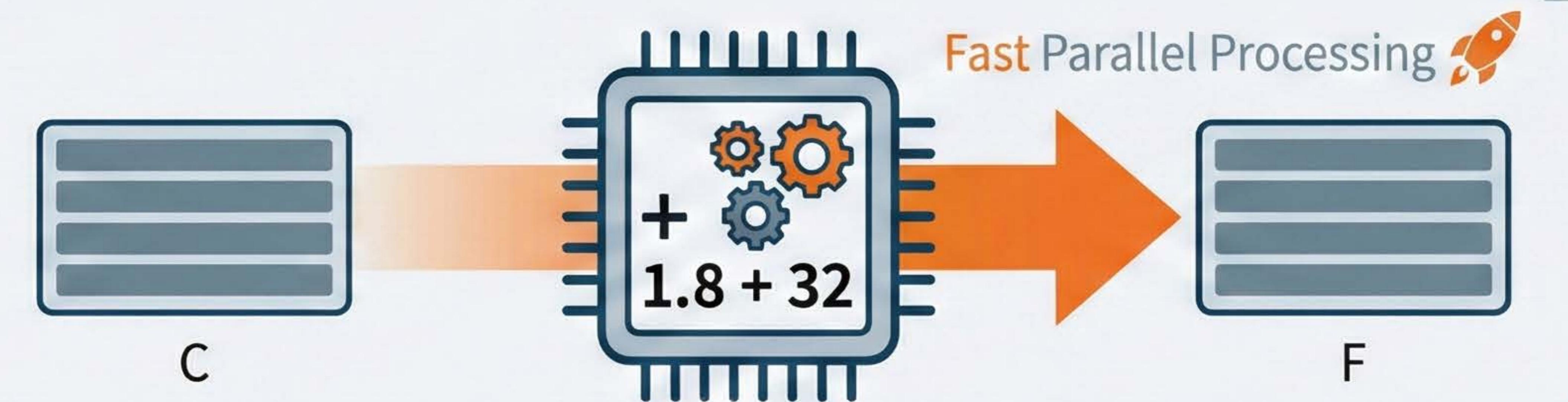
## Python Explicit Loop

```
f_temps = []
for c in c_temps:
    f = c * 1.8 + 32
    f_temps.append(f)
```



