Distributed Systems

Chun-Feng Liao

廖峻鋒

Department of Computer Science

National Chengchi University

Distributed Systems

CQRS/Event Sourcing

Chun-Feng Liao

廖峻鋒

Dept. of Computer Science

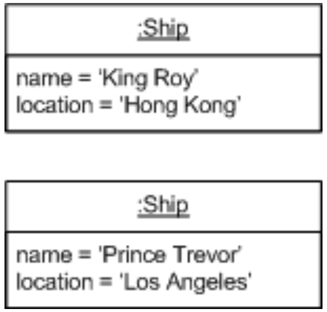
National Chengchi University

大綱

- Technology
- Discussion
- Case
- Conclusion

Event Sourcing: The idea

- 動機
  - 一般來說，我們會儲存應用程式的「目前狀態」
  - 有些時候，也會需要知道系統「如何演變成目前狀態」
    - 狀態變遷歷程
- 範例
  - 除了郵輪所在地，我們也想知道它們的航行歷程
    - King Roy: San Francisco → Hong Kong
    - Prince Trevor: Taipei → Los Angeles

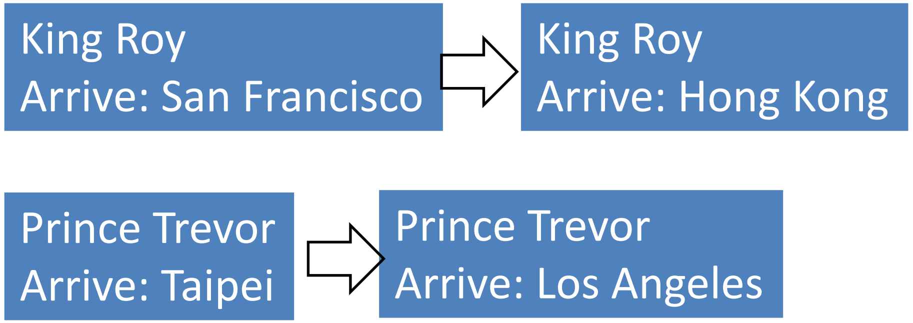


只記錄目前狀態，看不出航行歷程

範例來源: Fowler, M. (2005) https://martinfowler.com/eaaDev/EventSourcing.html

Event Sourcing: The idea

- 新的資料儲存方式
  - 將「每次資料異動」存為一個事件(Event)



King Roy
Arrive: San Francisco

King Roy
Arrive: Hong Kong

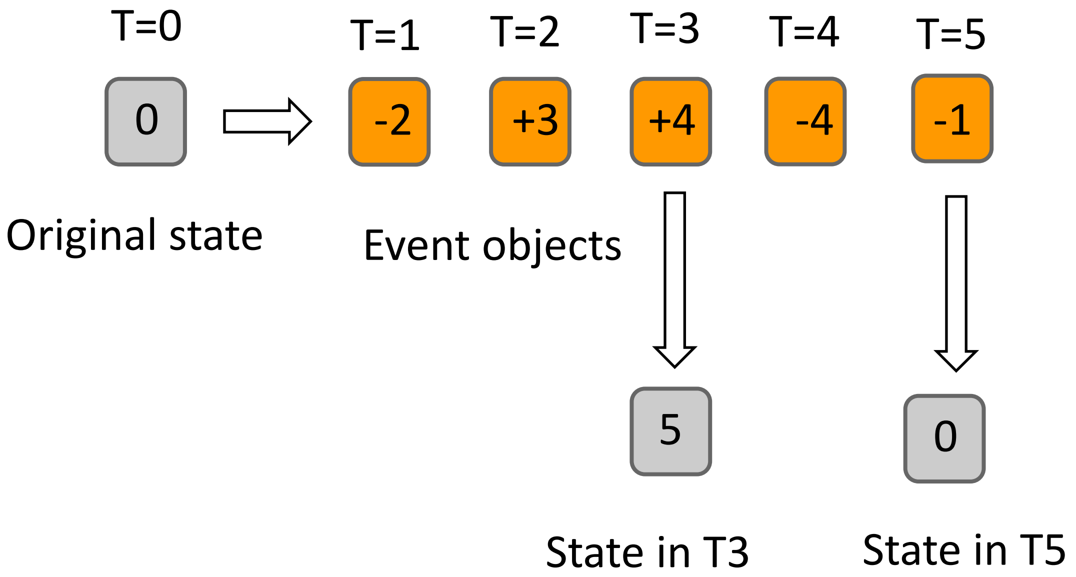
Prince Trevor
Arrive: Taipei

Prince Trevor
Arrive: Los Angeles

記錄狀態變遷的歷史，可以看出二艘船的航行路線

Event Sourcing: Key Concept

- 應用程式狀態<u>改變</u>存為events
  - Events依時間形成有序資料
  - 由「原始狀態」匯整「異動」後，就可得到某時間點的狀態

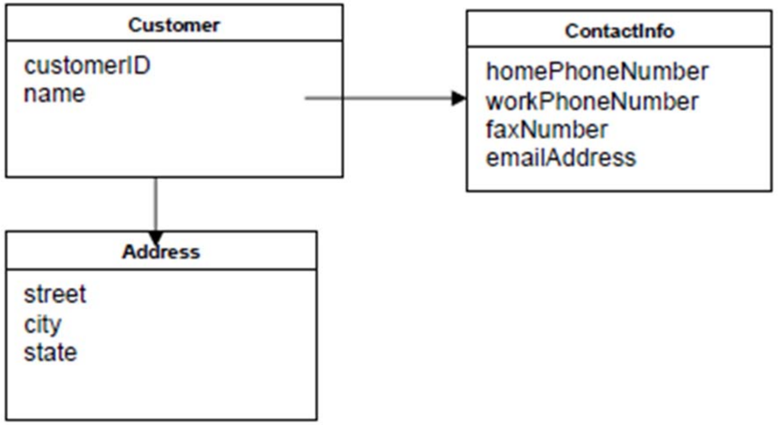


權威資料來源 (Source)

- 比較
  - Current state as official records (Source:「目前狀態」)
    - Current states held in a database
  - Event logs as official records (Source:「狀態變遷歷程」)
    - Historical events held in an event store
    - Current states can be built from them whenever needed
    - 故稱: Event Sourcing (ES)

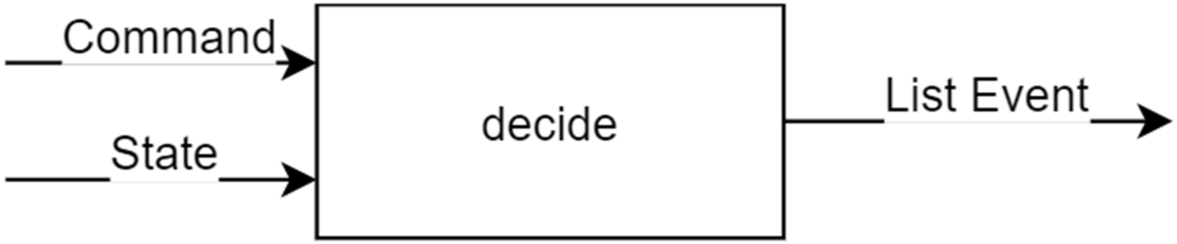
Terms

- Aggregate (一串關連緊密的領域物件)
  - 彼此有關連(relationship)，經常會一起被操作的領域物件
  - 記錄變動的基本單位
- Command
  - (可序列化的) 寫入性資料操作指令
    - Create
    - Update
    - Delete
- Query
  - (可序列化的) 讀取性的資料操作指令

