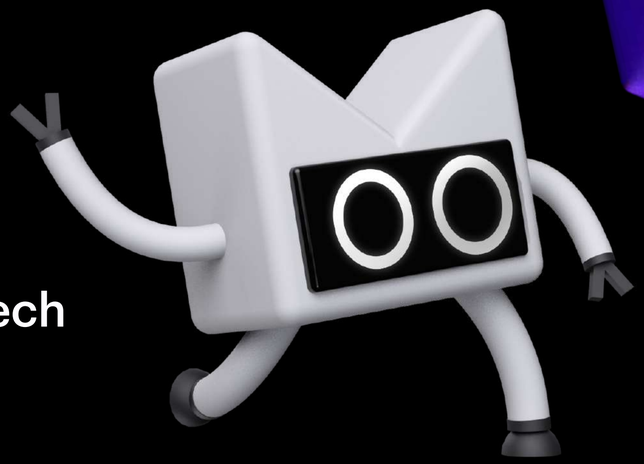
Kotlin

从零开始欣赏 Coroutine 的精湛设计

Gary LO

Gary LO
@Gap撈Tech
Experts



JVM (Java Virtual Machine)
Java 虛擬機

JVM

由開發到執行的過程

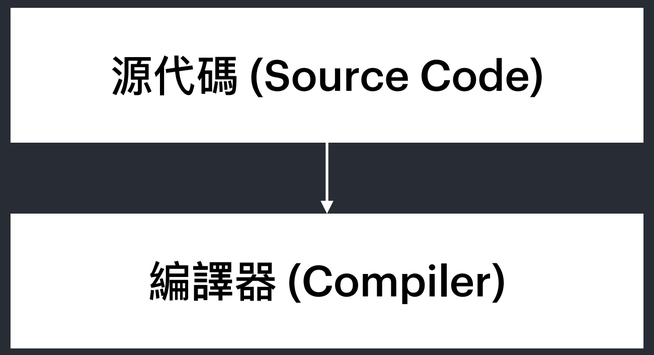
JVM

由開發到執行的過程

源代碼 (Source Code)

