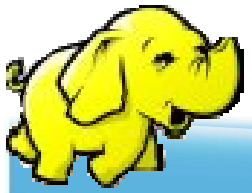
三、Map Reduce 與HDFS

3.A：HDFS 檔案系統
3.B：MapReduce 演算法

三、Map Reduce 與HDFS

3.A：HDFS - 檔案系統

HDFS = Hadoop Distributed File System，是Hadoop用來存放資料的分散式檔案系統，因此若要讓Hadoop運算資料，就要將待運算的資料放到這個空間才可以；同理可知運算後的資料...。

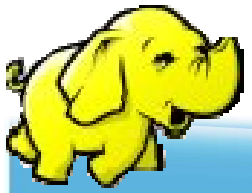


HDFS: 開發動機

- 基於 Google File System (GFS)
- 用便宜而不可靠的電腦，打造互為備援的儲存媒介，拿來存放海量資料。
- 為何不使用現存的檔案系統呢?
  - 不同的工作負載與設計優先權
  - 須處理比其他檔案系統還要更大的資料集

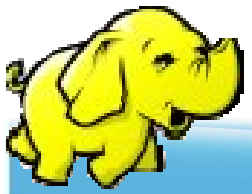
基本假設

- 高單元故障率
  - 便宜的個人電腦商品總是容易故障
- 海量檔案的『客觀』參考數字
  - 就只是幾百萬個檔案而已...
  - 每個大小約100MB或更大; 通常以數GB的檔案最常見
- 這些檔案都只寫一次(write-once)，多數是附加(append)
- 大量的串流讀取需求
- 持續而大量的資料通量(throughput)遠比較短的反應延遲(latency)來得重要