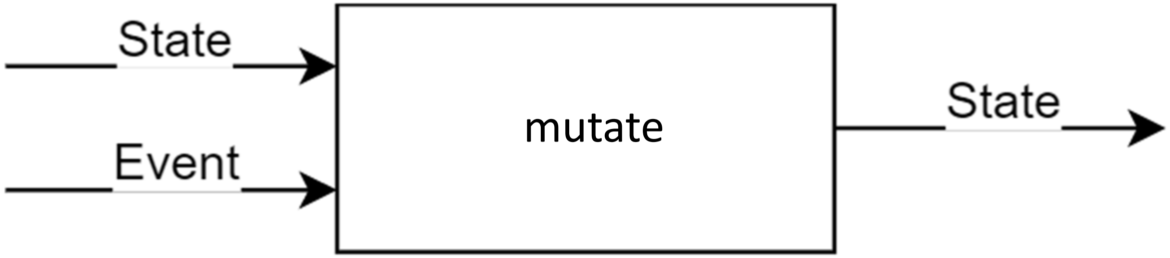


ES Core Operations

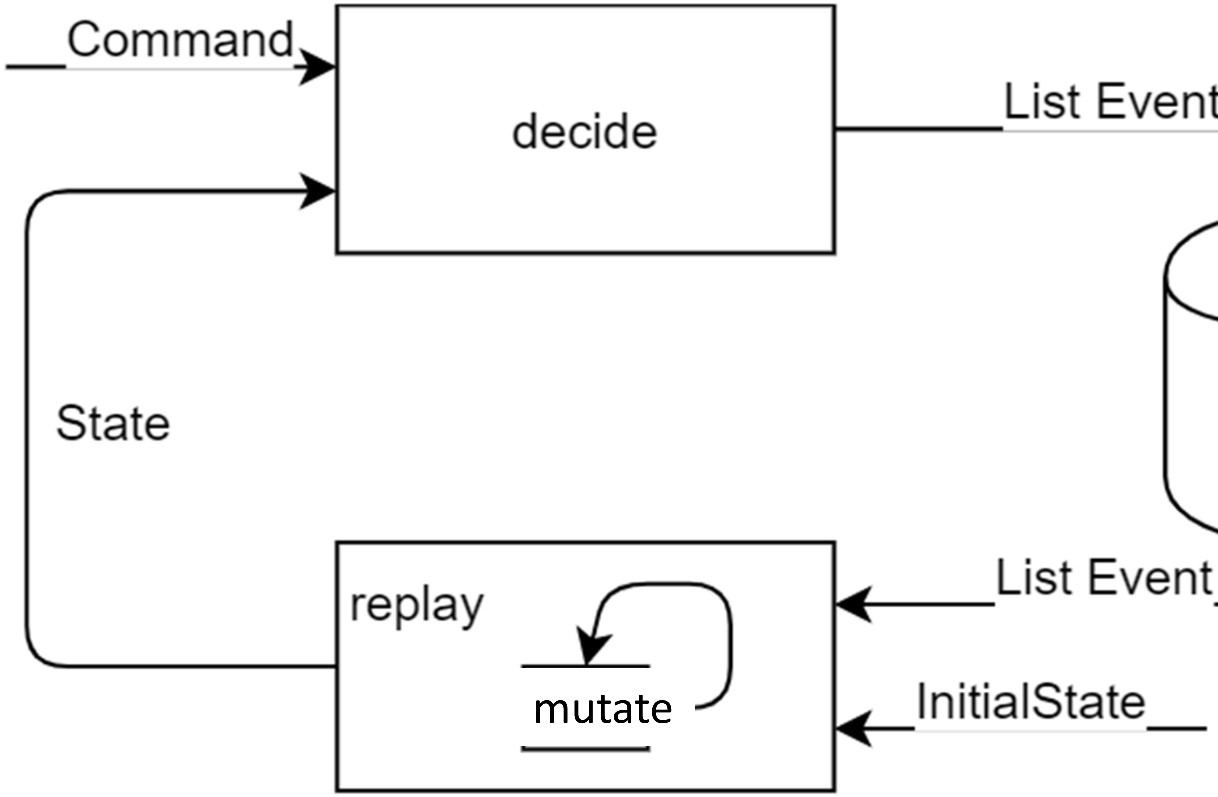
- Decide: Command轉Events



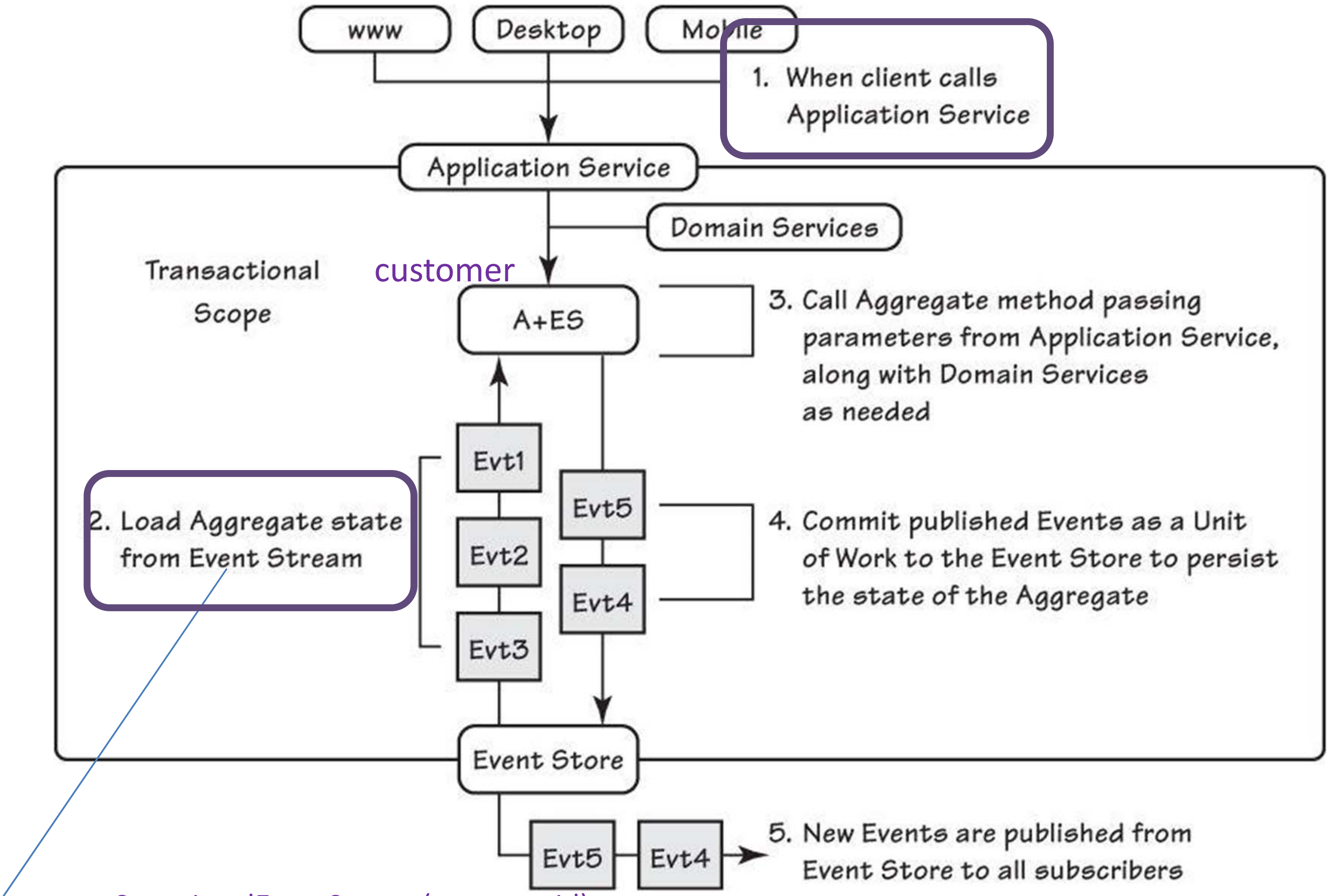
- Mutate
  - 套用event到aggregate上 (導致aggregate狀態改變)



ES Implementation (Conceptual)



Implementation: Step 1- Step 2


www
Desktop
Mobile
1. When client calls Application Service
Application Service
Domain Services
Transactional Scope
customer
A+ES
3. Call Aggregate method passing parameters from Application Service, along with Domain Services as needed
Evt1
Evt2
Evt3
Evt5
Evt4
2. Load Aggregate state from Event Stream
4. Commit published Events as a Unit of Work to the Event Store to persist the state of the Aggregate
Event Store
Evt5
Evt4
5. New Events are published from Event Store to all subscribers

```
var stream = _eventStore.LoadEventStream(customerId);
var customer = new Customer(stream.Events);
```

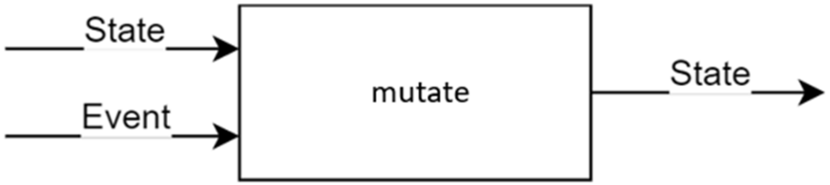
此處材料取自V. Vernon (2013) Implementing Domain-Driven Design

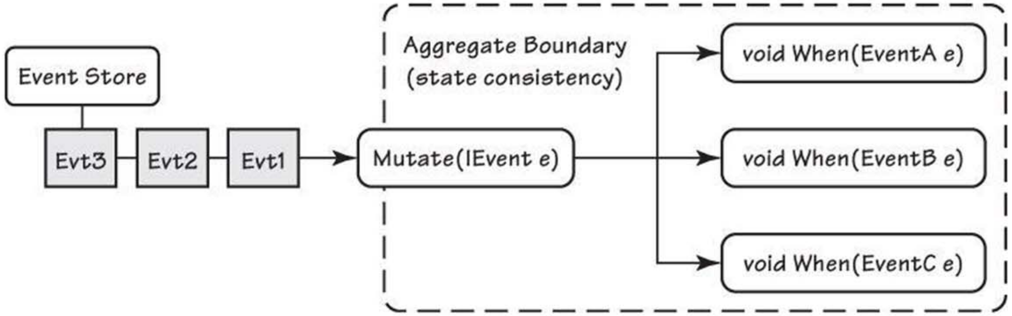
Implementation: Step 1- Step 2

```
public partial class Customer
{
  public Customer(IEnumerable<IEvent> events)
  {
    // reinstate this aggregate to the latest version
    foreach (var @event in events)
    {
      Mutate(@event);
    }
  }

  public bool ConsumptionLocked { get; private set; }

  public void Mutate(IEvent e)
  {
    // .NET magic to call one of 'When' handlers with
    // matching signature
    ((dynamic) this).When((dynamic)e);
  }

  public void When(CustomerLocked e)
  {
    ConsumptionLocked = true;
  }

  public void When(CustomerUnlocked e)
  {
    ConsumptionLocked = false;
  }

  // etc.
```

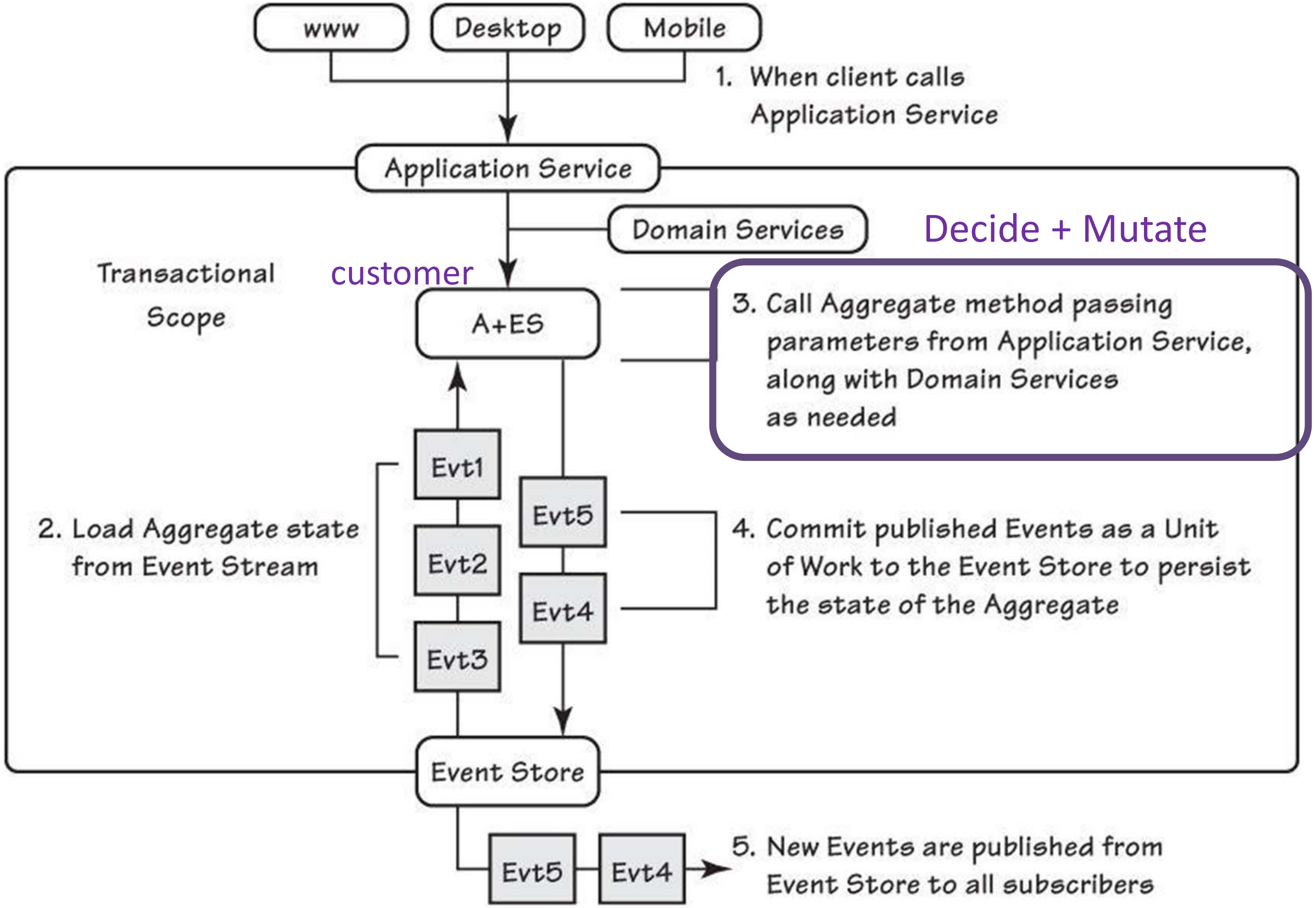
var customer = new Customer(stream.Events);