## NumPy Vectorization

```
F = C * 1.8 + 32
```



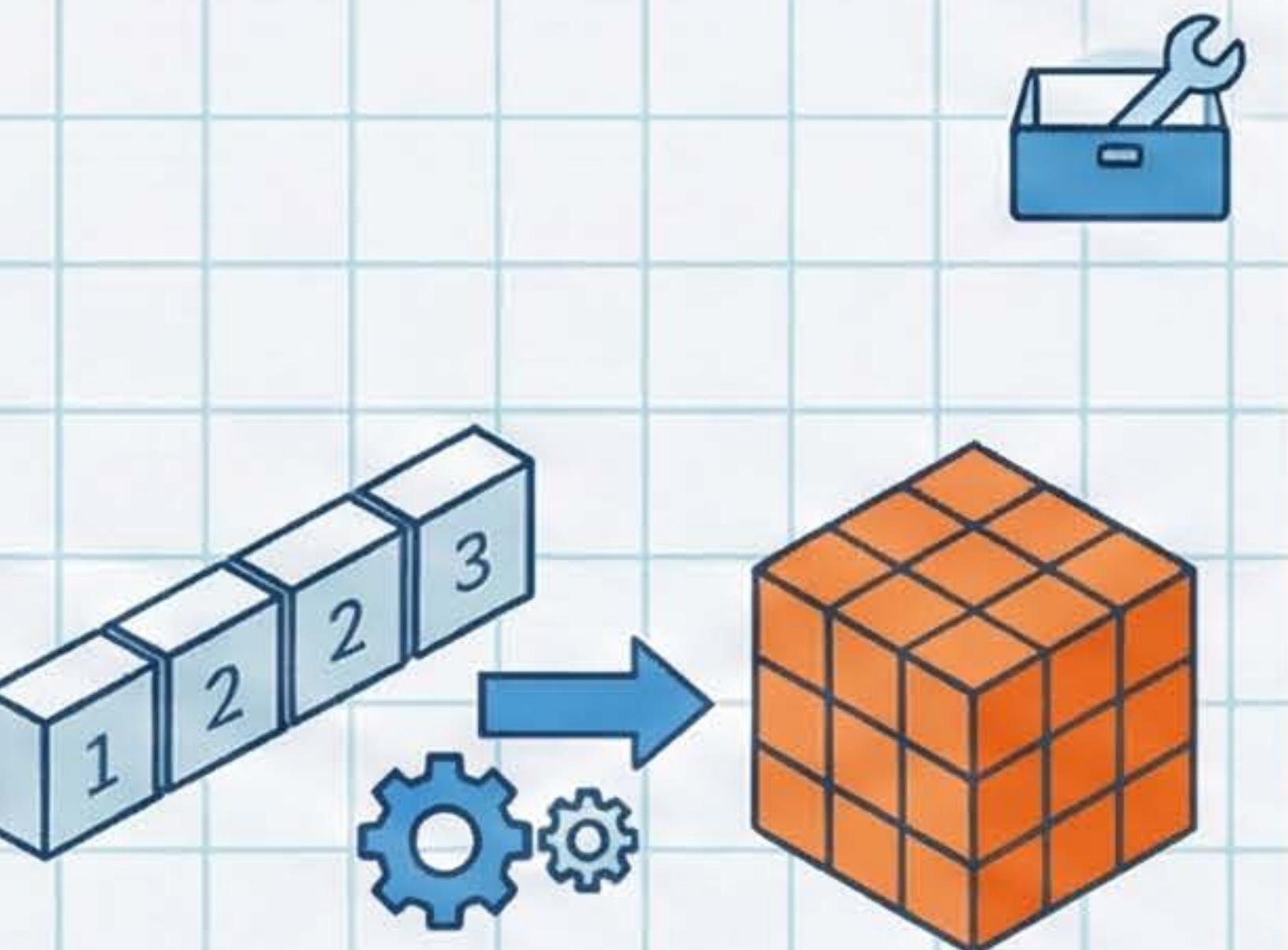
💡 陣列層級的運算，無需明確撰寫迴圈，程式碼更簡潔且高效。

# 數據藍圖：陣列的建立與初始化

## 工具 1: 原始材料

```
np.array([1, 2, 3])
```

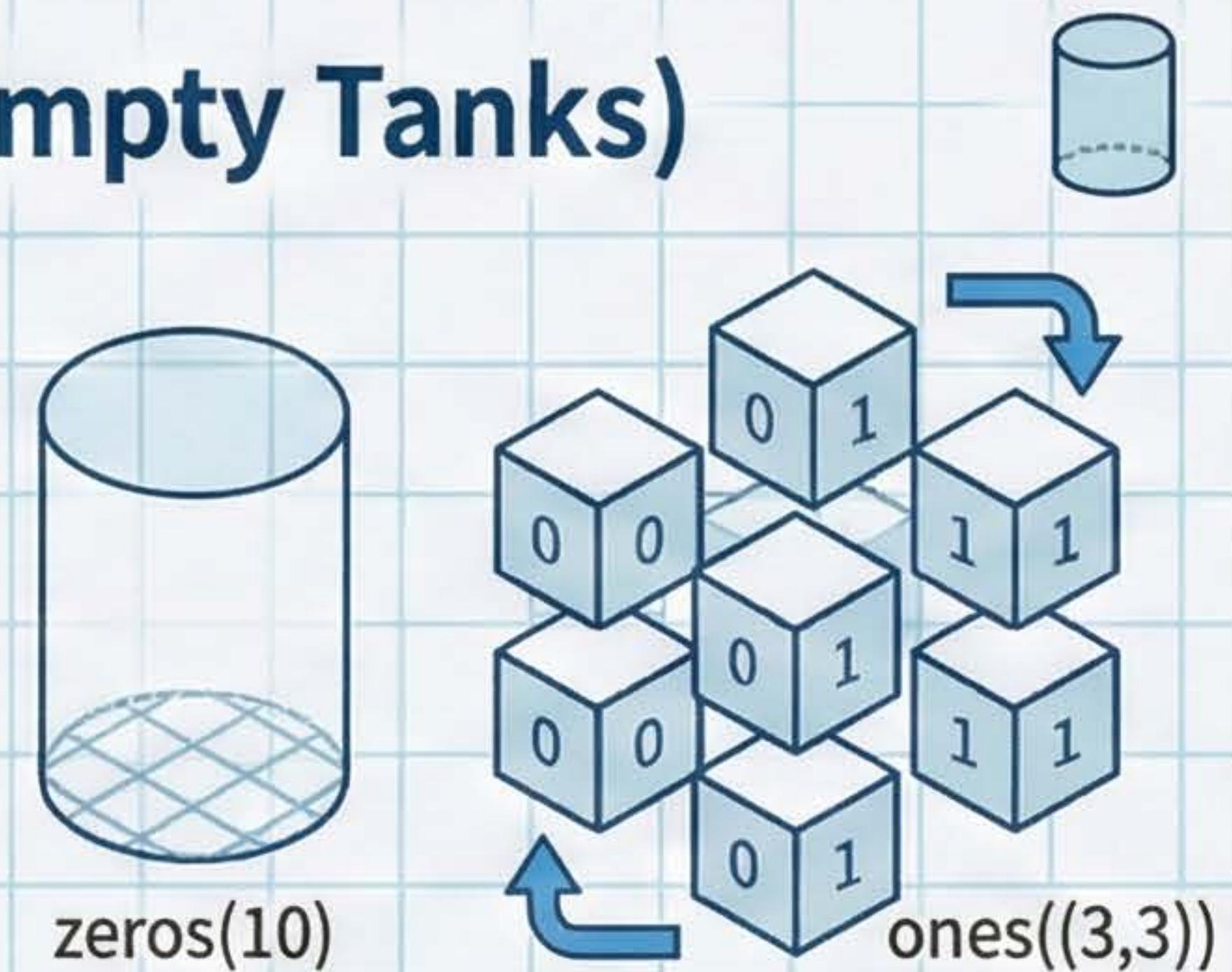
將 Python List 轉換為 Array。



## 工具 2: 初始狀態 (Empty Tanks)

```
np.zeros(10) /  
np.ones((3,3))
```

建立全零或全一的陣列，作為數據容器。

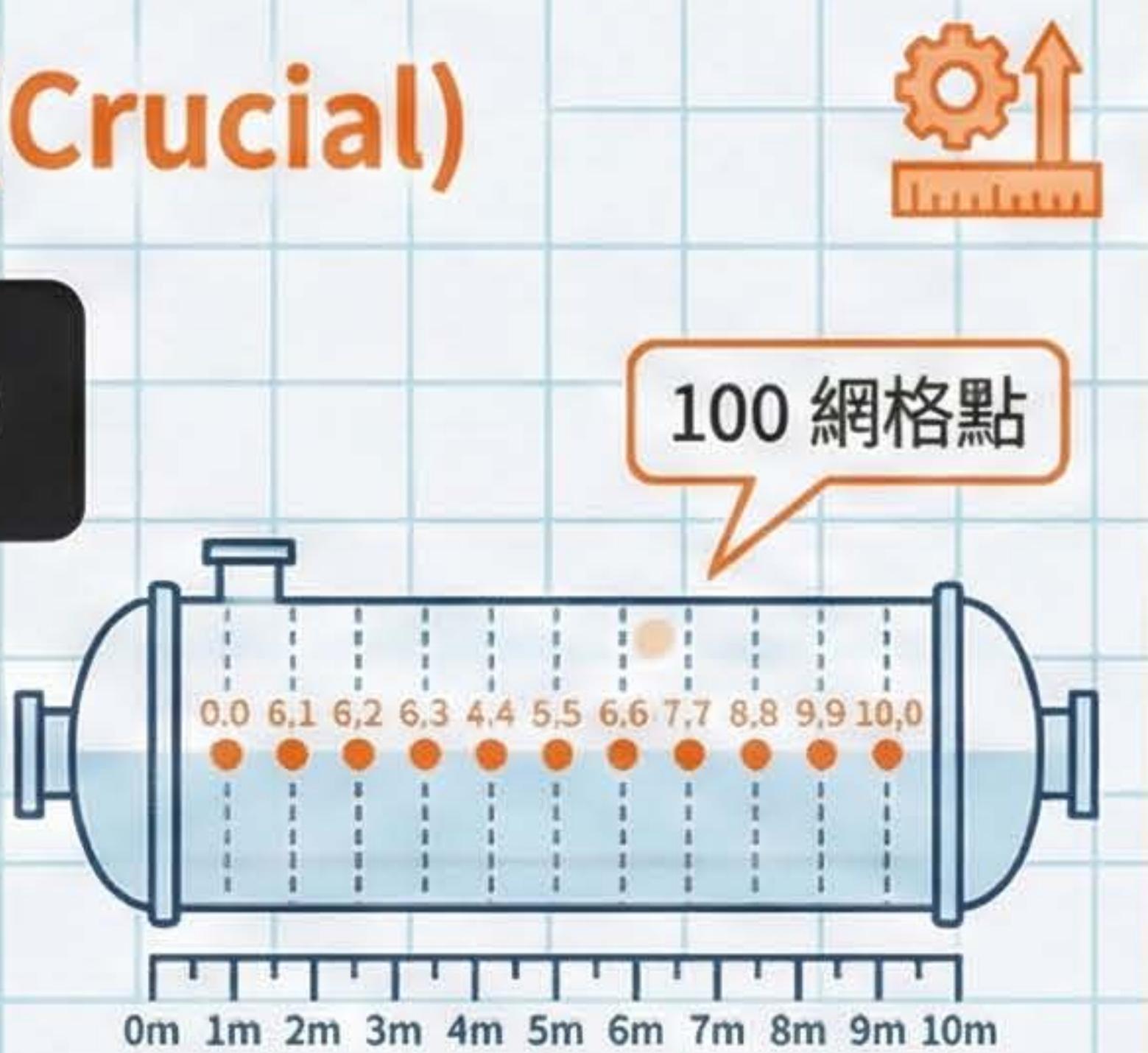


## 工具 3: 數值模擬關鍵 (Crucial)

```