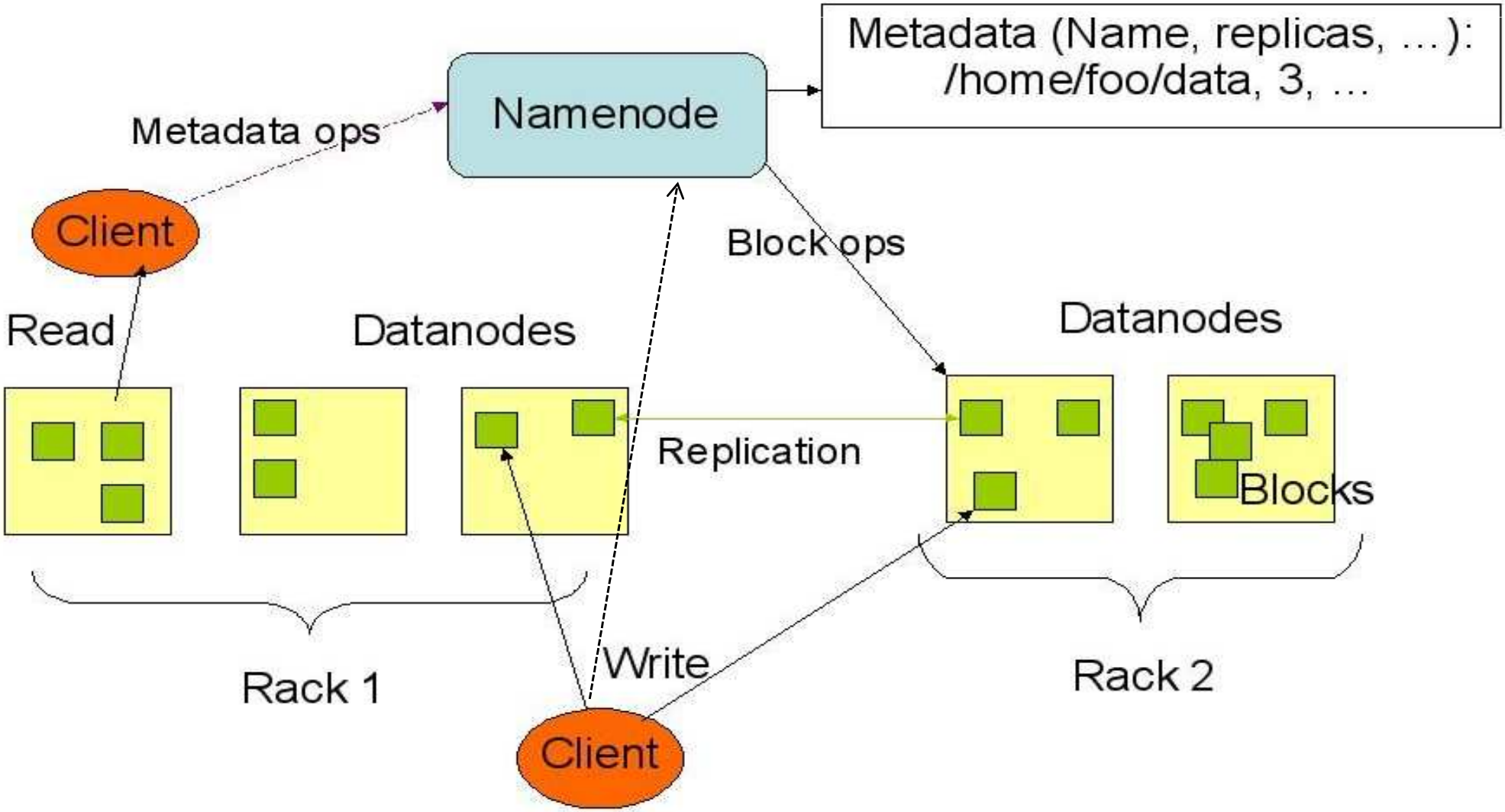
HDFS 設計準則

- 檔案以區塊(block)方式儲存
  - 每個區塊大小遠比多數檔案系統都來得大(預設值為64MB)
- 透過複本機制來提高可靠度
  - 每個區塊至少備分到三台以上的DataNode
- 單一master (NameNode) 來協調存取及屬性資料(metadata)
  - 簡易的集中控管機制
- 沒有資料快取機制(No data caching)
  - 快取對於大資料集與串流讀取沒太大幫助
- 熟悉的介面，但客制化的API
  - 簡化問題；專注於分散式應用



管理資料

HDFS Architecture

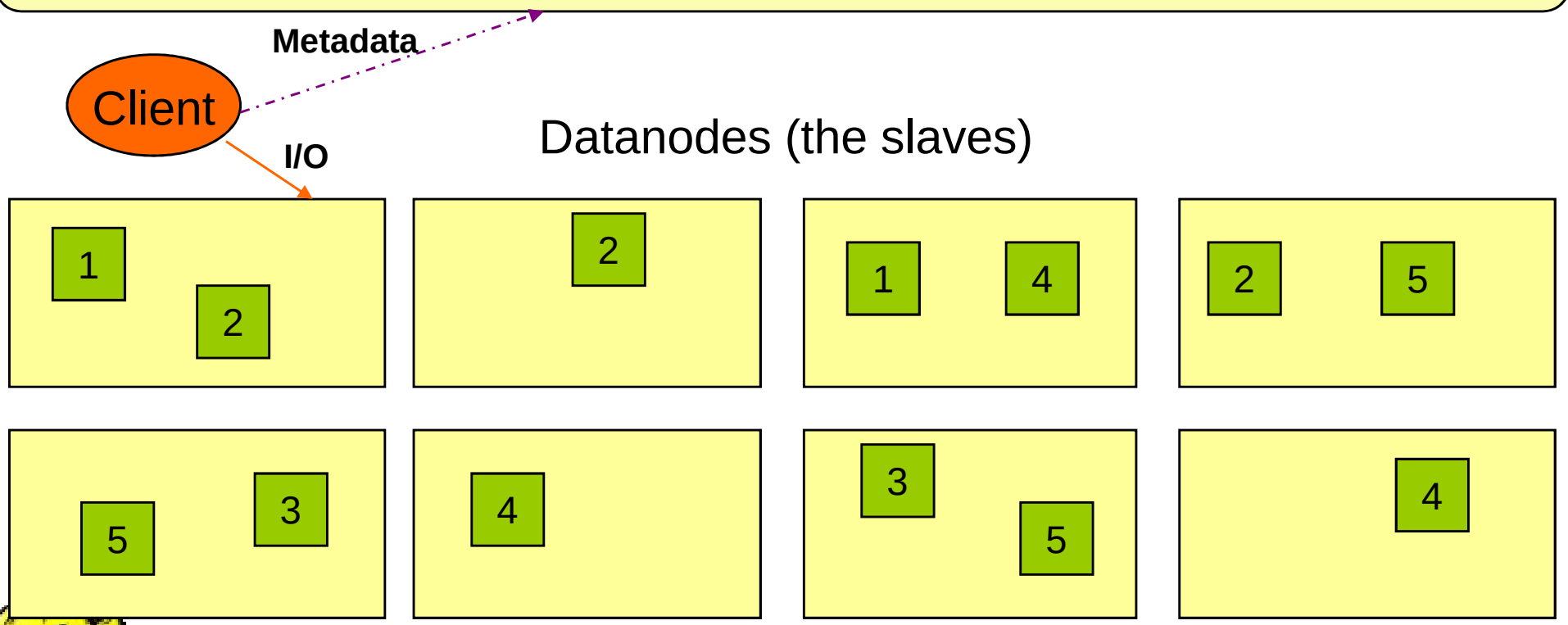


HDFS 運作

Namenode (the master)