Mutate: **依據進來的事件改變物件本身狀態**

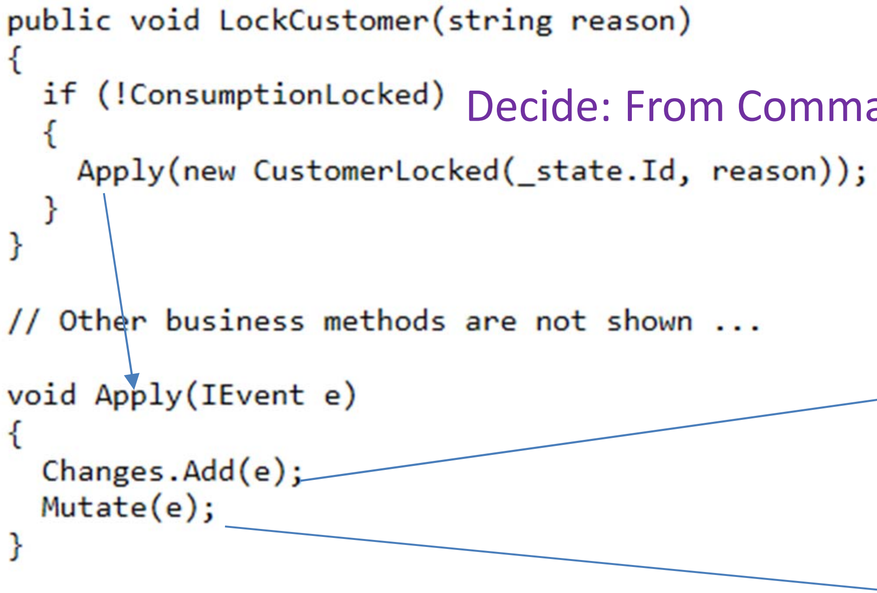




Implementation: Step 3

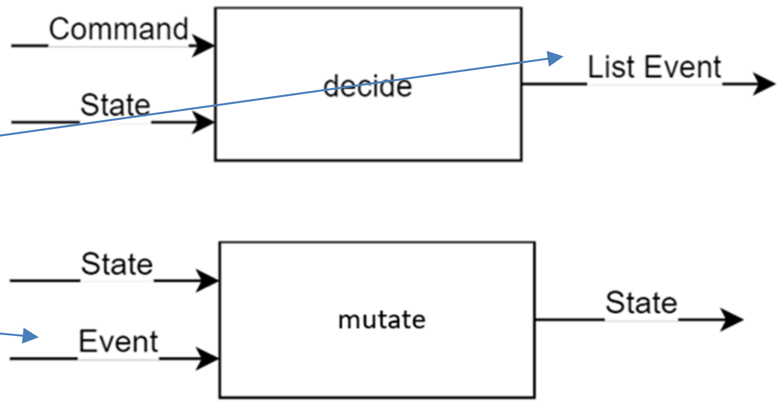

www
Desktop
Mobile
1. When client calls Application Service
Application Service
Domain Services
Decide + Mutate
Transactional Scope
customer
A+ES
3. Call Aggregate method passing parameters from Application Service, along with Domain Services as needed
Evt1
Evt2
Evt3
2. Load Aggregate state from Event Stream
Evt5
Evt4
4. Commit published Events as a Unit of Work to the Event Store to persist the state of the Aggregate
Event Store
Evt5
Evt4
5. New Events are published from Event Store to all subscribers

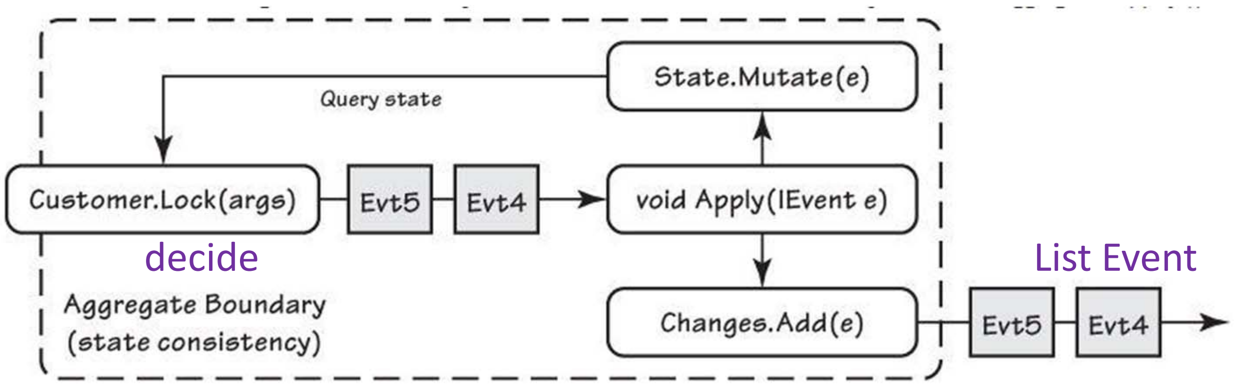
Implementation: Step 3



```
public void LockCustomer(string reason)
{
  if (!ConsumptionLocked)
  {
    Apply(new CustomerLocked(_state.Id, reason));
  }
}

// Other business methods are not shown ...

void Apply(IEvent e)
{
  Changes.Add(e);
  Mutate(e);
}
```

Decide: From Command to Events

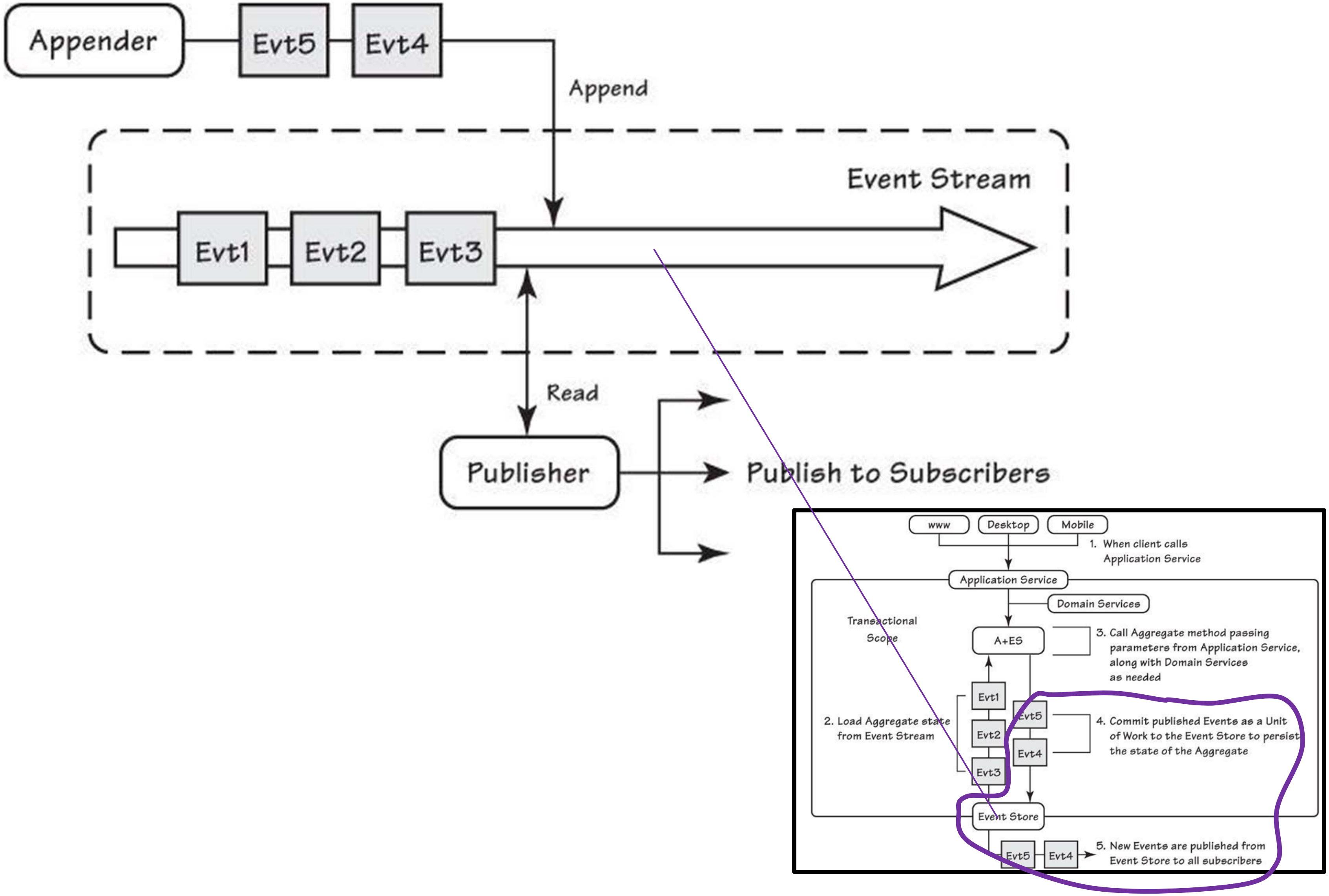




decide

List Event

Implementation: Step 4 and 5



1. When client calls Application Service

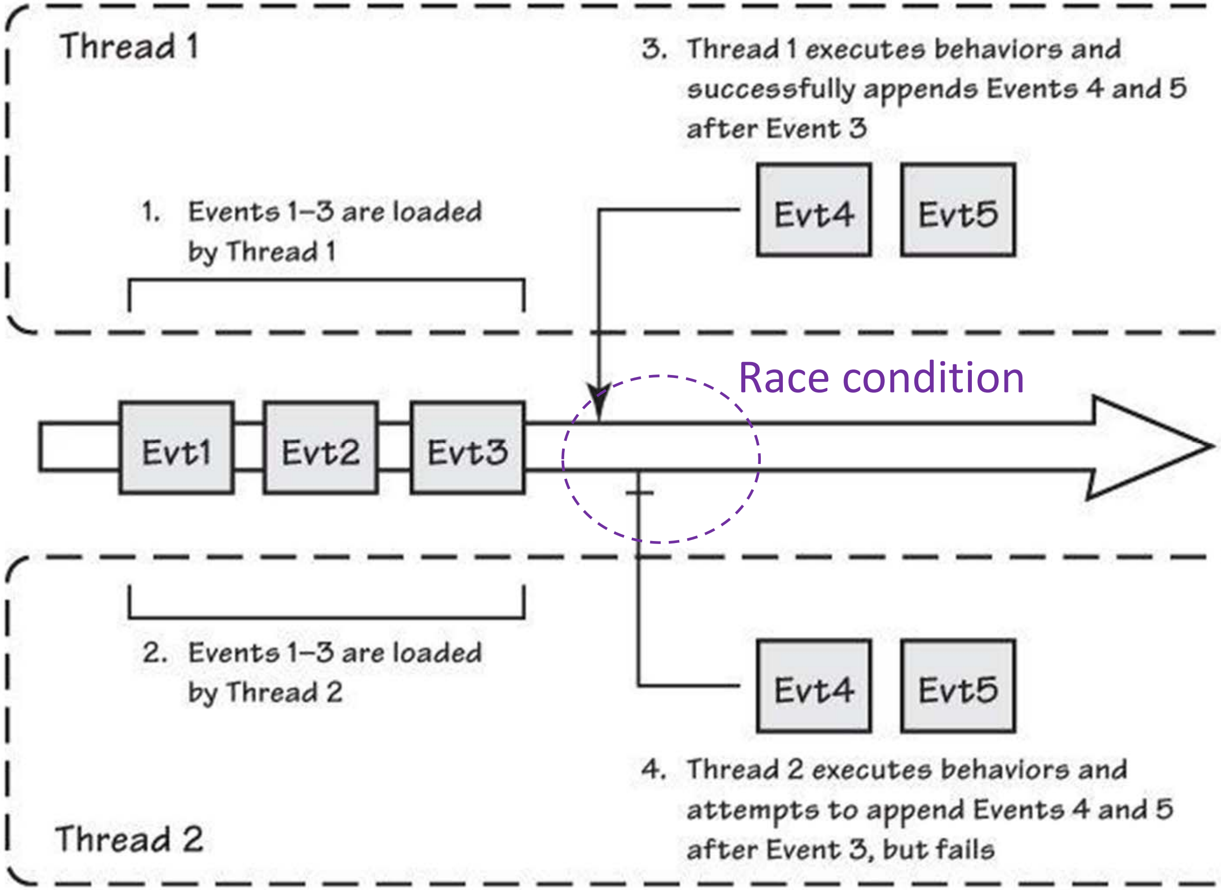
2. Load Aggregate state from Event Stream

