



San Francisco, by Frank S.C. Tseng
(http://www2.nkfust.edu.tw/~imfrank))

第十三章 分散式資料庫系統



本章內容

- 前言
- 分散式資料庫系統簡介
- 分散式資料庫系統上的問題探討
 - 查詢處理與最佳化 (Query Optimization)
 - 分散式的邏輯資料庫設計
 - 系統目錄管理 (System Catalog Management)
 - 回復處理 (Recovery)
 - 連鎖的更新問題
 - 並行控制 (Concurrency Control)
 - 資料的保密性與安全性控制
- **SQL Server 2008 跨越伺服器的查詢功能**



前言

- 分散式資料庫系統：
 - 同質性分散式資料庫系統 (Homogeneous Distributed Database Systems)
處理同一組織中的分散式資料庫
由上而下 (Top-Down) 的設計方式
 - 異質性分散式資料庫系統 (Heterogeneous Distributed Database Systems)
整合不同組織間的既存資料庫
由下而上 (Button-Up) 的整合方式



平行處理

- 強調儘量發掘上述計算過程中的可平行事件，並以同步 (Synchronous) 的協同計算方式完成任務
 - 「並行性」 (Parallelism) — 發生在同一個時段中的多個資源運作上
 - 「同時性」 (Simultaneity) — 同一時間點上的運算
 - 「管線性」 (Piplining) — 在同一個重覆時段中的運作上



平行處理 vs. 分散式處理

■ 電腦計算工作的四類層級

- 一項完整的計算工作或整段程式的層級—主要是透過分散式來執行。
- 一項由函數所完成的工作或副程式層級—主要是透過開發平行演算法 (Parallel Algorithms) 來加速執行。
- 不同運算指令間的運算層級—主要是透過多個 CPUs 來加速執行。
- 運算指令內容的運算層級—主要是透過多個暫存器 (Registers) 來加速執行。



對上述四個層級做平行處理

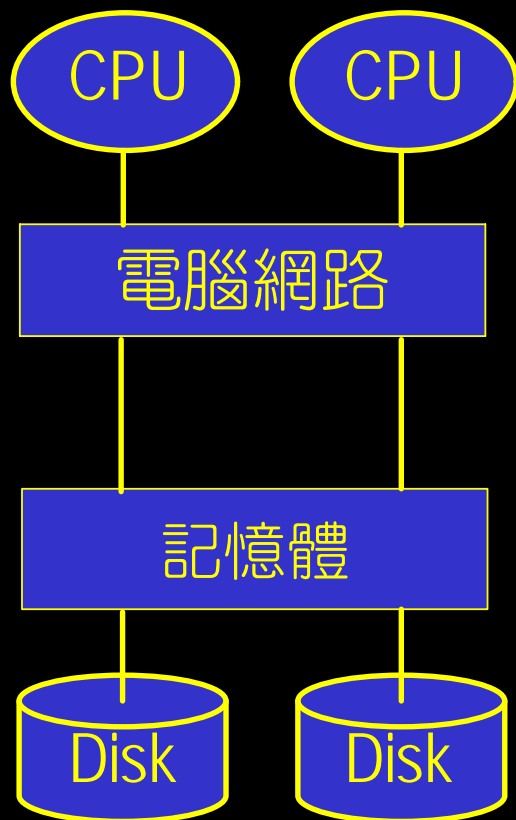
- 1. 將工作分配到不同的系統去執行
- 2. 將工作分配到不同的 CPU去執行。
- 3. 將工作分配到CPU中不同的暫存器 (Registers) 中去執行
- 4. 將工作分配到 CPU 中不同的暫存器中於同一個訊號觸發時執行。



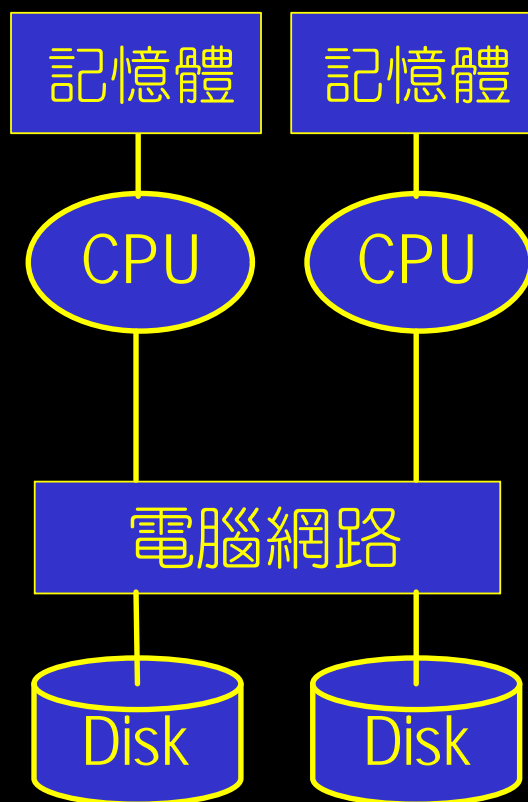
各層級平行處理的作法

- 第 3 與 4 層級的平行處理作法，都可以在設計 CPU 時，以硬體線路來解決。
- 要達到第 2 層級的平行處理，最簡單的作法就是運用與並聯多個 CPU 一同工作。
- 並聯多個 CPU 將衍生出不同的實體架構(如下頁所示) ...

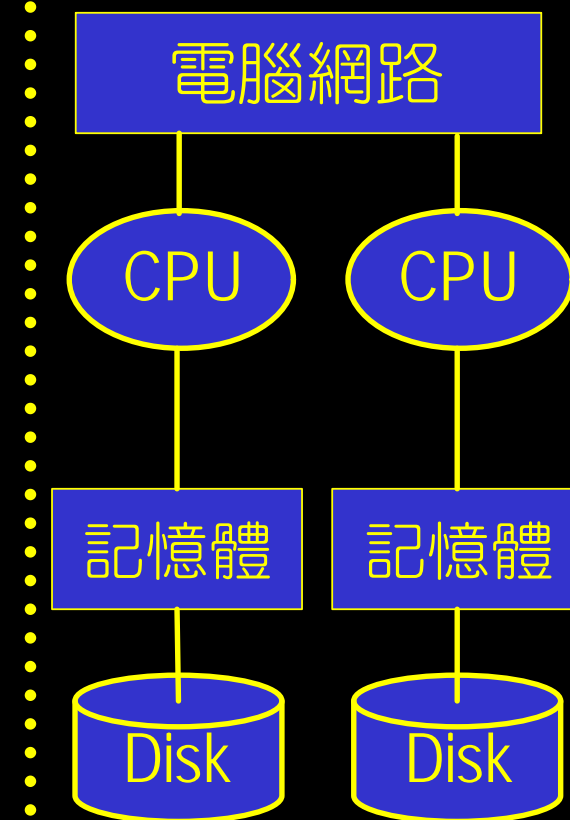
平行處理的基本架構



共享記憶體
(Shared Memory)



共享硬碟空間
(Shared Disk)



完全不共享
(Shared Nothing)



共享記憶體 (Shared Memory)

- 在一部主機中，讓多個 CPU 共享整體記憶體，以及硬碟儲存空間。
- 透過記憶體共享，CPU 彼此的溝通成本很低，
- 但會造成 Memory Contention 的情況發生。
- 當 CPU 數量增加，共享記憶體的同步作業 (Synchronization) 形成效能「瓶頸」(Bottleneck)。
- 依照不同硬體配置與應用情況，可以測出具備最佳效能的 CPU 個數。
- 超過該最佳數量後，再增加 CPU 反而會降低效能。



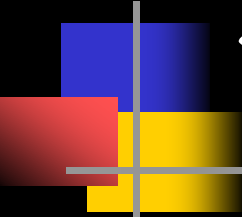
共享硬碟空間 (Shared Disk)

- 同一部主機中，多個 CPU 各自有自己的記憶體，但是整體卻使用一塊共享的硬碟空間。
- 讓 CPU 本身的自主權提高了，
- 雖然東西都可以透過硬碟共享，但是卻仍會造成 Disk Contention 的情況發生。
- 當 CPU 數量增加時，共享硬碟的同步作業 (Synchronization) 也形成了效能上的瓶頸。



完全不共享 (Shared Nothing)

- 讓 CPU 擁有自己的記憶體與儲存空間，算是一個完整而獨立的系統，
- 看成是多個系統互相以電腦網路串聯起來運作。
- 雖然將電腦網路變成了效能的瓶頸，但是透過軟體的最佳化規劃，反而可以降低網路流量，讓整體效能隨著獨立系統的加入呈現線性成長趨勢。
- 各種不同的獨立系統 也可以加入此種運算架構，
- 以「非同步」(Asynchronous) 方式透過軟體協調運作，形成所謂的「分散式處理」(Distributed Processing)，並達到第1層級所需的平行處理模式。



分散式處理

- 也可以看成是一種平行處理的機制，主要強調：
 - 將原本集中的資料分散後擺放在不同的子系統中
 - 以非同步 (Asynchronous) 協同計算方式完成任務
 - 各子系統在運算期間當然也要儘量發掘上述的第 2、3、4 層級可平行事件



平行處理 vs. 分散式處理

- 分散式處理 (Distributed processing) : 對已經被「分散」的資料來做處理，其探討重點在「資料」。
- 平行處理 (Parallel processing) : 將「處理程序」(Process) 盡量平行化，探討對象是「處理程序」。
- 平行處理 (Parallel Processing) ← 互為相反詞 → 循序處理 (Sequential Processing) ;
- 分散式處理 (Distributed/De-Centralized Processing) ← 互為相反詞 → 集中式處理 (Centralized Processing)

平行處理 vs. 分散式處理

從處理程序的角度看

平行式

循序式

平行處理
(多CPU但是分享記憶體或儲存空間)

分散式處理
(獨立的多部主機彼此用網路串聯起來)

單 CPU 主機系統

多台單 CPU 主機系統

集中式

分散式

從資料的分佈角度看



四個象限的說明

- **左下角象限:** 單 CPU 主機系統之運作模式——就是一般的單 CPU 主機的運作模式，**沒有討論的必要**。
- **右下角象限:** 多部單 CPU 主機系統之運作模式——指多部單 CPU 主機系統針對毫無關聯的資料做處理之運作模式，**在此也沒有討論的必要**。
- **左上角象限:** 平行處理 (Parallel Processing)——主要利用並行性、同時性與管線性進行前述 2, 3, 4 三種層級的平行處理。
- **右上角象限:** 分散式處理 (Distributed Processing)——主要利用並行性進行前述第 1 種層級的平行處理。



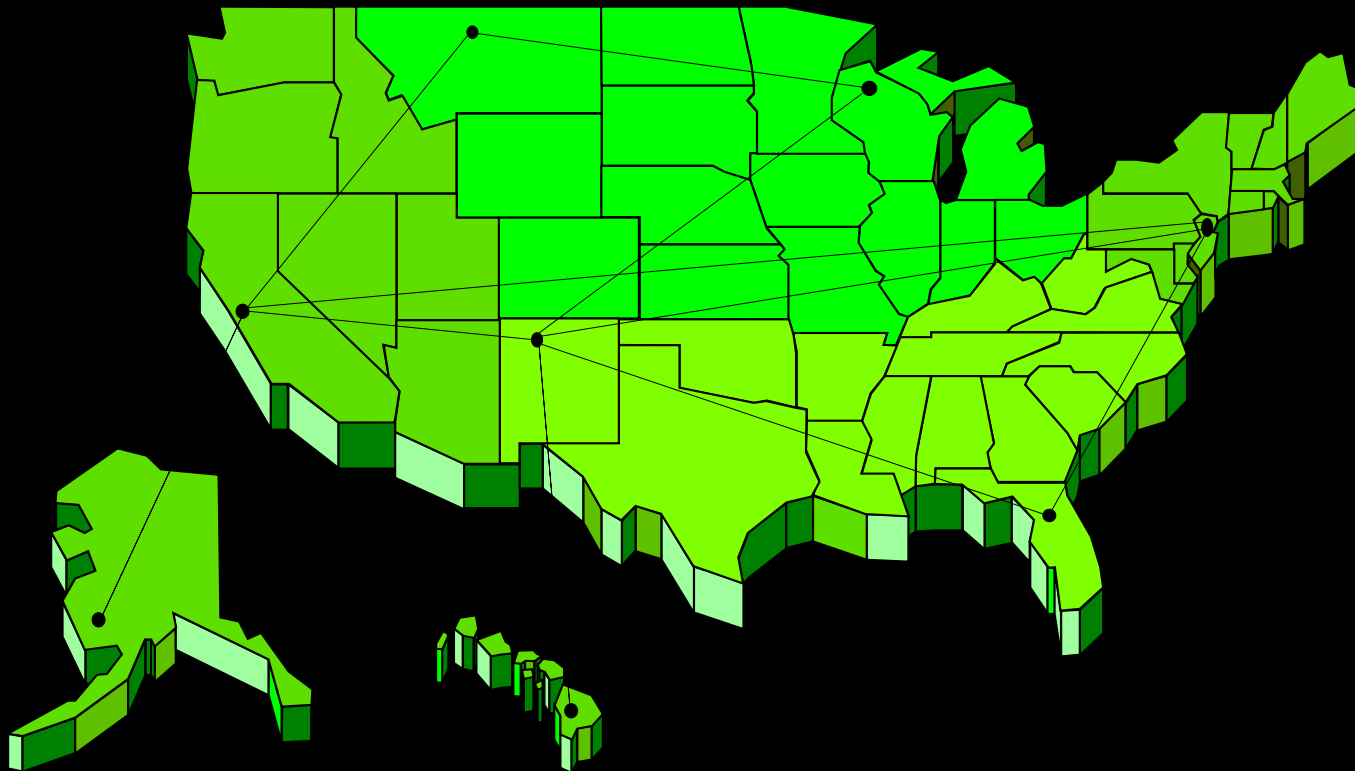
平行處理與分散式處理的比較

狹義的平行處理	分散式處理
緊密耦合 (Tightly-Coupled)	鬆散耦合 (Loosely-Coupled)
共用記憶體或硬碟，共用計時器	不共用記憶體 (含硬碟) 與計時器
沒有自主性	具有自主性
同步 (Synchronous) 運作	非同步 (Asynchronous) 運作
資料集中放在同一電腦系統中	資料分散在不同系統中
可利用並行性、同時性與管線性	只有利用到並行性