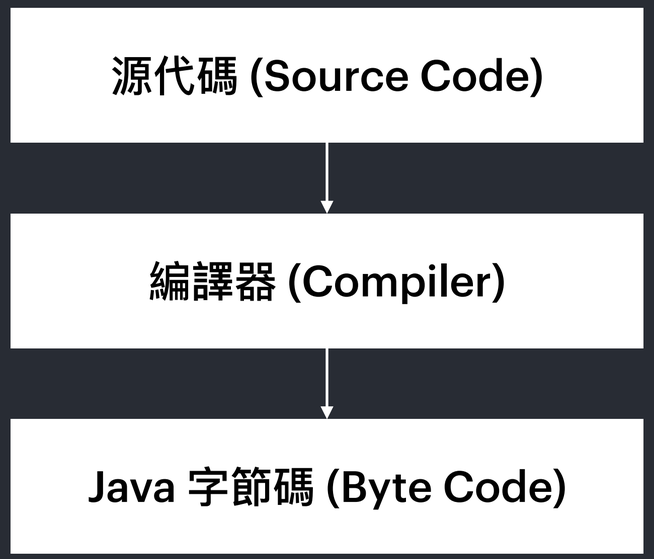
JVM

由開發到執行的過程



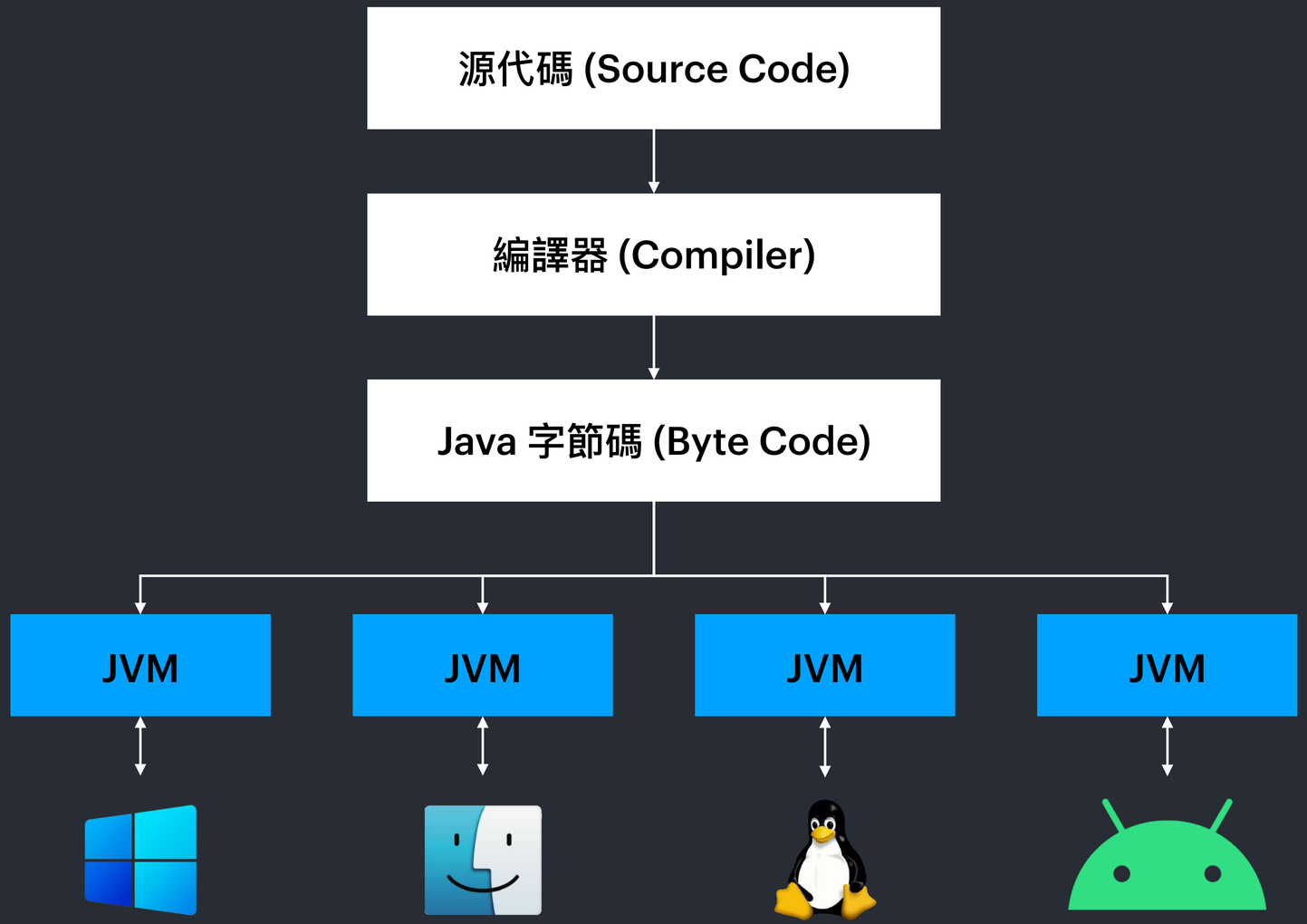
JVM

由開發到執行的過程



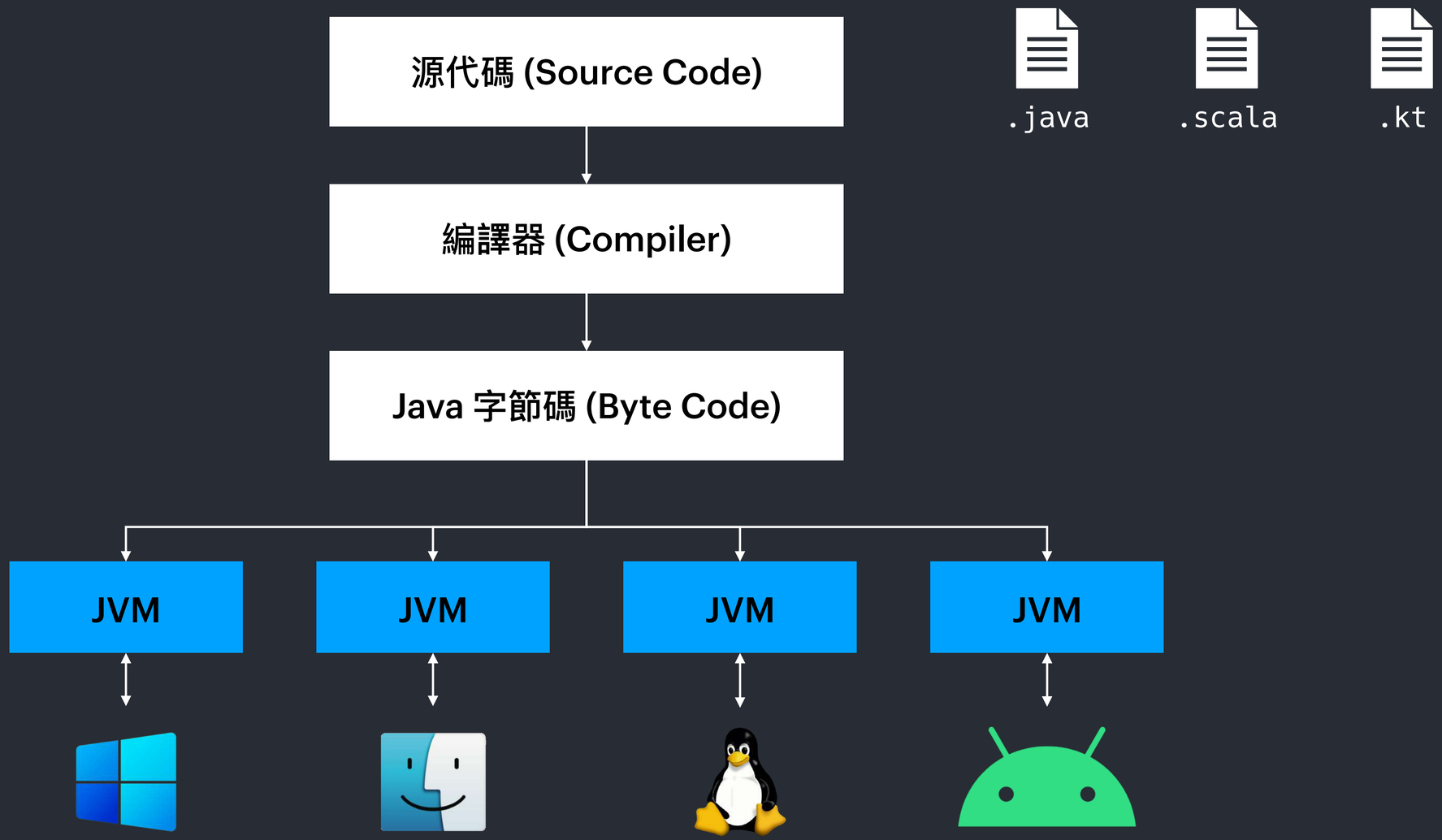
JVM

由開發到執行的過程



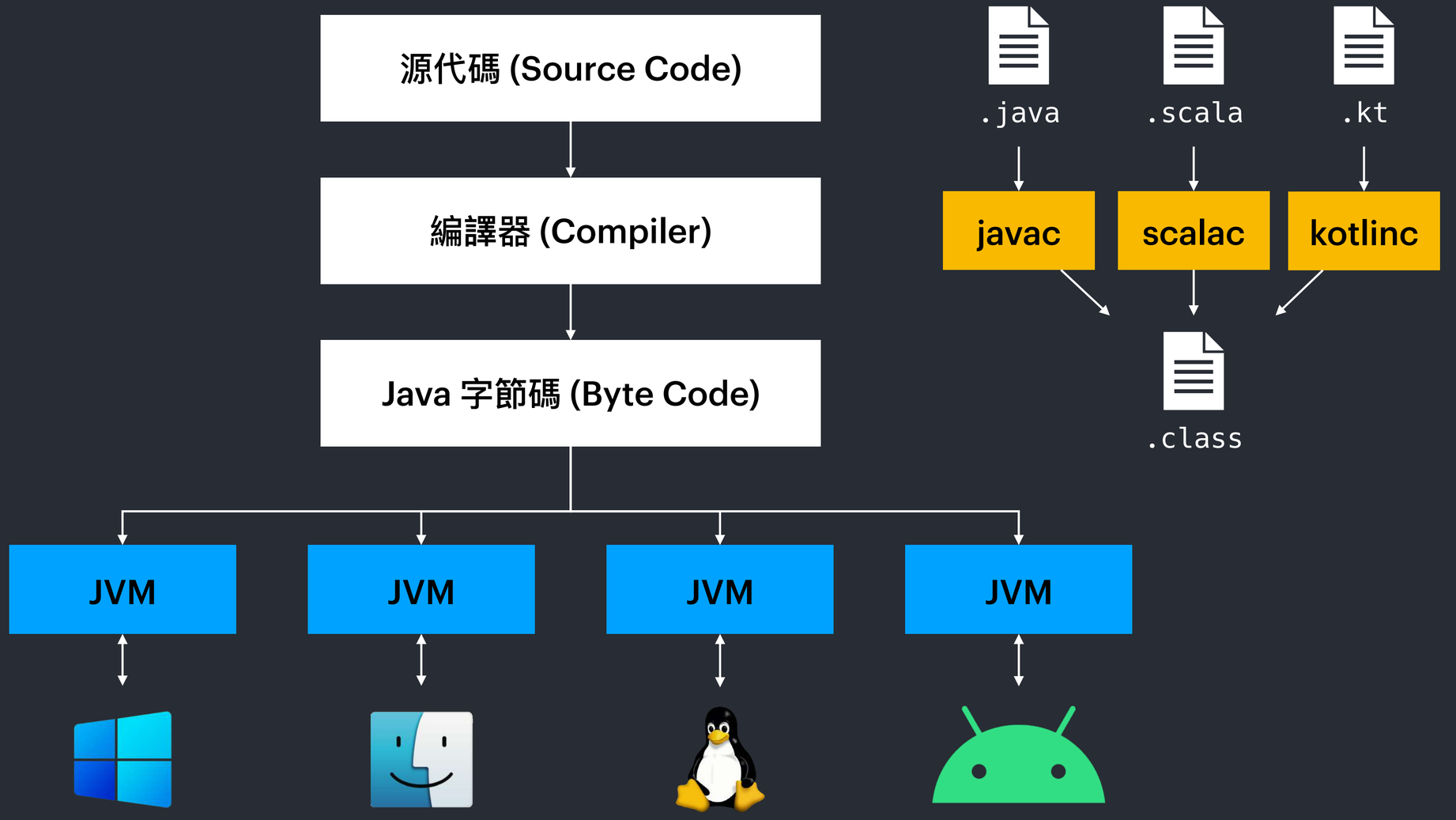
JVM

由開發到執行的過程



JVM

由開發到執行的過程



JVM 語言

JVM 語言

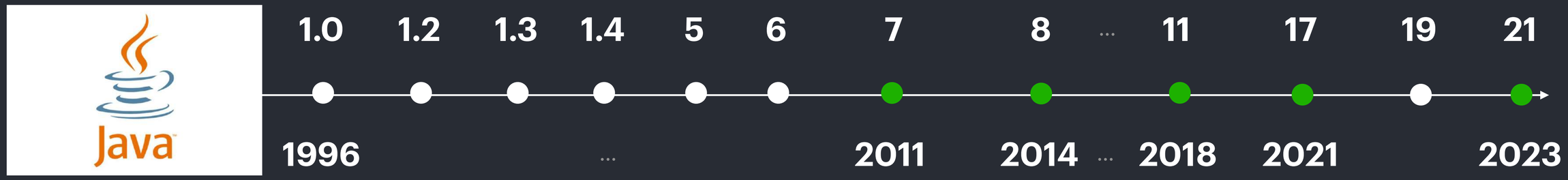
主流語言的成長歷史

JVM 語言

主流語言的成長歷史

JVM 語言

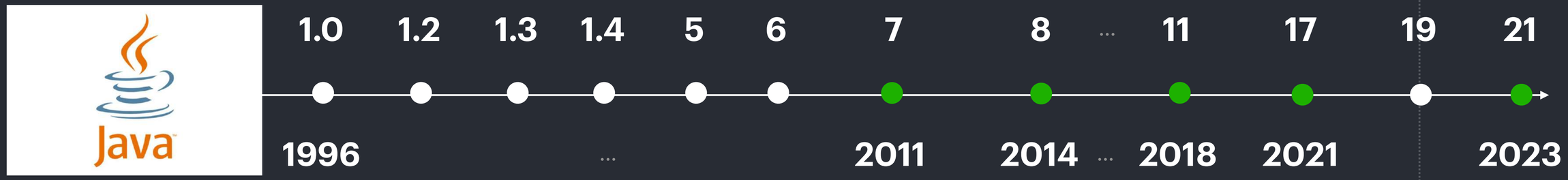
主流語言的成長歷史



JVM 語言

主流語言的成長歷史

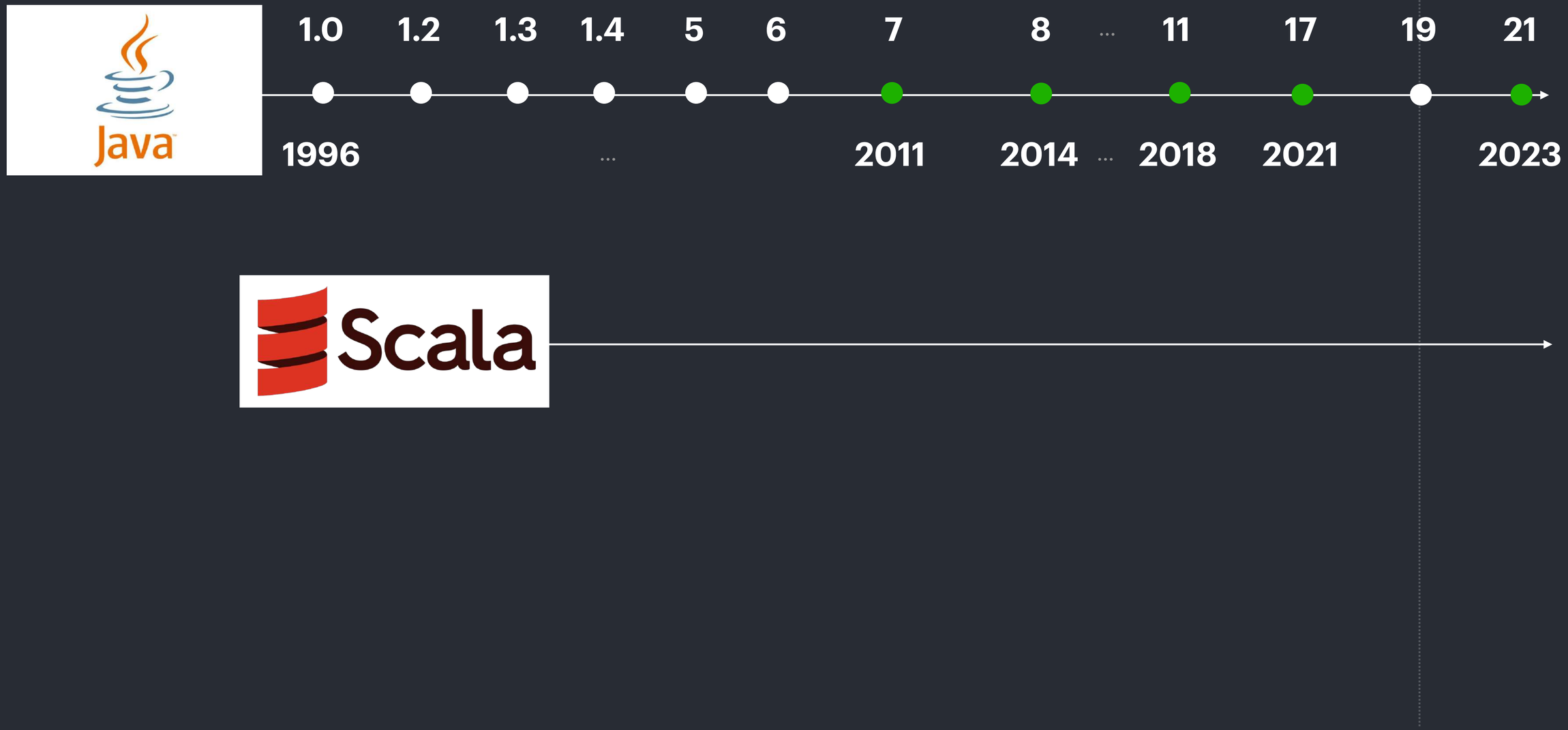
現在



JVM 語言

主流語言的成長歷史

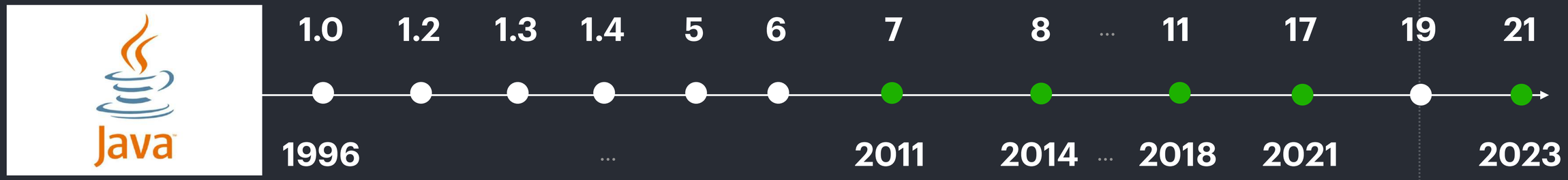
現在

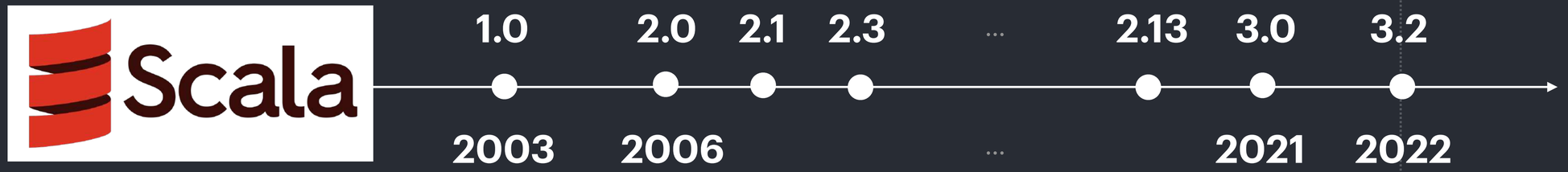


JVM 語言

主流語言的成長歷史

現在

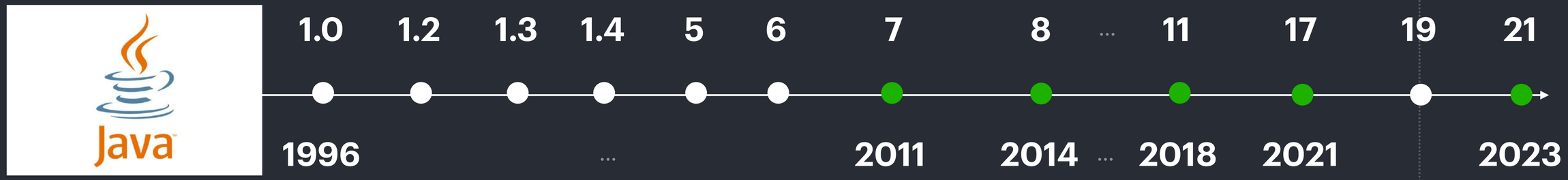


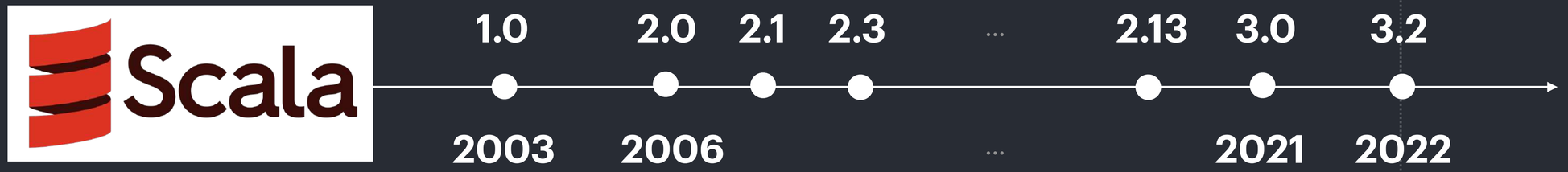


JVM 語言

主流語言的成長歷史

現在

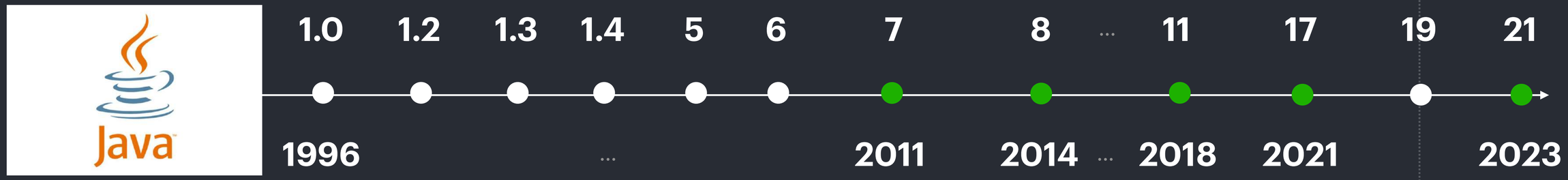


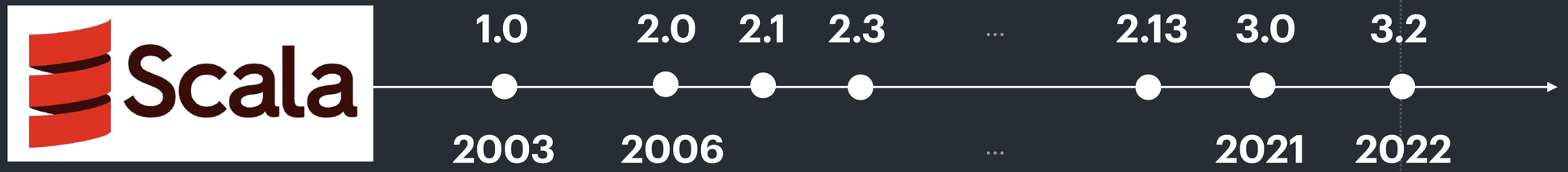


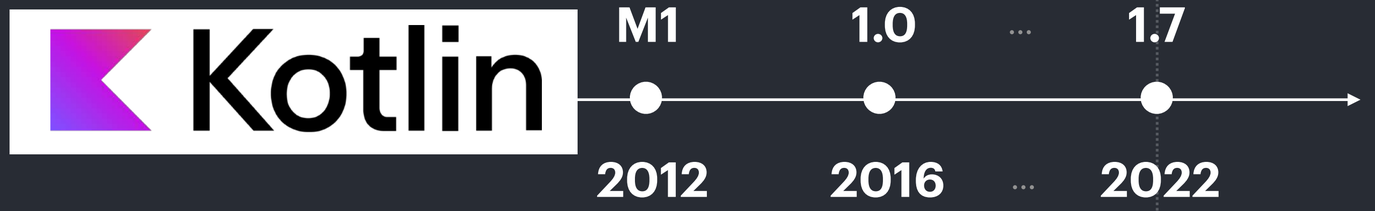
JVM 語言

主流語言的成長歷史

現在







JVM 有多快?

JVM 有多快?

節錄自一個關於語言能源效率的論文

JVM 有多快?

節錄自一個關於語言能源效率的論文

Total

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

	Time
(c) C	1.00
(c) Rust	1.04
(c) C++	1.56
(c) Ada	1.85
(v) Java	1.89
(c) Chapel	2.14
(c) Go	2.83
(c) Pascal	3.02
(c) Ocaml	3.09
(v) C#	3.14
(v) Lisp	3.40
(c) Haskell	3.55
(c) Swift	4.20
(c) Fortran	4.20
(v) F#	6.30
(i) JavaScript	6.52
(i) Dart	6.67
(v) Racket	11.27
(i) Hack	26.99
(i) PHP	27.64
(v) Erlang	36.71
(i) Jruby	43.44
(i) TypeScript	46.20
(i) Ruby	59.34
(i) Perl	65.79
(i) Python	71.90
(i) Lua	82.91

	Mb
(c) Pascal	1.00
(c) Go	1.05
(c) C	1.17
(c) Fortran	1.24
(c) C++	1.34
(c) Ada	1.47
(c) Rust	1.54
(v) Lisp	1.92
(c) Haskell	2.45
(i) PHP	2.57
(c) Swift	2.71
(i) Python	2.80
(c) Ocaml	2.82
(v) C#	2.85
(i) Hack	3.34
(v) Racket	3.52
(i) Ruby	3.97
(c) Chapel	4.00
(v) F#	4.25
(i) JavaScript	4.59
(i) TypeScript	4.69
(v) Java	6.01
(i) Perl	6.62
(i) Lua	6.72
(v) Erlang	7.20
(i) Dart	8.64
(i) Jruby	19.84

JVM 有多快?

節錄自一個關於語言能源效率的論文

Total

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

	Time
(c) C	1.00
(c) Rust	1.04
(c) C++	1.56
(c) Ada	1.85
(v) Java	1.89
(c) Chapel	2.14
(c) Go	2.83
(c) Pascal	3.02
(c) Ocaml	3.09
(v) C#	3.14
(v) Lisp	3.40
(c) Haskell	3.55
(c) Swift	4.20
(c) Fortran	4.20
(v) F#	6.30
(i) JavaScript	6.52
(i) Dart	6.67
(v) Racket	11.27
(i) Hack	26.99
(i) PHP	27.64
(v) Erlang	36.71
(i) Jruby	43.44
(i) TypeScript	46.20
(i) Ruby	59.34
(i) Perl	65.79
(i) Python	71.90
(i) Lua	82.91

	Mb
(c) Pascal	1.00
(c) Go	1.05
(c) C	1.17
(c) Fortran	1.24
(c) C++	1.34
(c) Ada	1.47
(c) Rust	1.54
(v) Lisp	1.92
(c) Haskell	2.45
(i) PHP	2.57
(c) Swift	2.71
(i) Python	2.80
(c) Ocaml	2.82
(v) C#	2.85
(i) Hack	3.34
(v) Racket	3.52
(i) Ruby	3.97
(c) Chapel	4.00
(v) F#	4.25
(i) JavaScript	4.59
(i) TypeScript	4.69
(v) Java	6.01
(i) Perl	6.62
(i) Lua	6.72
(v) Erlang	7.20
(i) Dart	8.64
(i) Jruby	19.84

JVM 有多快?

節錄自一個關於語言能源效率的論文

Total

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

	Time
(c) C	1.00
(c) Rust	1.04
(c) C++	1.56
(c) Ada	1.85
(v) Java	1.89
(c) Chapel	2.14
(c) Go	2.83
(c) Pascal	3.02
(c) Ocaml	3.09
(v) C#	3.14
(v) Lisp	3.40
(c) Haskell	3.55
(c) Swift	4.20
(c) Fortran	4.20
(v) F#	6.30
(i) JavaScript	6.52
(i) Dart	6.67
(v) Racket	11.27
(i) Hack	26.99
(i) PHP	27.64
(v) Erlang	36.71
(i) Jruby	43.44
(i) TypeScript	46.20
(i) Ruby	59.34
(i) Perl	65.79
(i) Python	71.90
(i) Lua	82.91

	Mb
(c) Pascal	1.00
(c) Go	1.05
(c) C	1.17
(c) Fortran	1.24
(c) C++	1.34
(c) Ada	1.47
(c) Rust	1.54
(v) Lisp	1.92
(c) Haskell	2.45
(i) PHP	2.57
(c) Swift	2.71
(i) Python	2.80
(c) Ocaml	2.82
(v) C#	2.85
(i) Hack	3.34
(v) Racket	3.52
(i) Ruby	3.97
(c) Chapel	4.00
(v) F#	4.25
(i) JavaScript	4.59
(i) TypeScript	4.69
(v) Java	6.01
(i) Perl	6.62
(i) Lua	6.72
(v) Erlang	7.20
(i) Dart	8.64
(i) Jruby	19.84

JVM 有多快?