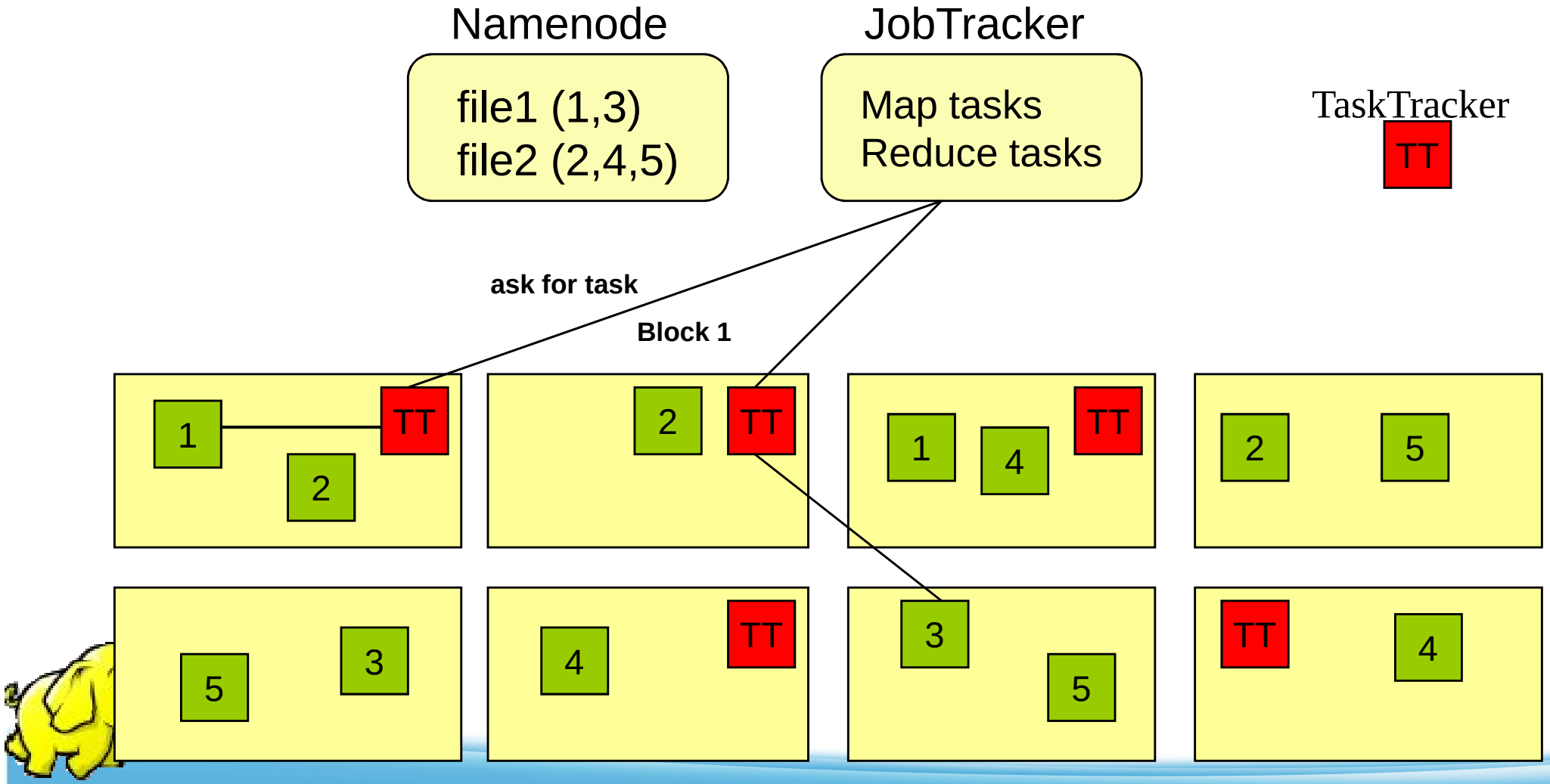
檔案路徑－ 副本數， 由哪幾個block組成

name:/users/joeYahoo/myFile - copies:2, blocks:{1,3}
name:/users/bobYahoo/someData.gzip, copies:3, blocks:{2,4,5}