兩者共同構成「廣義的平行處理」模式

分散式資料庫系統

80 年代在美國就有不錯的成就
整個企業的組織與架構型態徹底反應在資料庫系統上



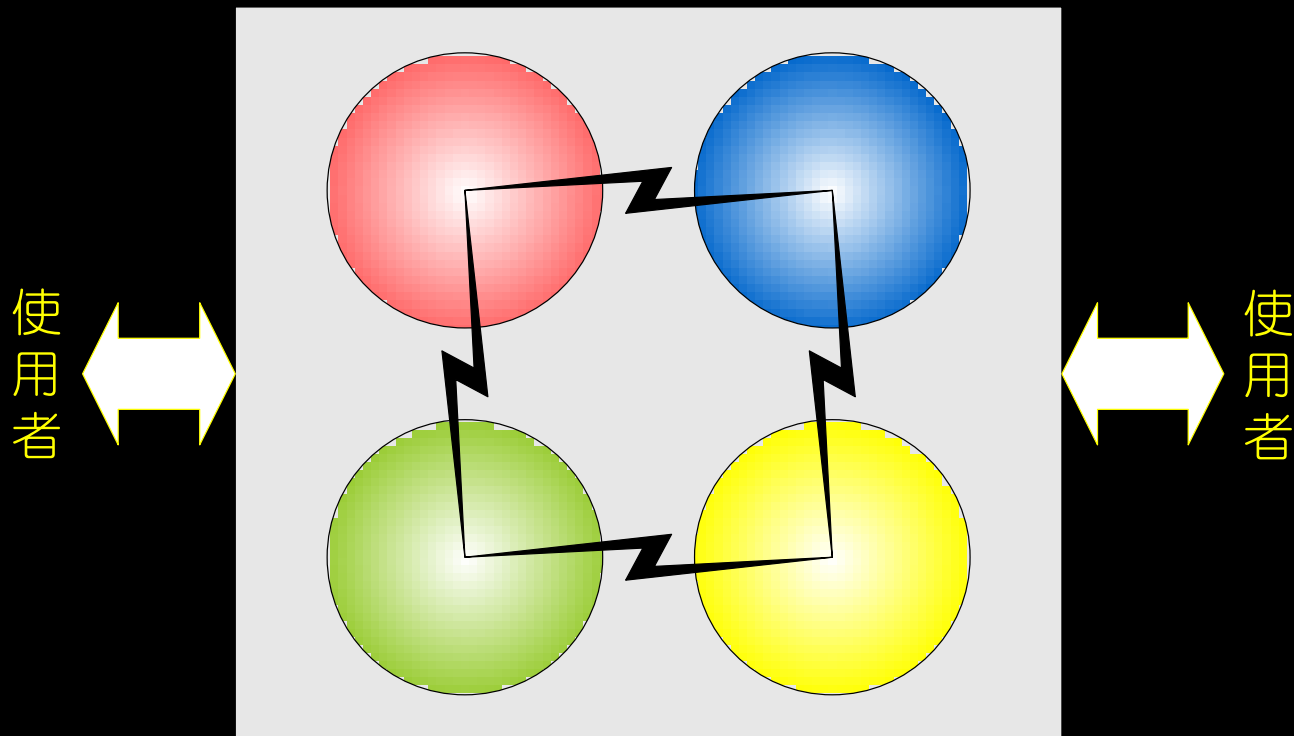


分散式資料庫系統簡介

- 兩個重要的觀點：
 - 一個是資料的分散性 (Distribution) ，
 - 另一個則是資料的邏輯相關性與整合性 (Integration) 。
- 以使用者的眼光來看，整個系統是集中式單一系統，
- 內部運作與管理方式卻是分散的。

分散式資料庫系統

- 內部的處理是透過網路環境，
- 外部的介面則是透過資料庫系統來提供





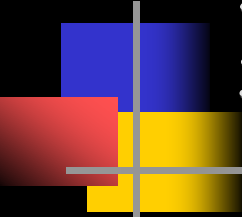
分散式資料庫系統的優點

- 真實反應企業的組織、架構。
- 允許區域的自主性。
- 降低網路的流量，增加處理速度，80/20 Rule 顯示 80 % 都在 Local Database 存取。
- 隨著企業的成長而做漸進式的擴充。
- 技術上改善集中式資料庫系統無法應付使用者需要跨越多個資料站做整體運算的缺點。
- 增進可靠度以及妥善率。
- 將來可透過網路連接已存在的資料庫系統。



分散式資料庫系統的缺點

- 內部處理與系統管理的複雜度為一大缺點。
- 資料在網路上的傳輸頻繁，因此資料的保密性 (Privacy) 與安全性 (Security) 均會受到嚴格的挑戰。



集中式 vs. 分散式 資料庫系統的差異

- 資料庫的管理 (集中式只有一位 DBA)
 - 區域的資料庫管理師 (Local DBA) ,
 - 整體的資料庫管理師 (Global DBA)
 - 區域資料庫管理師有充分的自主性 (Autonomy)
- 資料的獨立性 (集中式強調讓 AP 不受影響, 分散式則強調下列透通性)
 - 位置透通性 (Location Transparency)
 - 資料分割的透通性 (Fragmentation Transparency)
 - 資料重覆的透通性 (Replication Transparency)

集中式 vs. 分散式 資料庫系統的差異 (續)

- 重覆資料的儲存
- 內部的儲存結構與快速存取—在分散式的環境下，很難建立與維持跨越多個資料站的快速儲存結構
- 整合性 (Integrity)、回復性 (Recovery) 與並行控制 (Concurrency Control)—2 phase commit 或 3 phase commit
- 資料的保密性 (Privacy) 與安全性 (Security)



資料獨立 (Data Independence)

- **反義詞** 「資料相依」 (Data Dependence) : 只要資料的儲存結構 (Internal Data Structure) 或存取方式 (Access Method) 一更改，則該應用程式就必須要跟著修改
- 「**資料獨立**」 (Data Independence) : 應用程式不會因內層結構的改變而需更改
- 請詳見第一章 1.5 節內容

位置的透通性

在使用者觀念中，只知資料庫含 Books 關聯表
所以下的指令與集中式的下法一樣

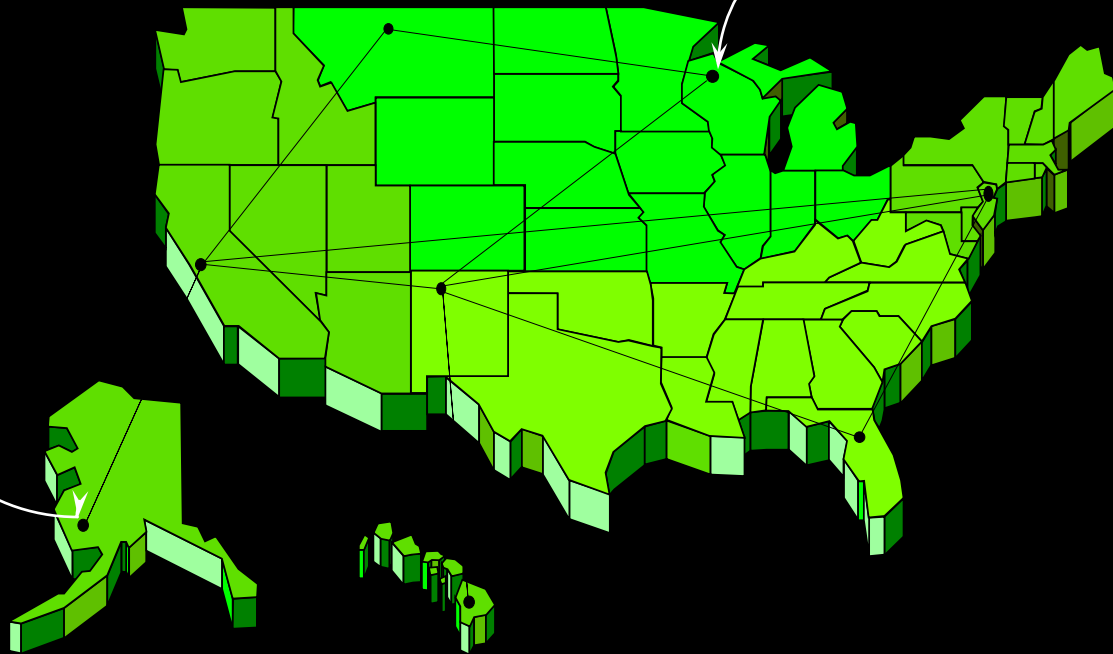
Select * from Books 而不是

Select * From Anchorage.Books

黃色部分是系統
轉換後的結果

Books 放在
Anchorage

使用者在
Chicago



資料分割的透通性

使用者下的指令與集中式的下法一樣

Select * from Books 而不是

Select * From Anchorage.Books

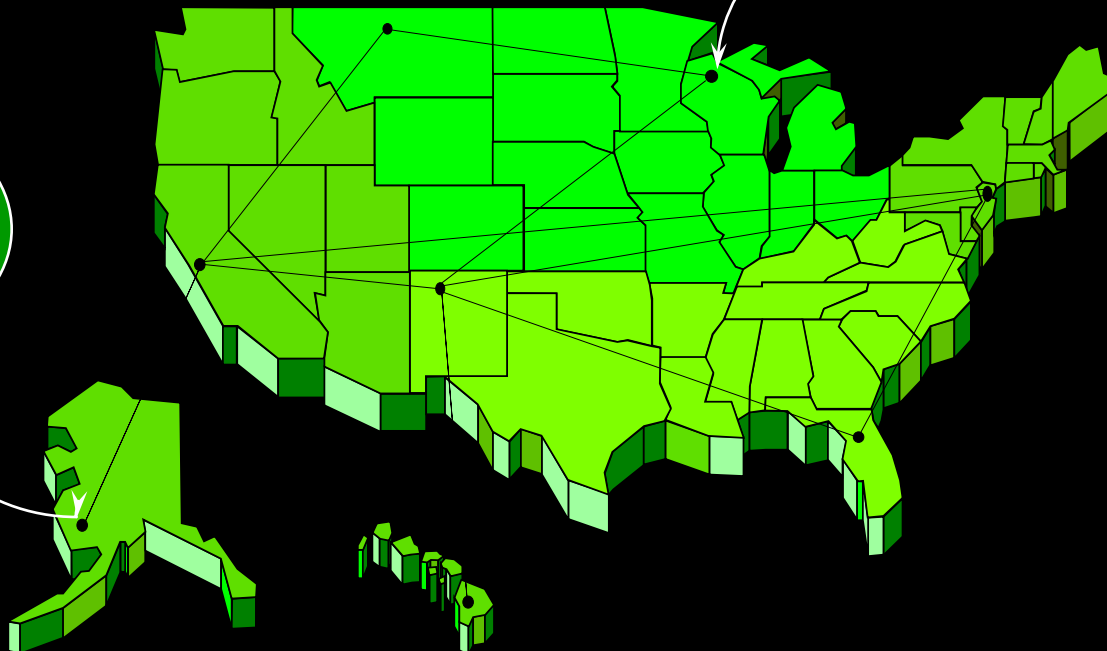
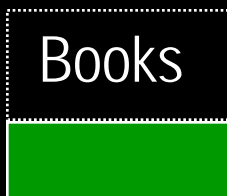
Union