節錄自一個關於語言能源效率的論文

Total

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

	Time
(c) C	1.00
(c) Rust	1.04
(c) C++	1.56
(c) Ada	1.85
(v) Java	1.89
(c) Chapel	2.14
(c) Go	2.83
(c) Pascal	3.02
(c) Ocaml	3.09
(v) C#	3.14
(v) Lisp	3.40
(c) Haskell	3.55
(c) Swift	4.20
(c) Fortran	4.20
(v) F#	6.30
(i) JavaScript	6.52
(i) Dart	6.67
(v) Racket	11.27
(i) Hack	26.99
(i) PHP	27.64
(v) Erlang	36.71
(i) Jruby	43.44
(i) TypeScript	46.20
(i) Ruby	59.34
(i) Perl	65.79
(i) Python	71.90
(i) Lua	82.91

	Mb
(c) Pascal	1.00
(c) Go	1.05
(c) C	1.17
(c) Fortran	1.24
(c) C++	1.34
(c) Ada	1.47
(c) Rust	1.54
(v) Lisp	1.92
(c) Haskell	2.45
(i) PHP	2.57
(c) Swift	2.71
(i) Python	2.80
(c) Ocaml	2.82
(v) C#	2.85
(i) Hack	3.34
(v) Racket	3.52
(i) Ruby	3.97
(c) Chapel	4.00
(v) F#	4.25
(i) JavaScript	4.59
(i) TypeScript	4.69
(v) Java	6.01
(i) Perl	6.62
(i) Lua	6.72
(v) Erlang	7.20
(i) Dart	8.64
(i) Jruby	19.84

JVM 有多快?

節錄自一個關於語言能源效率的論文

Total

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

	Time
(c) C	1.00
(c) Rust	1.04
(c) C++	1.56
(c) Ada	1.85
(v) Java	1.89
(c) Chapel	2.14
(c) Go	2.83
(c) Pascal	3.02
(c) Ocaml	3.09
(v) C#	3.14
(v) Lisp	3.40
(c) Haskell	3.55
(c) Swift	4.20
(c) Fortran	4.20
(v) F#	6.30
(i) JavaScript	6.52
(i) Dart	6.67
(v) Racket	11.27
(i) Hack	26.99
(i) PHP	27.64
(v) Erlang	36.71
(i) Jruby	43.44
(i) TypeScript	46.20
(i) Ruby	59.34
(i) Perl	65.79
(i) Python	71.90
(i) Lua	82.91

	Mb
(c) Pascal	1.00
(c) Go	1.05
(c) C	1.17
(c) Fortran	1.24
(c) C++	1.34
(c) Ada	1.47
(c) Rust	1.54
(v) Lisp	1.92
(c) Haskell	2.45
(i) PHP	2.57
(c) Swift	2.71
(i) Python	2.80
(c) Ocaml	2.82
(v) C#	2.85
(i) Hack	3.34
(v) Racket	3.52
(i) Ruby	3.97
(c) Chapel	4.00
(v) F#	4.25
(i) JavaScript	4.59
(i) TypeScript	4.69
(v) Java	6.01
(i) Perl	6.62
(i) Lua	6.72
(v) Erlang	7.20
(i) Dart	8.64
(i) Jruby	19.84

JVM 有多快?

節錄自一個關於語言能源效率的論文

Total

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

	Time
(c) C	1.00
(c) Rust	1.04
(c) C++	1.56
(c) Ada	1.85
(v) Java	1.89
(c) Chapel	2.14
(c) Go	2.83
(c) Pascal	3.02
(c) Ocaml	3.09
(v) C#	3.14
(v) Lisp	3.40
(c) Haskell	3.55
(c) Swift	4.20
(c) Fortran	4.20
(v) F#	6.30
(i) JavaScript	6.52
(i) Dart	6.67
(v) Racket	11.27
(i) Hack	26.99
(i) PHP	27.64
(v) Erlang	36.71
(i) Jruby	43.44
(i) TypeScript	46.20
(i) Ruby	59.34
(i) Perl	65.79
(i) Python	71.90
(i) Lua	82.91

	Mb
(c) Pascal	1.00
(c) Go	1.05
(c) C	1.17
(c) Fortran	1.24
(c) C++	1.34
(c) Ada	1.47
(c) Rust	1.54
(v) Lisp	1.92
(c) Haskell	2.45
(i) PHP	2.57
(c) Swift	2.71
(i) Python	2.80
(c) Ocaml	2.82
(v) C#	2.85
(i) Hack	3.34
(v) Racket	3.52
(i) Ruby	3.97
(c) Chapel	4.00
(v) F#	4.25
(i) JavaScript	4.59
(i) TypeScript	4.69
(v) Java	6.01
(i) Perl	6.62
(i) Lua	6.72
(v) Erlang	7.20
(i) Dart	8.64
(i) Jruby	19.84

“遺憾的事，異步編程以往是一件十分痛苦的事；”

“異步 I/O 是現今寫高並發（*High Concurrency*）的基本框架。”

“直至今天，開發者仍會避免這些痛苦而選擇同步編程及同步 I/O。”