3. Call Aggregate method passing parameters from Application Service, along with Domain Services as needed

4. Commit published Events as a Unit of Work to the Event Store to persist the state of the Aggregate

5. New Events are published from Event Store to all subscribers

Implementation: Step 4 and 5

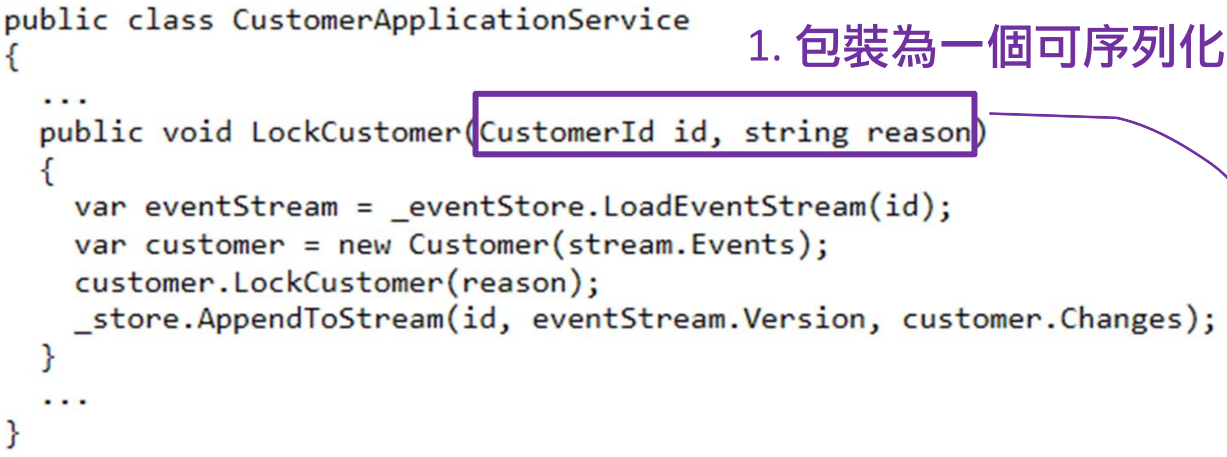

Thread 1
3. Thread 1 executes behaviors and successfully appends Events 4 and 5 after Event 3
Evt4
Evt5
1. Events 1–3 are loaded by Thread 1
Race condition
Evt1
Evt2
Evt3
2. Events 1–3 are loaded by Thread 2
Evt4
Evt5
4. Thread 2 executes behaviors and attempts to append Events 4 and 5 after Event 3, but fails
Thread 2

這裡使用樂觀鎖: 寫入時檢查store最新版是否為之前load的版本

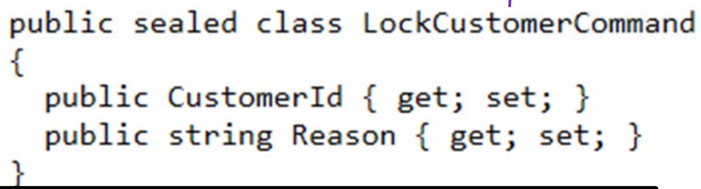
若不是就寫入失敗，要再re-load並重來 (考慮一下失敗的代價)

Command Handler

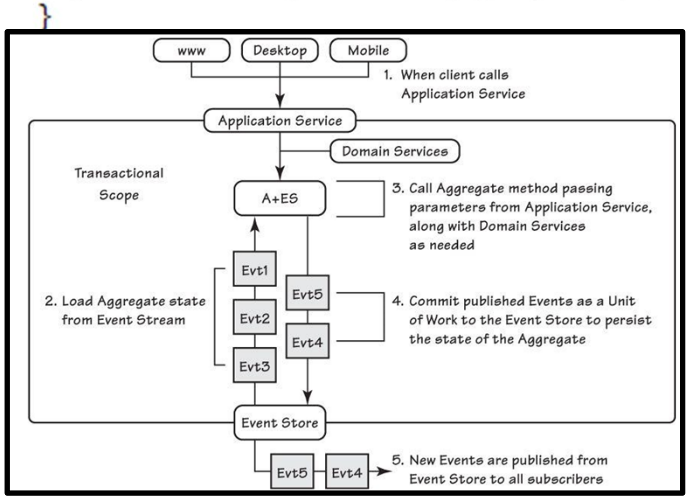
1. 包裝為一個可序列化的類別 (Command)



```
public class CustomerApplicationService
{
  ...
  public void LockCustomer(CustomerId id, string reason)
  {
    var eventStream = _eventStore.LoadEventStream(id);
    var customer = new Customer(stream.Events);
    customer.LockCustomer(reason);
    _store.AppendToStream(id, eventStream.Version, customer.Changes);
  }
  ...
}
```



```
public sealed class LockCustomerCommand
{
  public CustomerId { get; set; }
  public string Reason { get; set; }
}
```



www
Desktop
Mobile
1. When client calls
Application Service
Application Service
Domain Services
Transactional
Scope
A+ES
3. Call Aggregate method passing
parameters from Application Service,
along with Domain Services
as needed
Evt1
Evt5
2. Load Aggregate state
from Event Stream
Evt2
Evt4
4. Commit published Events as a Unit
of Work to the Event Store to persist
the state of the Aggregate
Evt3
Event Store
Evt5
Evt4
5. New Events are published from
Event Store to all subscribers

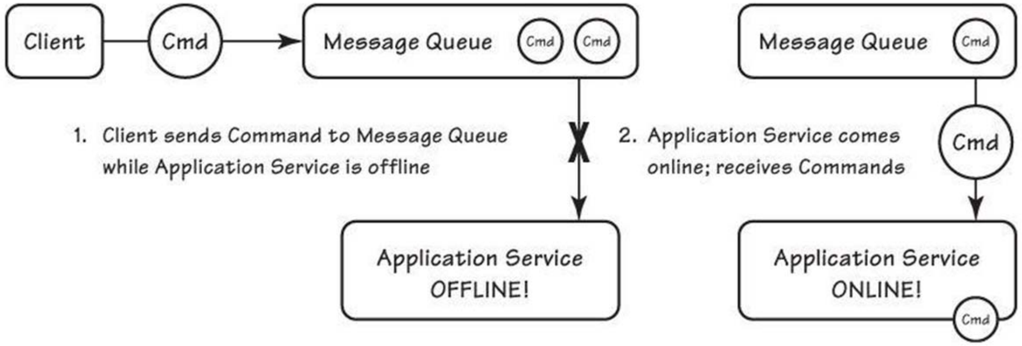
2. Client改透過MQ提交序列化後的Command物件，來觸發update程序

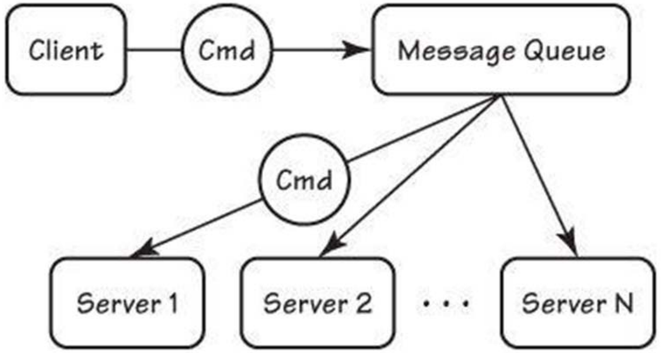
3. 此時Application Service變成Command Handler

Command Handler有什麼好處?

- Robust to temporary failure
  - MQ as a buffer
- 可實現Load balance
- 可實現Circuit Breaker
  - Deal with failures in a single location
  - Retry / resend management
- 可實現Decorator
  - 可彈性增減居中處理邏輯
  - Audit、Logging、Validation…

Message Buffer

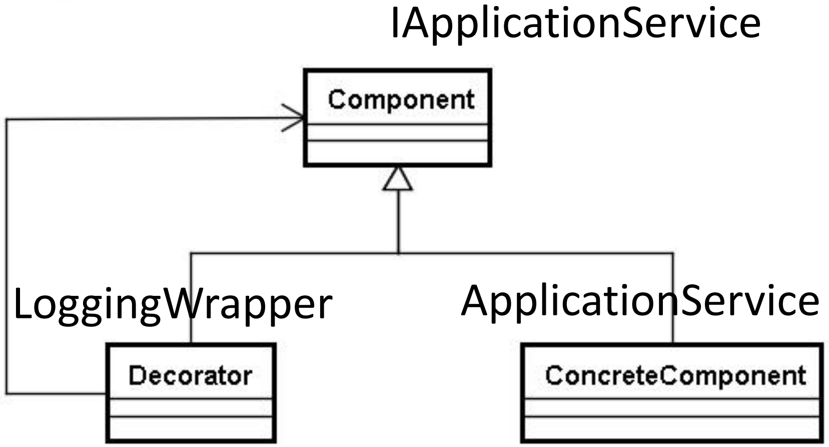

Client
Cmd
Message Queue
Cmd
Cmd
Message Queue
Cmd
1. Client sends Command to Message Queue while Application Service is offline
2. Application Service comes online; receives Commands
Cmd
Application Service OFFLINE!
Application Service ONLINE!
Cmd



Load balance

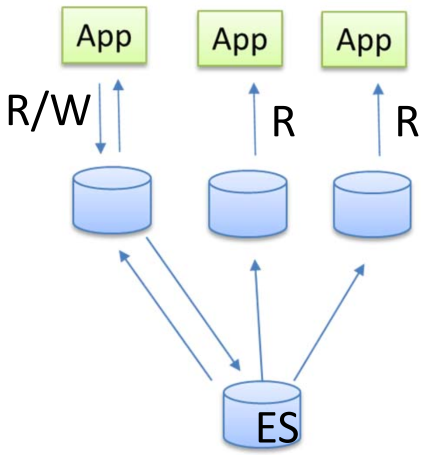
Decorator

```
public class LoggingWrapper : IApplicationService
{
  readonly IApplicationService _service;

  public LoggingWrapper(IApplicationService service)
  {
    _service = service;
  }

  public void Execute(ICommand cmd)
  {
    Console.WriteLine("Command: " + cmd);
    try
    {
      var watch = Stopwatch.StartNew();
      _service.Execute(cmd);
      var ms = watch.ElapsedMilliseconds;
      Console.WriteLine("  Completed in {0} ms", ms);
    }
    catch( Exception ex)
    {
      Console.WriteLine("Error: {0}", ex);
    }
  }
}
```