np.linspace(0, 10, 100)
```

在區間內產生均勻分佈的點。

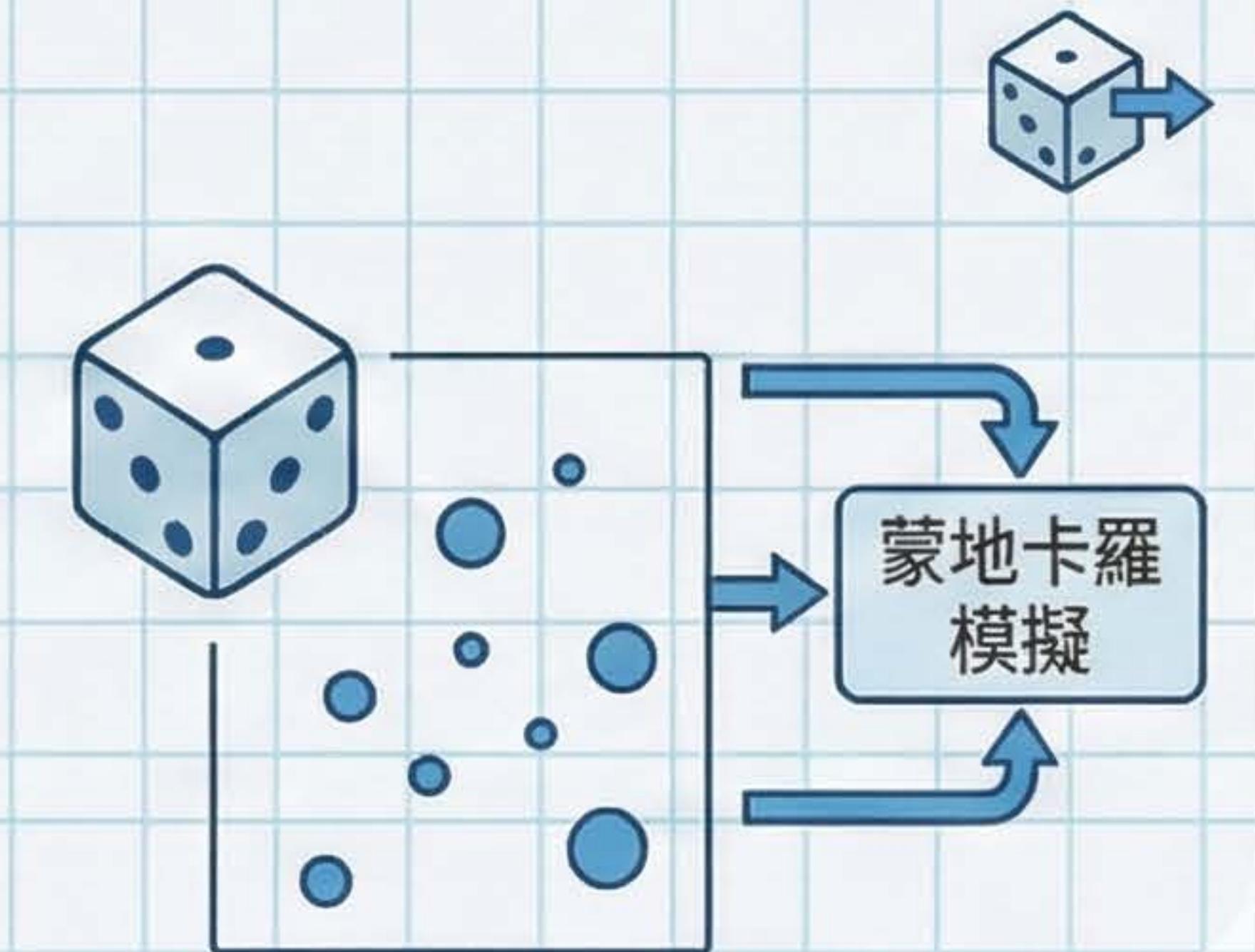
**應用：**將反應器長度 0-10m  
切分為 100 個網格點。



## 工具 4: 隨機模擬

```
np.random.rand(5)
```

生成隨機數據，用於蒙地卡羅模擬 (Monte Carlo)。



# 精準控制：索引 (Indexing) 與切片 (Slicing)

反應器壁感測器矩陣 (Sensor Matrix)				
45.2	51.0	48.3	49.7	52.1
47.5	50.2	46.8	48.9	53.4
49.1	54.7	50.5	47.2	55.6
50.8	46.4	49.3	51.9	48.7
46.1	52.8	47.6	49.5	50.3