高並發場景例子

高並發場景例子

後台微服務 A

數據庫 A

數據庫 B

後台微服務 B

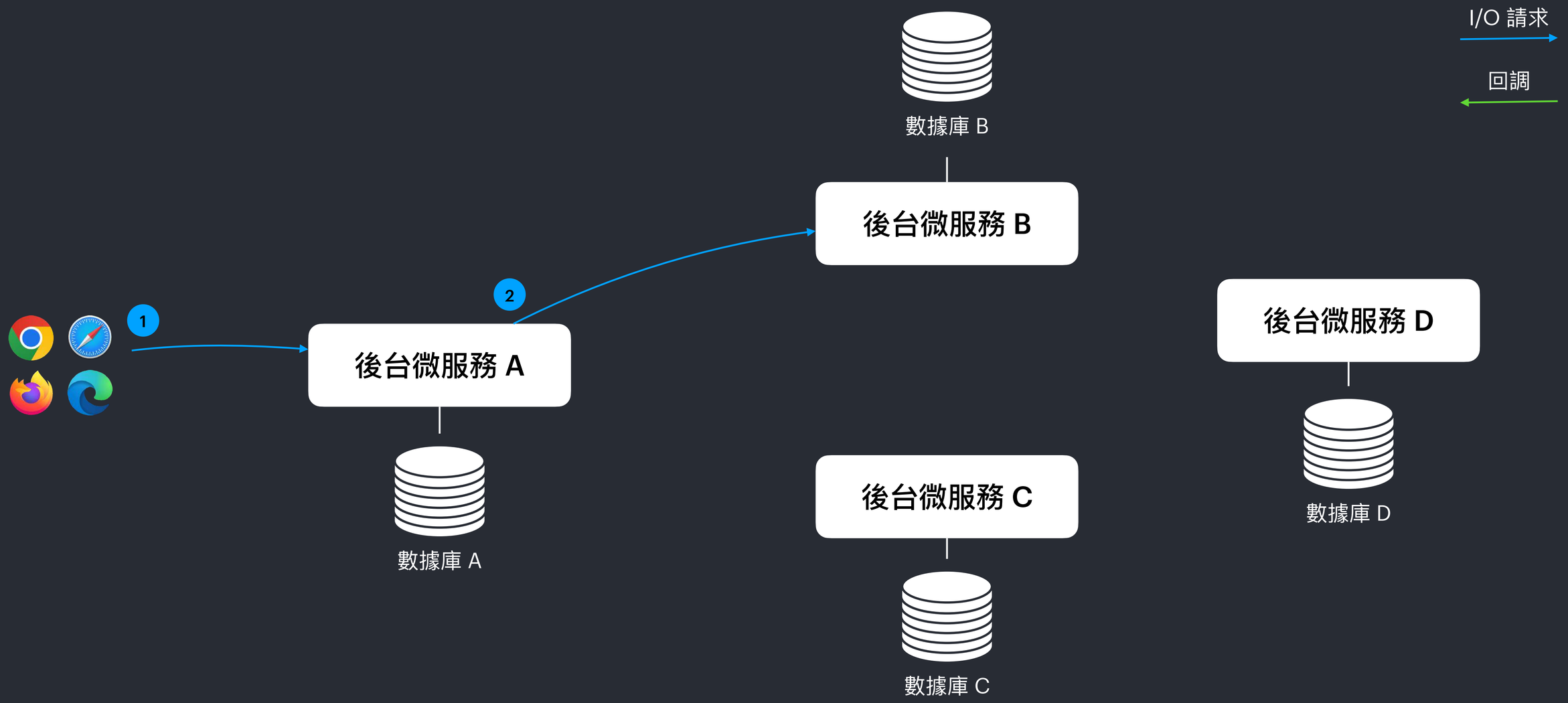
後台微服務 C

數據庫 C

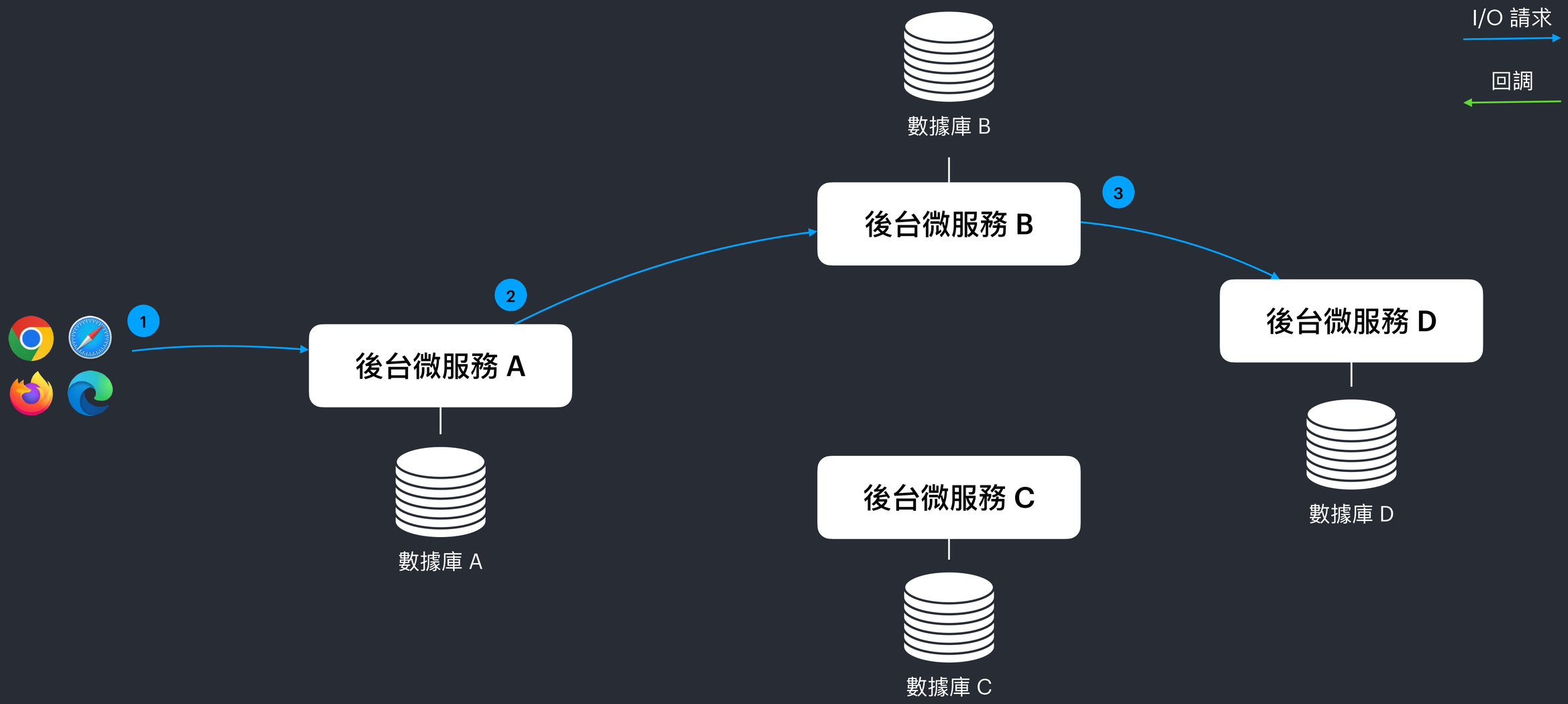
後台微服務 D

數據庫 D

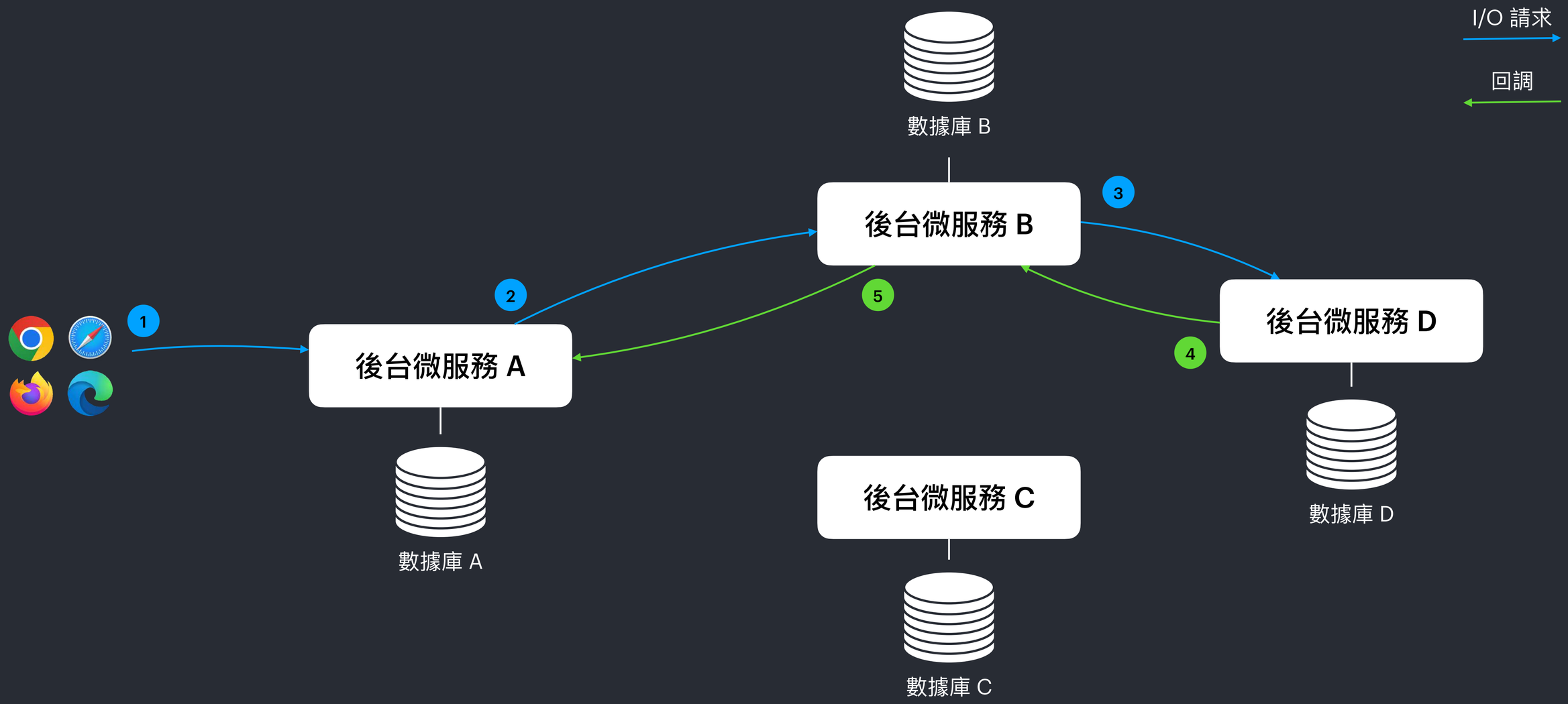
高並發場景例子



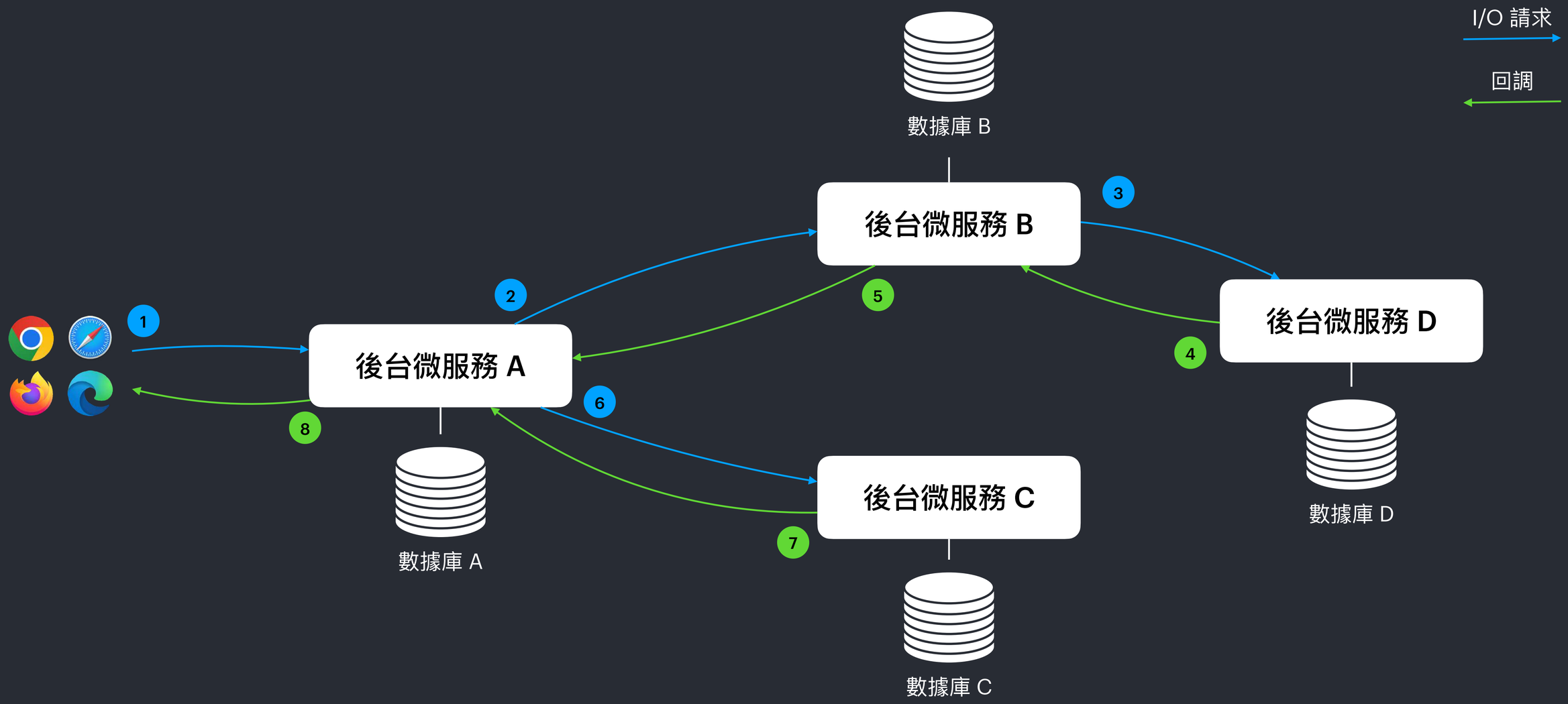
高並發場景例子



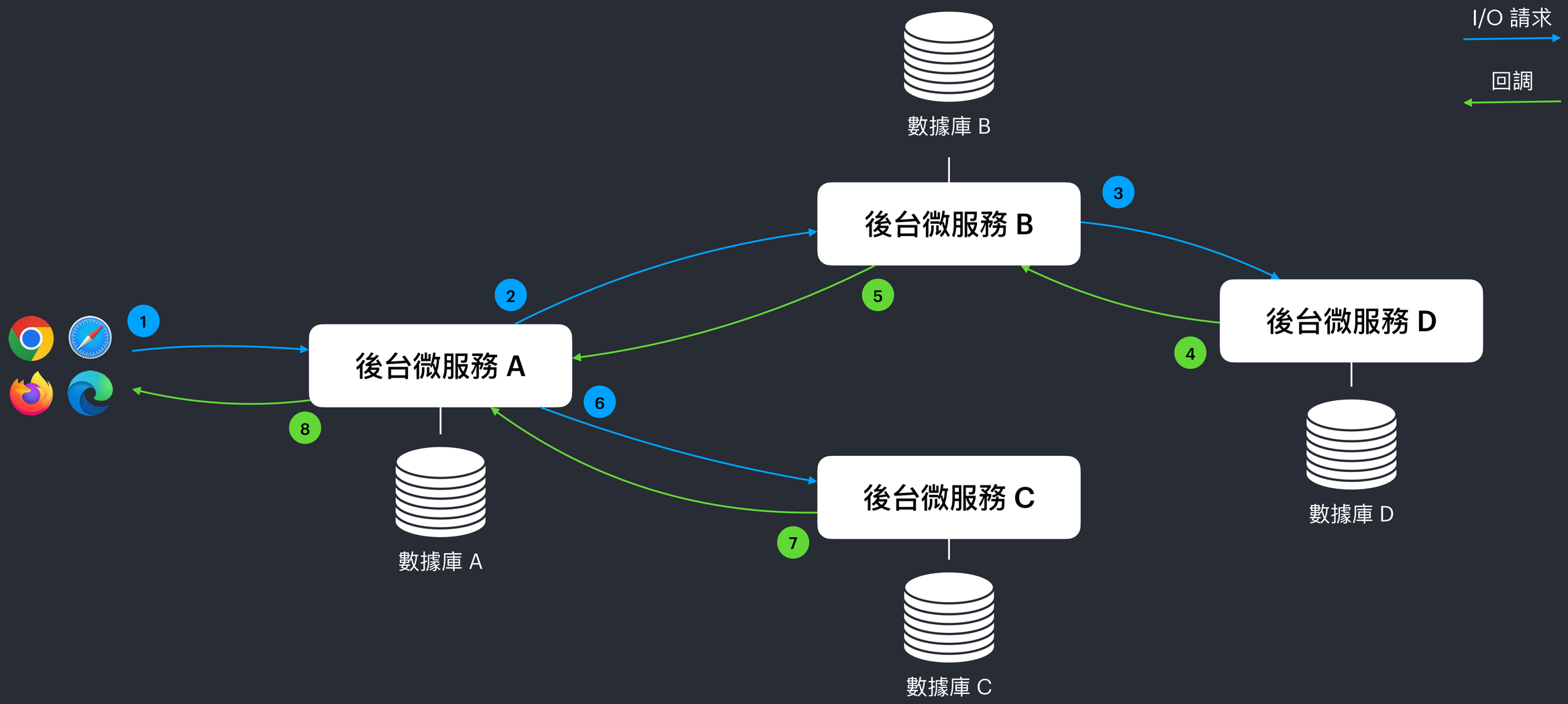
高並發場景例子


I/O 請求
回調
數據庫 B
後台微服務 B
3
2
5
1
後台微服務 A
後台微服務 D
4
數據庫 A
後台微服務 C
數據庫 D
數據庫 C

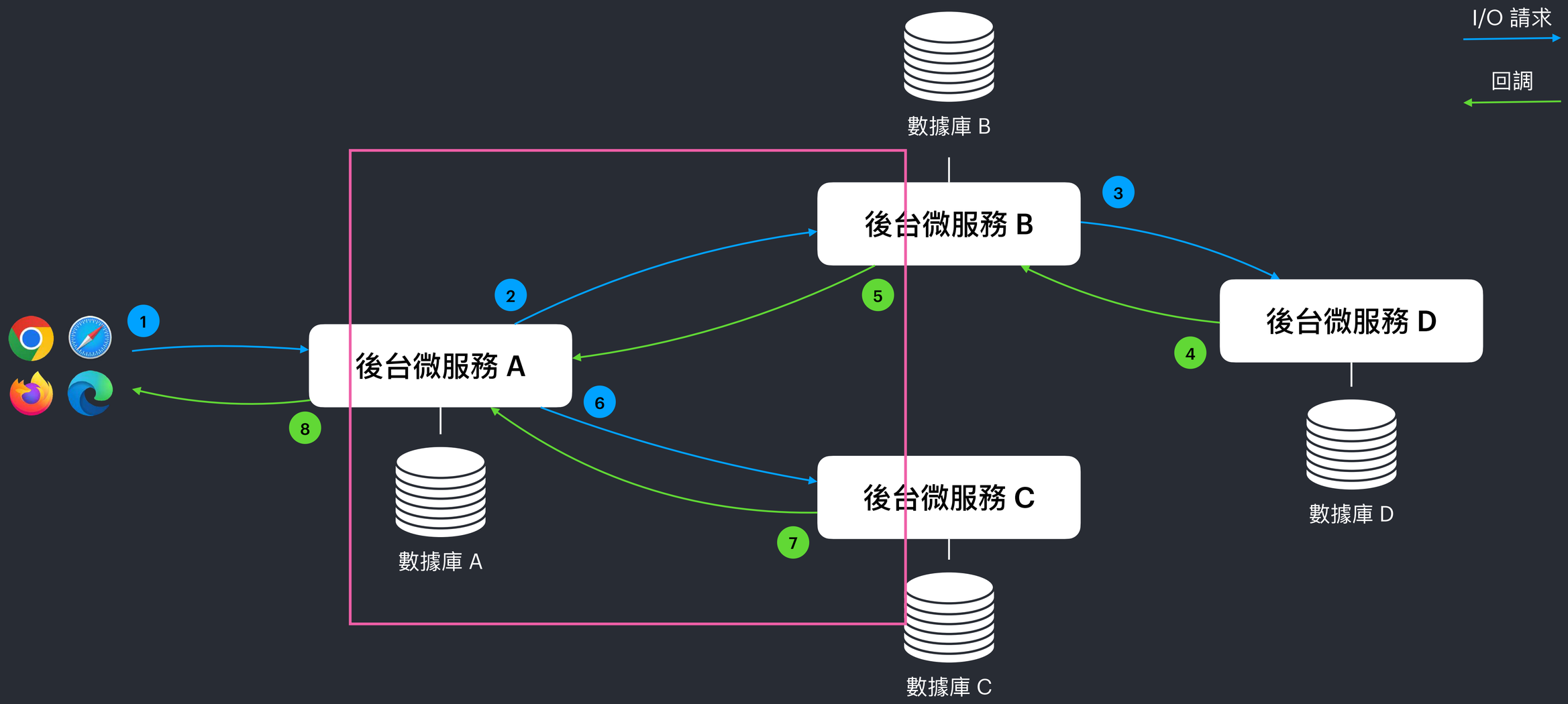
高並發場景例子


I/O 請求
回調
數據庫 B
後台微服務 B
3
2
5
1
後台微服務 A
後台微服務 D
4
6
8
數據庫 A
後台微服務 C
數據庫 D
7
數據庫 C

高並發場景例子


I/O 請求
回調
數據庫 B
後台微服務 B
3
2
5
1
後台微服務 A
後台微服務 D
4
6
8
數據庫 A
後台微服務 C
數據庫 D
7
數據庫 C

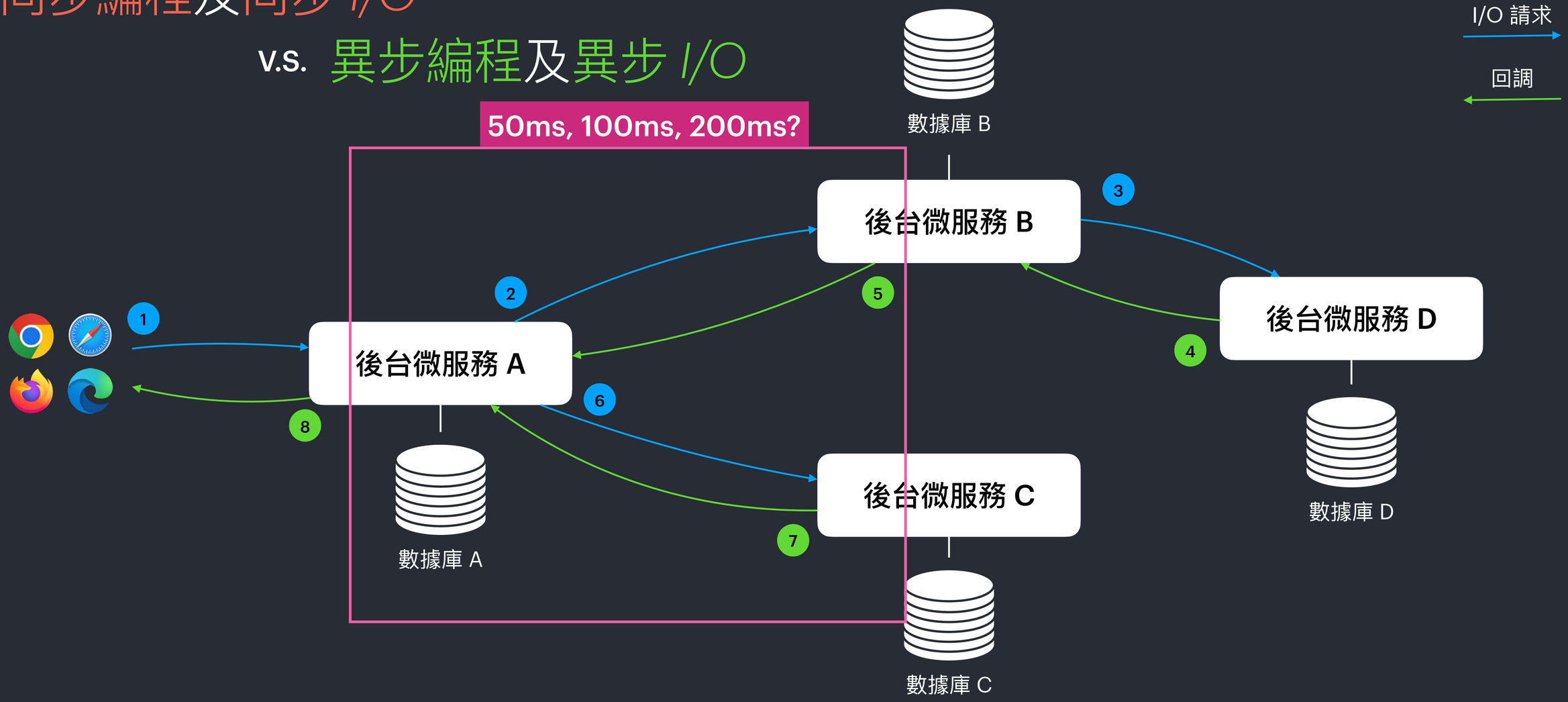
高並發場景例子


I/O 請求
回調
數據庫 B
後台微服務 B
3
2
5
1
後台微服務 A
後台微服務 D
4
6
8
數據庫 A
後台微服務 C
數據庫 D
7
數據庫 C

高並發場景例子

同步編程及同步 I/O

v.s. 異步編程及異步 I/O



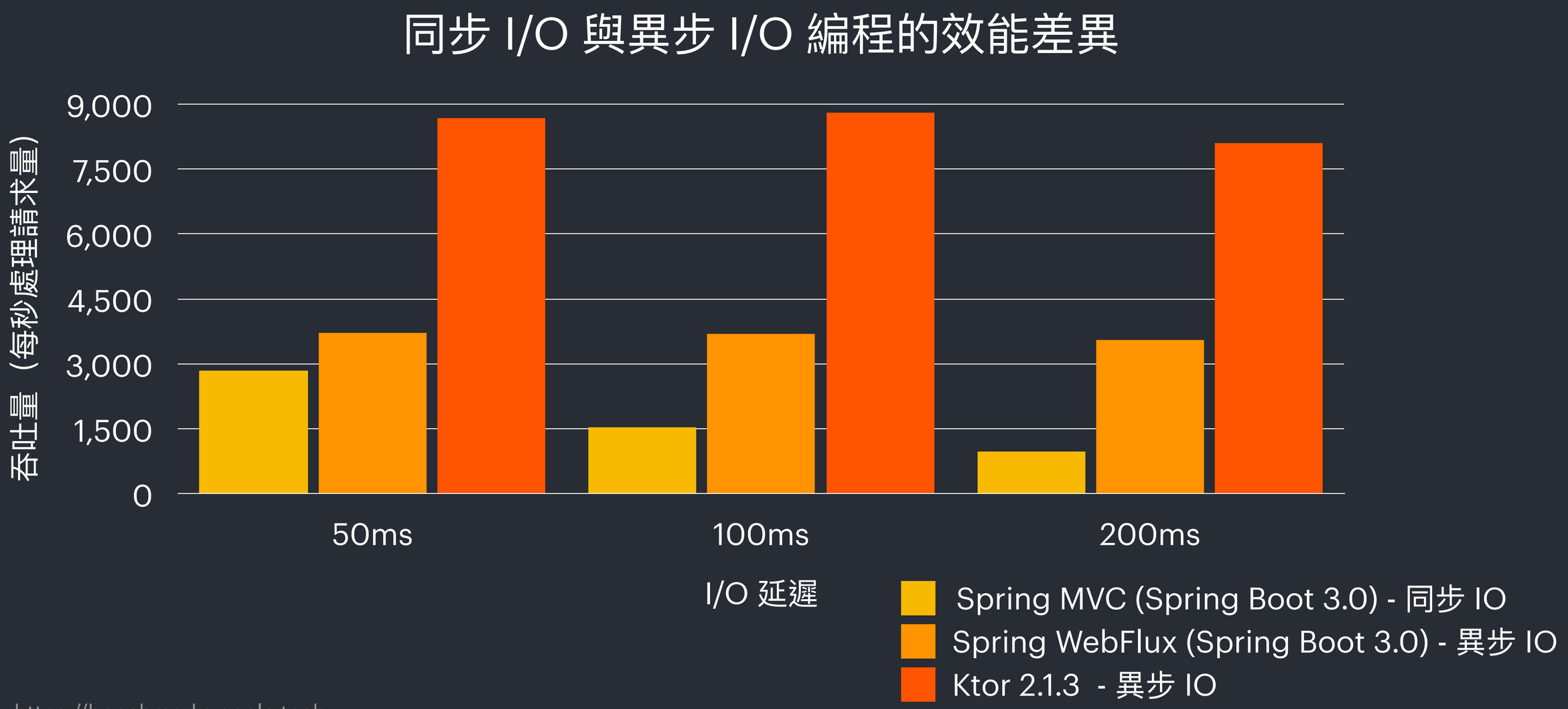
高並發效能測試

預設軟件配置 | 硬件: 1vCPU 4GB Memory | 16,000 並發請求

完整跑分: https://benchmarks.gaplo.tech

高並發效能測試

預設軟件配置 | 硬件: 1vCPU 4GB Memory | 16,000 並發請求



完整跑分: https://benchmarks.gaplo.tech

同步編程及同步 I/O

IO 線程池數目調整
(IO Thread Pool Tuning)