Select * From Chicago.Books



Books 部份
在 Chicago

Books 另一
部份放在
Anchorage



關聯表的水平分割

Employees

<i>eid</i>	<i>name</i>	<i>age</i>	<i>salary</i>	<i>dept_location</i>
N1	張三	25	28k	臺北
N2	李四	27	30k	臺北
S1	王五	30	40k	高雄
S2	毛六	40	50k	高雄

Employees (存放在臺北分公司)

<i>eid</i>	<i>name</i>	<i>age</i>	<i>salary</i>	<i>dept_location</i>
N1	張三	25	28k	臺北 (此欄可省略)
N2	李四	27	30k	臺北 (此欄可省略)

Employees (存放在高雄分公司)

<i>eid</i>	<i>name</i>	<i>age</i>	<i>salary</i>	<i>dept_location</i>
S1	王五	30	40k	高雄 (此欄可省略)
S2	毛六	40	50k	高雄 (此欄可省略)

分割透通性的效果

- 如果我們在高雄分公司將

S2	毛六	40	50k	高雄
----	----	----	-----	----

- 更新成為

S2	毛六	40	50k	臺北
----	----	----	-----	----

- 的話，則此資料便會被移往臺北分公司存放了。由於使用者對關聯表的分割有透通性，所以使用者與應用程式對該關聯表的感覺並不會受到任何影響。

關聯表的垂直分割

Employees

<i>eid</i>	<i>name</i>	<i>age</i>	<i>salary</i>	<i>dept_location</i>
N1	張三	25	28k	臺北
N2	李四	27	30k	臺北
S1	王五	30	40k	高雄
S2	毛六	40	50k	高雄

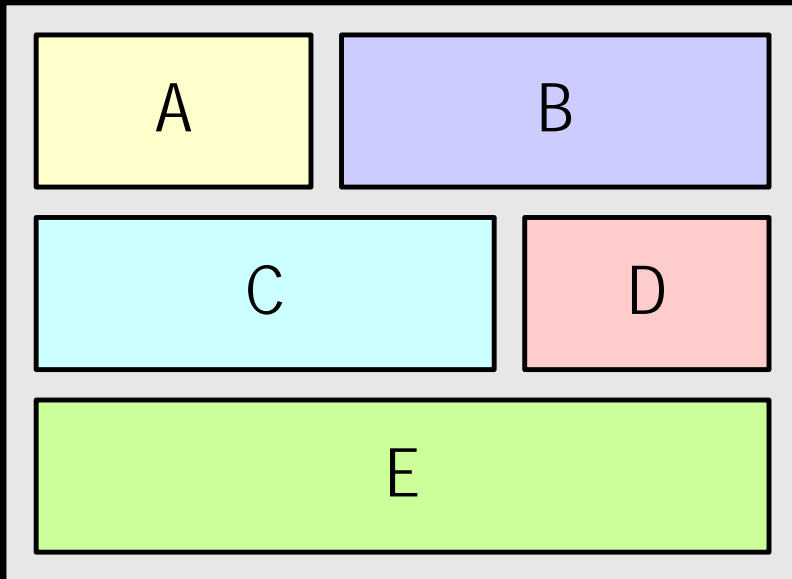
要保留主鍵方可以造
回原表

<i>eid</i>	<i>name</i>	<i>age</i>
N1	張三	25
N2	李四	27
S1	王五	30
S2	毛六	40

要保留主鍵方可以造
回原表

<i>eid</i>	<i>salary</i>	<i>dept_location</i>
N1	28k	臺北
N2	30k	臺北
S1	40k	高雄
S2	50k	高雄

關聯表的水平與垂直分割



- 將 R 切割成五大部份，分別放在 A, B, C, D, E 五個資料站中
- 垂直切割的部份要複製主鍵造回 R 時才能夠不失真

$$\blacksquare R = (A \bowtie B) \cup (C \bowtie D) \cup E$$



依據外來鍵參考圖進行分割

- 在外來鍵參考圖中被參考的關聯表 (假設稱為 **S**) 會被拿來當做首要切割的對象
- 隨後參考 **S** 的所有關聯表便完全依據 **S** 切割出來的主鍵集進行切割。
 - 如：關聯表 Books 被 Orders 所參考，因此先對 Books 切割，
 - 假設將 Books 切成包含 Books.id 值 {1, 2}、{3, 4} 與 {5, 6}，
 - Orders 隨後也據此分割 Orders.id 屬性成為 {1, 2}、{3, 4}、{5, 6}
 - 讓 Books.id 與 Orders.id 為 {1, 2} 的放在同一個資料站；
 - 讓 Books.id 與 Orders.id 為 {3, 4} 的放在另一個相同的資料站，依此類推
 - 原因：這些值進行合併運算 (Join) 的機率很高，擺在一起效能較好。

先分割 Books 再據此分割 Orders

Orders

no	id	quantity
1	1	30
1	2	20
2	1	30
2	2	40
3	2	20
4	2	20

Books

id	bookname	author	price	publisher
1	三國演義	羅貫中	120	古文出版社
2	水滸傳	施耐庵	170	中庸出版社

Site A

id	bookname	author	price	publisher
3	紅樓夢	曹雪芹	170	春秋出版社
4	西遊記	吳承恩	140	聊齋出版社

Site B

id	bookname	author	price	publisher
5	水經注	酈道元	120	易經出版社
6	道德經	老子	190	大唐出版社

Site C

id	bookname	author	price	publisher
1	5	10		
1	6	10		
4	5	40		

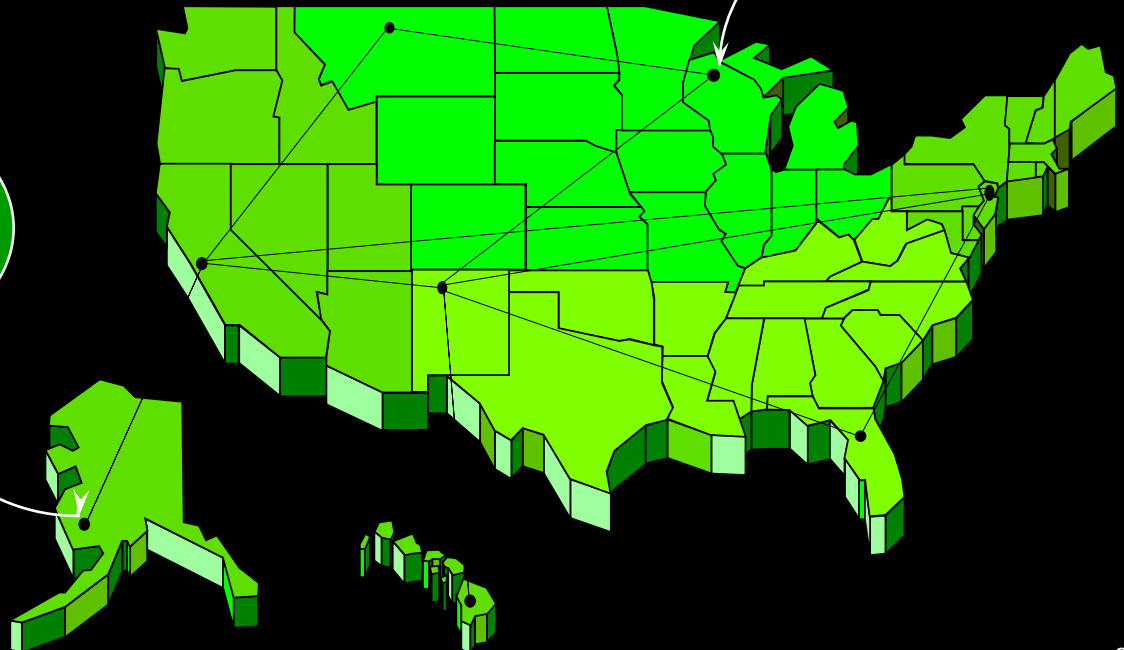
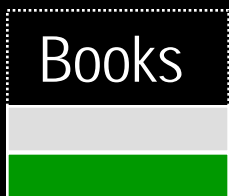
資料重覆的透通性

Books 中間的部份重覆放在不同地點
使用者 下的指令與集中式一樣
Update Books where id = 3 而不是
Update Chicago.Books where id = 3
Update Anchorage.Books where id = 3



Books 部份
在 Chicago

Books 另一
部份放在
Anchorage





分散式資料庫系統的目標

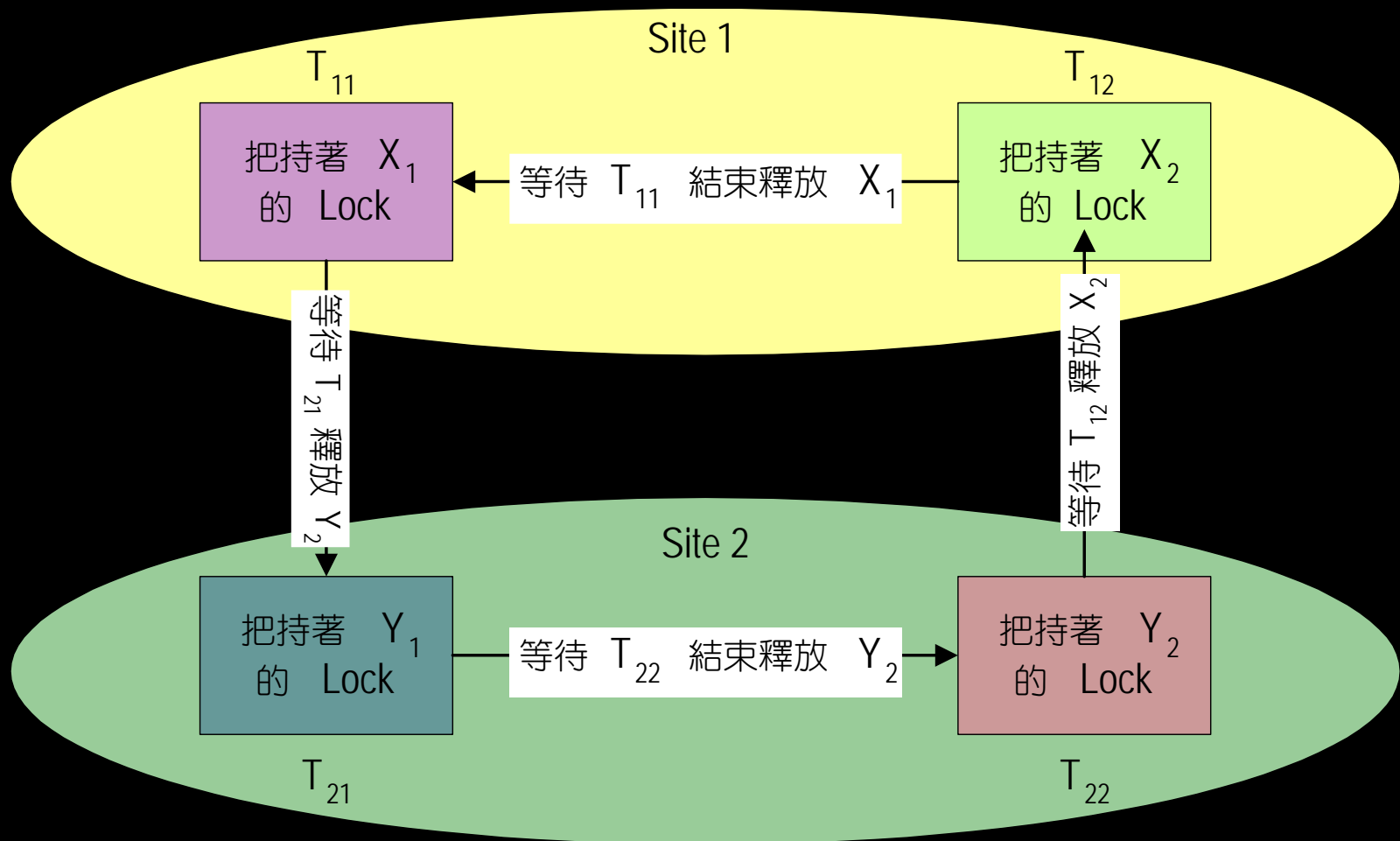
- 每個資料站都應有自主性 (Local Autonomy) ，但是卻不是百分之百擁有自主權
- 所有的資料站都具有平等的地位—無主導者
- 持續地運作 (Continue Operation)—新加入一個資料站或將某資料站 Upgrading 時可以不須要停止系統的運作
- 位置透通性、分割透通性、複製透通性（如前所述）



分散式資料庫系統的目標(續)

- 系統要有處理分散式查詢的能力
- 系統要有處理分散式異動的能力—維持異動的四個特性
 - 「兩段式委任」(Two-Phase Commit) 或
 - 「三段式委任」(Three-Phase Commit)
 - 要能解決分散式系統的「死結」(Deadlock) 問題
- 資料站的資料庫管理系統可以不同，而且要與硬體設備、作業系統、網路系統無關

分散式系統的「死結」





分散式資料庫系統上的問題

- 有效率處理跨資料站的存取運算與查詢處理。
- 跨資料站的資料轉換與整合處理。
- 如何將資料以最佳方式切割放到各個資料站。
- 透過網路存取資料的存取控制，與安全防護。
- 提供快速而安全的回復處理程序。
- 確保各區域資料站與整體資料站上所存資料能真實反應真實情況，避免因區域與整體異動的相互干擾而有不相容的情況產生。



分散式系統上的技術發展

- 查詢處理與最佳化
- 分散式的邏輯資料庫設計
- 系統目錄管理
- 回復處理
- 連鎖的更新問題
- 並行控制
- 資料的保密性與安全性控制



分散式系統上的管理問題

- 必須評估網路上的資料流量與需求
- 要對分散式資訊系統上的應用程式與資料做一番規劃。例如：決定特定資料，如系統目錄，與應用程式要如何擺放與其執行方式。
- 在人為的爭論點上解決不同資料站之間的資源分配問題。
- 維資料保密性與安全性的策略與規範制定。
- 人員的教育訓練問題。
- 資料的備份策略與做法。
- 系統停擺時的替代方案



分散式的查詢處理與最佳化

範例

Bookstores

<i>no</i>	<i>name</i>	<i>rank</i>	<i>city</i>
...
(共10,000筆) (存放在資料站A)			
...