## 切片 (Slicing)



`data[0:2, :]`

提取前兩排感測器的數據 (提取特定區域)。

## 布林索引 (Boolean Indexing)



`data[data > 50]`

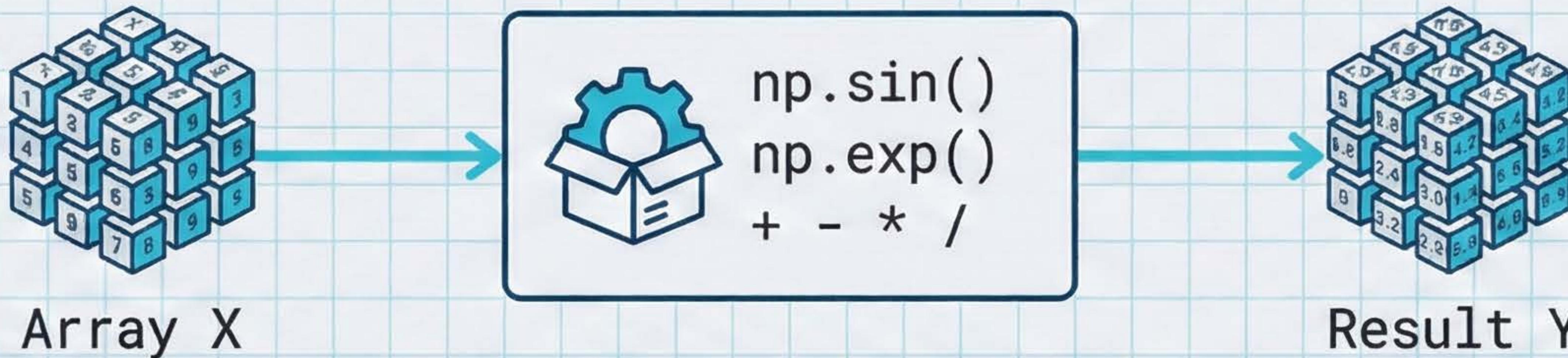
篩選壓力  $> 50$  bar 的異常感測器。

## \*\*花式索引 (Fancy Indexing):\*\*

使用整數陣列提取特定順序的資料，例如

`data[[1, 3, 4]]`。

# 數學運算：通用函式 (Universal Functions)



## 元素級運算 (Element-wise)

基本運算: `+`, `-`, `*`, `/`

科學運算: `np.sin()`, `np.exp()`  
**(Arrhenius Equation 必備)**

比較運算: `>`, `<`, `==`

## 聚合統計 (Aggregations)

`np.mean()`: 計算平均產率

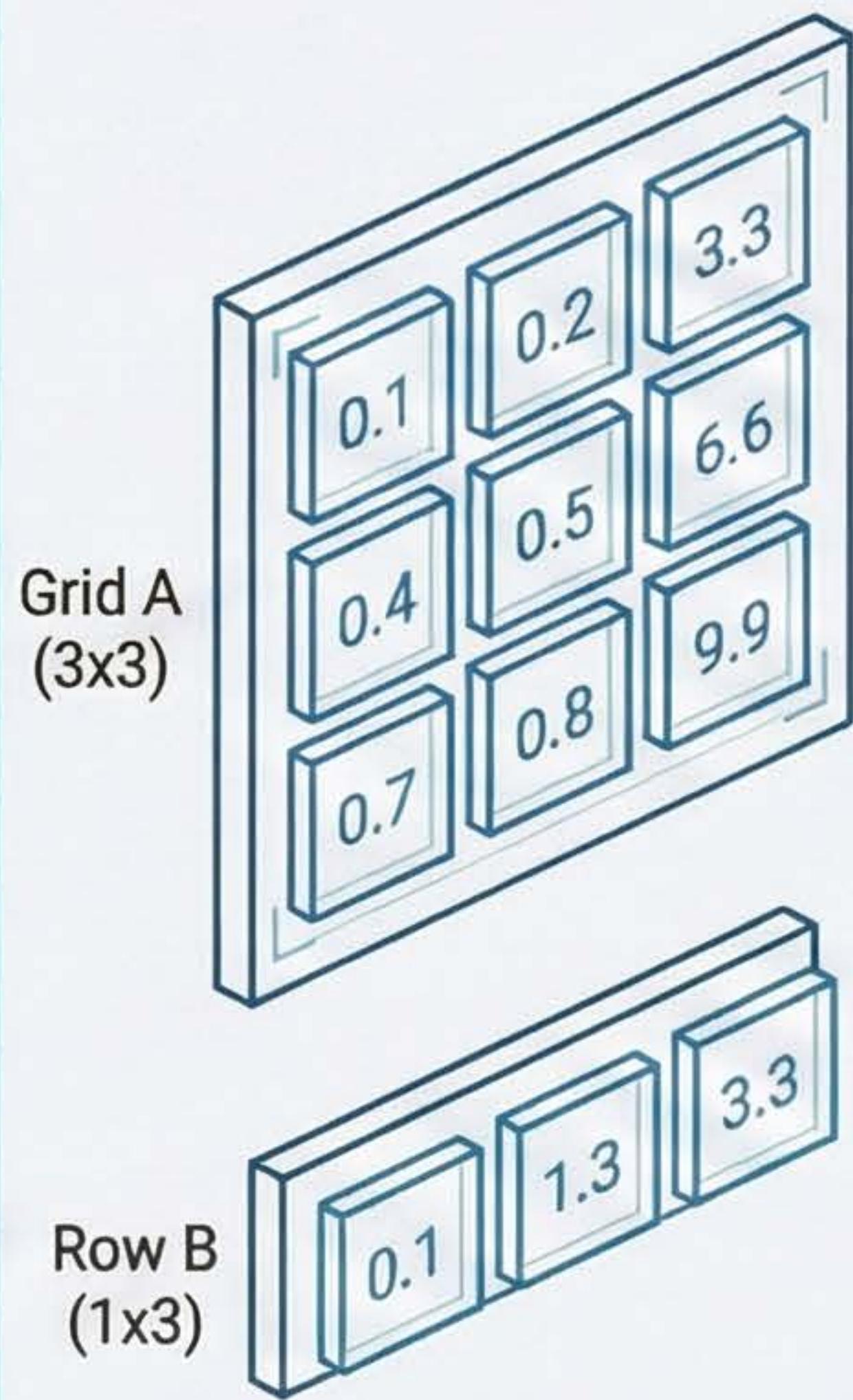
`np.std()`: 計算製程雜訊 (Noise)  
標準差

`np.max()` / `np.min()`: 監控峰值溫度

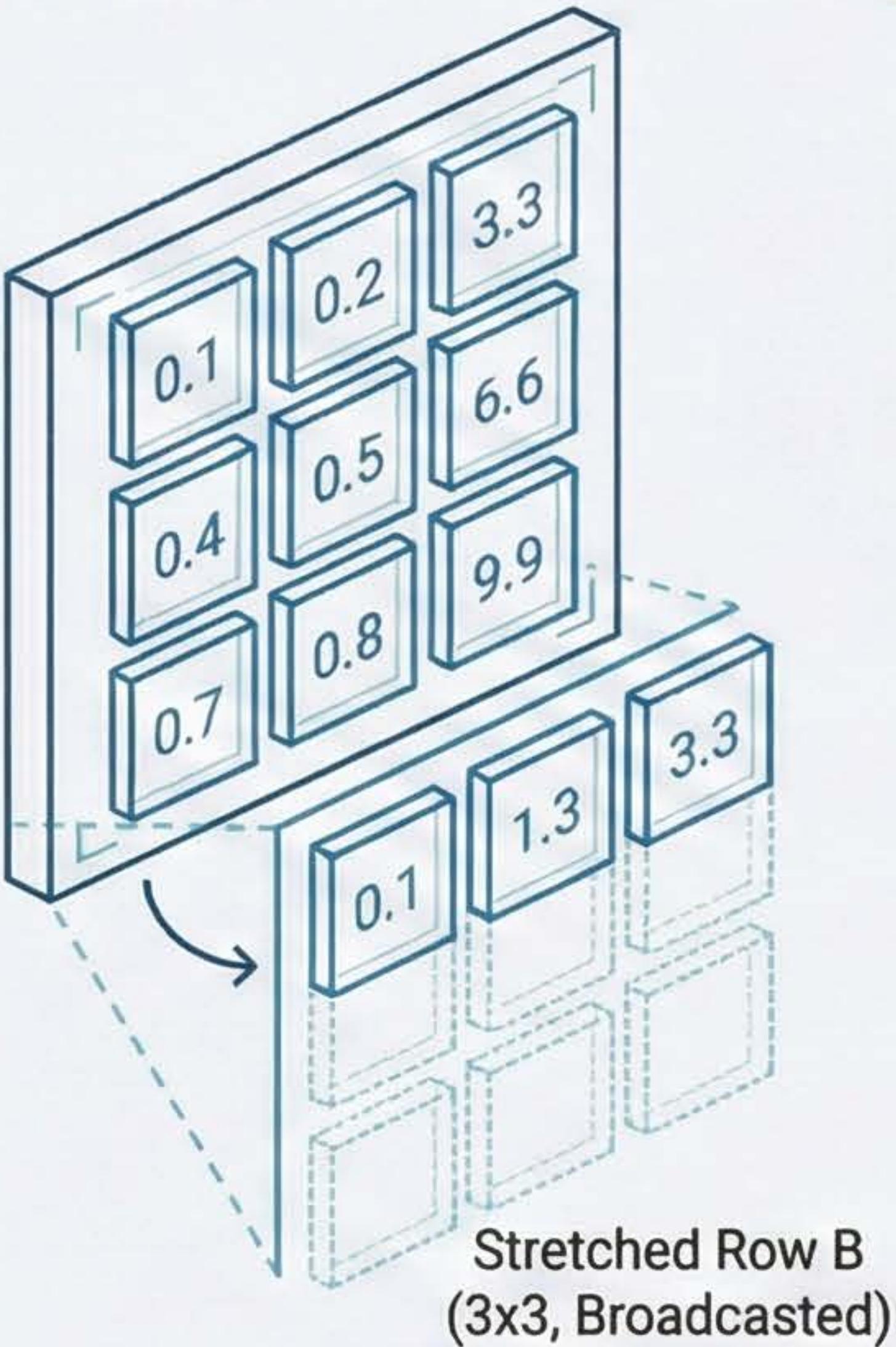
這些函式經過高度優化，速度遠快於 Python 內建 `math` 函式庫。

# 隱藏的超能力：廣播機制 (Broadcasting)

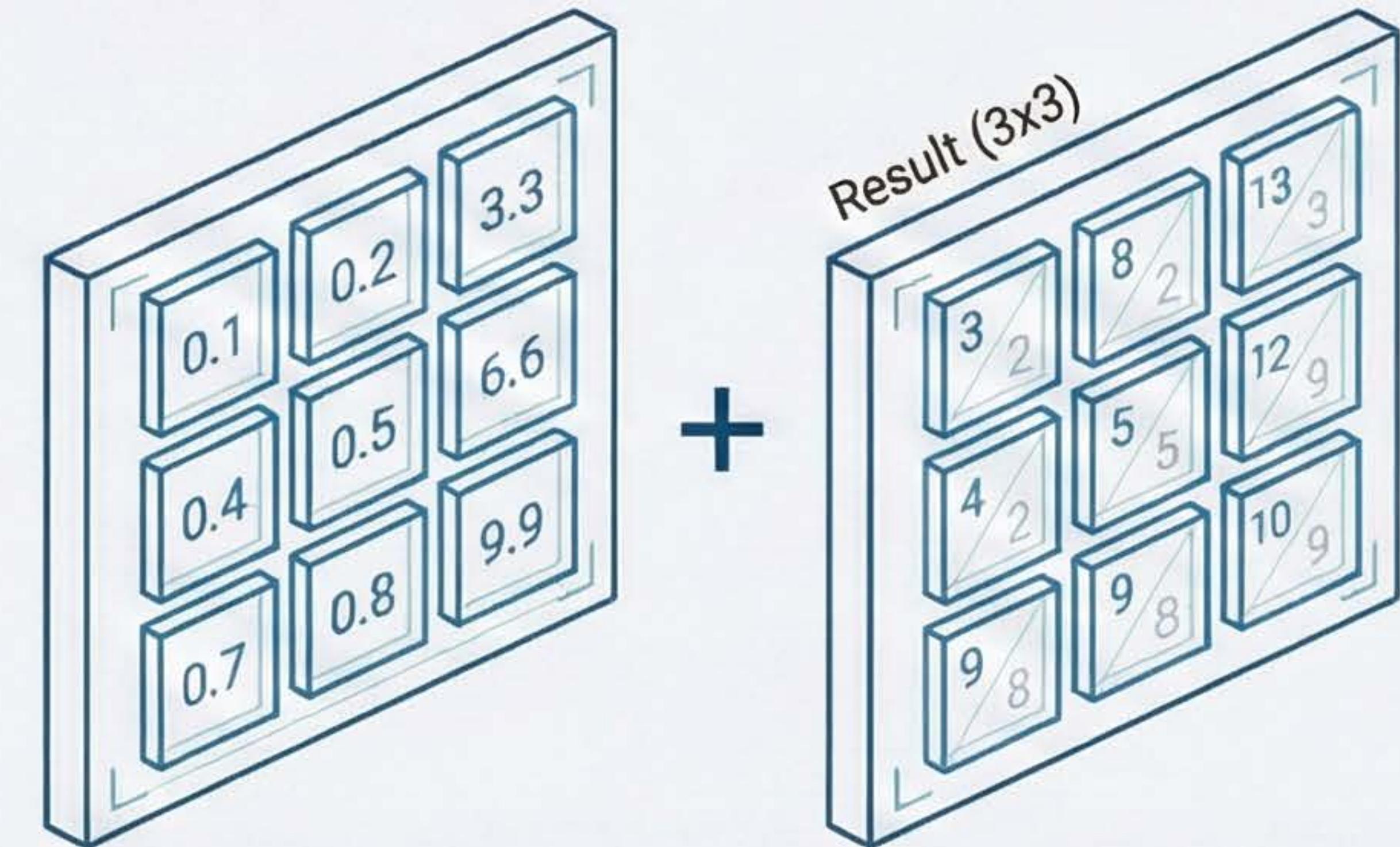
原始數據 (The Problem)