同步編程及同步 I/O

IO 線程池數目調整
(IO Thread Pool Tuning)

線程池耗盡
(Thread Pool Exhaustion)

同步編程及同步 I/O

IO 線程池數目調整
(IO Thread Pool Tuning)

線程池耗盡
(Thread Pool Exhaustion)

線程數目邊際收益遞減
(Diminishing of Return)

同步編程及同步 I/O

IO 線程池數目調整
(IO Thread Pool Tuning)

線程池耗盡
(Thread Pool Exhaustion)

多線程安全
(Thread Safety)

線程數目邊際收益遞減
(Diminishing of Return)

同步編程及同步 I/O

IO 線程池數目調整
(IO Thread Pool Tuning)

線程池耗盡
(Thread Pool Exhaustion)

多線程安全
(Thread Safety)

線程數目邊際收益遞減
(Diminishing of Return)

同步編程及同步 I/O

多線程共享變量
(Shared memory)

IO 線程池數目調整
(IO Thread Pool Tuning)

線程池耗盡
(Thread Pool Exhaustion)

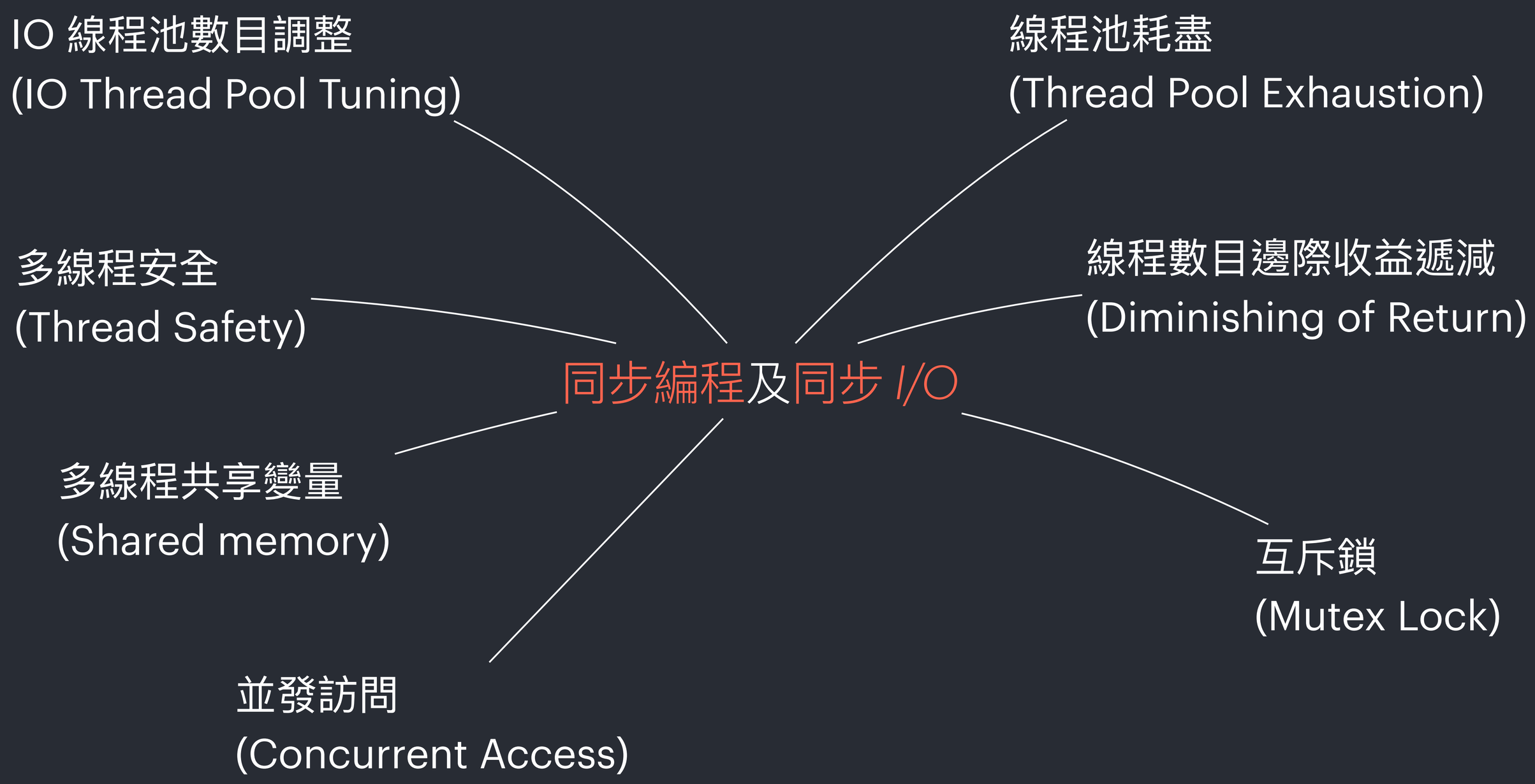
多線程安全
(Thread Safety)

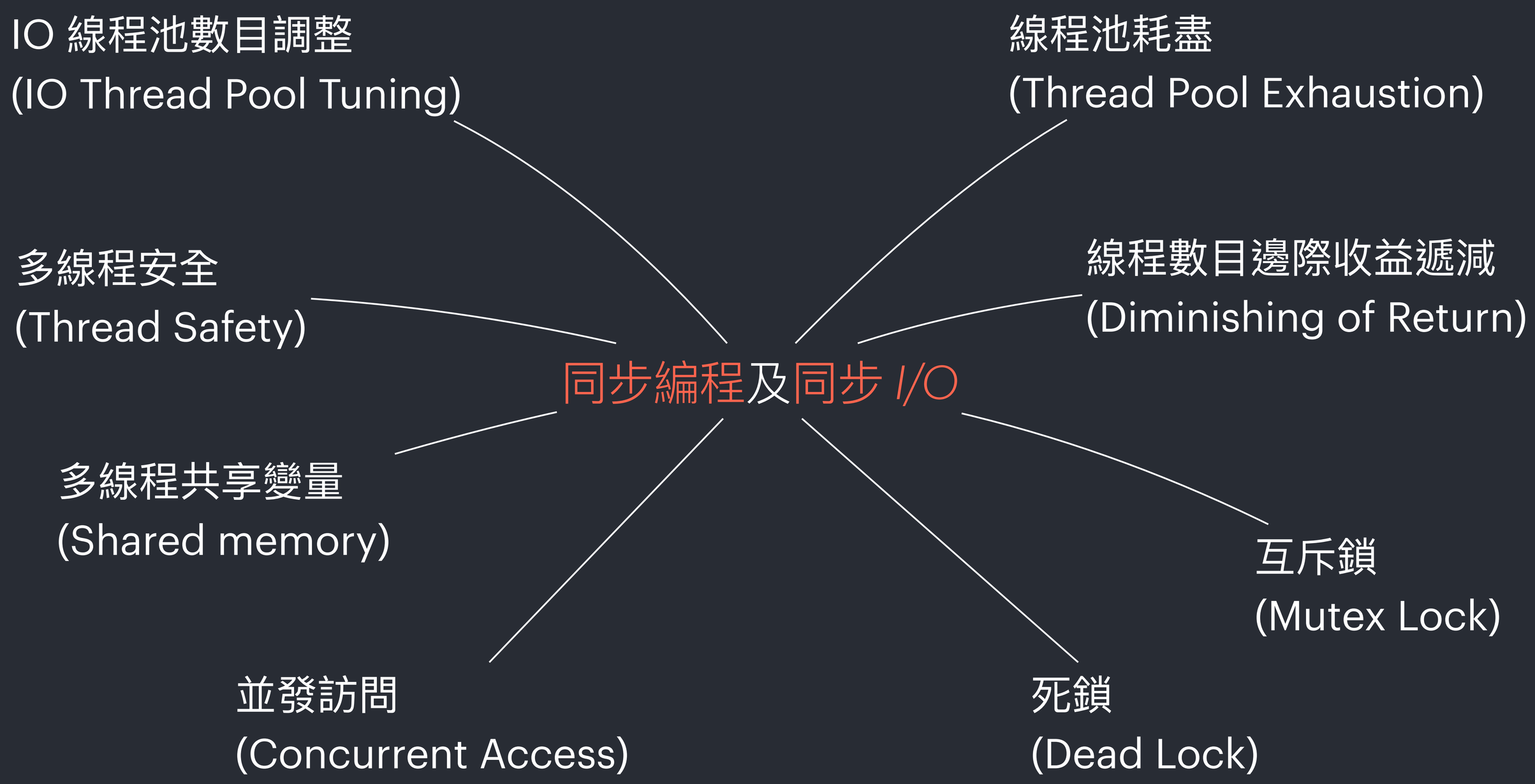
線程數目邊際收益遞減
(Diminishing of Return)

同步編程及同步 I/O

多線程共享變量
(Shared memory)

並發訪問
(Concurrent Access)


IO 線程池數目調整
(IO Thread Pool Tuning)
線程池耗盡
(Thread Pool Exhaustion)
多線程安全
(Thread Safety)
線程數目邊際收益遞減
(Diminishing of Return)
同步編程及同步 I/O
多線程共享變量
(Shared memory)
互斥鎖
(Mutex Lock)
並發訪問
(Concurrent Access)


IO 線程池數目調整
(IO Thread Pool Tuning)
線程池耗盡
(Thread Pool Exhaustion)
多線程安全
(Thread Safety)
線程數目邊際收益遞減
(Diminishing of Return)
同步編程及同步 I/O
多線程共享變量
(Shared memory)
互斥鎖
(Mutex Lock)
並發訪問
(Concurrent Access)
死鎖
(Dead Lock)

異步編程及異步 I/O

回調地獄
(Callback Hell)

異步編程及異步 I/O

回調地獄
(Callback Hell)

低代碼可讀性 高維護成本
(Low Readability, High Maintenance Cost)

異步編程及異步 I/O

回調地獄
(Callback Hell)

低代碼可讀性 高維護成本
(Low Readability, High Maintenance Cost)

異步編程及異步 I/O

複雜的異常處理
(Complex Exception Handling)

回調地獄
(Callback Hell)

低代碼可讀性 高維護成本
(Low Readability, High Maintenance Cost)

加工變換及組裝
(Composability)

異步編程及異步 I/O

複雜的異常處理
(Complex Exception Handling)

回調地獄
(Callback Hell)

低代碼可讀性 高維護成本
(Low Readability, High Maintenance Cost)

加工變換及組裝
(Composability)

異步編程及異步 I/O

複雜的異常處理
(Complex Exception Handling)

線程轉換
(Context Switching)

回調地獄
(Callback Hell)

低代碼可讀性 高維護成本
(Low Readability, High Maintenance Cost)

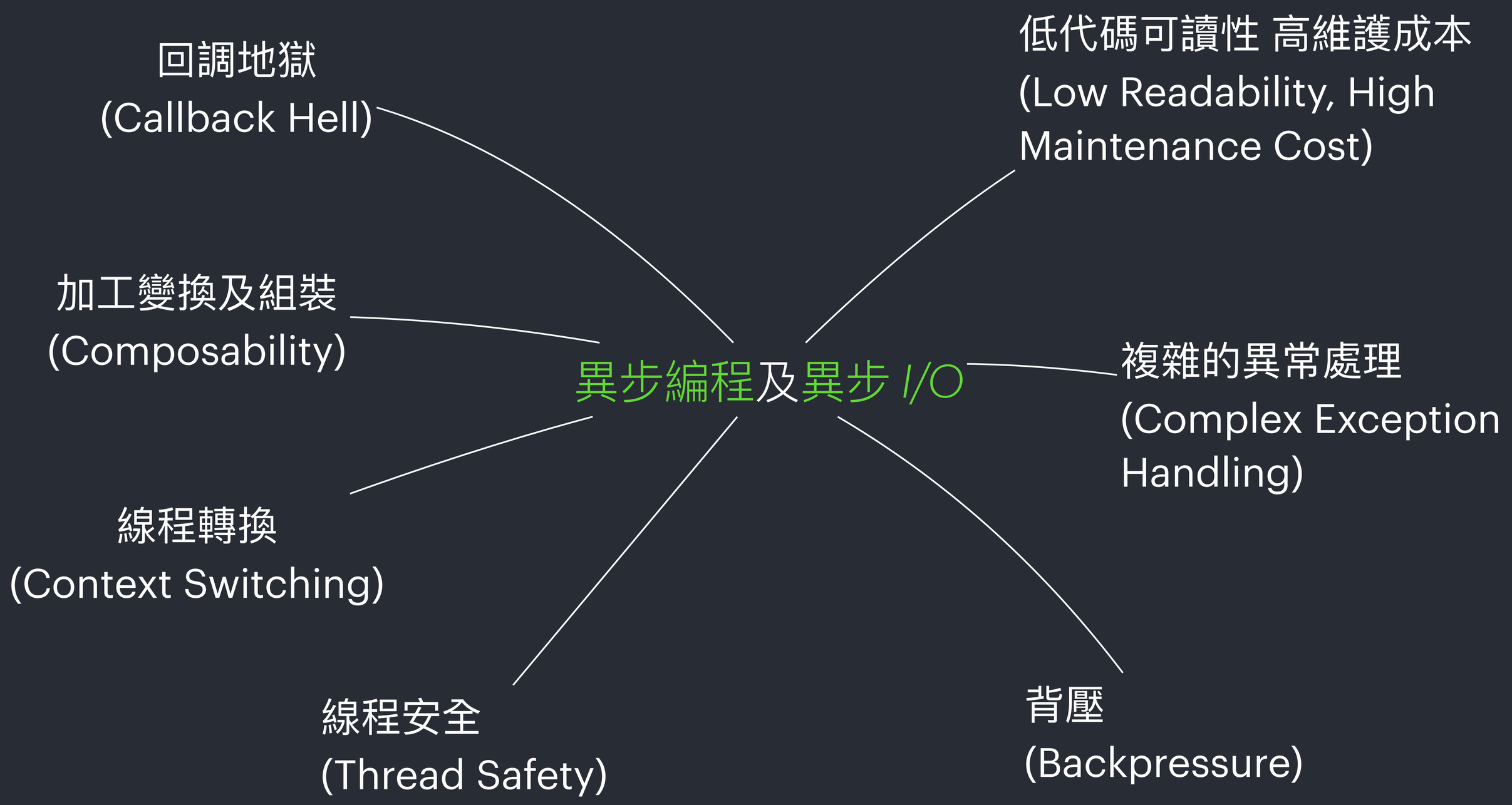
加工變換及組裝
(Composability)