Orders

<i>no</i>	<i>id</i>	<i>quantity</i>
...
(共1,000,000筆) (存放在資料站A)		
...

Books

<i>id</i>	<i>bookname</i>	<i>price</i>
...
(共100,000筆) (存放在資料站B)		
...

假設每一筆記錄佔有 400 個位元 (bits)



SQL 查詢範例

```
■ select Bookstores.no  
from Bookstores, Orders, Books  
where Bookstores.city = '臺中市' and  
Bookstores.no = Orders.no and Orders.id = Books.id  
and Books.price = 160;
```



假設條件

- 在 Books 中 price 等於 160 的記錄有 10 筆
- 與位於 ‘臺中市’ 的書店交易的有 100,000 筆
- 網路傳輸的速率為 100,000 bits/sec ,
- 存取時的延遲時間為 0.01 秒鐘
- 以上均為正常的網路傳輸速度



整體查詢花在網路上的時間

- 對於任一種查詢處理策略而言，整體花在網路傳輸上的時間為：

整體的存取延遲時間 + 整體資料量 / 網路傳輸速度 (秒)

$$= \left(\text{所有往來訊息的數量} / 100 \right) + \left(\text{整體資料量的位元數} / 100,000 \right) \text{ 秒}$$



第一種處理策略

- 在資料站 A 中合併 Bookstores 與 Orders 得：
Bookstores \bowtie Orders
- 由 Bookstores \bowtie Orders 中選出 City = '臺中市' 的記錄，共 100,000 筆。
- 對於上面所得的記錄，——到資料站 B 去檢查是否其 Price = 160，每一次都要使用兩個溝通訊息，一個用來向資料站 B 查詢，一個乃是來自資料站 B 的回應，所以共需
(200,000/100) \div 2000 秒 (33.3分鐘)。



第二種處理策略

- 將 Books 移到資料站 A，並且在 A 中完成整個查詢：
將 Bookstores、Orders 與 Books 合併後得到：
Bookstores \bowtie Orders \bowtie Books
- 由 Bookstores \bowtie Orders \bowtie Books 中選出符合 City = '臺中市' 以及 Price = 160 的記錄，共 100,000 筆。
- 共需 $0.01 + (100,000 * 400) / 100,000 \doteq 400$ 秒 (6.67分鐘)。
- 在本例中我們都假設在單一資料站上的計算時間遠比網路的傳輸速度少很多, 故予以忽略。



第三種處理策略

- 將 Bookstores 與 Orders 移到資料站 B，並且在B 中完成整個查詢：Bookstores \bowtie Orders \bowtie Books
- 由 Bookstores \bowtie Orders \bowtie Books 中選出符合 City = '臺中市' 以及 Price = 160 的記錄，共 100,000 筆。
- 共需 $0.02 + ((1,000,000 + 10,000) * 400) / 100,000 \doteq 4040$ 秒 (67.3分鐘)。



第四種處理策略

- 在資料站 B 中以 $\text{price} = 160$ 將 Books 過濾後得到10筆資料，
- 針對上面所得的記錄，一一到資料站 A 去檢查是否在 Orders 中與這些書籍有交易的書店其所在城市是“臺中市”，每一次都要使用兩個溝通訊息，一個用來向資料站 A 查詢，一個乃是來自資料站 A 的回應，所以共需 $(20/100) \div \underline{0.2}$ 秒。



第五種處理策略

- 在資料站 B 中以 $\text{price} = 160$ 將 Books 過濾後得到 10 筆資料，
- 將上面所得的記錄整個送到資料站 A 去和 Bookstores 與 Orders 做合併，並以城市是“臺中市”的條件來過濾結果，所以共需

$$0.01 + (10 * 400/100,000) = \underline{0.05 \text{ 秒}}。$$



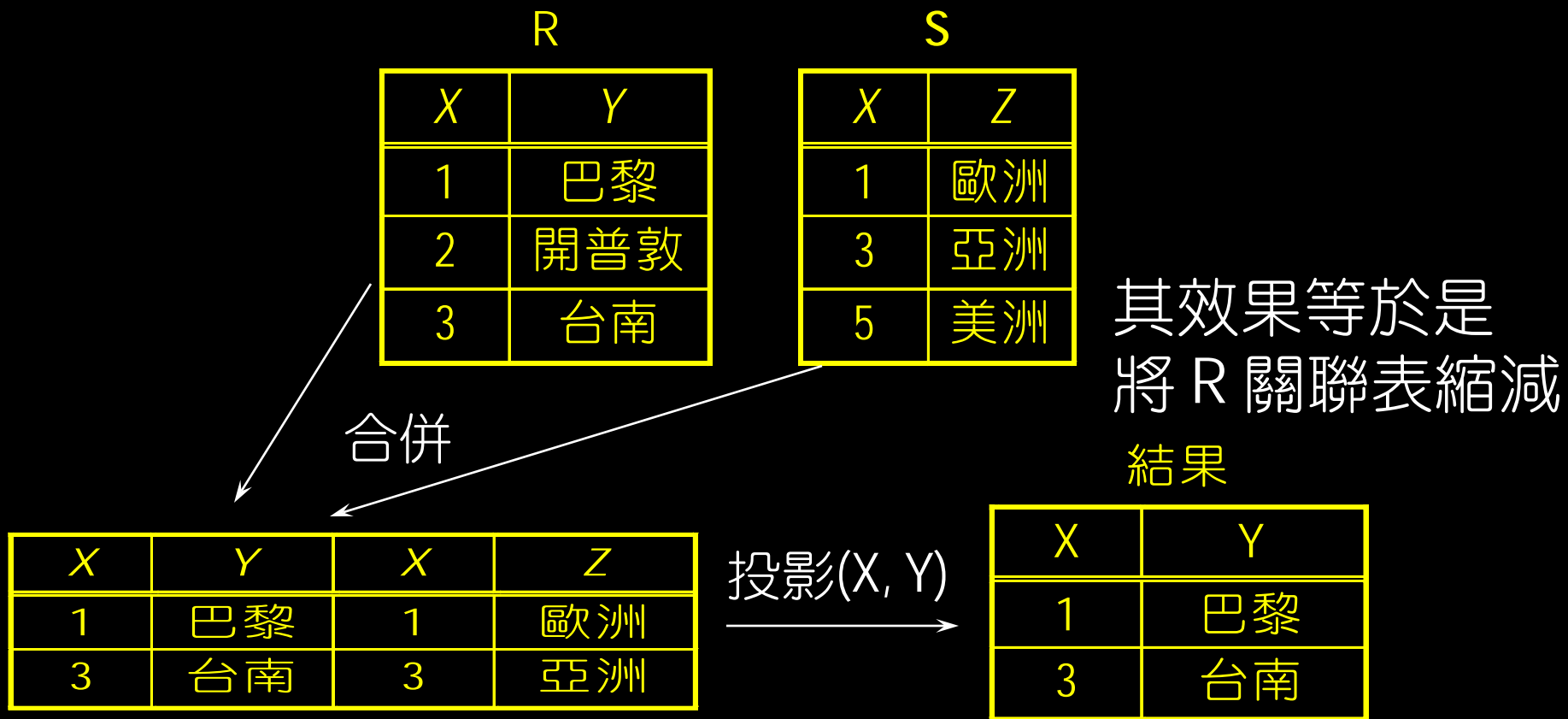
「半合併」 (Semi-Join) 運算

- 定義 6.1 : [半 合 併] (Semi-Join) :
關聯表 $R(X,Y)$ 與關聯表 $S(X,Z)$ 做半合併運算，
寫成 $R(X,Y) \bowtie S(X,Z)$ ，的結果定義為

$$R(X,Y) \bowtie S(X,Z) = \pi_{X,Y}(R(X,Y) \Join S(X,Z))$$

也就是說：先將 $R(X, Y)$ 與 $S(X, Z)$ 做相等合併後，
再將 R 中的所有屬性投影出來。

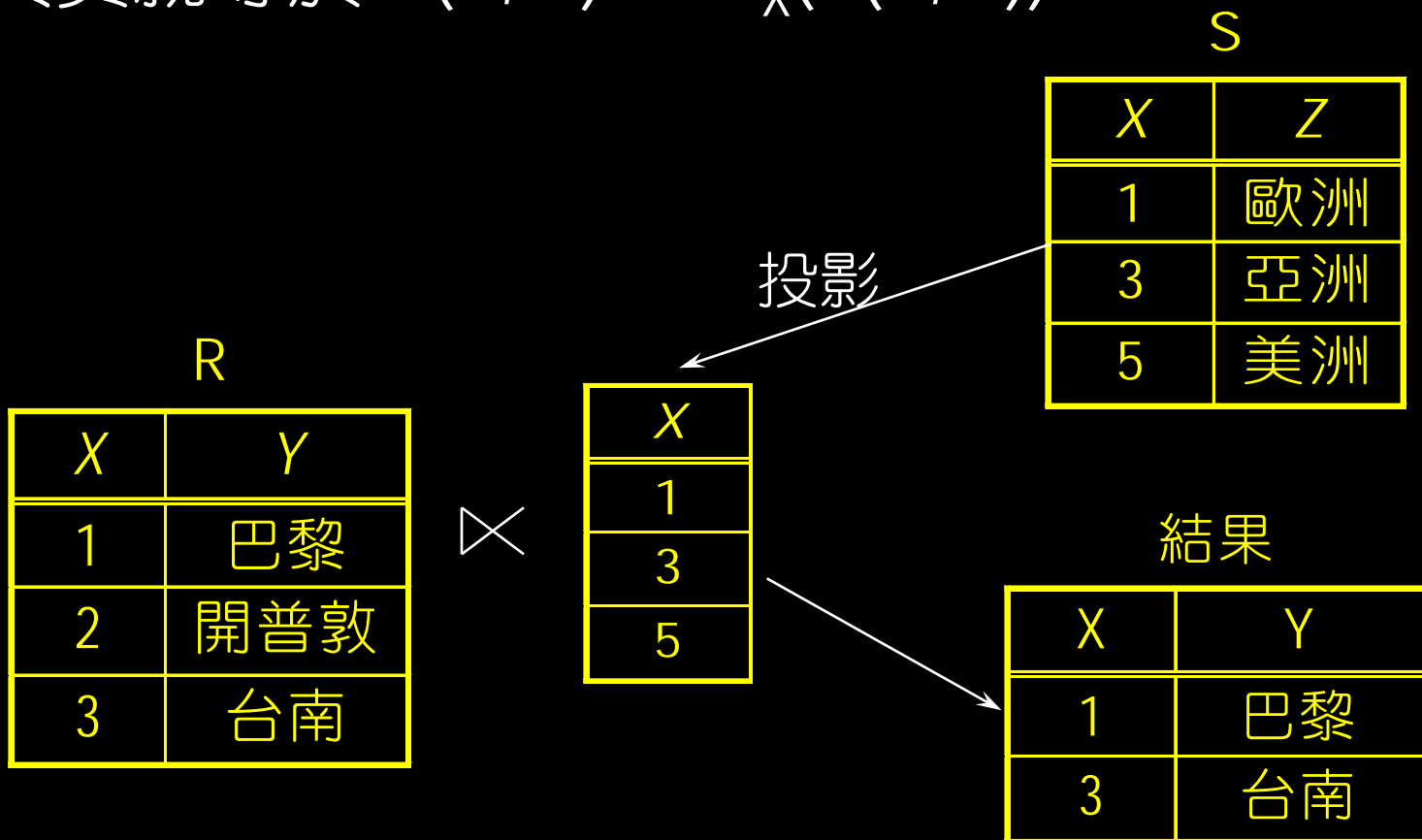
半合併運算的範例



再敘半合併

$$R(X, Y) \bowtie S(X, Z) = \pi_{X, Y}(R(X, Y) \bowtie S(X, Z))$$

其實就等於 $R(X, Y) \bowtie \pi_X(S(X, Z))$



半合併沒有交換律

S

X	Z
1	歐洲
3	亞洲
5	美洲

R

X	Y
1	巴黎
2	開普敦
3	台南

$$R \bowtie S \neq S \bowtie R$$

其效果等於是
將 S 關聯表縮減

合併

X	Z	X	Y
1	歐洲	1	巴黎
3	亞洲	3	台南

投影 (X, Z)