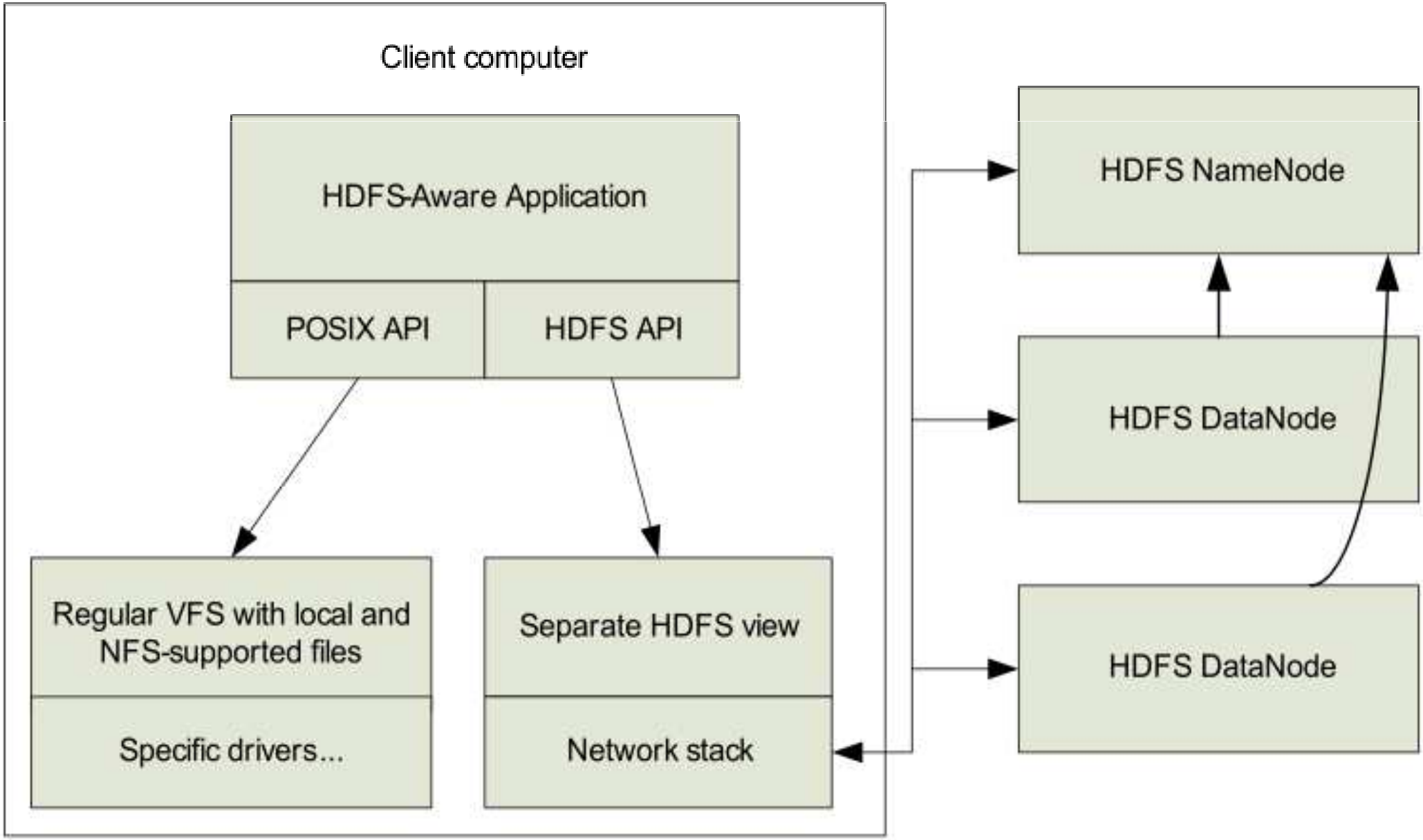


HDFS 運作

- 目的：提高系統的可靠性與讀取的效率
  - 可靠性：節點失效時讀取副本已維持正常運作
  - 讀取效率：分散讀取流量（但增加寫入時效能瓶頸）



HDFS 系統流程圖

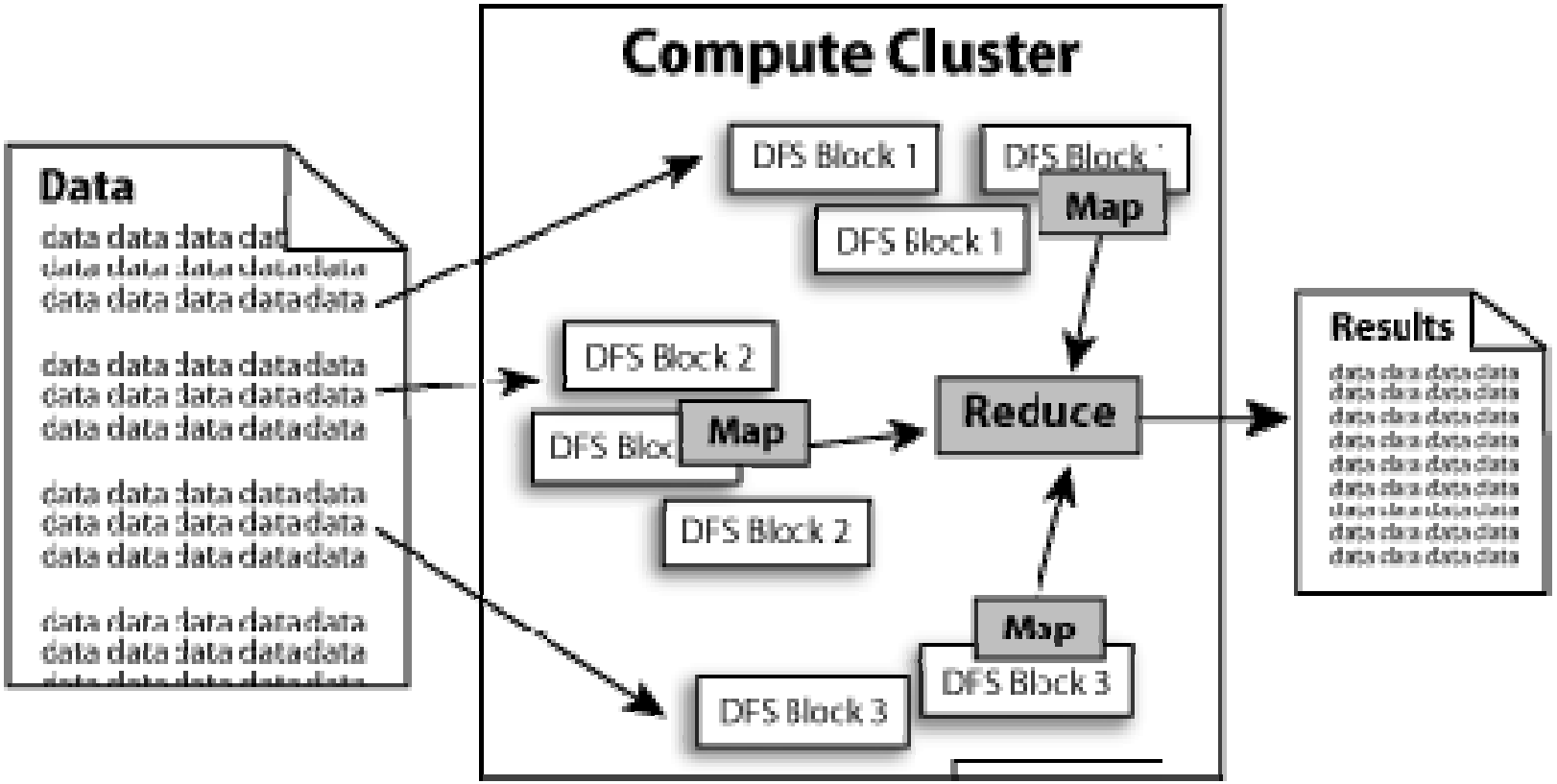

Client computer
HDFS-Aware Application
POSIX API
HDFS API
Regular VFS with local and NFS-supported files
Specific drivers...
Separate HDFS view
Network stack
HDFS NameNode
HDFS DataNode
HDFS DataNode

三、Map Reduce 與HDFS

3.B : MapReduce 演算法

Hadoop 的運算方式是透過Map/Reduce 演算法構成，也意謂著，要透過Hadoop API 撰寫平行分散式程式，則一定要懂 Map / Reduce

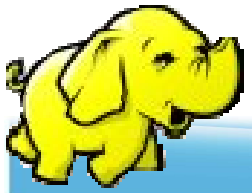
Map / Reduce 定義



MapReduce is a software framework to support distributed computing on large data sets on clusters of computers.

演算法

- Functional Programming : Map Reduce
  - map(...) :
    - [1,2,3,4] – (*2) -> [2,4,6,8]
  - reduce(...):
    - [1,2,3,4] - (sum) -> 10
  - 對應演算法中的Divide and conquer
  - 將問題分解成很多個小問題之後，再做總和



58

Map

- One-to-one Mapper

```
let map(k, v) =
Emit(k.toUpper(),
v.toUpper())
```

("Foo", "other") → ("FOO", "OTHER")
("key2", "data") → ("KEY2", "DATA")

- Explode Mapper

```
let map(k, v) =
foreach char c in v:
  emit(k, c)
```

("A", "cats") → ("A", "c"), ("A", "a"), ("A", "t"), ("A", "s")

- Filter Mapper

```
let map(k, v) =
if (isPrime(v))
then emit(k, v)
```

("foo", 7) → ("foo", 7)
("test", 10) → (nothing)