廣播過程 (The Broadcasting)



運算結果 (The Result)



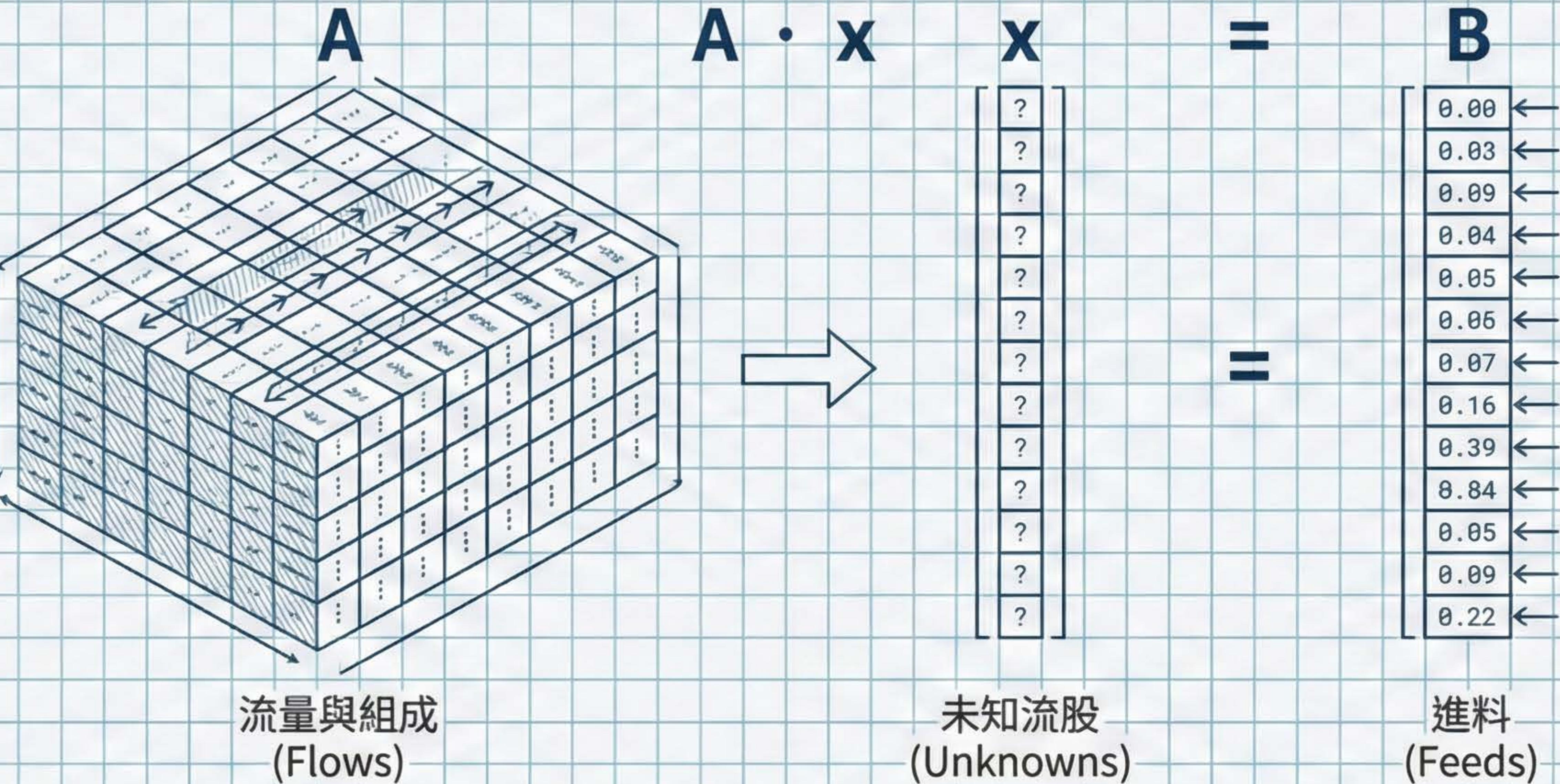
## 廣播規則 (Broadcasting Rules)

1. 維度靠右對齊
2. 維度為 1 的部分自動延伸

## 化學工程應用場景 (ChemE Example)

應用場景：`Matrix (Sensors)` - `Vector (Ambient\_Temp)`  
將反應器所有位置的溫度讀數 (矩陣) 同時減去環境溫度，進行修正。