ES Impact

- 得到的新功能
  - Complete rebuild
    - 最新狀態=初始狀態+所有改變 (改變都存在events中)
  - Temporal query
    - 查詢任何時間點的狀態
    - 建構特定時間點的狀態
- 提升的軟體品質
  - Scalability
    - 讀: 建構大量用完即棄的查詢副本(cache)
    - 寫: 事件immutable→只能insert不能update
      - (相較一般資料庫用法) 能承受更多寫入
  - Robustness
    - 將出錯的process/repository直接移除，資料重新依event store建置即可



Comparison

- 比較: 集中式資料架構
  - Pros
    - Simplicity: design, maintain (backup and restore)
    - Consistency: single source of facts
  - Cons
    - Repository becomes performance bottleneck
      - More transactions → more locks→ lower concurrency
    - Hard to scale
    - Single point of failure

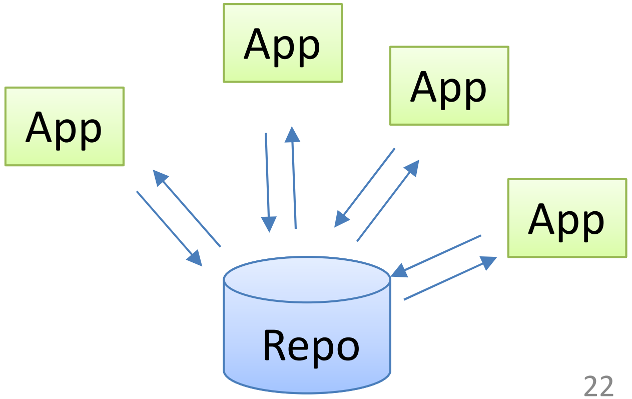
ES: Complex

ES: Eventually Consistent

ES: higher concurrency

ES: scalable

ES: robust



Discussion: ES適用場合

- When ES is a natural choice?
  - Audit log
  - Debugging
  - Scalable architecture (EDA)
    - Many readers; few writers
  - Advanced usage
    - Parallel models (需要同時保有Relational 和 De-Norm Model)
    - Retroactive events

Discussion: ES 的限制/問題

- Eventually consistency
  - 無法達成即時一致性
- Complexity of Design
  - 如何評估適合的Aggregate
  - 學習曲線
- 實作充滿陷井，欠缺通用參考實作
  - 大部份案例都是量身訂做，各式設計各有優劣
  - 地雷區
    - Concurrency
    - Schema evolution (Overeem et al. 17): The dark side of event sourcing
    - External interaction (Fowler 05): 可能造成每次event reply結果不同
    - Robustness of infrastructure (變Robust的是使用ES的AP)

CQRS/Event Sourcing工具

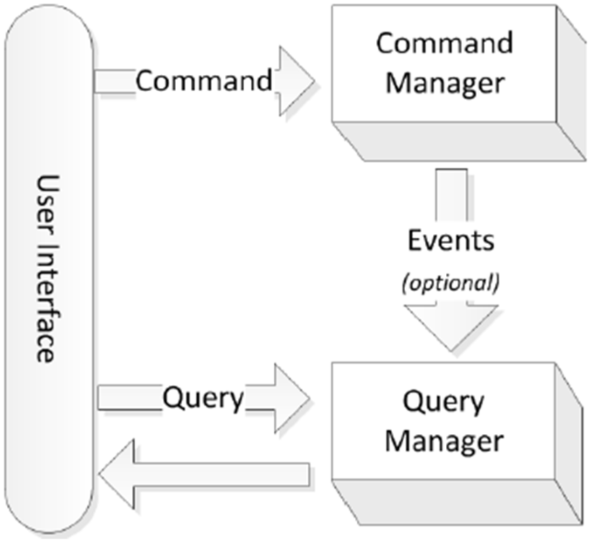
- Event Store
  - Event Store (now EventStoreDB) by Greg Young (.NET)
  - Lagom (Java or Scala)
- Application Framework
  - Axon (Java) by Allard Buijze
  - Eventuate (Java) by Chris Richardson
  - Event Sourcing in Python (Python) by John Bywater
  - reSolve (JavaScript)

Command Query Separation (CQS)

- Proposed by B. Meyer (1988)
  - in the book “Object Oriented Software Construction”
- Core idea
  - Use different methods for queries (no side effects) and commands (changes the states)
    - (Query) If you have a return value you cannot mutate state
    - (Command) If you mutate state your return type must be void
    - 例外: Stack’s pop method

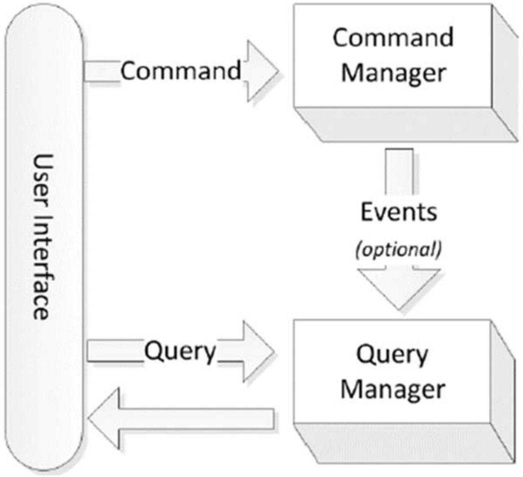
CQRS and CQS

- CQRS: a conceptual extension of CQS
  - Core: split query and command functions in distributed systems
    - Use a model (service object and/or store) to modify information
    - Use another model (service object and/or store) to read information



CQRS + Events

- CQRS and event-based programming model
  - Split services communicating with Event Collaboration
    - Allows these services to easily take advantage of Event Sourcing
    - Command model更新後，丟出changed event，query model接到更新，直接apply來更新資料 (透過event來sync資料)
  - Having separate models raises a model consistency issue
    - Only eventual consistency is possible (Query)



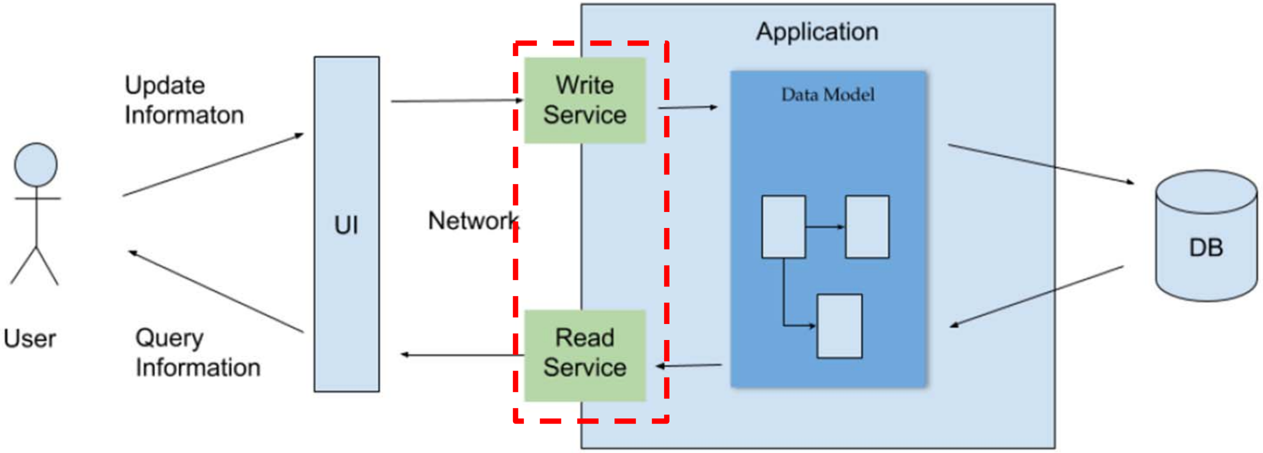
Event Collaboration: Multiple components work together by communicating with each other by sending events when their internal state changes.

CQRS Benefits

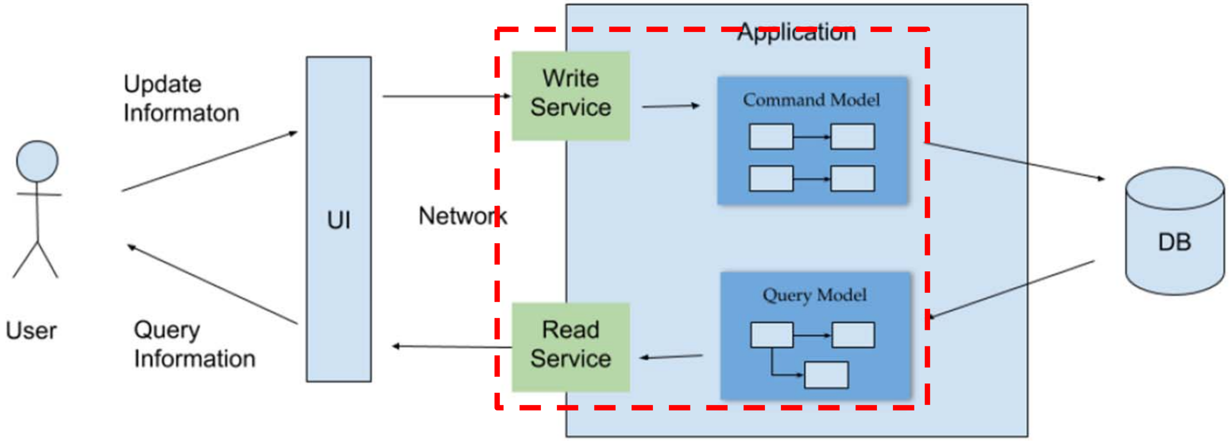
- Command和Query特性不同，區分後較好對症下藥
  - Consistency
    - Command: 著重立即的 consistency
    - Query: 通常可接受 eventually consistency
  - Data model
    - Command: store data in a more normalized way
    - Query: de-normalized way, avoid join to improve performance
  - Scalability
    - Command: 相對不重要: 一般應用程式只有較少比例在modify data
    - Query: 相對重要-經常需要serve大量query