Reduce

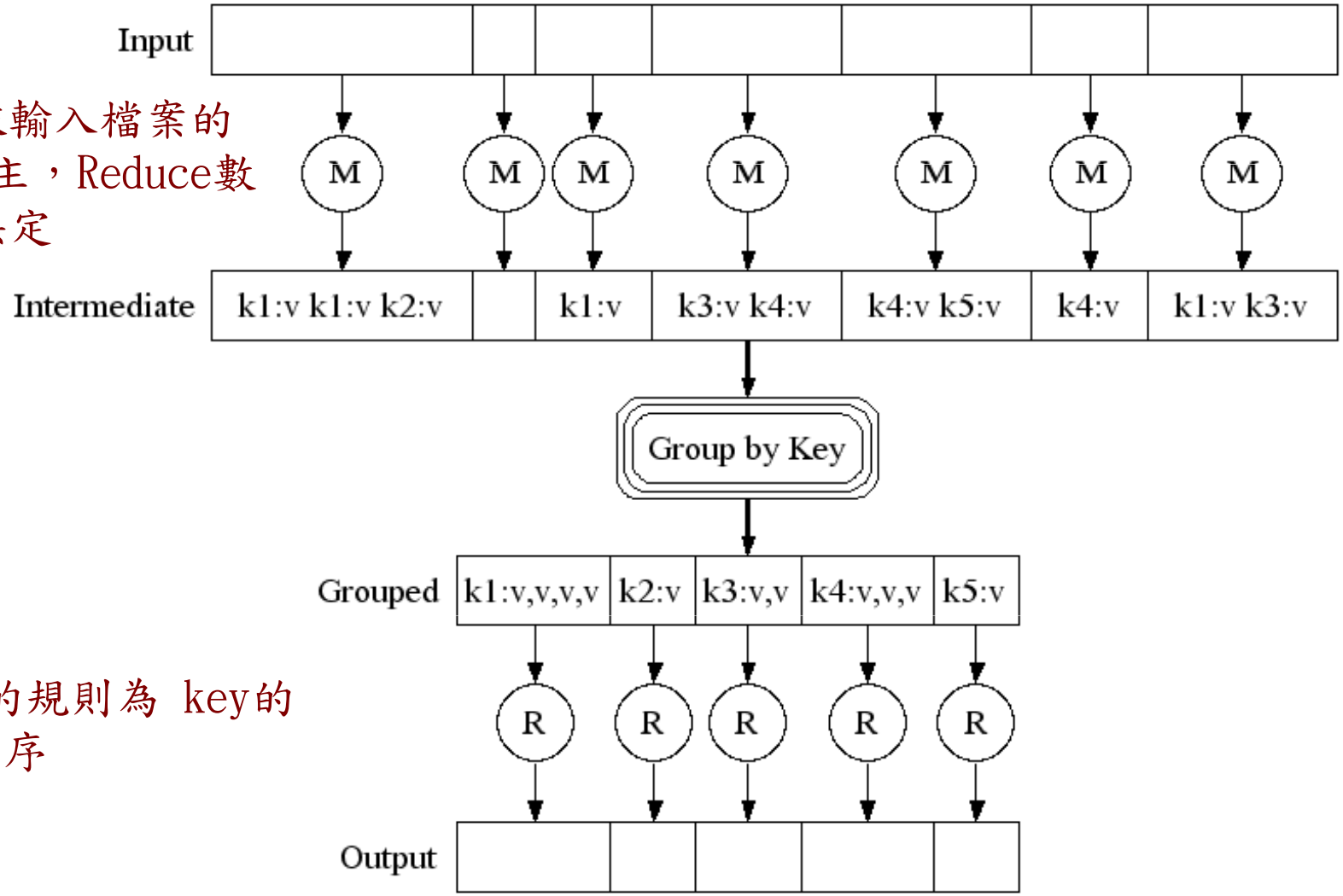
Example: Sum Reducer

```
let reduce(k, vals) =
   sum = 0
   foreach int v in vals:
      sum += v
   emit(k, sum)
```

("A", [42, 100, 312]) → ("A", 454)
("B", [12, 6, -2]) → ("B", 16)