結果

X	Z
1	歐洲
3	亞洲



半合併的用處

- 用來做不同資料站間的關聯表合併
- 只要傳送合併欄位到被合併的關聯表端，不用在網路上傳輸整個關聯表
- 半合併可以將被合併的關聯表大小減少
- 並將減少後的結果一次傳回
- 降低了大量的網路流量



分散式邏輯資料庫設計

- 除了考慮正規化的問題以外，還要考慮下列的幾個問題：
 - “同一時間內能完成最多的值組計算”——分散資料時儘量提高平行處理的程度
 - “同一地點中能完成最多的值組計算”——分散資料時儘量提高區域處理的程度
 - 系統方面則需考量：
 - 如何將整體查詢轉換到區域資料站上的子查詢，
 - 如何將各區域所傳回的結果整合成完整的整體結果。

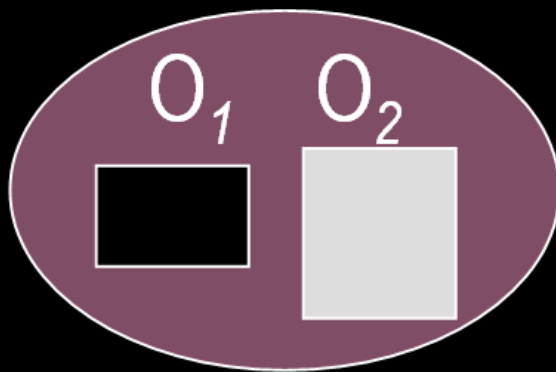


分散式資料庫的規劃問題

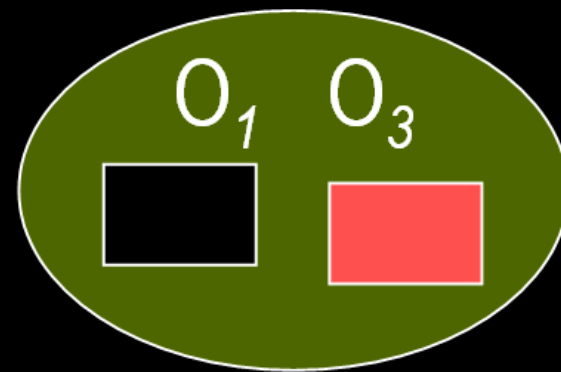
- 「檔案分配問題」 (File Allocation Problems) : 考慮如何將 n 個關聯表— $\{F_1, F_2, \dots, F_n\}$ 分配到 m 個資料站上一— $\{S_1, S_2, \dots, S_m\}$ ，以使得整體查詢最有利。
- 「檔案放置問題」 (File Placement Problems) : 考慮如何將相關的值組儘量擺放到檔案中的同一頁中 (Pages)，或者是拆散分別放在不同的資料站 (以利平行處理)，加速整體的查詢

關聯表的切割與複製

- 有些資料，例如 O_1 ，可能會面臨既要與 O_2 擺在一起，卻又有必要與 O_3 擺在一起的兩難情況，此時我們可以將資料做適當的切割 (Fragmentation) 或複製 (Replication) 以達成所需的目的



Site 1



Site 2



分散式系統目錄管理

- 分散式資料庫系統目錄要包含
 - 所有基底關聯表、索引、視界與使用者資料
 - 關聯表被切割、複製以及所放的資料站位址
 - 使用者跨資料站的存取取權限
 - 其它相關資訊
 - 最重要的是要讓所有資料站都能很容易地存取。



分散式系統目錄管理(續)

- 系統目錄的儲存方式
 - 1. 集中放在某資料站上 (Centralized)：違反了所有的資料站都具有平等的地位的原則，也會造成該資料站上的網路瓶頸。
 - 2. 所有資料站存一份系統目錄 (Full Replicated)：資料站會失去自主性 (Autonomy)，每次對系統目錄的異動都要連帶反應到其它資料站上的系統目錄，造成網路上許多額外的負擔。

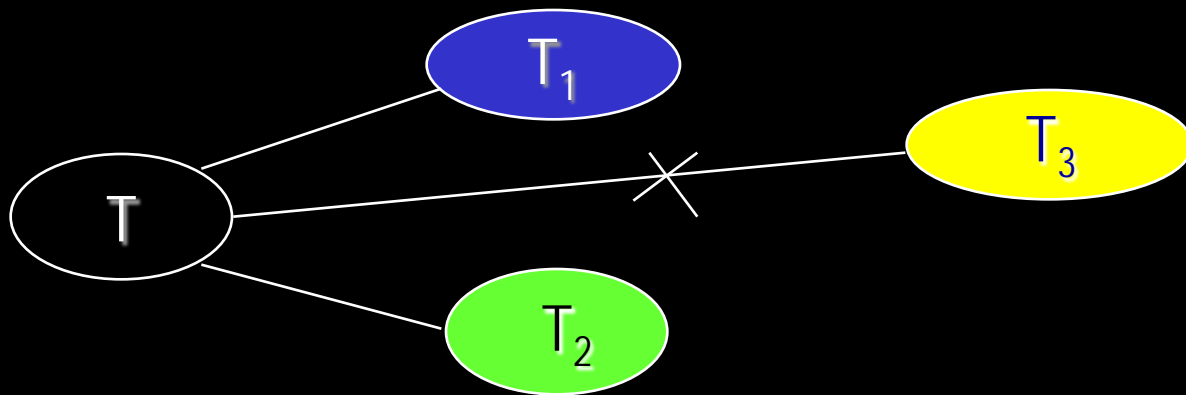


分散式系統目錄管理(續)

- 系統目錄的儲存方式 (續)
 - 3. 將系統目錄做適當分割分別擺放 (Partitioned) : 會使得存取遠端系統目錄的代價很高，平均說來會使每一次存取系統目錄都要去存取一半以上的資料站。
 - 4. 結合 1. 與 3. : 讓每一個資料站存放與本身相關的部份，除此之外並選定一個資料站來存放整體的系統目錄：雖然違反了所有的資料站都具有平等的地位原則，但效率比 3. 好，因為存取系統目錄只要對一個資料站即可。

分散式環境下的回復處理

- 單純的主機錯誤 (Site Failures Only)
- 網路斷線但不造成網路分隔
- 網路斷線但造成網路分隔 (Site Failure with Network Partitions)
- 在分散式環境下維持異動的單元性 (Atomicity)



兩段式委任 (Two-Phase Commit)

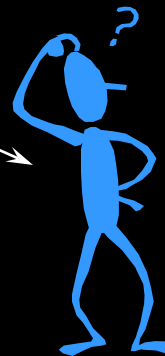
- 想像西洋式的婚禮

你願意嫁給他嗎?



我願意!

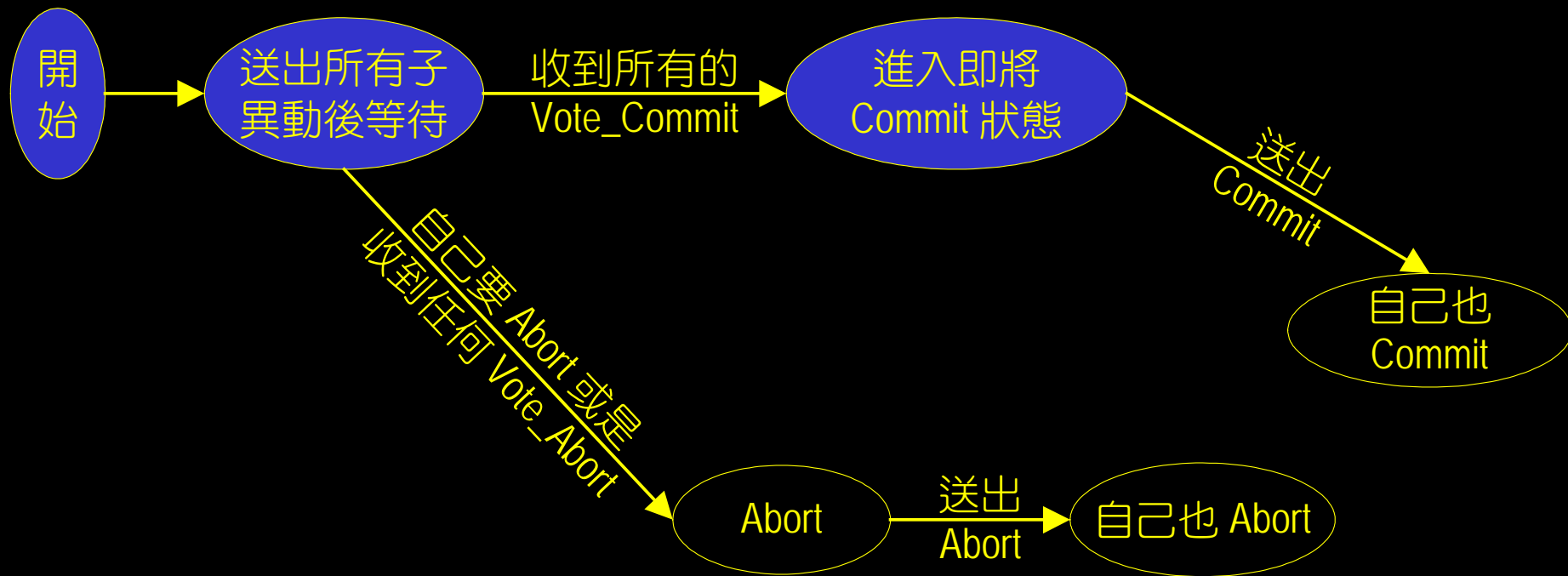
你願意娶她嗎?



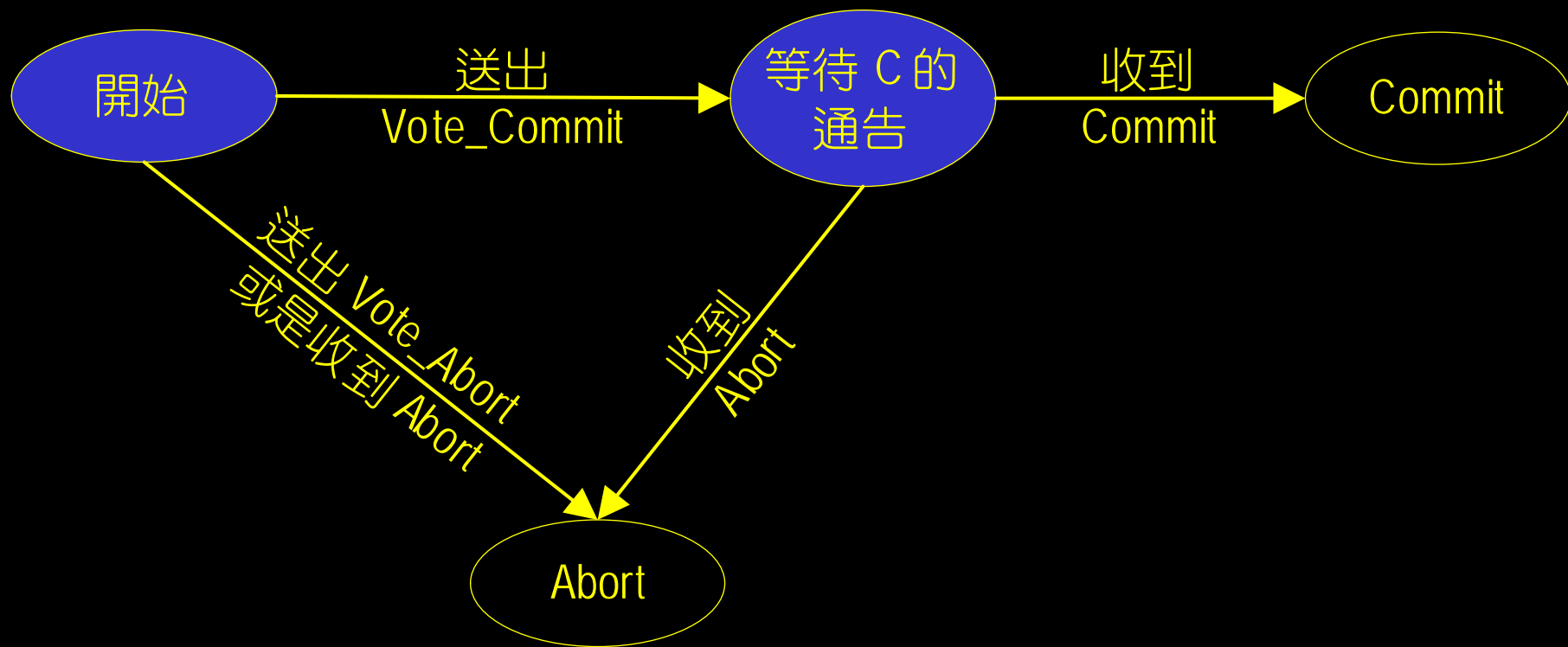
我願意 (Commit)!