# 線性代數：工程問題的求解器



## Code Snippets:

矩陣乘法：

```
np.dot(A, B) or A @ B
```

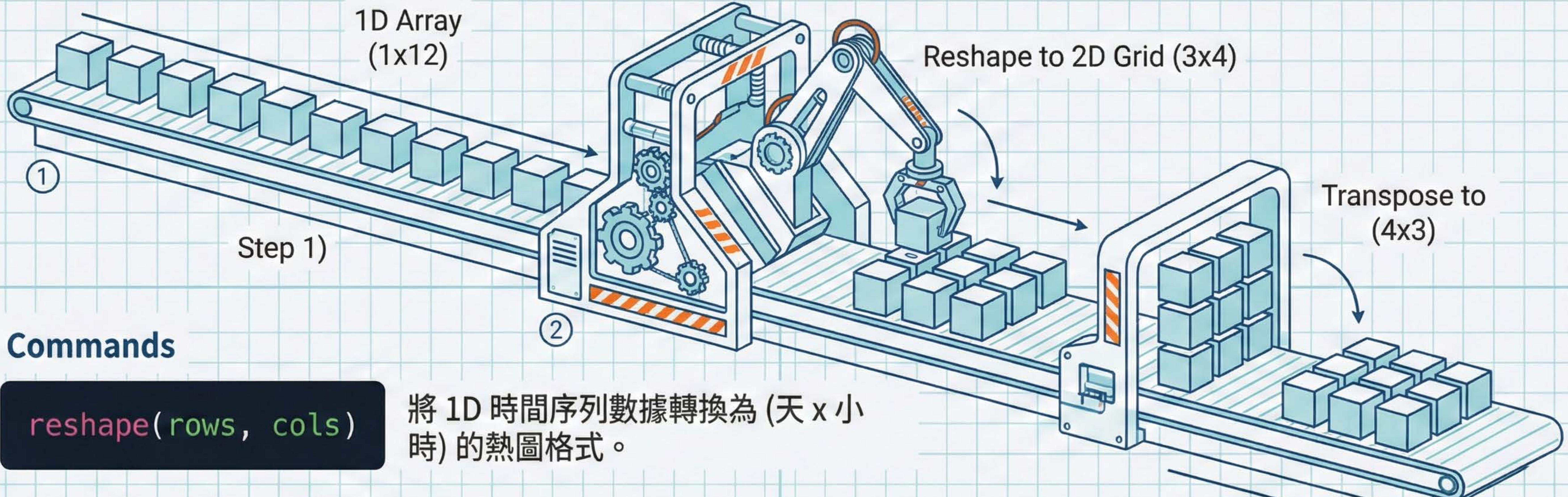
求解方程組：

```
np.linalg.solve(A, B)
```

## 化工應用：質量平衡 (Mass Balance)

直接求解複雜化工製程中的多股流體平衡問題，無需手動迭代。

# 形狀操作：資料的重塑與變形



## Commands

`reshape(rows, cols)`

`flatten() / ravel()`

`transpose() / .T`

將 1D 時間序列數據轉換為 (天 x 小時) 的熱圖格式。

將多維數據攤平為一維 (為了輸入某些機器學習模型)。

矩陣轉置 (交換列與行)。

**Insight:\*\*** 準備數據以符合模型輸入要求的必要步驟。

# 實務應用：反應動力學模擬

## Problem Statement:

模擬一階反應 (First-order reaction) 濃度隨時間變化。

方程式： $C(t) = C_0 \cdot e^{-kt}$

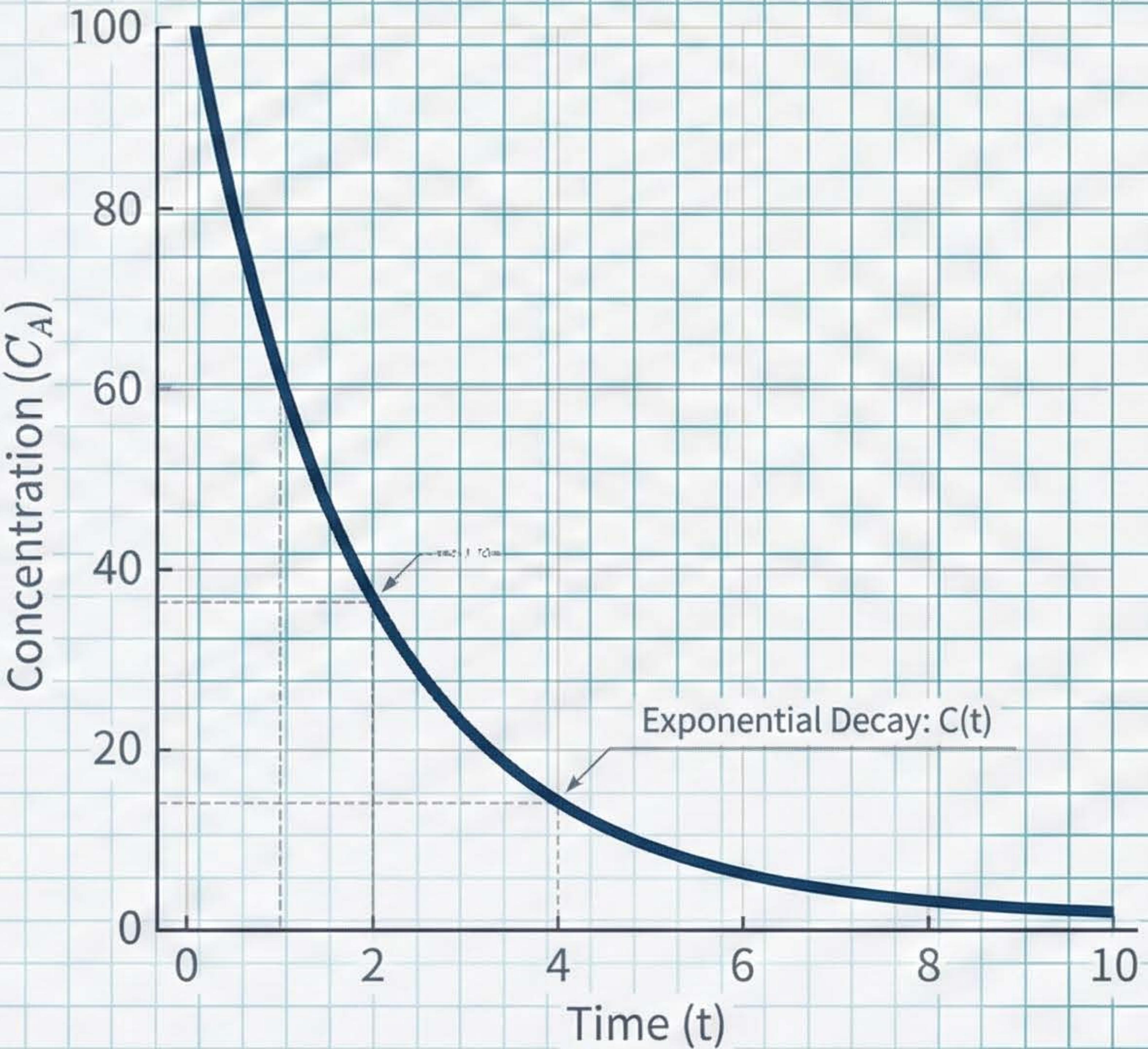
## NumPy Solution

```
# 1. 定義時間切片 (Time Slicing)
t = np.linspace(0, 10, 100)

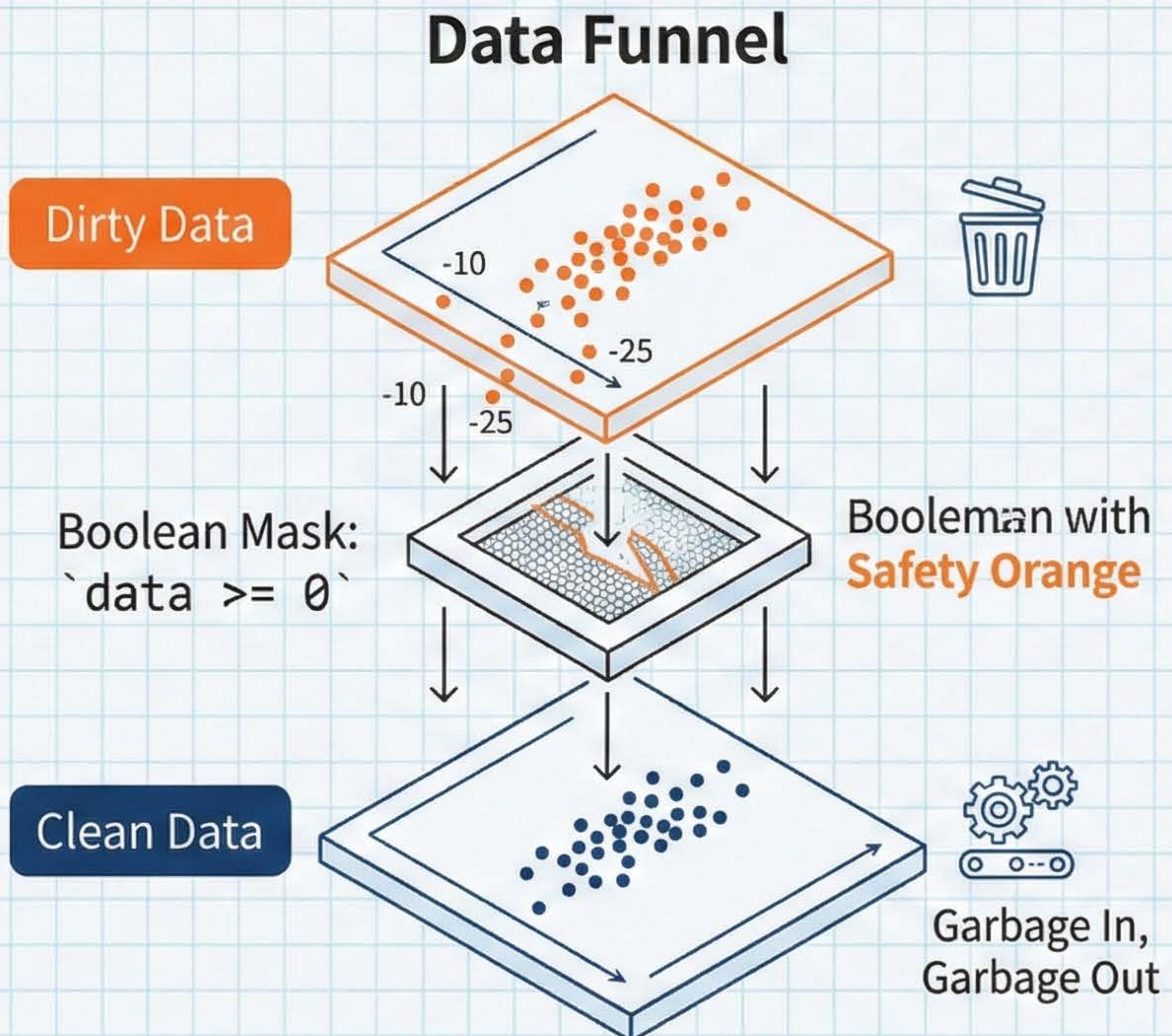
# 2. 向量化計算濃度 (Vectorized Calculation)
k = 0.5
C0 = 100
C = C0 * np.exp(-k * t)
```