MapReduce 單台

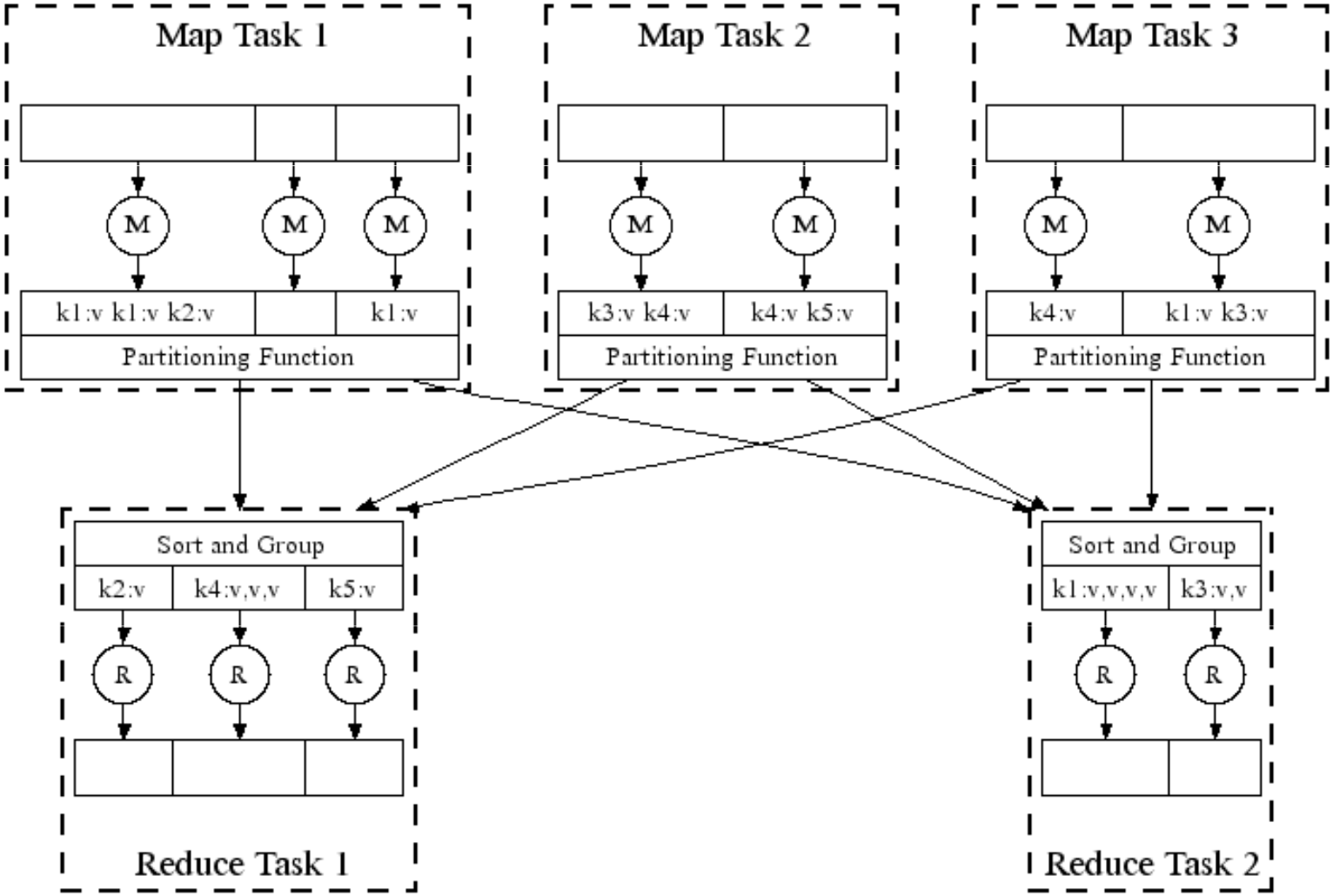
Map 數量依輸入檔案的block數為主，Reduce數量由系統決定



Sort 的規則為 key的字母順序

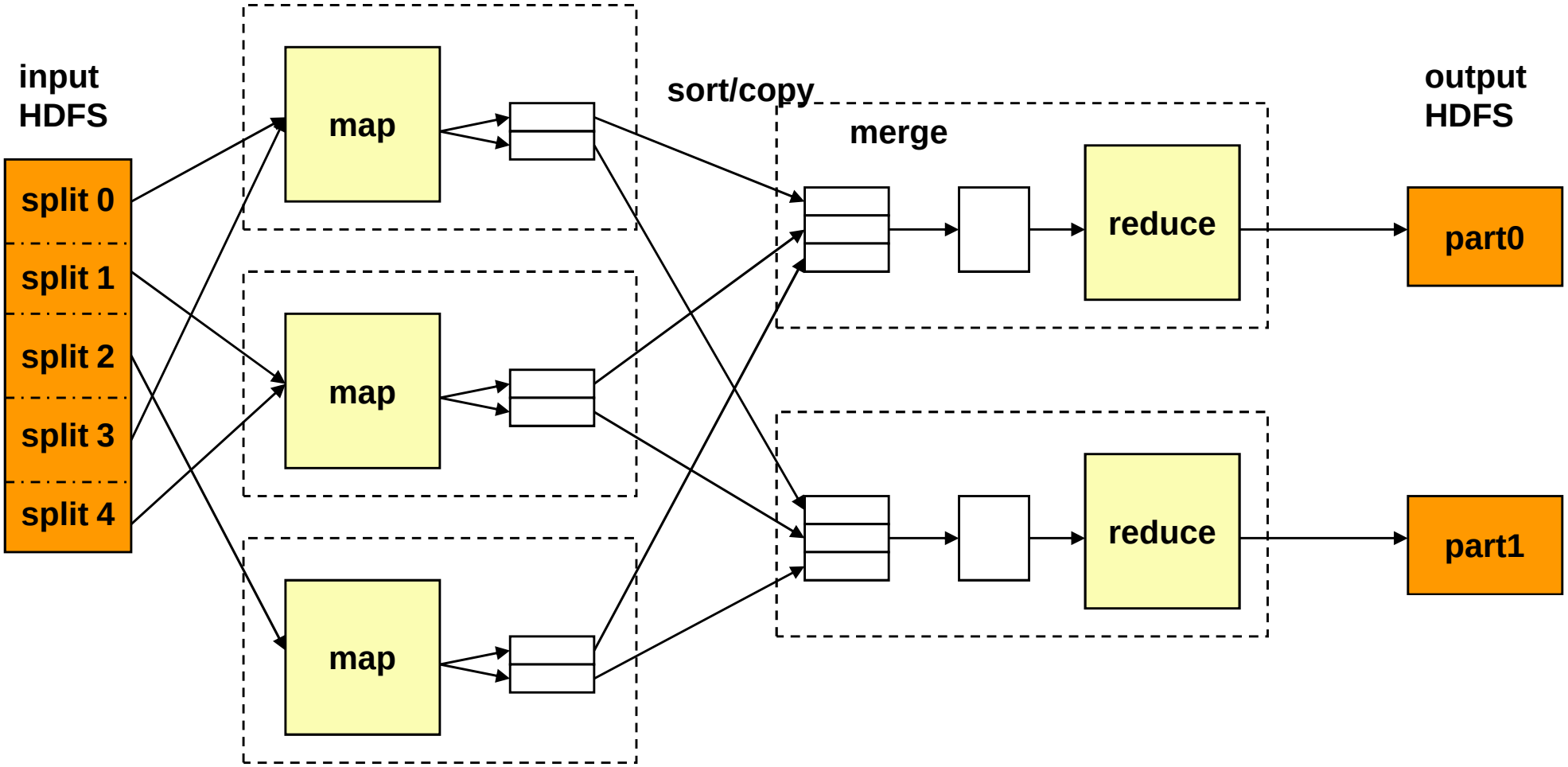
61

MapReduce 多台



62

MapReduce 運作流程



JobTracker跟NameNode取得需要運算的blocks

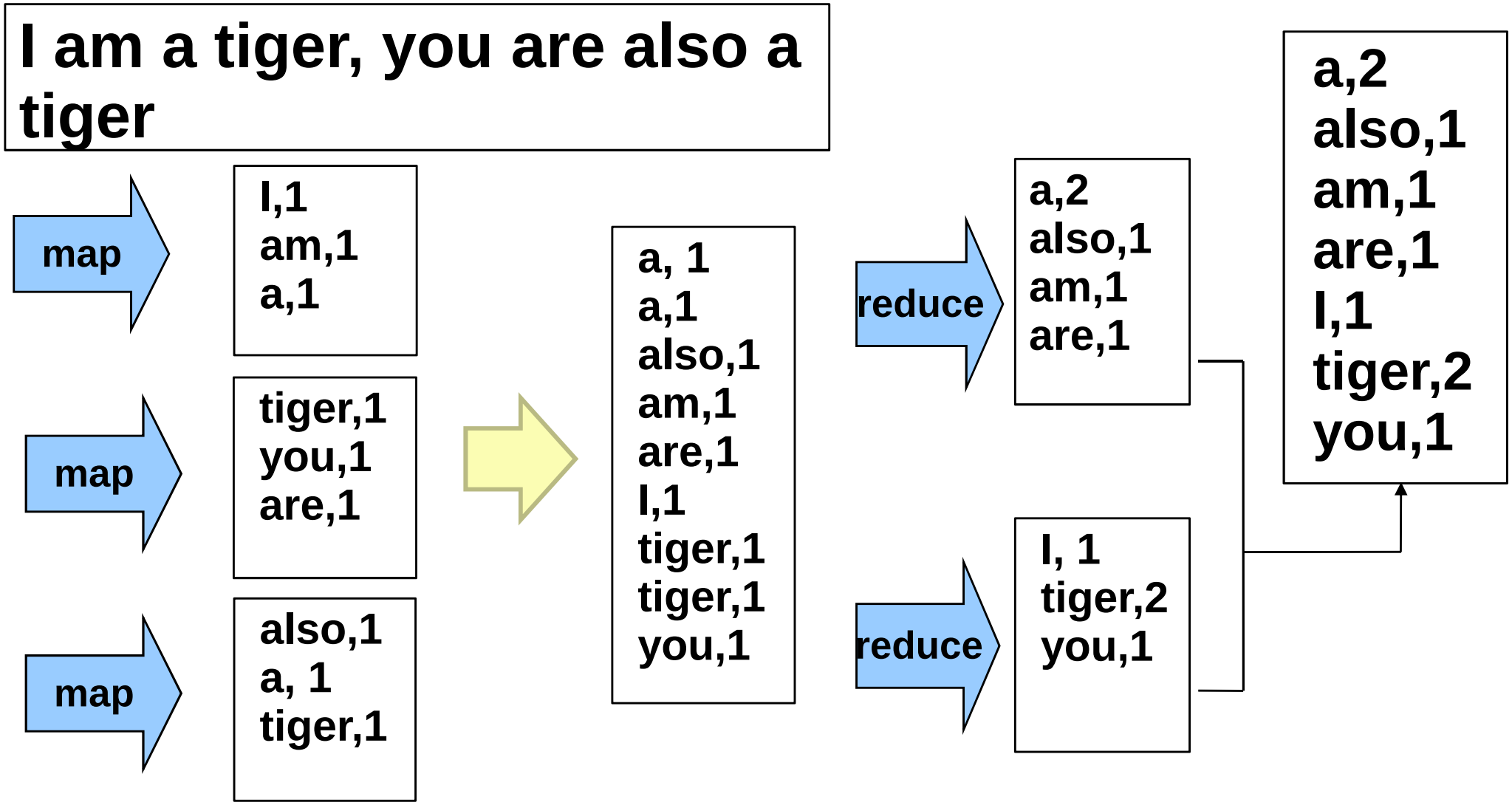
JobTracker選數個TaskTracker來作Map運算，產生些中間檔案

JobTracker將中間檔案整合排序後，複製到需要的TaskTracker去

JobTracker派遣TaskTracker作reduce

reduce完後通知JobTracker與Namenode以產生output

實例範例



JobTracker先選了三個Tracker做map

Map結束後，hadoop進行中間資料的整理與排序

JobTracker再選兩個TaskTracker作reduce