CQRS 分隔層次

介面層次

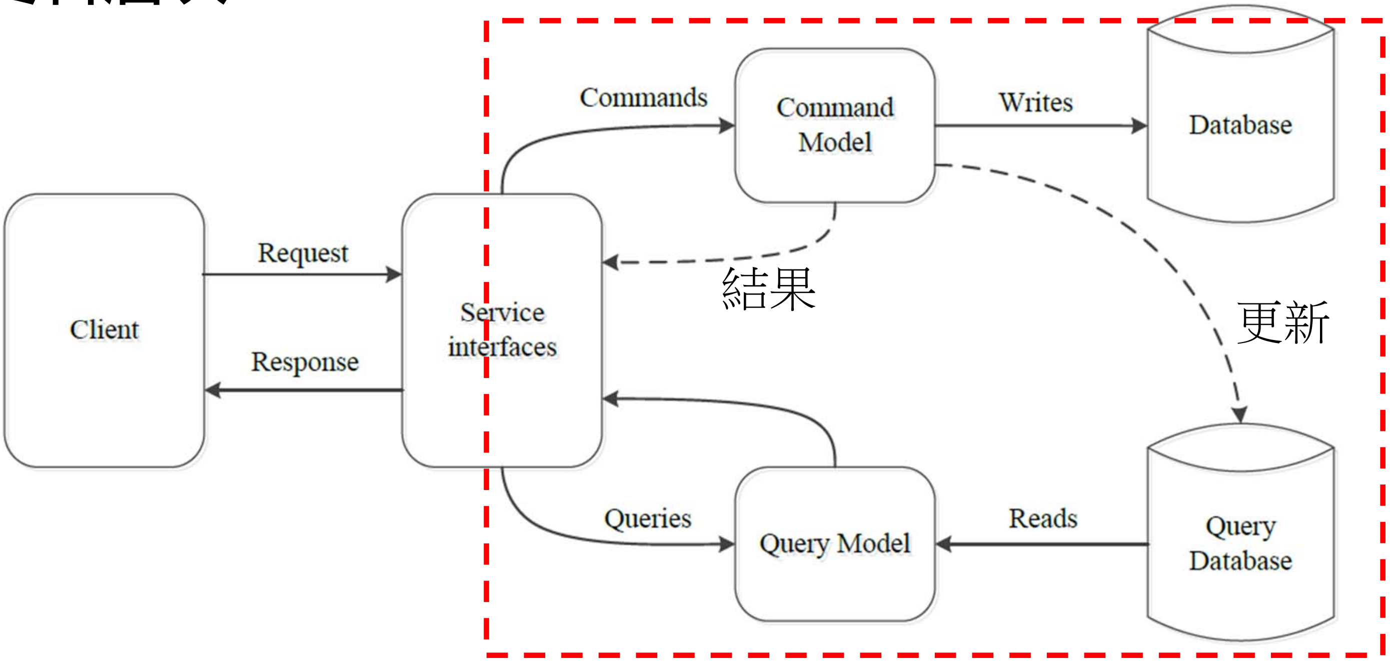


Domain層次



CQRS 分隔層次

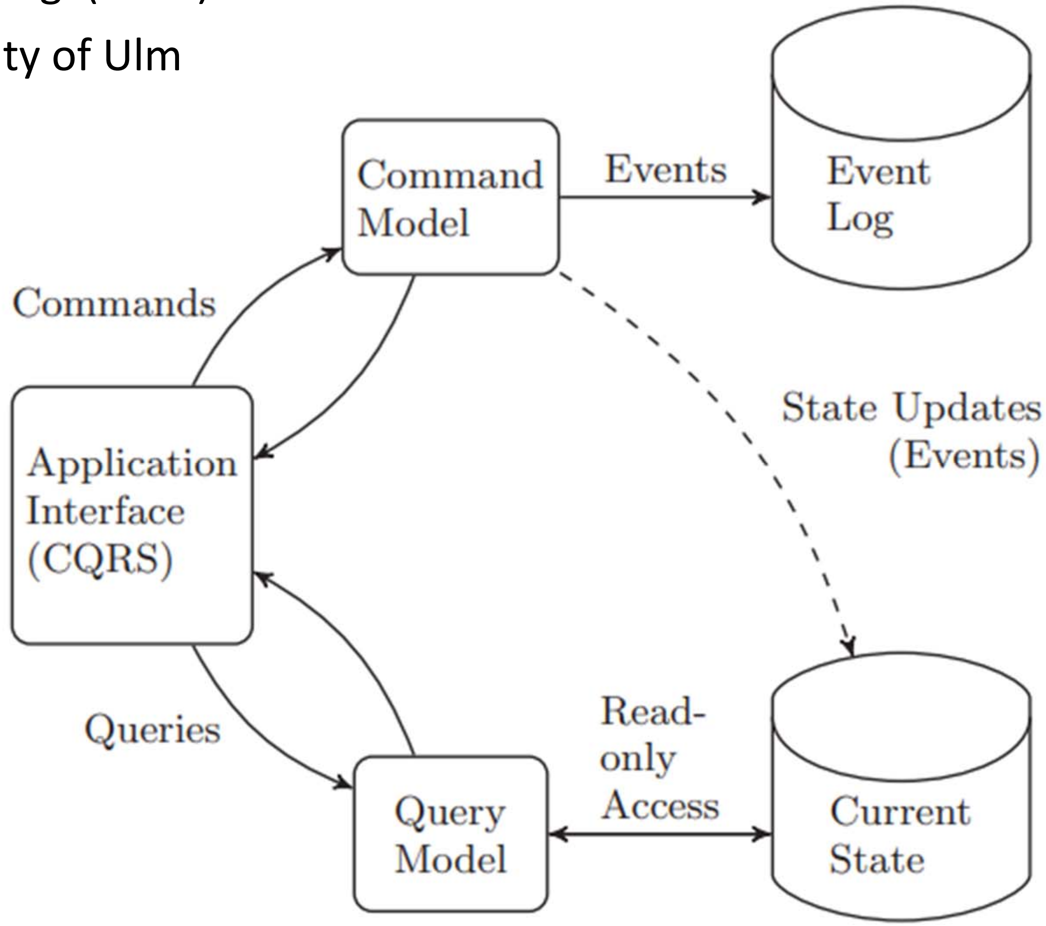
- 資料層次



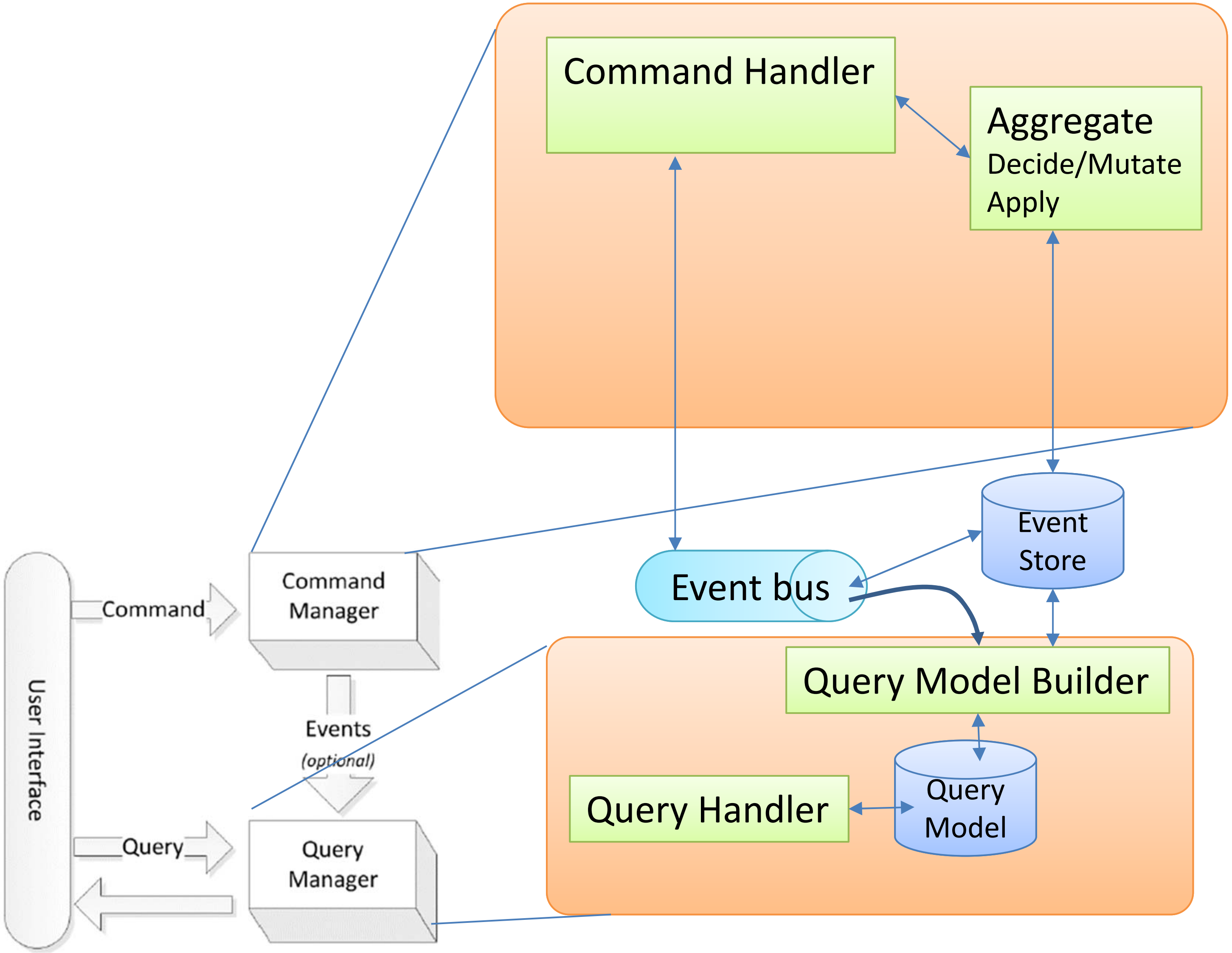
CQRS + Event Sourcing

Erb and Kargl (2014)

University of Ulm

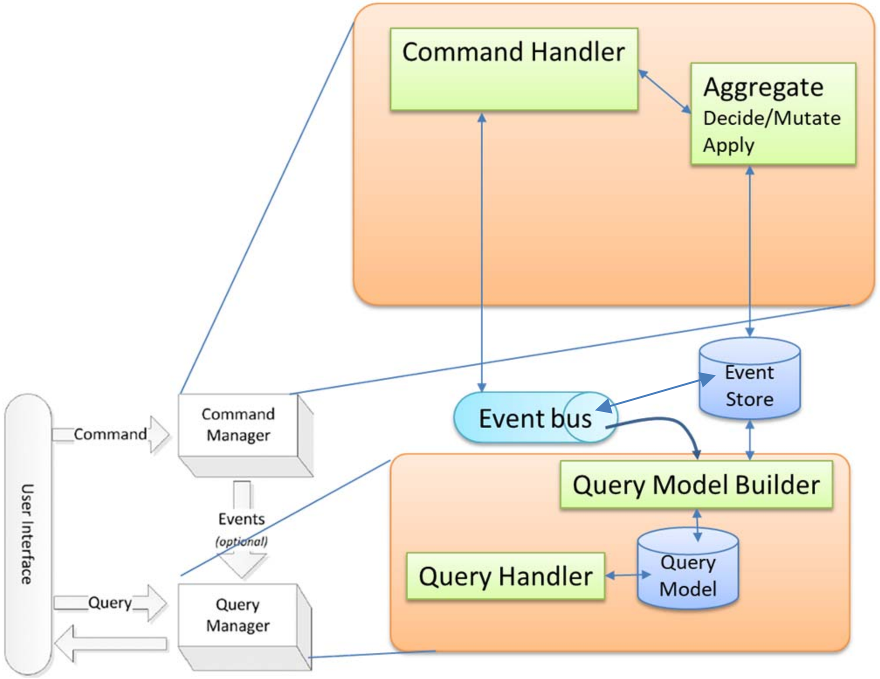


CQRS+ ES Implementation Details



CQRS+ ES Implementation Details

- 如何更新Query Model
  - 同步接收來將送到Event Store儲存的events並據以更新
  - 定期從Event Store取出新events並據以更新