### Value Add

無需迴圈，一行程式碼即可算出整條反應曲線，並可立即調整  $k$  值進行參數敏感度分析。



# 數據前處理：清洗與過濾



**Technique:** 使用布林索引去除異常值  
(Outlier Removal)

**Scenario:** 感測器故障導致出現負的絕對溫度或濃度。

```
clean_data = data[data >= 0]
```

- Missing Values : 處理 `NaN` (缺失值) :  
`np.isnan(data)`` 檢測並填補。

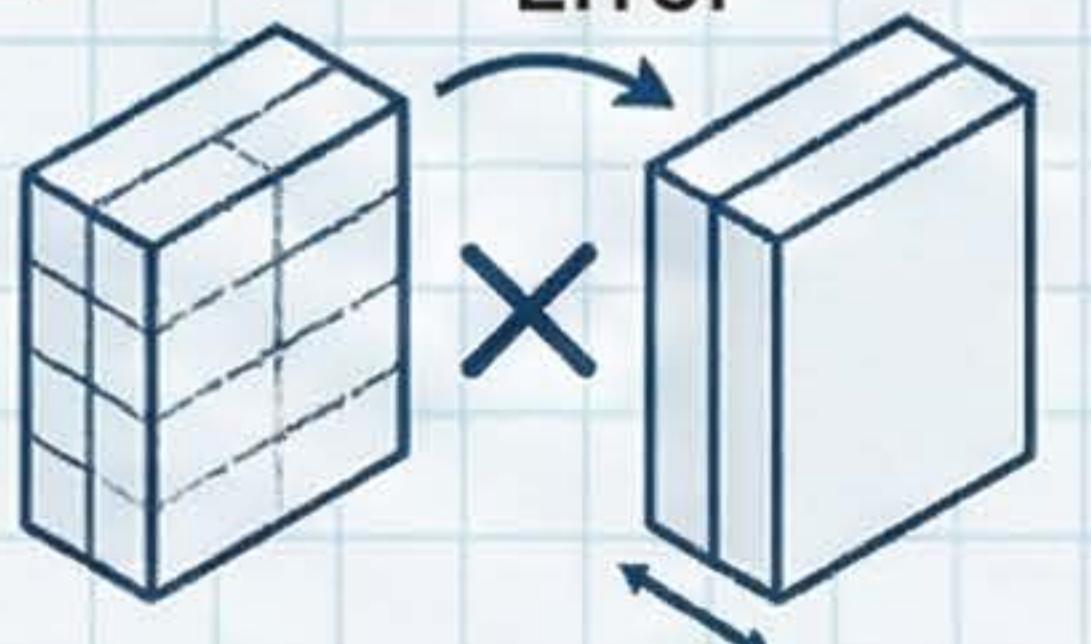
這是進入 Machine Learning 模型訓練前的關鍵步驟 (Pre-processing)。