我不要 (Abort)!

兩段式委任中協調者 C 之狀態



兩段式委任中參與者 P_i 之狀態

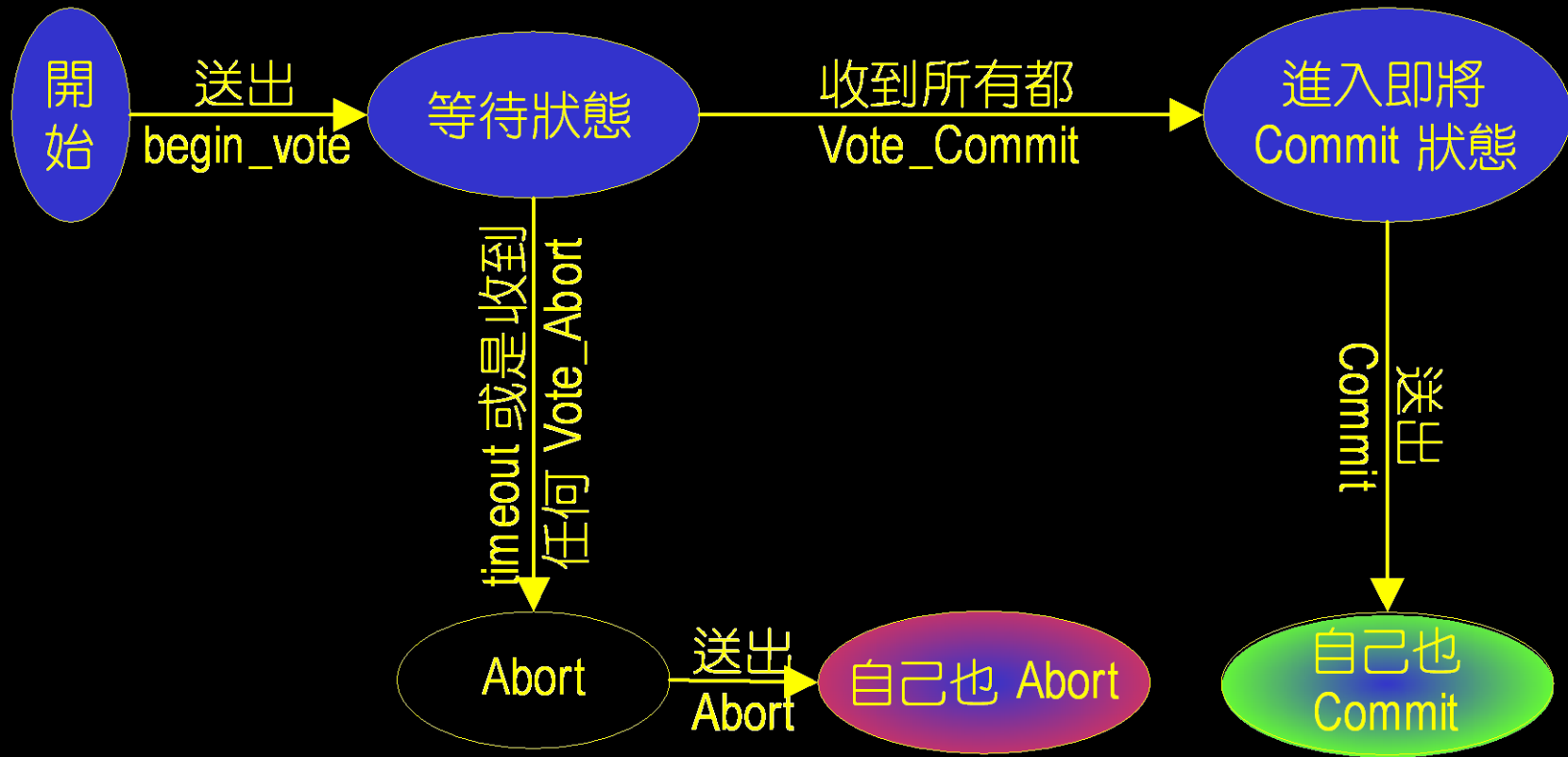




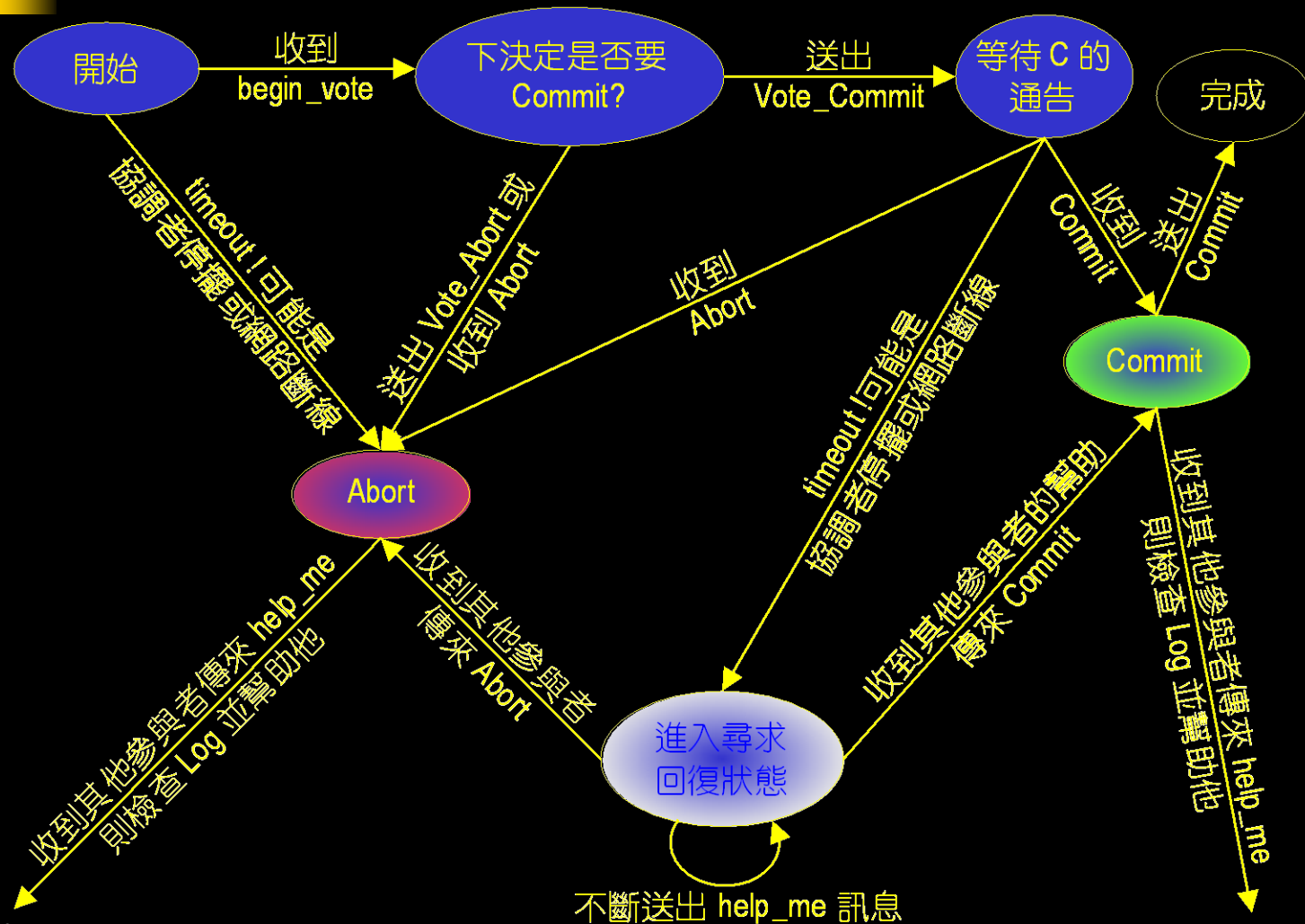
溝通過程網路斷線或主機停擺

- 導致整體異動無法完成，甚至無限延期 Blocking
- 解決 Blocking 狀態的補強方案：
 - Time-out 機制
 - 協調者增加送出 begin_vote (開始投票) 的公告，其中並包含所有參與者的列表與其網路位址，以便與下面的 help_me 訊息配合。
 - 參與者 P_i 一直收不到協調者 C 所發出的訊息時，可以送出 help_me 的訊息

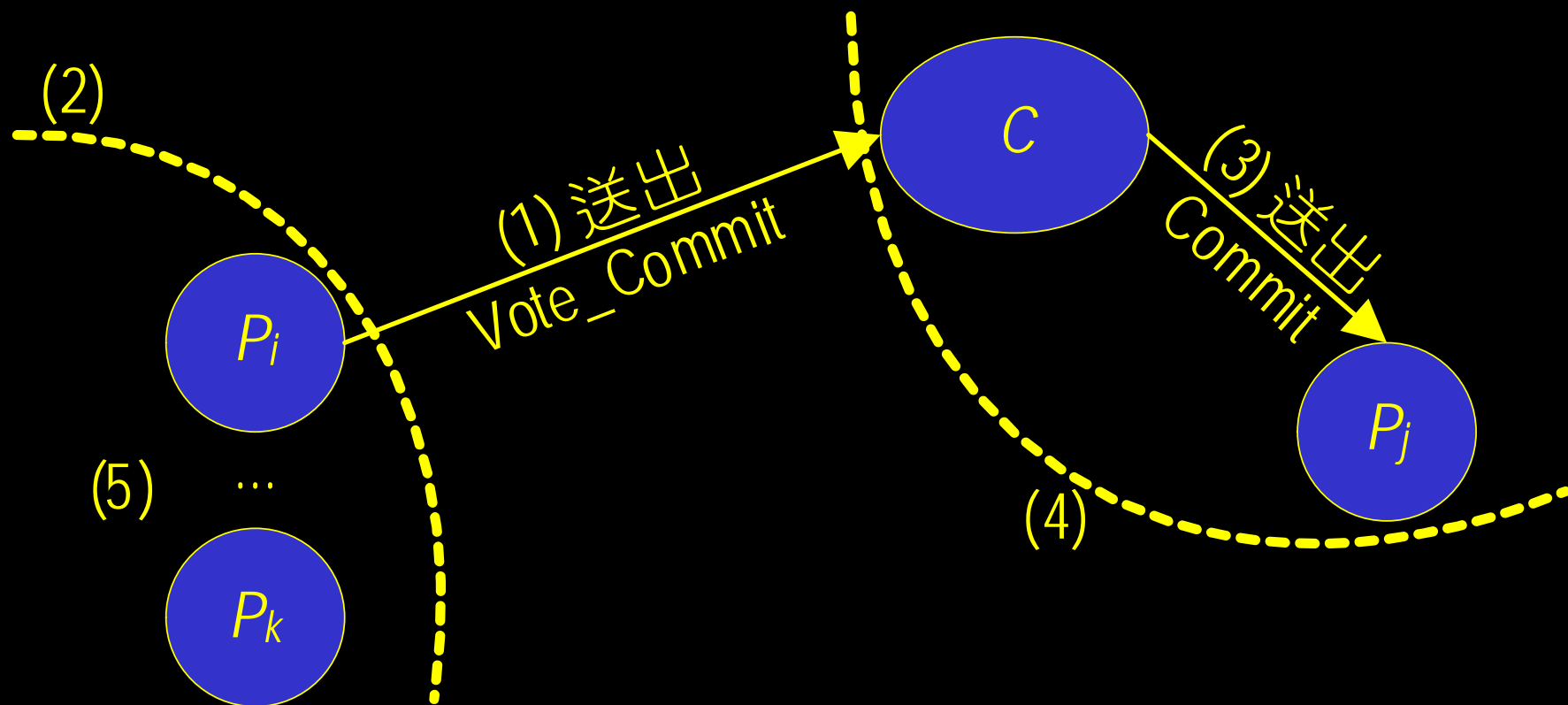
改良協調者 (Coordinator) 角色



改良參與者 (Participant) 角色



兩段式委任仍會造成 Blocking





Blocking 的問題癥結

- 我們允許任意一個參與者只要在“知道”所有其他參與者都同意 Commit 的時候，便可 Commit，但是這造成了有一些參與者不知道是否其他的參與者都同意 Commit？所以 Blocking。
- 在真實的網路運作情況下，沒有任何一種作法能完全防止 Blocking，但是我們可以儘量降低 Blocking 的可能性。