CQRS Impact

- Pros
  - Improves separation of (NFR) concerns
  - 加速分散式查詢效率
  - 加速多樣化查詢效率
  - 讓Eventstore也能被有效率查詢
- Cons
  - More complex architecture
  - Dealing with the replication latency

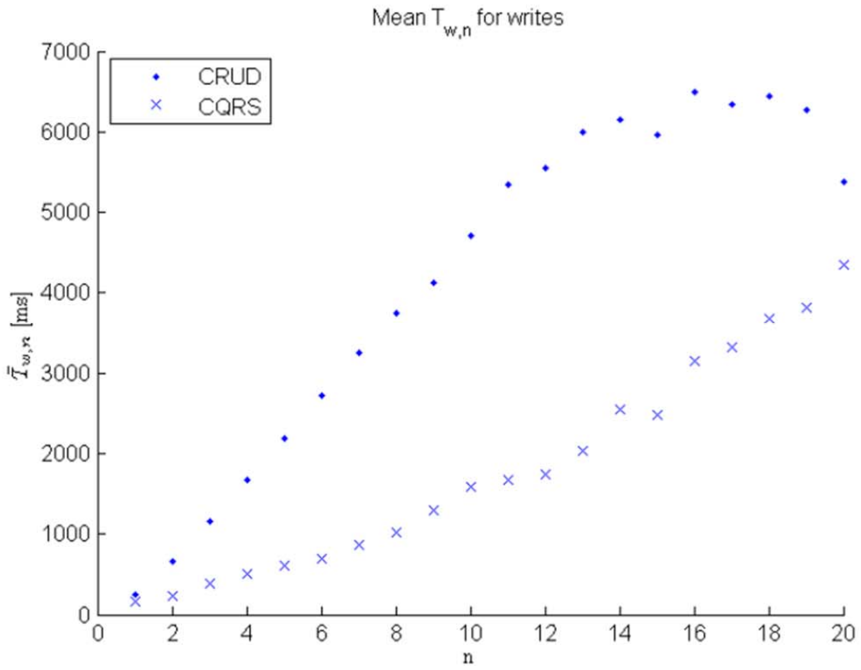
CQRS Limitations

- **MF: You should be very cautious about using CQRS**
- CQRS is useful in some places, but not in others
  - CQRS reduces productivity and increasing risk
  - Should only be used on specific portions of a system
  - Each Bounded Context needs its own decisions on how it should be modeled
- When appropriate?
  - complex domains (minority case)
  - handling high performance applications
    - If the application sees a big disparity between reads and writes
    - separate the load from reads and write (scale each independently)

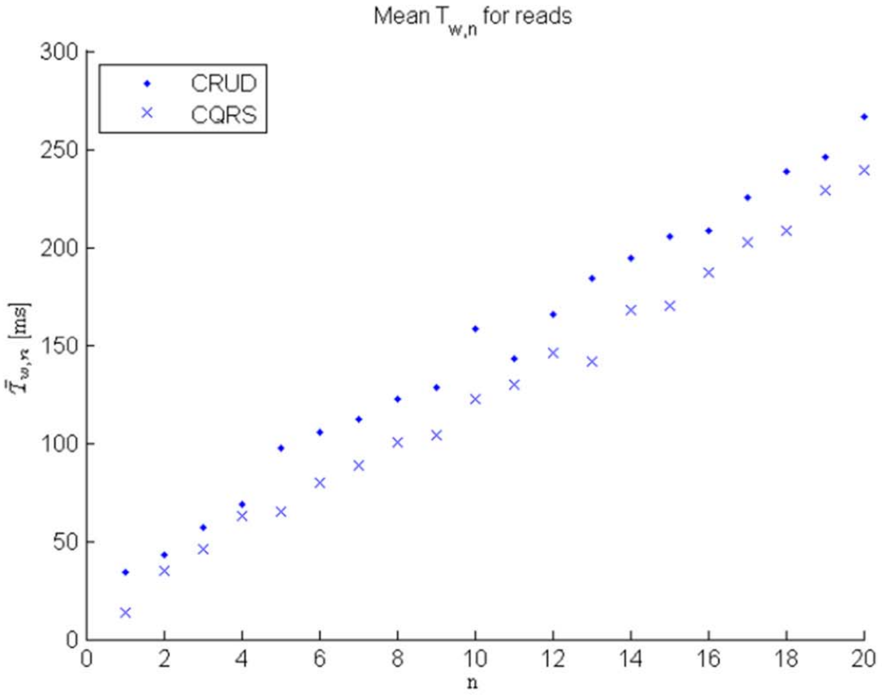
效能評估 (Niltoft & Pochill, 2013)

(本實驗由瑞典隆德大學資料系設計執行)

寫入



讀取



隨著client個數的上升，CQRS/ES在寫入turnaround time 表現好且相對穩定
讀取效能的部份並沒有顯著提升

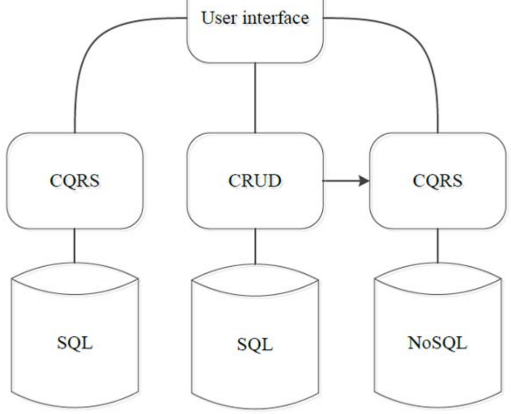
專家訪談 (Korkmaz & Nilsson, 2014)

- 瑞典隆德大學研究團隊針對7位親身實踐過CQRS/ES於專案中的技術主管進行深度質性訪談
  - Alignment (套用)
    - 需要刻意讓use cases map 到CQRS/ES的設計 (more code)
      - Often pays off in the long run
    - 促進stakeholder和設計團隊更好的溝通
    - Components are less reusable
  - 開發
    - Modules can be developed in parallel; usually appropriate for outsourcing
    - 受訪者認為query model相對容易外包; CQRS/ES有助於這種區分
      - 寫入部份由核心團隊開發
      - 整合較為容易

專家訪談

- 質性訪談2
  - 彈性
    - 過去長期依賴RDBMS，很難放心完全使用ES
    - 有人認為，不用整個架構改，只要使用NoSQL DB來存denorm資料，就可大幅改善效能 (提到RavenDB有類似功能)
  - 模組化
    - CQRS is not a way to decompose the application and shall never be
    - 受訪者認為不應該整個系統都是CQRS/ES，它只適用在一小部份

可能的情況



專家訪談

- 質性訪談3
  - 複雜性
    - Distributed transactions can take longer if query model is updated consistently (讀寫比例會影響CQRS/ES的效能)
    - CQRS/ES的實現有點像翻天覆地的改變，modeling、設計到實作、維運都需要，它所得到的好處通常不足以讓我下決定採用

業界案例

- 德國 漢莎航空 (Debski et al. 18; IEEE Software)
  - Actor model (Akka)
  - Build a prototype for flight scheduling; partially used in real world
  - Increase scalability, elasticity, and responsiveness
  - Unacceptable rebuilt speed
  - Lacks field-proven tools, developer guides, and best practices
- 塞爾維亞 (Rajkovic et al. 13)
  - CQRS Improved 40% response time in Medical Information System
- 荷蘭 ERPSoft (Kabbedijk et al. 12; in EuroPLoP)
  - Gain scalability and performance during load peaks
  - Result in a high level of variability within a software product (和隆德大學研究不一致)

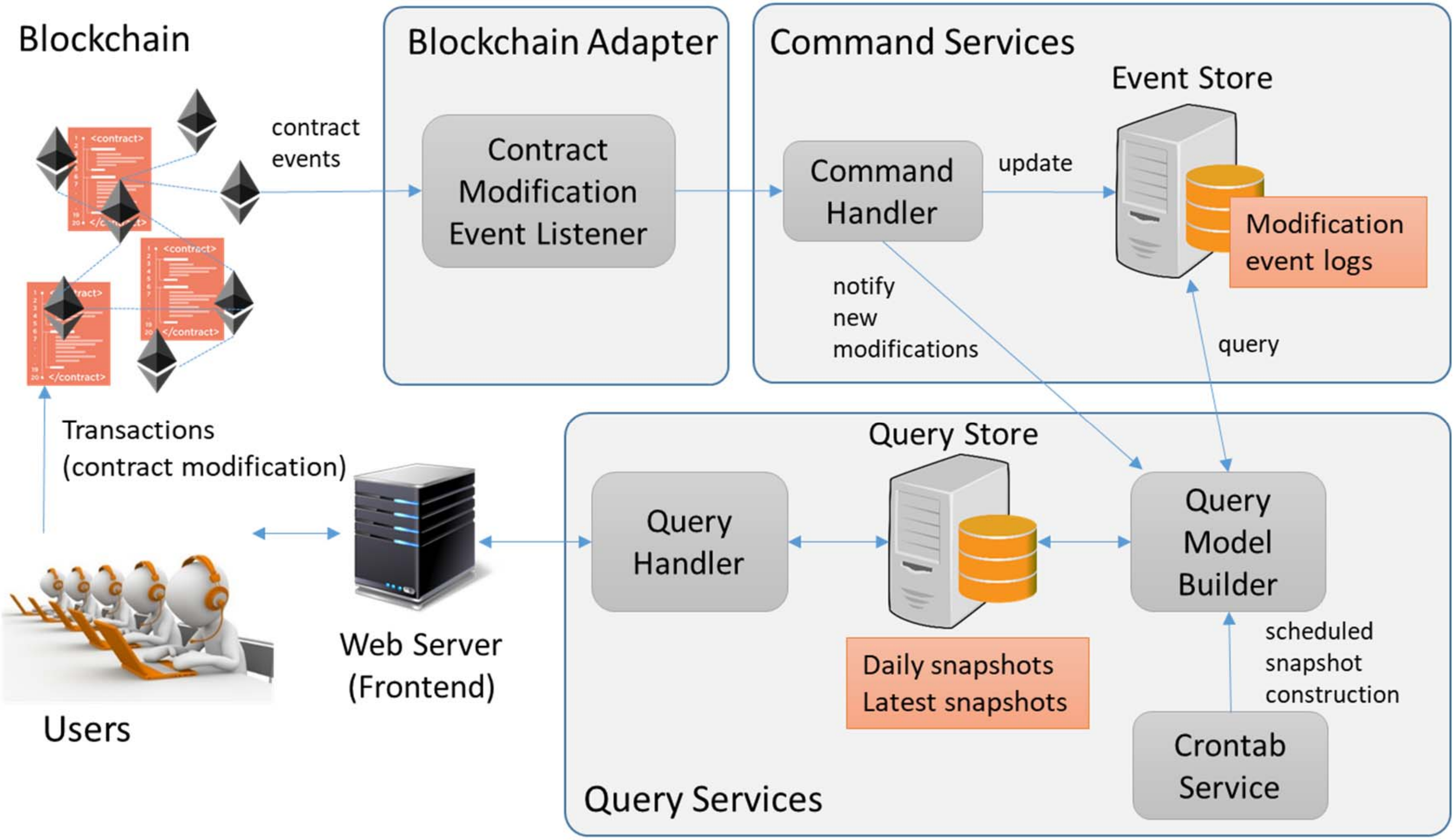
智能合約稽核 (2020)

- 2020年與政大FinTech研究中心合作之研究
- 區塊鏈常被用於存證與稽核
  - 合約狀態變動記錄下來永久保存
  - 難以篡改、非集中特性
- 鏈上取出合約狀態歷史資料，理論可行，實務困難
  - 區塊鏈內部資料結構主要被設計來記錄「區塊本身被確認的歷史」，而非「合約實體狀態(欄位值)被修改的歷史」
  - 多數情況使用暴力法查詢
    - 區塊中混雜來自不同合約產生的交易記錄，沒有內建相關索引(index)
    - 在沒有其它機制支援的情況下，要從頭到尾找一次，並挑出符合的交易

解決方案

- 基於 Event Sourcing，提出一個相容於區塊鏈運作方式的合約狀態歷史保留與查詢機制
- 引進CQRS架構，預先產生Snapshot，得以快速回應不同需求且大量的稽核查詢

系統架構



鄭智豪, 廖峻鋒, 鄭宇軒, 陳恭 “初探基於 CQRS 與 Event Sourcing 智能合約稽核機制設計,” 台灣軟體工程研討會 (TCSE), 嘉義, 台灣, 2020.