# 最佳實踐與常見陷阱

## ⚠ Pitfalls (陷阱)



Broadcasting  
Error



### 形狀不匹配 (Shape Mismatch)

廣播錯誤是 #1 常見錯誤。

解法：善用 `shape` 檢查維度。

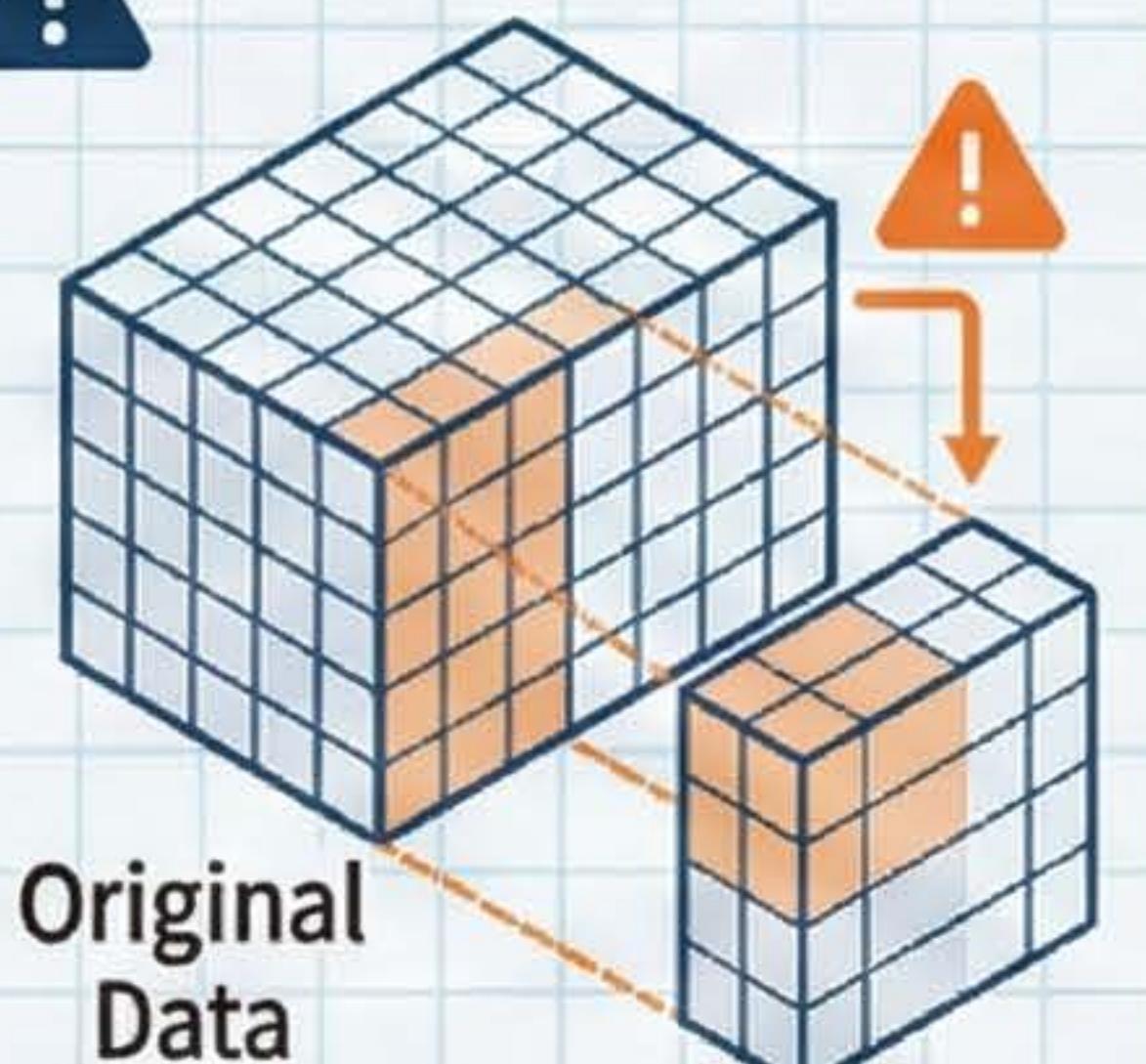
```
data.shape - (100, 5)  
(10, )
```



### 複製 vs. 視圖 (Copy vs. View)

切片通常回傳 View (共用記憶體)，修改切片會影響原始資料。

解法：若需獨立資料，請使用 `copy()`。



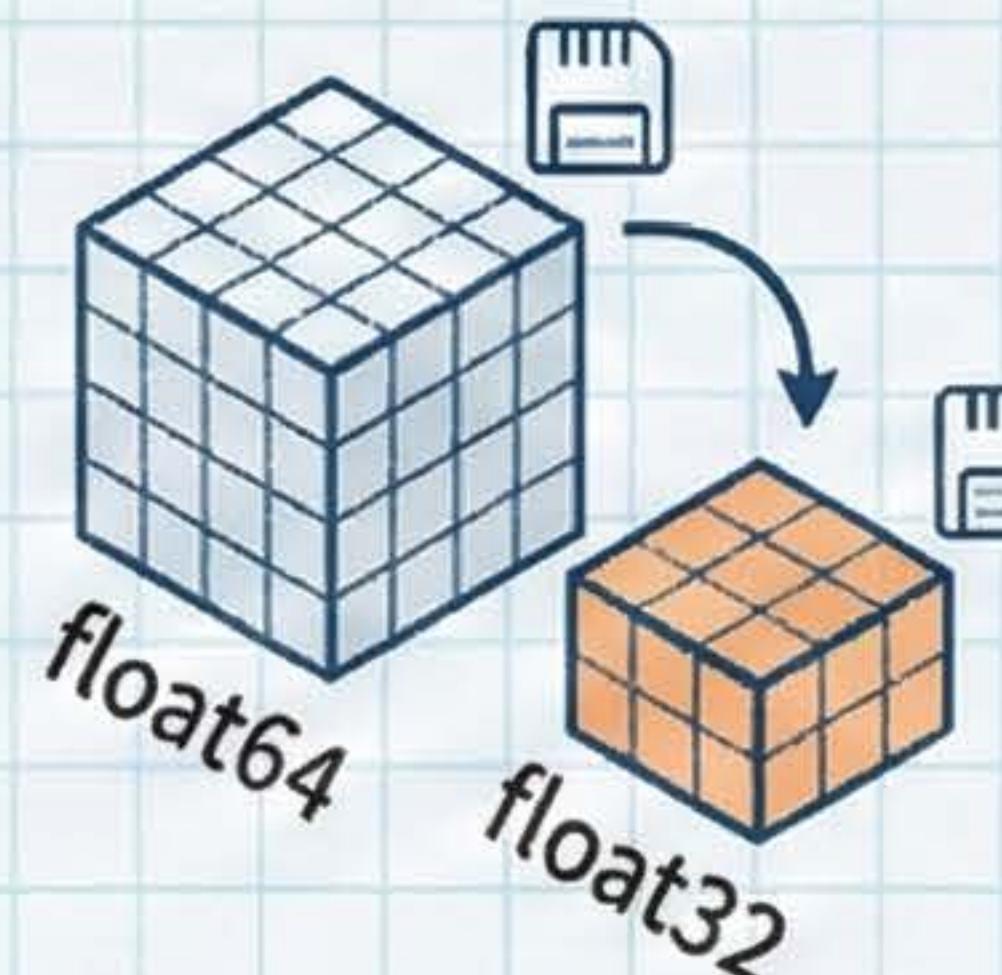
Original  
Data

Slice/View

```
# Problem:  
view = original[0:5]  
view[0] = 999 # Affects original  
  
# Solution:  
independent = original[0:5].copy()  
independent[0] = 999 # Does not affect
```



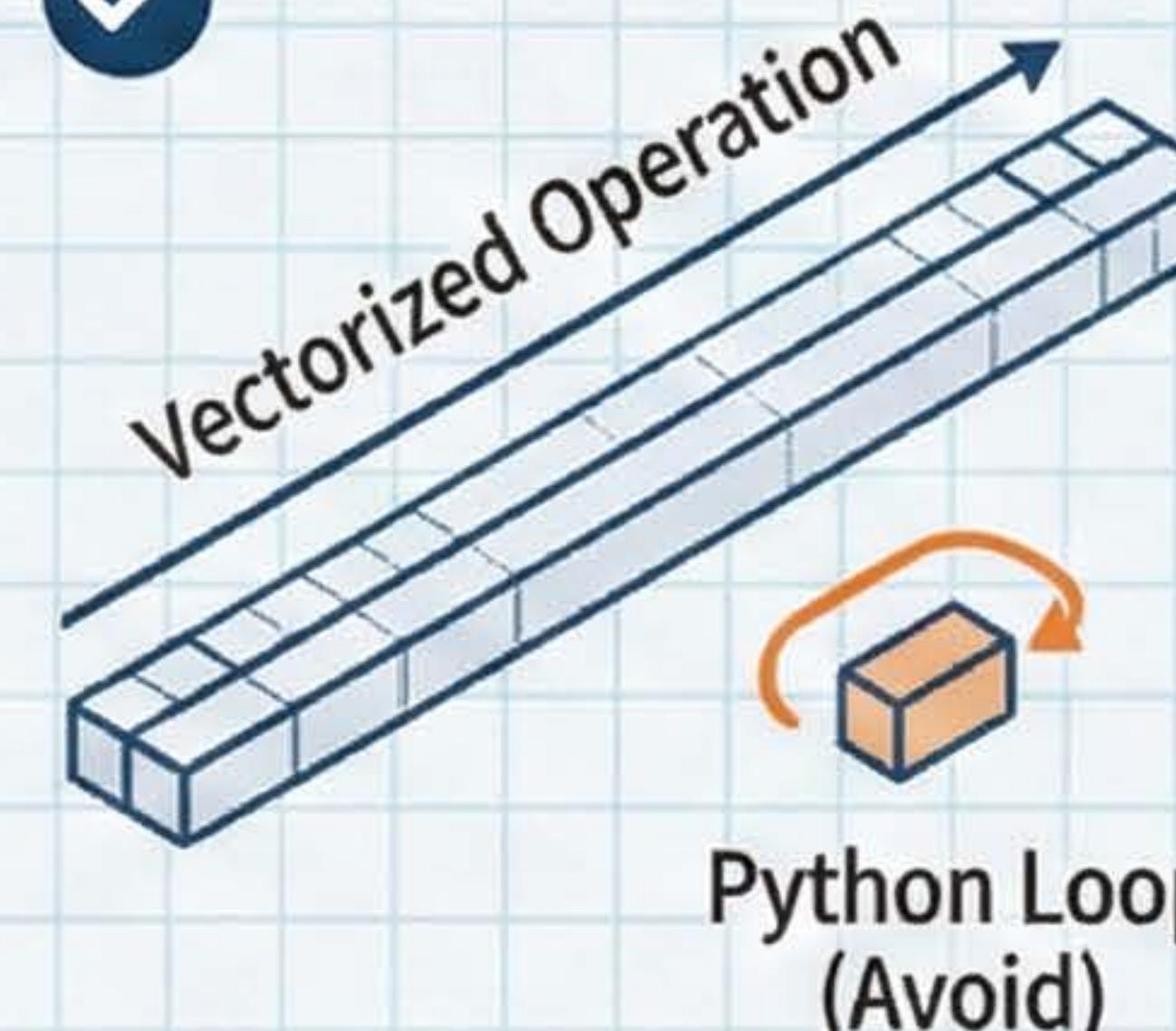
## Pro Tips



### 記憶體優化 (Memory Optimization)

選擇適當的資料型態 (如 `float32` vs `float64`) 以節省記憶體。

```
dtype=np.float32
```



### 效能優先 (Performance Priority)

永遠優先使用向量化，避免在 Array 上使用 Python 迴圈。

```
# Good:  
result = data * 2
```

```
# Bad:  
for i in range(len(data)):  
    result[i] = data[i] * 2
```

# 學習地圖：NumPy 的戰略位置



NumPy 是這條數據流水線的地基。掌握陣列操作，就是掌握了與所有 AI 工具溝通的語言。

# 結語：通往資料科學的第一步

## 工程總結

- ✿ **速度**：C 語言核心帶來的極致效能 (**100x Speedup**)。
- ✿ **思維**：從「迴圈」轉向「**向量化**」思考。
- ✿ **應用**：解決**化工模擬**與數據清洗的強大工具。

### 動手實作 (Exercise) :

試著用 NumPy 解決三元物料平衡方程組 (Exercise 8)。

“ NumPy 不只是工具，它是現代工程師處理數據的直覺本能。”