三段式委任 (Three-Phase Commit)

- 規定任何一個參與者必須要：「“知道”其他所有參與者也都“知道”大家都同意要 Commit 時，才能 Commit。」
- 如此則會減少許多 Blocking 的機會，但是效能不彰
- 礙於篇幅，煩請讀者自行找尋三段式委任的相關資料

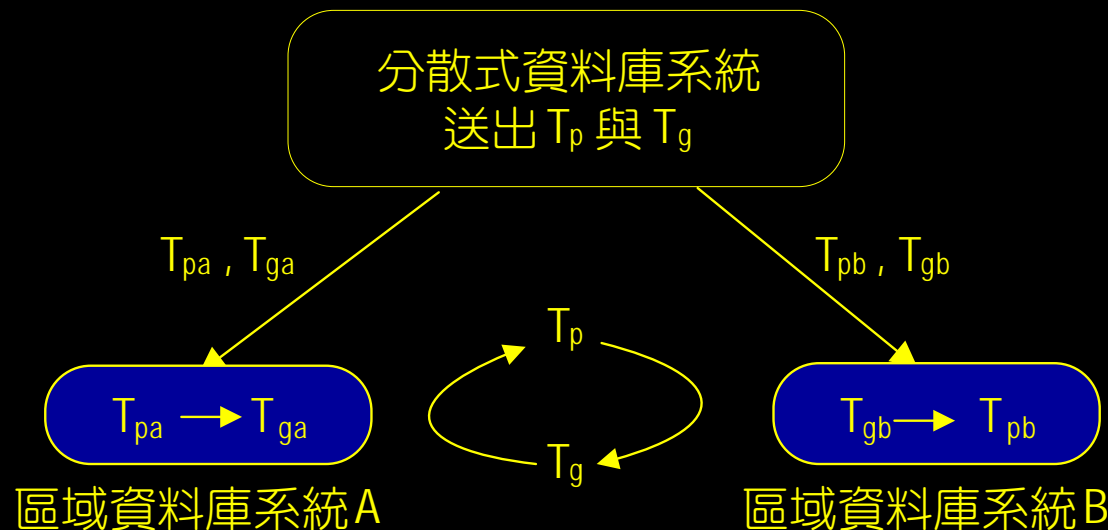


連鎖的更新問題

- 兩段式委任效能低落
- 更不用說三段式委任了！
- 有些系統對於較不敏感的資料是以線外複製 (Offline Replicate) 的方式來達成，
- 例如：SQL Server 2008 與 Sybase 支援 Replication 的作法，雖然不是同步更新，卻也只差幾秒鐘而已。
- SQL Server 2008 做法上是使用所謂的 Publication 物件來達成此項工作。

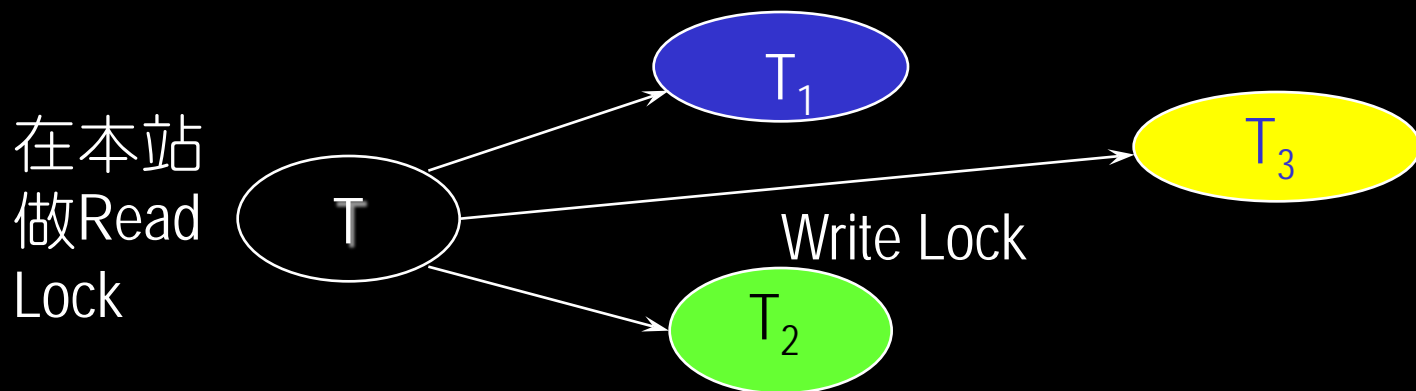
並行控制

- 除了要求同一區域資料站的子異動之間要符合「循序性」(Serializability) 以外，
- 同時也要求整體異動之間也要符合循序性。

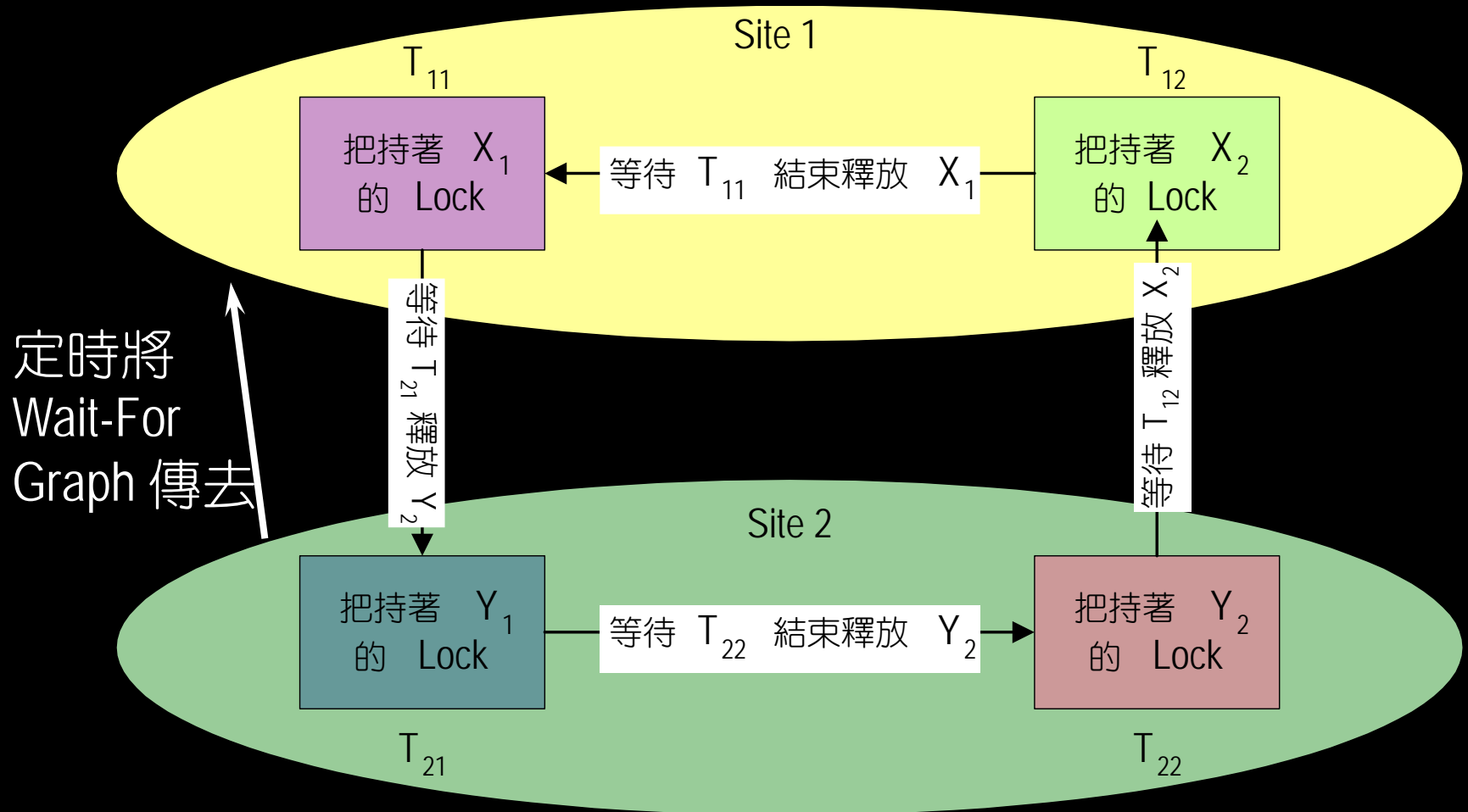


分散式環境中的資料鎖定

- 有許多的關聯表被複製，並放在不同的資料站
- 讀取時只要鎖定該資料站的那一份複本即可— Read-Lock-One
- 更新時則要鎖定所有資料站上的複本— Write-Lock-All



整體死結的偵測問題





資料的保密性與安全性控制

- 建立稽核追蹤系統 (Audit Trail System)
- 以視界 (Views) 將受保護的資料遮蔽
- 分別賦予不同的使用者不同的存取權限，在 SQL 中有 GRANT 與 REVOKE 可以達成。
- 建立「授權矩陣」 (Authorization Matrix) 供稽核人員稽查
- 只提供使用者統計數據—統計式資料庫 (Statistical Databases)



授權矩陣的例子

Data Users \	關聯表 R1	關聯表 R2	關聯表 R3	關聯表 R4
Frank	All	All	All	All
John	Select	Select	Insert	Delete
Annie	All	Select	None	Select
Mary	Select	None	Delete	Select
Mike	All	None	Insert	Delete

Grant 用法

■ GRANT {ALL [PRIVILEGES] [column_list]| ...
ON {table_name [(column_list)] | view [(column_list)]
| stored_procedure}
TO {PUBLIC | name_list}
[WITH GRANT OPTION] 可透過他再授權給別人

人



Revoke 用法

■ REVOKE {[GRANT OPTION FOR]
ALL [PRIVILEGES] | *permissions*} [(*columns*)] ON
{*table_name* [(*columns*)] | *view_name* [(*columns*)]
| *stored_procedure* / *extended_store_procedure*}
FROM {PUBLIC | *name_list*}
[CASCADE] 將所有透過他再授權的權利取回





Grant 與 Revoke 範例

- Grant All on Table R1 to Annie;
- Grant Select on Table R2 to Annie;
- Grant Select on Table R4 to Annie;
- Revoke All on Table R3 from Annie;



統計資料庫 (Statistical Databases)

- 資料庫中包含了許多機密的個人資料，
- 使用者只能看到整體資料庫中的統計數據，無法看到某個人的詳細資料。
- 例如：使用者只能查詢公司的平均薪水以及公司人數等，不能查詢某位員工的薪水。
- 注意：此種方式卻無法防止聰明的有心人窺探個別的資料記錄，或者透過推算導出某一筆資料的值。

破解統計式資料庫範例

Employees

<i>name</i>	<i>gender</i>	<i>specialty</i>	<i>salary</i>
Andy	M	Programming	55K
Clark	M	Programming	38K
John	M	Programming	38K
Tom	M	Programming	36K
Annie	F	Programming	50K
Betty	F	Testing	49K
Doris	F	Planning	59K
Frances	F	Accounting	65K
Grace	F	Training	45K
Shirley	F	Marketing	48K



破解統計式資料庫範例

- 由於關聯表中只有一個負責程式設計 (Specialty 為 Programming) 的員工是女性 (F) 所以

```
Select count(*)  
From Employees  
Where Gender = 'F' AND Specialty =  
'Programming'  
可得到答案為 1。
```



破解統計式資料庫範例

- 如果

```
Select sum(Salary)
From Employees
Where Gender = 'F' AND Specialty = 'Programming'
```

則可以得到答案為 50K，也就是說 Annie 的薪資便一覽無遺了。



解決上述的問題

- 限制像上述會取得單一答案的查詢，
- 只有當答案會超過某個值 C 以上 (例如：3 以上) 才回答。
- 也就是說：如果查詢結果只有一個則系統要拒絕回答
- 不過，即使做如此的限制，我們仍然可以由反面的查詢得到同樣的結果，