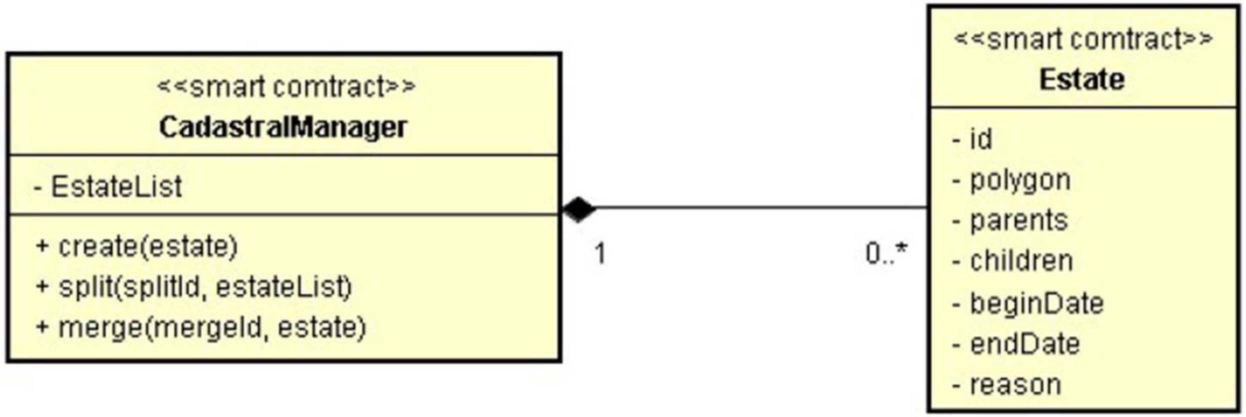
案例: 土地分割合併歷史變遷追朔機制

- 2020年與政大社科院數位轉型研究中心合作之研究
- 目前地籍資料系統不足之處
  - 缺點1：難以提供地籍歷史變遷追蹤：
    - 地區發展脈絡不齊全
    - 產權研究、社會研究無法有效利用地籍資訊
  - 缺點2：整合性低
    - 現行系統: 不定期離線批次移轉，資訊傳遞緩慢
    - 造成土地產權轉移的不確定性

案例: 土地分割合併歷史變遷追朔機制

- 研究目的
  - 以區塊鏈為基礎，建立即時、可追朔的地籍圖資平台
  - 使用Event Sourcing管理地籍圖資的變動版本資訊
  - 以CQRS架構樣式建立高效率且多樣的查詢系統

Aggregate示意圖



<<smart comtract>>
CadastralManager

- EstateList

+ create(estate)
+ split(splitId, estateList)
+ merge(mergeId, estate)

1

0..*

<<smart comtract>>
Estate

- id
- polygon
- parents
- children
- beginDate
- endDate
- reason

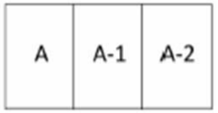


圖 8: 5 月 2 日土地狀況

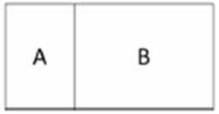


圖 9: 5 月 3 日土地狀況

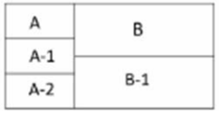


圖 10: 5 月 4 日土地狀況

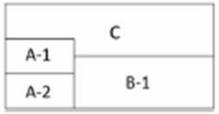


圖 11: 最新土地狀況

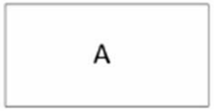
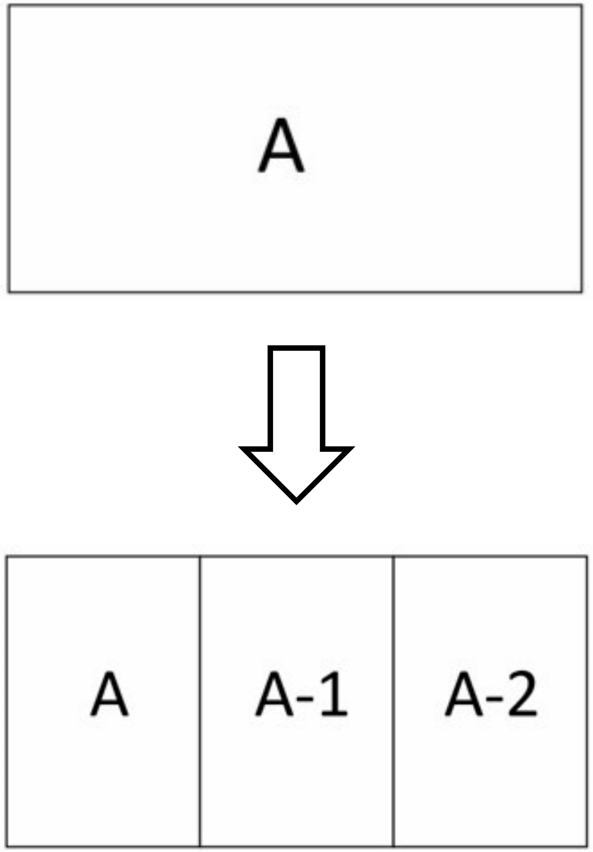


圖 12: 初始土地狀況

範例：土地分割

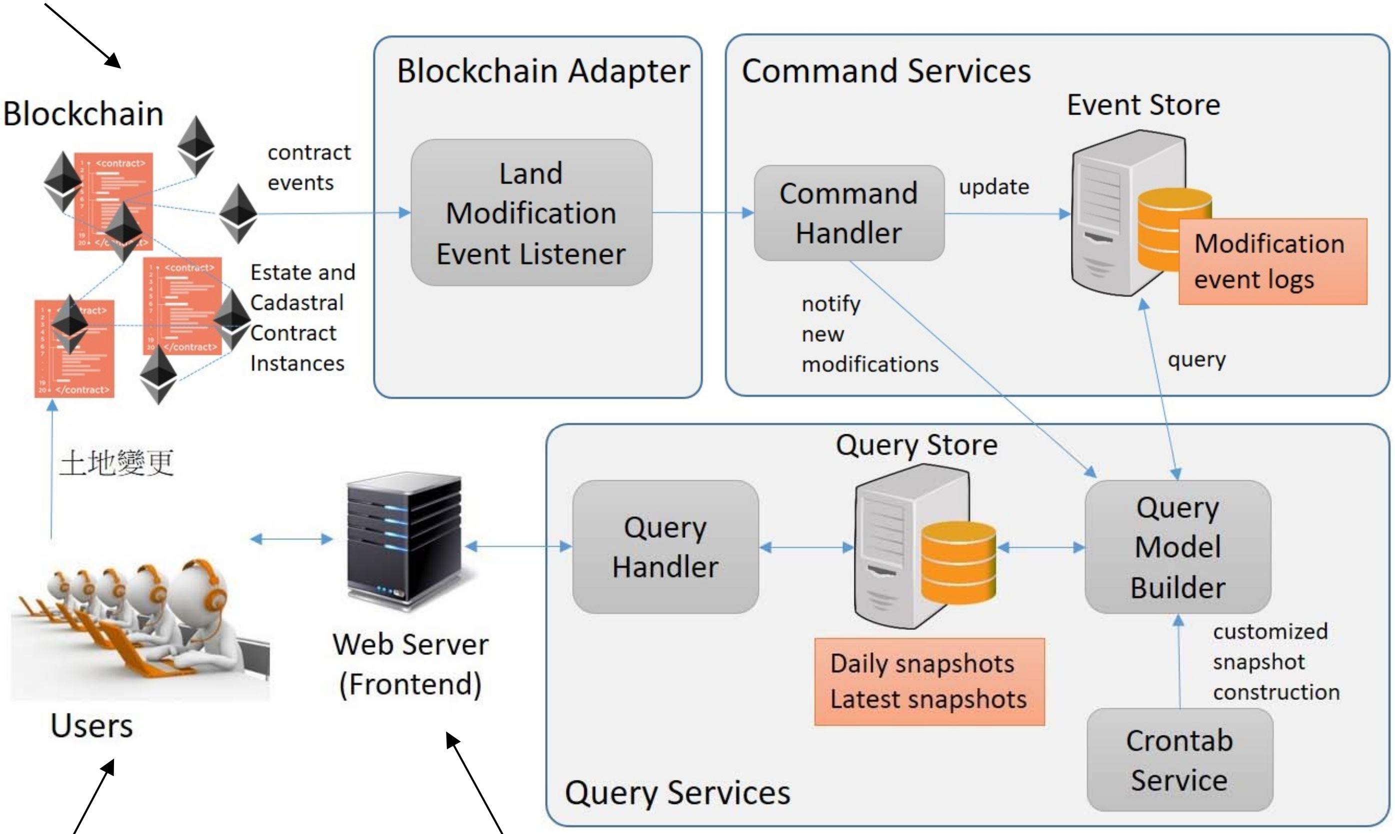
- 以土地分割(右圖)為例：
  - A地被分割為A、A-1、A-2三地
  - 分割模擬困難點：
    - A地和分割後三地之關係
    - 分割後三地之記錄方式



系統設計：CQRS架構

2.地籍區塊鏈

3.寫入服務



1.使用者

5.前端

4.查詢服務

Conclusion

- Event Sourcing和CQRS
  - 本質是二個不同類型的架構概念，可以獨立實現，也可以併用
  - Event Sourcing 比較像Data Architecture
  - CQRS比較像Application Architecture
  - 在效能提升、朔源技術與儲存備份帶來革命性的(新?)思維
- Event Sourcing和CQRS可能不適合做為「聖杯」
  - 被放棄過的「聖杯」
    - San Francisco(IBM), ESB(IBM), EJB2(Sun), OSGi (Pivotal)
  - 理由
    - 並非適用於所有場合的架構 (甚至於不適用於大多數場合)
    - 相關機制皆重大改變，維運面亦有重大影響
    - 目前相關成功使用案例、支援技術與工具還不夠多

Q and A