異步編程及異步 I/O

複雜的異常處理
(Complex Exception Handling)

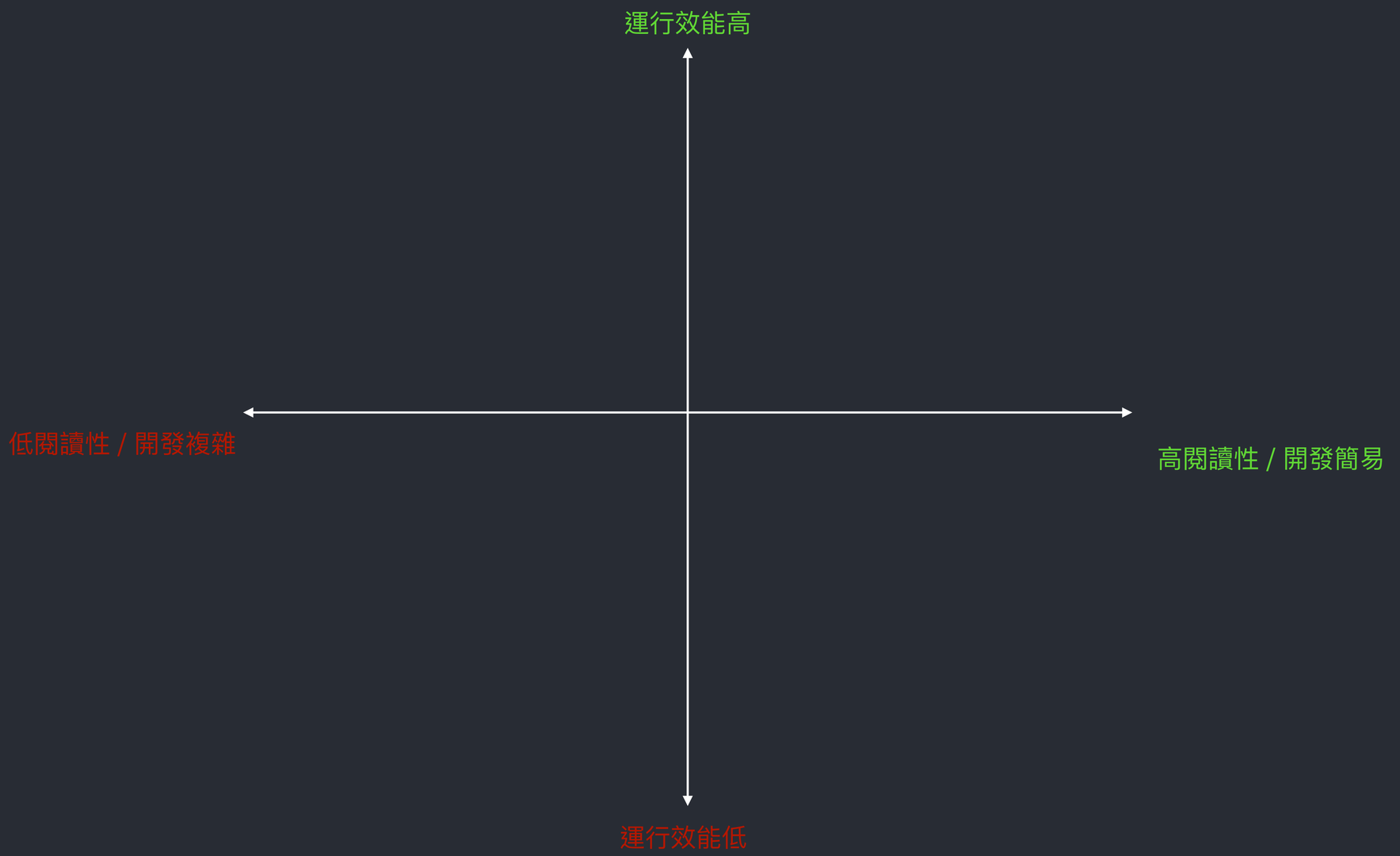
線程轉換
(Context Switching)

線程安全
(Thread Safety)


回調地獄
(Callback Hell)
低代碼可讀性 高維護成本
(Low Readability, High Maintenance Cost)
加工變換及組裝
(Composability)
異步編程及異步 I/O
複雜的異常處理
(Complex Exception Handling)
線程轉換
(Context Switching)
線程安全
(Thread Safety)
背壓
(Backpressure)