上述解法仍可以破解

- 假定：
Select count(*) From Employees 答案為 10。
- 如果
Select count(*) From Employees
Where NOT(Gender = 'F' AND Specialty =
'Programming')
得到答案為 9，則可知負責程式設計的女性
員工只有一位，因為 $10 - 9 = 1$ 。



上述解法仍可以破解 (續)

- 如果

Select sum(Salary) From Employees 答案為 483K，
所有員工的新水總合是 483K。

- 接下來，如果

Select sum(Salary) From Employees
Where NOT(Gender = 'F' AND Specialty =
'Programming')

答案為 433K，則負責程式設計的女性員工
薪水是 50K，因為 $483K - 433K = 50K$ 。



更進一步修正

- 更進一步限制答案的筆數要落在 $C \leq A \leq T - C$ 之間才回答
- 不過，道高一尺魔高一丈，即使如此限制，我們可以發現利用集合交集的特性，在不違反上述規定的情況下來導出 Annie 的個人資料落在下面集合中

Set(Gender = 'F' AND Specialty = 'Programming')



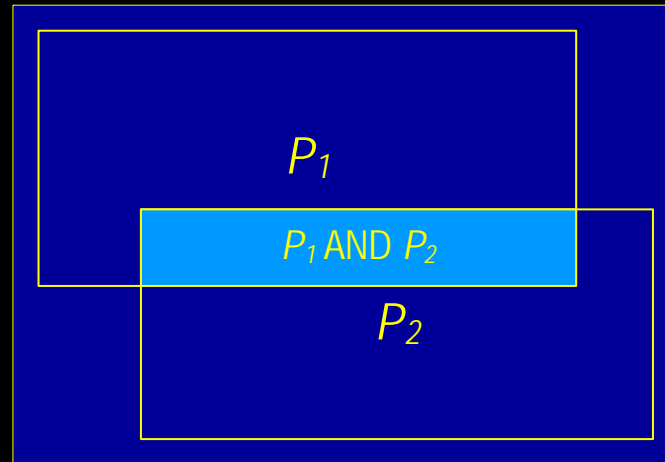
Where is the beef ?

- $P1 = \text{「Gender = 'F'」}$
 $P2 = \text{「Specialty = 'Programming'」}$ ，則
- $\text{Set}(P1 \text{ AND } P2) = \text{Set}(P1) - \text{Set}(P1 \text{ AND NOT } P2)$ 。
- 也就是說：Annie 的資料值組為
- $\sigma_{P1 \text{ AND } P2}(\text{Employees})$
 $= \sigma_{P1}(\text{Employees}) - \sigma_{P1 \text{ AND NOT } P2}(\text{Employees})$



Here you are !!!

整個關聯表 R



Select count(*)
From Employees
Where Gender = 'F'

答案是 6

Select Count(*)
From Employees
Where NOT(Specialty = 'Programming')

答案是 5



所以我們可以知道...

- `Select count(*) From Employees`
`Where Gender = 'F' AND Specialty = 'Programming'`
的結果為 1。所以，只要再下達
 - `Select sum(Salary)`
`From Employees`
`Where Gender = 'F'`
 - `Select Sum(Salary)`
`From Employees`
`Where not(gender = 'F' and specialty = 'Programming')`
- 得到結果分別為 316K 與 266K，所以符合 `Gender = 'F' AND Specialty = 'Programming'` 的人 (只有 Annie 一人而已) 之薪資總合為 50K



單一追蹤器與通用追蹤器

- 查詢條件可以用來追查某一特定值組的內容，我們稱之為「單一追蹤器」(Single Tracker)。如
Gender = 'F' AND NOT(Specialty = 'Programming')
- 「通用追蹤器」(General Tracker)，可以用來追蹤任何個別值組的內容。
- 通用追蹤器幾乎可以在各種資料庫中找到，而且也不是很難猜到，
- 此乃統計式資料庫的一大隱憂。



跨資料庫的查詢功能

- 由光碟中我們可以直接 “附加” (Attach) BOB 與 BSB (列於第八章) 兩個資料庫，所以我們可以實作一下跨越兩個資料庫做合併的動作 (假設目前的工作資料庫為 BOB)：

Select B.*, O.*, S.*

From Books as B, BSB.[Order] as O,
BSB.Bookstores as S

Where B.id = O.id and O.no = S.no



跨越伺服器的資料查詢

- 要跨越資料庫查詢時，只要使用類似 *relation.attribute* 的語法往前擴大成 *database.dbo.relation.attribute* 即可。
- 這種語法甚至還可以擴大成跨越不同伺服器：*server_name.database.dbo.relation.attribute* 的語法。



跨伺服器的分散式查詢

- 使用 `sp_addlinkedserver`、`sp_addlinkedsrvlogin` 與 `sp_serveroption`
- 使用 `sp_addlinkedserver` 建立連結伺服器之後，即可透過 OLE DB 資料來源對該伺服器執行分散式查詢
- 如果連結伺服器為 SQL Server 的執行個體，還可以執行遠端預儲程序。
- 由於 SQL Server 2008 的安全性設定非常嚴格，所以要先執行以下指令，才能讓下述指令正常執行：

```
sp_configure 'show advanced options', 1; reconfigure  
go  
sp_configure 'Ad Hoc Distributed Queries', 1; reconfigure  
go
```



跨伺服器的分散式查詢

- 假設有兩部 SQL Servers，一部稱為 Kidlab (裡面包含 BOB 資料庫)，另一部稱為 Kingbird (裡面包含 BSB 資料庫)，而我們目前位於 Kidlab，工作資料庫為 BOB
- 先將工作資料庫切換到 Master 資料庫，然後將 Kingbird 加入 Kidlab 的伺服器連結清單：

```
if exists (select name from sys.servers where name = 'Kingbird')
exec sp_dropserver 'Kingbird', 'droplogins'
go
sp_addlinkedserver @server = 'Kingbird', @srvproduct = '',
    @provider = 'SQLNCLI', @datasrc = 'Kidlab', @catalog = 'BSB'
go
```



將 Kingbird 換成其它主機的作法

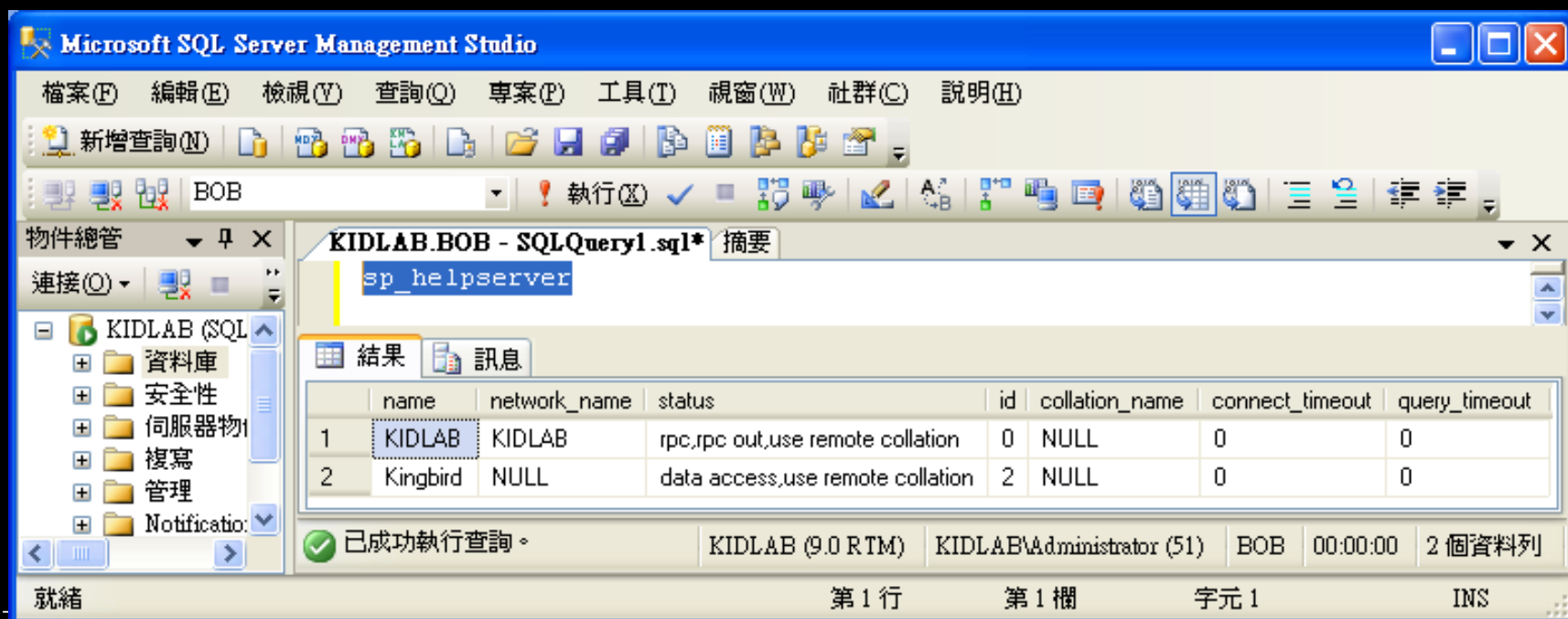
- 若要將 Kingbird 換成另一部主機的 SQL Server 2008，則將 @datasrc 參數換成該主機 IP 位址 (@provider 參數也可以更換成其它，如：SQLOLEDB)：

```
sp_addlinkedserver @server = 'Kingbird',  
@srvproduct = '',  
@provider = 'SQLOLEDB', @datasrc = 'ip_address',  
@catalog = 'BSB'  
Go
```

- 其中的 *ip_address* 為 Kingbird 的 IP 網路位址

查看所有連結的伺服器名稱

- Kingbird 會被加到 sys.servers 系統表格中。
- 可以使用 sp_helpserver 查看Kingbird 是不是在裏面？如圖所示：





接著設定資料存取的選項

- 使用 `sp_serveroption` 將資料存取的選項變成 TRUE 或是設定 Timeout 時間，如下：
- `sp_serveroption 'Kingbird', 'rpc', 'true'`
`go`
`sp_serveroption 'Kingbird', 'rpc out', 'true'`
`go`
`sp_serveroption 'Kingbird', 'connect timeout', 120`
`go`
`sp_serveroption 'Kingbird', 'query timeout', 120`
`go`



跨伺服器的分散式查詢

- 假設在 Kidlab 伺服器上, 工作資料庫為 BOB
- 以下指令可以直接對 Kidlab 的 BOB.dbo.Books 與 Kingbird 上的 BSB.dbo.[order] 與 BSB.dbo.Bookstores 做合併查詢

- `Select B.*, O.*, S.*`

`From Books as B, Kingbird.BSB.dbo.[Order] as O,`

`Kingbird.BSB.dbo.Bookstores as S`

`Where B.id = O.id and O.no = S.no`



密碼設定問題

- 如果發生 'Login failed' 的問題，建議你將兩部伺服器的使用者名稱、密碼都改成一樣的，或是使用 sp_addlinkedsrvlogin 輸入另一位使用者的 Username 與 Password 再連結一次 (以 sa 這個使用者的 password 是 xxxxxx 為例)：

```
sp_addlinkedsrvlogin  
    @rmtsrvname = 'Kingbird',  
    @useself = 'false',  
    @rmtuser = 'sa',  
    @rmtpassword = 'xxxxxx'
```

go



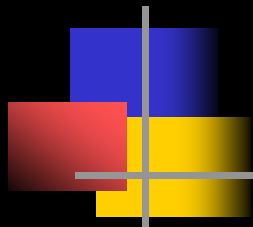
使用 OpenRowset() 函數

- 將光碟中的 Access 資料庫 BSB.mdb 複製到 'C:\' 下
- 以 Access 資料庫為例，利用 OpenRowset() 函數達成
`select B.*, S.* from Bob..Bookstores as B Inner Join
OPENROWSET('Microsoft.Jet.OLEDB.4.0', 'C:\BSB.mdb';
'admin'; ", [order]) as S
on B.no = S.no`
- 或是單純擷取該資料庫中的 Books 表格：
`select * from Openrowset('Microsoft.Jet.OLEDB.4.0',
'C:\BSB.mdb'; 'admin'; ", Books)`
- 不管任何形式的資料表, 只要具備 OLE DB 的 Driver,
SQL Server 都可以與之合併查詢，請同學上機練習...



跨越伺服器的異動

- USE BOB
- GO
- **BEGIN DISTRIBUTED TRANSACTION**
Update Kingbird.bsb.[order]
SET quantity = 0 WHERE id = 3
Update Kidlab.BOB.Orders set quantity = 0 where id = 3
COMMIT TRANSACTION
- GO
- 要使用 Begin Distributed Transaction 與 Commit Transaction 將分散式異動夾起來



本章結束
The End.