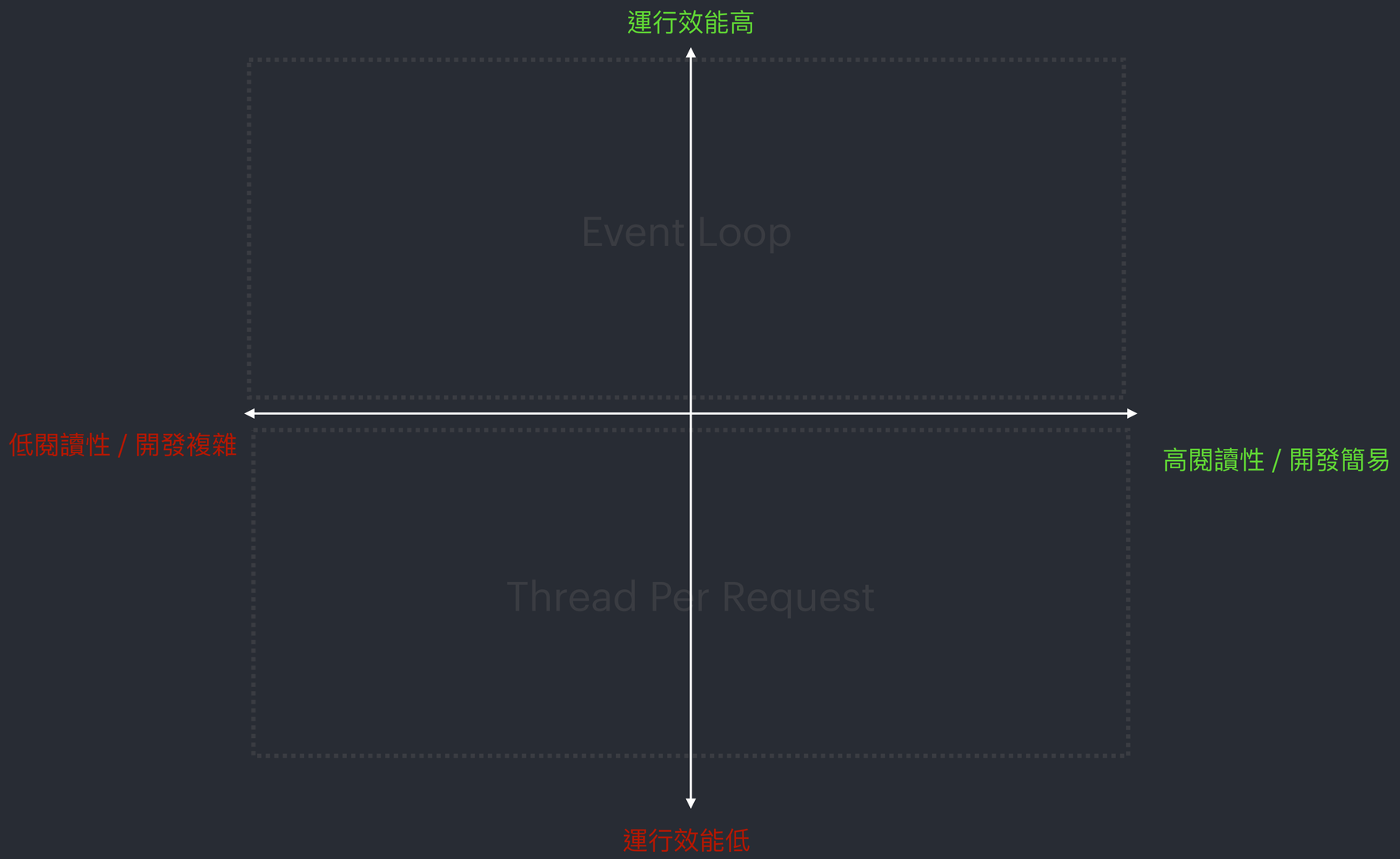
現今的設計方案

設計方案的開發者與運行效能二維分佈圖



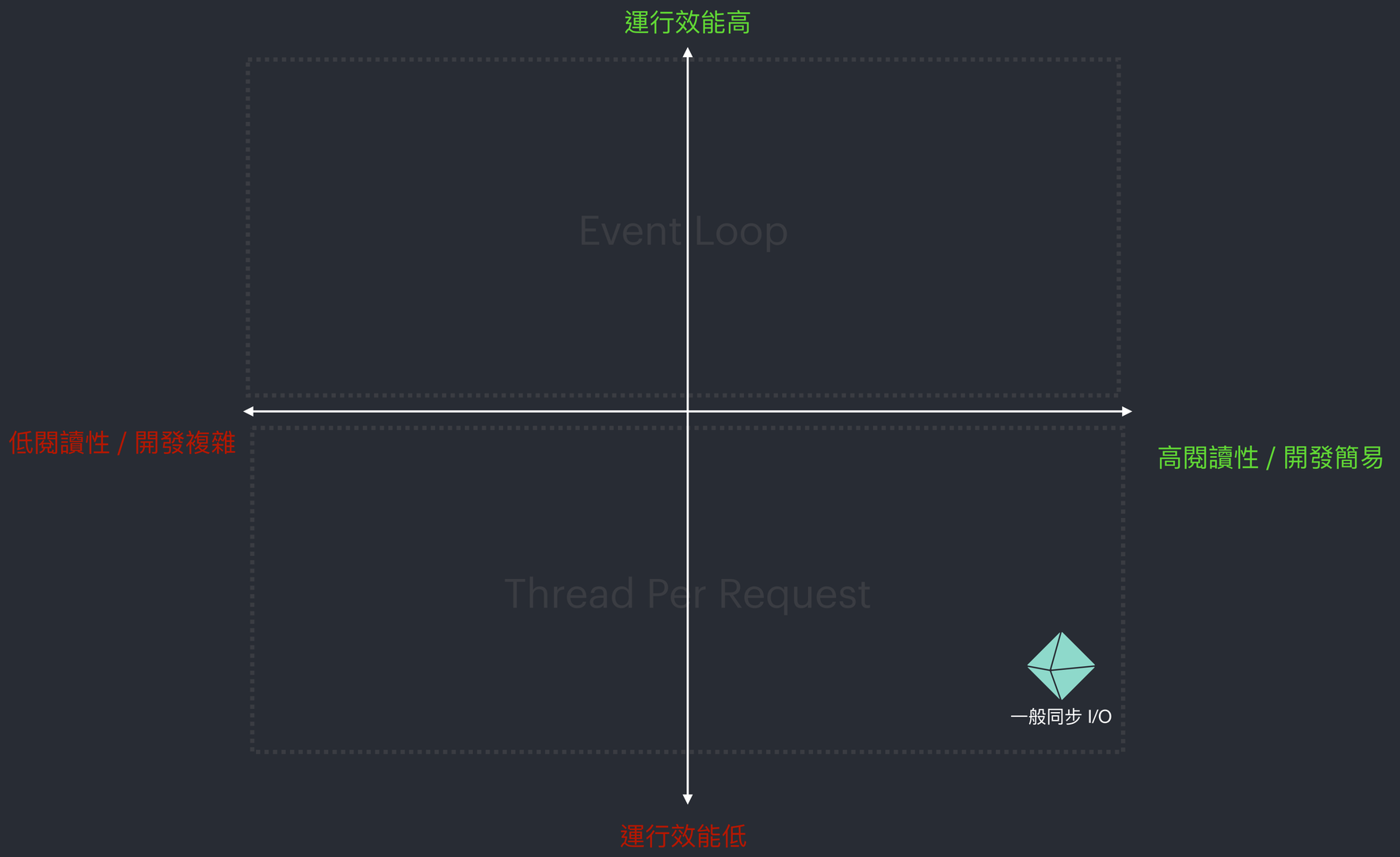
現今的設計方案

設計方案的開發者與運行效能二維分佈圖



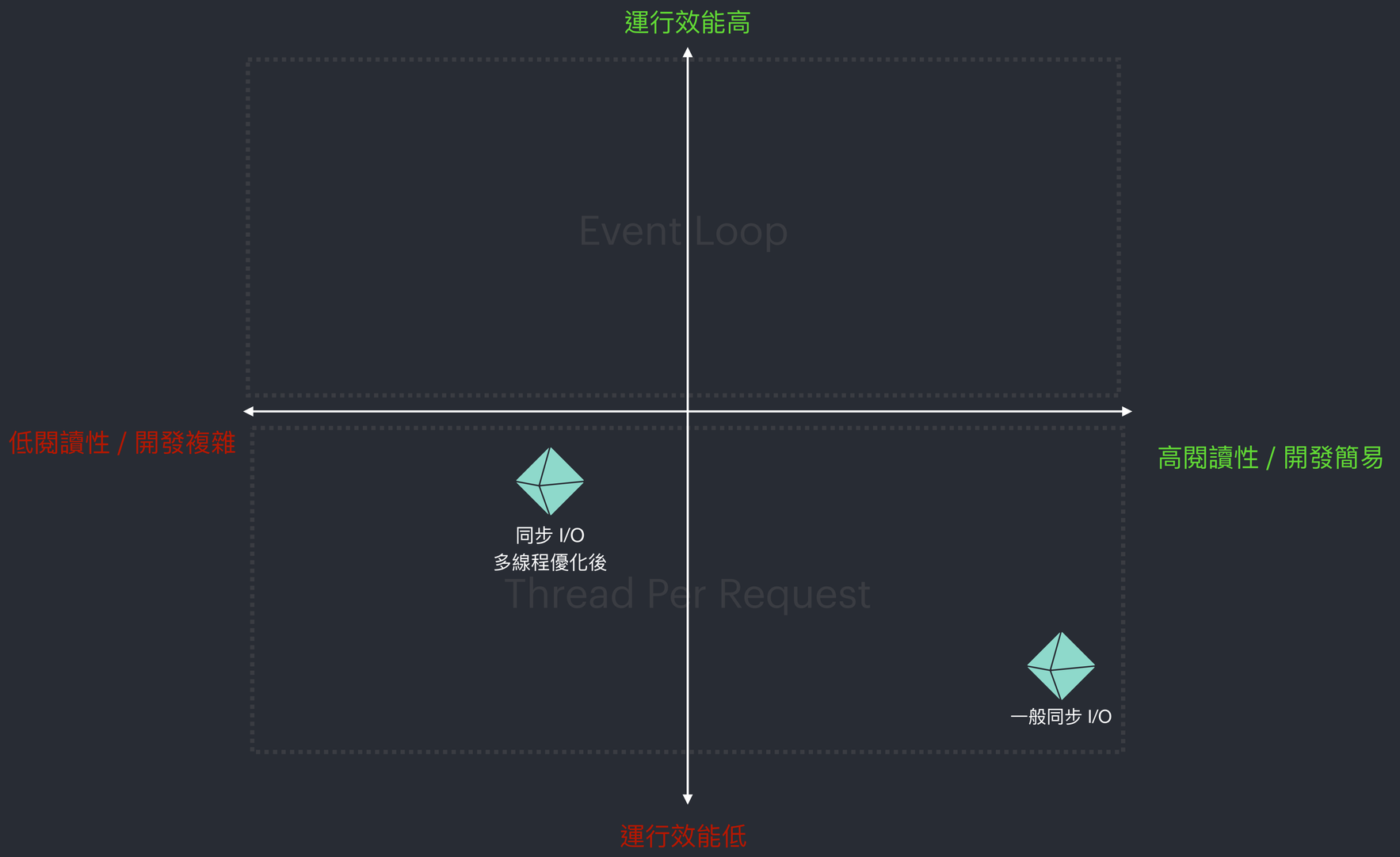
現今的設計方案

設計方案的開發者與運行效能二維分佈圖



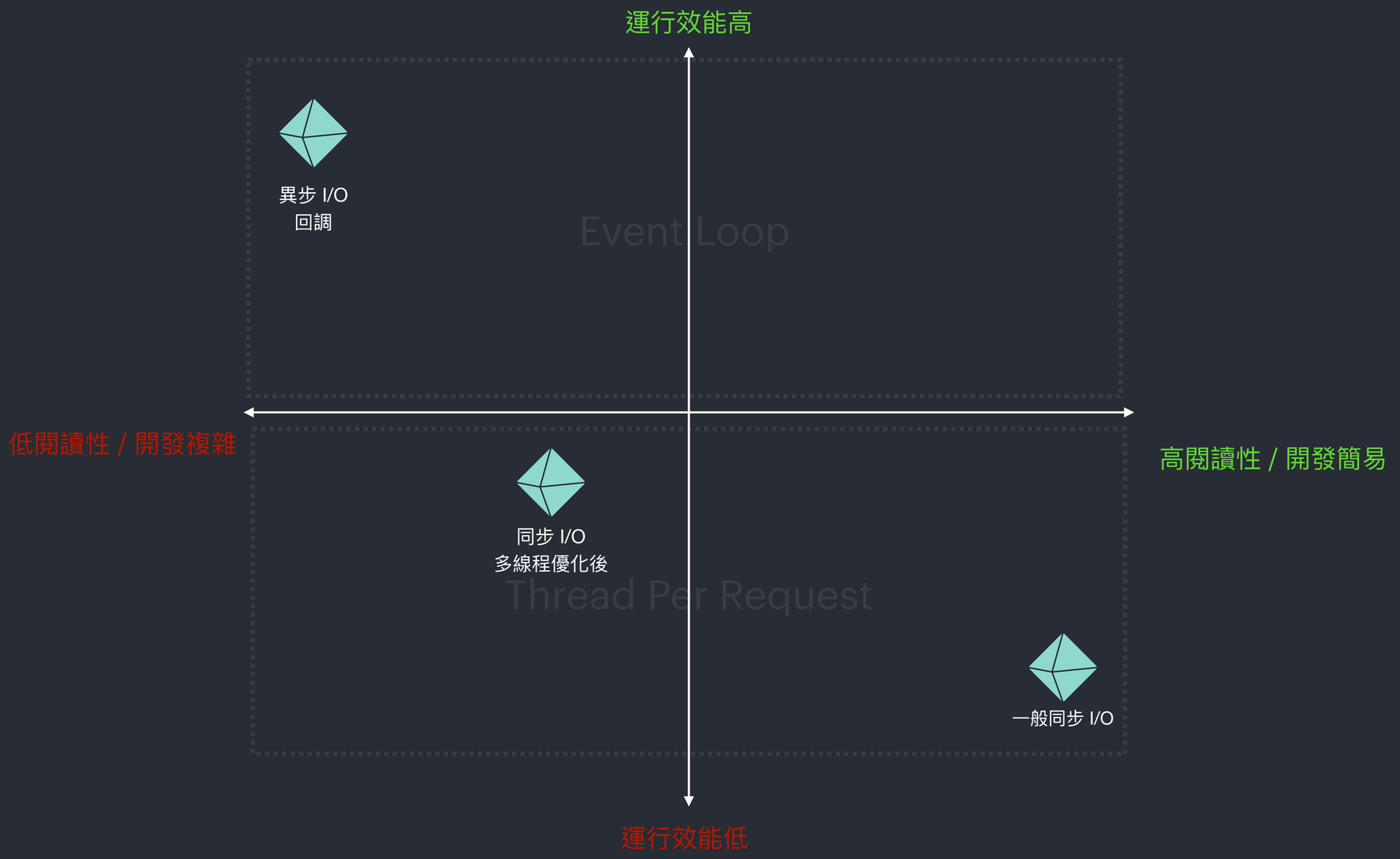
現今的設計方案

設計方案的開發者與運行效能二維分佈圖



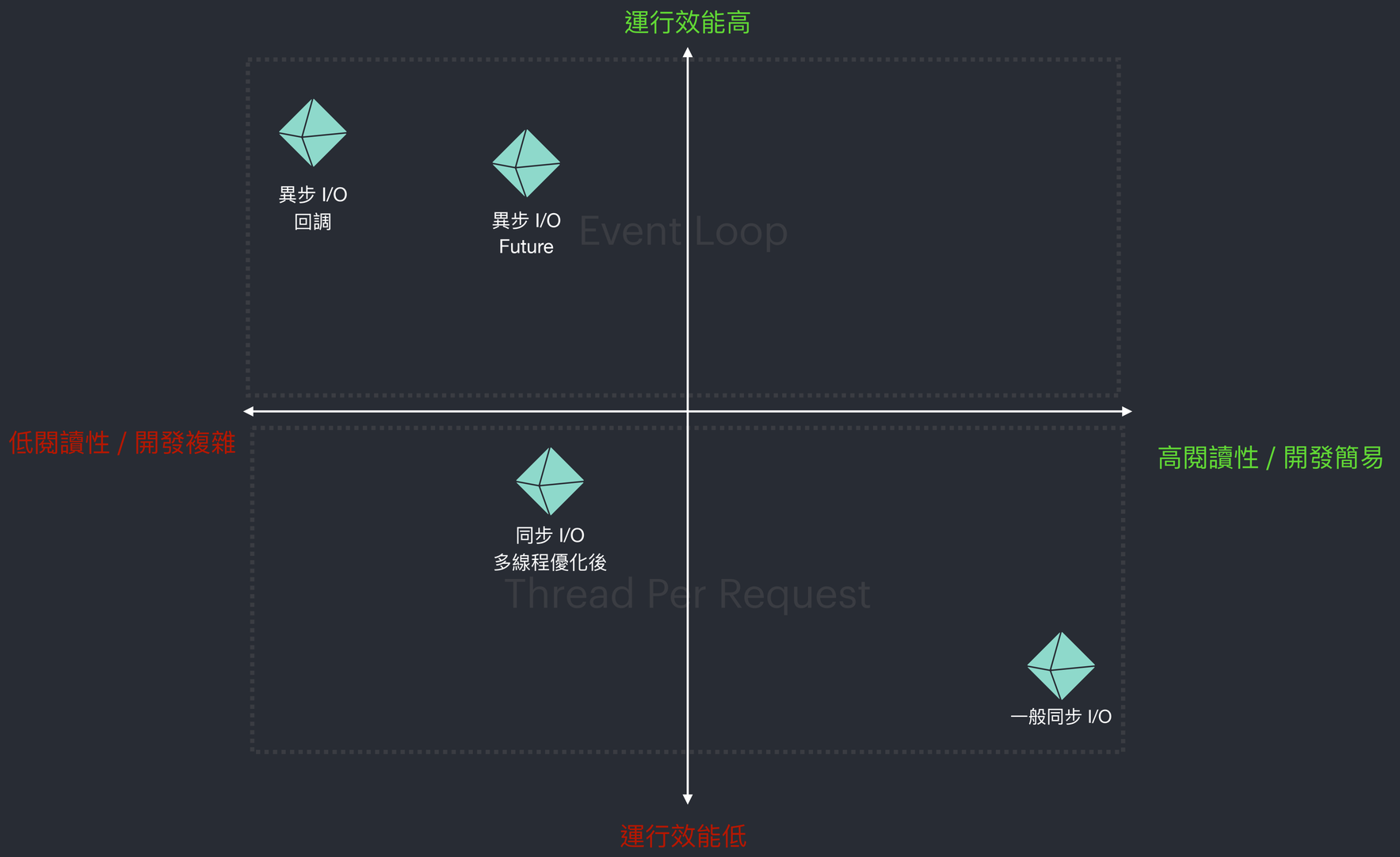
現今的設計方案

設計方案的開發者與運行效能二維分佈圖



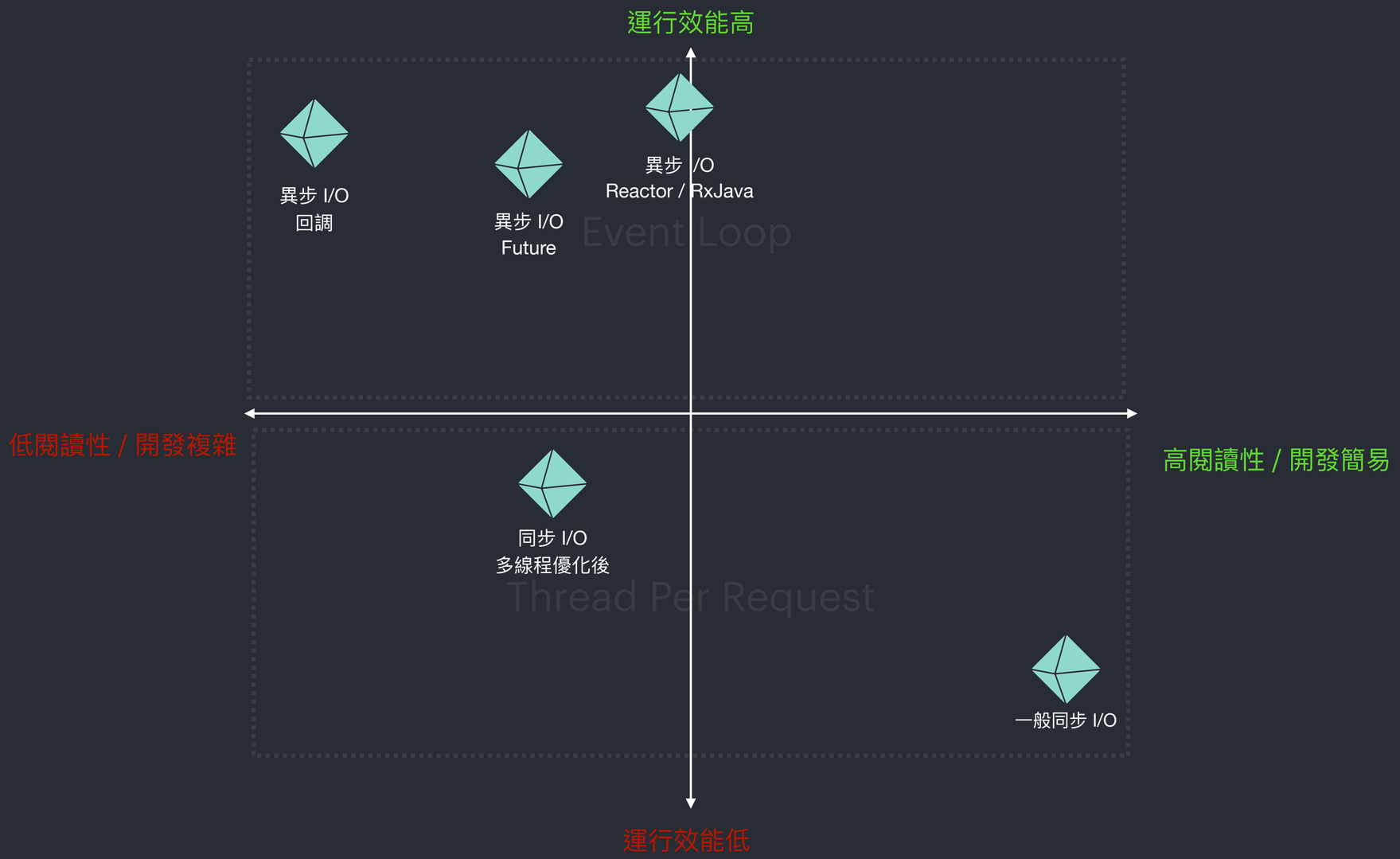
現今的設計方案

設計方案的開發者與運行效能二維分佈圖



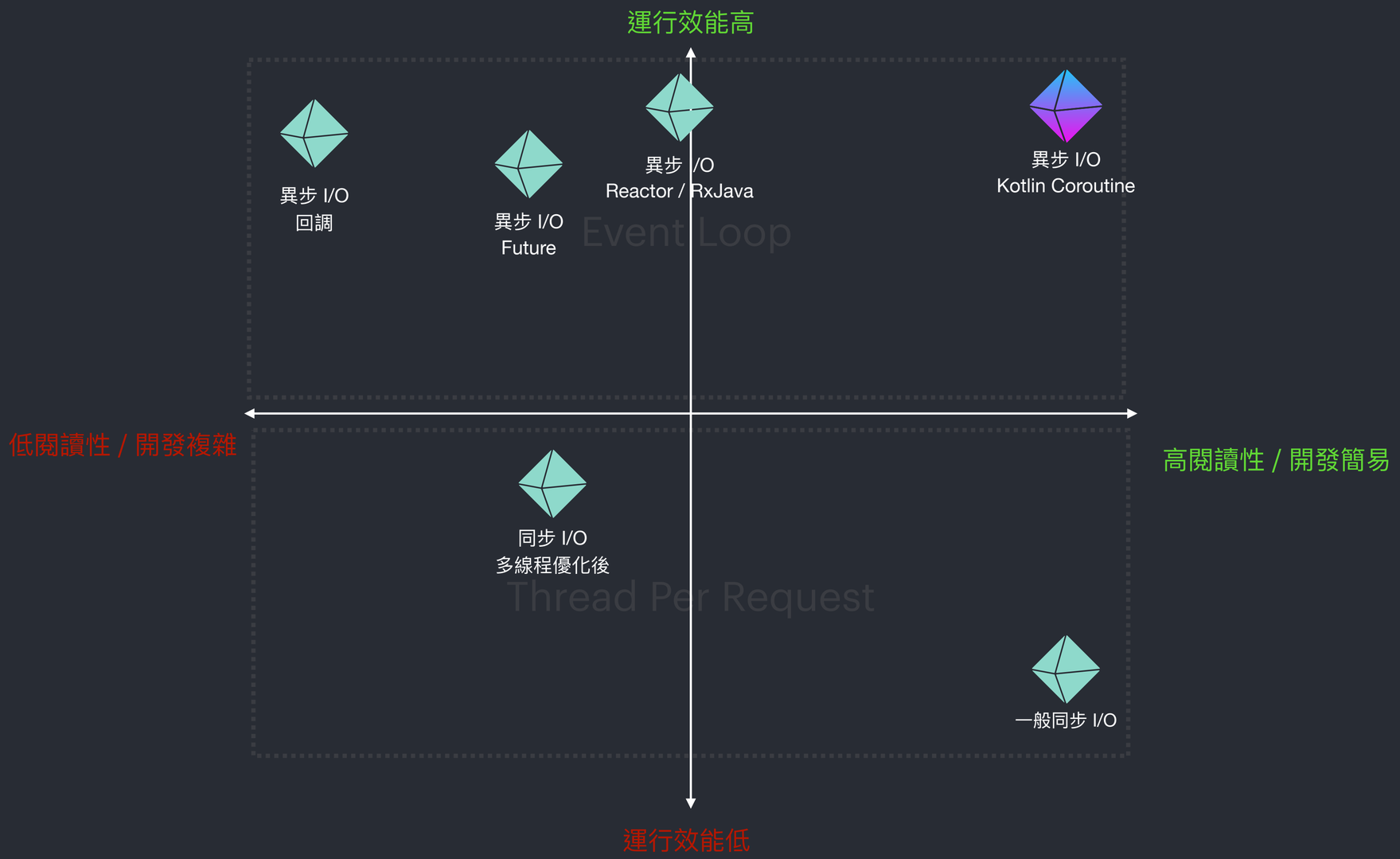
現今的設計方案

設計方案的開發者與運行效能二維分佈圖



現今的設計方案

設計方案的開發者與運行效能二維分佈圖



Kotlin 精湛的協程設計

現場代碼展示

CRUD 場景

Java 同步 I/O

假如有 3 個服務，接口都是同步 I/O

```
class RedeemResponse {
    int statusCode;
    int redeemRef;
}

interface Database {
    int getLoyaltyMemberId(int id);
}

interface RemoteService {
    RedeemResponse redeem(int memberId, int rewardId);
}

interface AnalyticsService {
    void trackRedeemAttempt(int userId, int memberId, int rewardId);
    void trackRedeemSuccess(int userId, int memberId, int rewardId, int redeemRef);
}
```

Java 同步 I/O

一個簡單的例子會發起 4 個 I/O

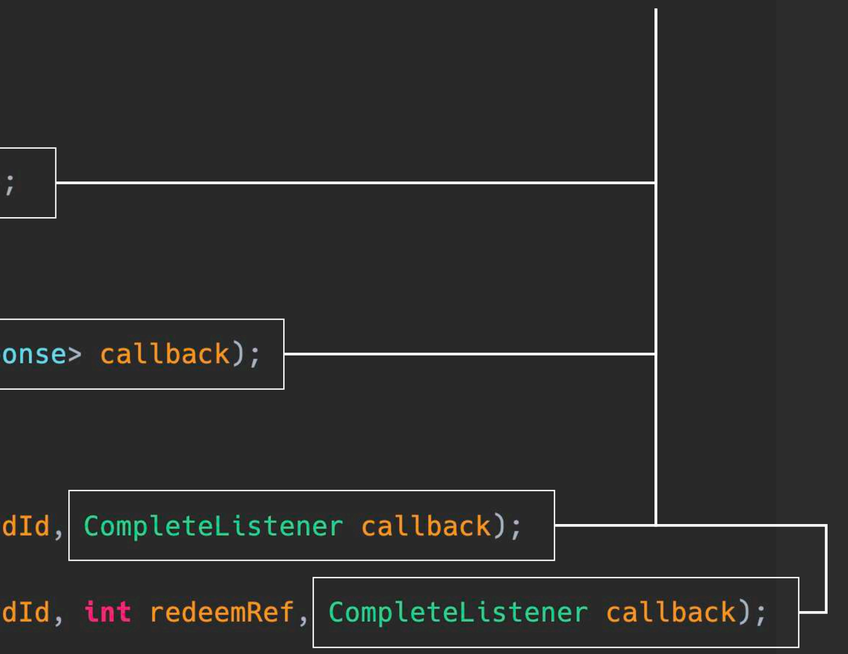
```java
// java, synchronous I/O
class LoyaltyService {
    Database db;
    RemoteService remoteService;
    AnalyticsService analyticsService;

    RedeemResponse redeemReward(int userId, int rewardId) {
        final var memberId :int = db.getLoyaltyMemberId(userId);

        // track the redeem action
        analyticsService.trackRedeemAttempt(userId, memberId, rewardId);

        final var redeemResponse :RedeemResponse = remoteService.redeem(memberId, rewardId);

        if(redeemResponse.statusCode == 200) {
            // only track redeem success
            analyticsService.trackRedeemSuccess(
                    userId,
                    memberId,
                    rewardId,
                    redeemResponse.redeemRef
            );
        }
        return redeemResponse;
    }
}
```

Java 異步 I/O (回調)

一個簡單的例子會發起 4 個 I/O

```java
class RedeemResponse {
    int statusCode;
    int redeemRef;
}

interface CompleteListener {
    void onComplete();
}

interface Database {
    void getLoyaltyMemberId(int id, Consumer<Integer> callback);
}

interface RemoteService {
    void redeem(int memberId, int rewardId, Consumer<RedeemResponse> callback);
}

interface AnalyticsService {
    void trackRedeemAttempt(int userId, int memberId, int rewardId, CompleteListener callback);

    void trackRedeemSuccess(int userId, int memberId, int rewardId, int redeemRef, CompleteListener callback);
}
```

需要使用 Lambda 處理回調



Java 異步 I/O (回調)

一個簡單的例子會發起 4 個 I/O

```java
// java, Asynchronous I/O (callback)
class LoyaltyService {
    Database db;
    RemoteService remoteService;
    AnalyticsService analyticsService;

    void redeemReward(int userId, int rewardId, Consumer<RedeemResponse> callback) {
        db.getLoyaltyMemberId(userId, memberId -> {
            analyticsService.trackRedeemAttempt(userId, memberId, rewardId, () -> {
                remoteService.redeem(memberId, rewardId, (redeemResponse) -> {
                    if (redeemResponse.statusCode == 200) {
                        // only track redeem success
                        analyticsService.trackRedeemSuccess(
                                userId,
                                memberId,
                                rewardId,
                                redeemResponse.redeemRef,
                                () -> callback.accept(redeemResponse)
                        );
                    }
                });
            });
        });
    }
}
```

回調地獄 (Callback Hell)

- 異常處理很困難
- 線程控制很困難
- 要額外考慮線程安全

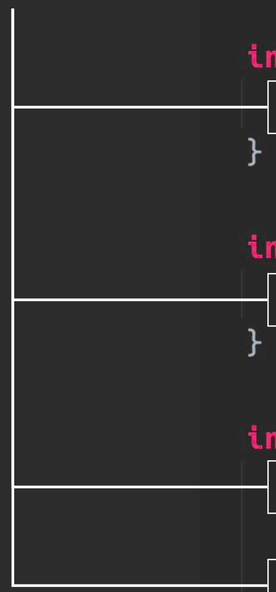
Color Scheme GapStyle

Java 異步 I/O (反應式 Reactor)

一個簡單的例子會發起 4 個 I/O

額外的封裝

提高代碼開發難度



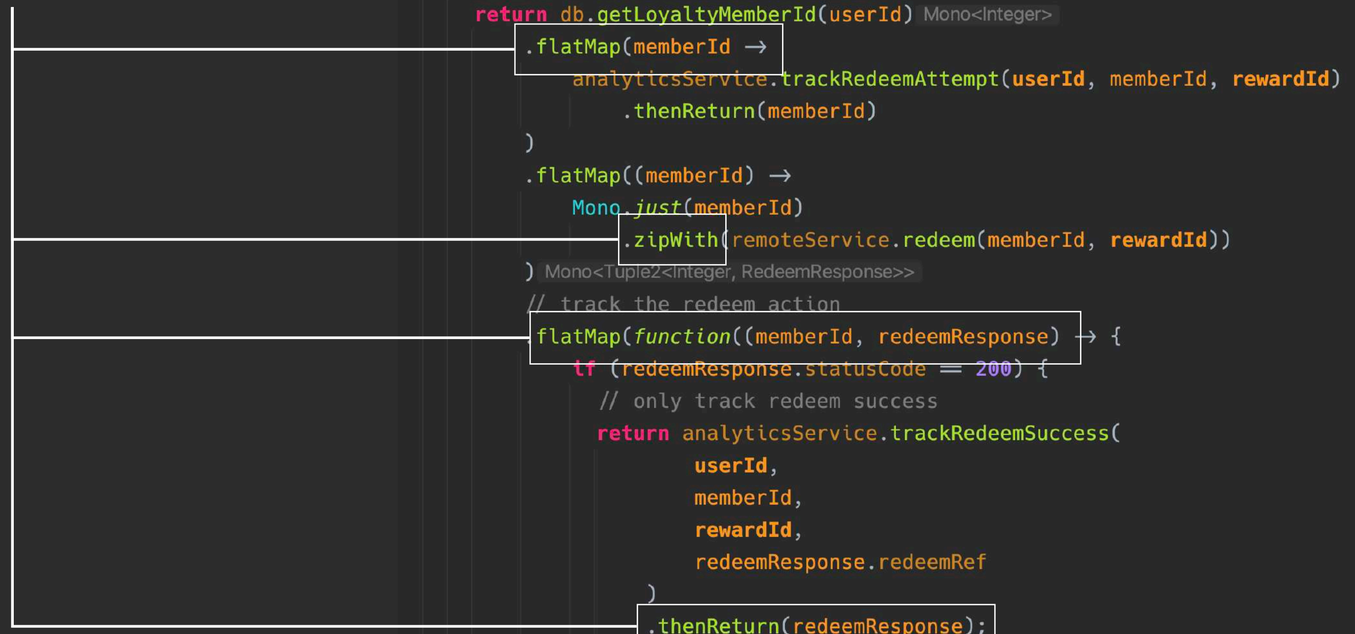
```java
class RedeemResponse {
  int statusCode;

  int redeemRef;
}

interface Database {
  Mono<Integer> getLoyaltyMemberId(int id);
}

interface RemoteService {
  Mono<RedeemResponse> redeem(int memberId, int rewardId);
}

interface AnalyticsService {
  Mono<Void> trackRedeemAttempt(int userId, int memberId, int rewardId);

  Mono<Void> trackRedeemSuccess(int userId, int memberId, int rewardId, int redeemRef);
}
```

Java 異步 I/O (反應式 Reactor)

一個簡單的例子會發起 4 個 I/O

額外的 Functional Programming 技巧

提高代碼開發難度



```java
// java, Asynchronous I/O, Reactive
class LoyaltyService {

  Database db;

  RemoteService remoteService;

  AnalyticsService analyticsService;

  Mono<RedeemResponse> redeemReward(int userId, int rewardId) {
    return db.getLoyaltyMemberId(userId) Mono<Integer>
        .flatMap(memberId →
            analyticsService.trackRedeemAttempt(userId, memberId, rewardId)
                .thenReturn(memberId)
        )
        .flatMap((memberId) →
            Mono.just(memberId)
                .zipWith(remoteService.redeem(memberId, rewardId))
        ) Mono<Tuple2<Integer, RedeemResponse>>
        // track the redeem action
        .flatMap(function((memberId, redeemResponse) → {
            if (redeemResponse.statusCode == 200) {
              // only track redeem success
              return analyticsService.trackRedeemSuccess(
                      userId,
                      memberId,
                      rewardId,
                      redeemResponse.redeemRef
                  )
                  .thenReturn(redeemResponse);
            }
            return Mono.just(redeemResponse);
          }));
  }
}
```

- 學習曲線高
- 需要額外的異常處理技巧
- 需要大量重構現有邏輯

Color Scheme GapStyle

Kotlin 異步 I/O (Coroutine)

加入一個 suspend 語法

與同步 I/O 的調用設計

```
class RedeemResponse {
  var statusCode = 0
  var redeemRef = 0
}

interface DatabaseCo {
  suspend fun getLoyaltyMemberIdFromAsync(id: Int): Int
  suspend fun getLoyaltyMemberIdFromSync(id: Int): Int
}

interface RemoteServiceCo {
  suspend fun redeem(memberId: Int, rewardId: Int): RedeemResponse
}

interface AnalyticsServiceCo {
  suspend fun trackRedeemAttempt(userId: Int, memberId: Int, rewardId: Int)
  suspend fun trackRedeemSuccess(userId: Int, memberId: Int, rewardId: Int, redeemRef: Int)
}
```

Kotlin 異步 I/O (Coroutine)

一個簡單的例子會發起 4 個 I/O

IDE 的清晰 Suspend 提示

```kotlin
// kotlin, Coroutine and structured concurrency
internal class LoyaltyService {
  lateinit var db: DatabaseCo
  lateinit var remoteService: RemoteServiceCo
  lateinit var analyticsService: AnalyticsServiceCo

  suspend fun redeemReward(userId: Int, rewardId: Int): RedeemResponse {
    val memberId = db.getLoyaltyMemberIdFromAsync(userId)

    // track the redeem action
    analyticsService.trackRedeemAttempt(userId, memberId, rewardId)
    val redeemResponse = remoteService.redeem(memberId, rewardId)
    if (redeemResponse.statusCode == 200) {
      // only track redeem success
      analyticsService.trackRedeemSuccess(
        userId,
        memberId,
        rewardId,
        redeemResponse.redeemRef
      )
    }
    return redeemResponse
  }
}
```

- 學習曲線低
- 一樣的異常處理技巧
- 不用重構現有邏輯

Color Scheme GapStyle

Coroutine 封裝現有 I/O 場景

Kotlin Coroutine 封裝現有 I/O 例子

加入 suspend 語法

```
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.withContext
import java.util.function.Consumer
import kotlin.coroutines.resume
import kotlin.coroutines.suspendCoroutine

interface DatabaseSync {
  fun getLoyaltyMemberId(id: Int): Int
}
interface DatabaseAsync {
  fun getLoyaltyMemberId(id: Int, callback: Consumer<Int>)
}
interface DatabaseCo {
  suspend fun getLoyaltyMemberIdFromAsync(id: Int): Int
  suspend fun getLoyaltyMemberIdFromSync(id: Int): Int
}

class DatabaseCoImpl: DatabaseCo {
  lateinit var dbAsync: DatabaseAsync
  override suspend fun getLoyaltyMemberIdFromAsync(id: Int): Int = suspendCoroutine { cont →
    dbAsync.getLoyaltyMemberId(id) { memberId →
      cont.resume(memberId)
    }
  }
  lateinit var dbSync: DatabaseSync
  override suspend fun getLoyaltyMemberIdFromSync(id: Int): Int = withContext(Dispatchers.IO) {
    dbSync.getLoyaltyMemberId(id)
  }
}
```

創建 suspendCoroutine
定義 resume

轉換到 I/O 線程池
同時傳回 Lambda 最後值

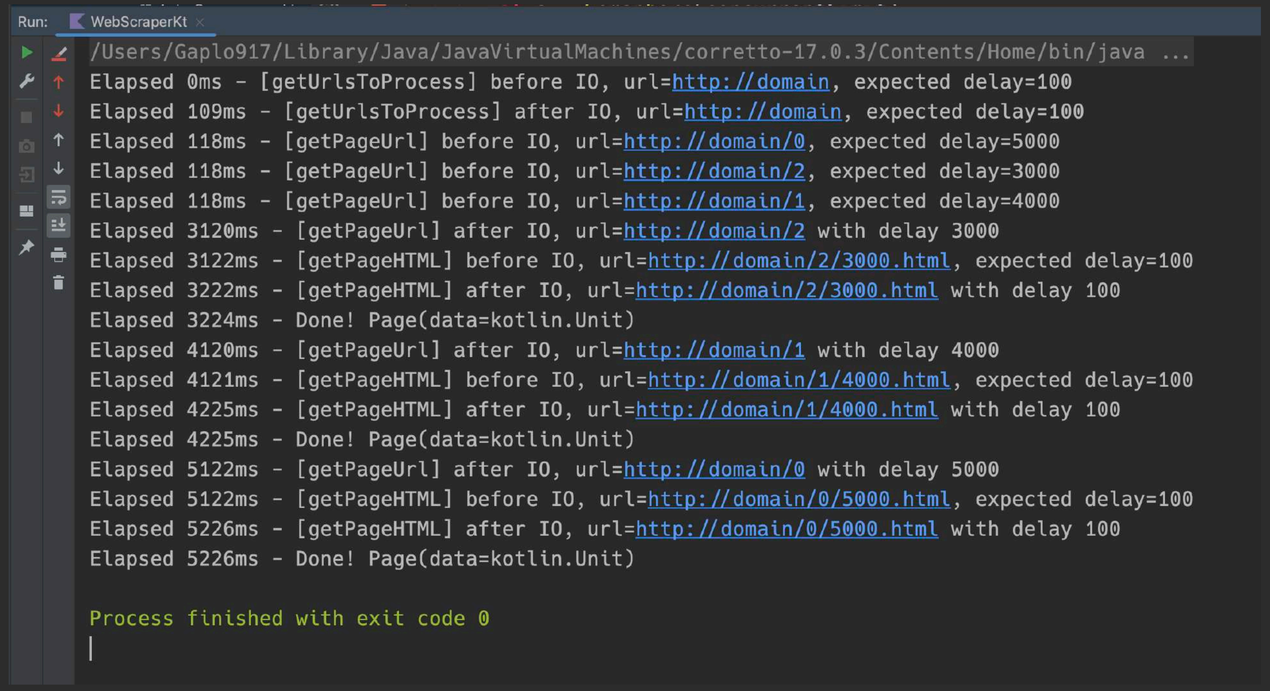
Color Scheme GapStyle

網頁爬蟲工具場景

網頁爬蟲工具

利用 Kotlin Coroutine ChannelFlow 設計實現高並發爬蟲

```
import kotlinx.coroutines.flow.*
import kotlinx.coroutines.sync.Semaphore
import kotlinx.coroutines.sync.withPermit

data class Page(val data: Any)
suspend fun getUrlsToProcess(url: String): List<String> {...}
suspend fun getPageUrl(id: Int, url: String): String {...}
suspend fun getPageHTML(url: String): Page {...}

private fun runWebScraper() = runBlocking { this: CoroutineScope
  channelFlow { this: ProducerScope<String>
    getUrlsToProcess( url: "http://domain").forEachIndexed { index, url →
      launch { this: CoroutineScope
        send(getPageUrl(index, url))
      }
    }
  }
    .map { itemDetailUrl →
      getPageHTML(itemDetailUrl)
    }
    .flowOn(Dispatchers.Default)
    .collect { it: Page
      log( s: "Done! $it")
    }
}
```



兩句代碼實現 N 並發請求 (concurrent request)

Color Scheme GapStyle

網頁爬蟲工具

限制 Kotlin Coroutine ChannelFlow 並發數目

```kotlin
fun main() {
    runWebScraperLimit( concurrentLimit: 2)
}

fun runWebScraperLimit(concurrentLimit: Int) {
    runBlocking { this: CoroutineScope
        channelFlow { this: ProducerScope<String>
            val s = Semaphore(concurrentLimit)
            getUrlsToProcess( url: "http://domain").forEachIndexed { index, url →
                launch { this: CoroutineScope
                    s.withPermit {
                        send(getPageUrl(index, url))
                    }
                }
            }
        }
            .map { itemDetailUrl →
                getPageHTML(itemDetailUrl)
            }
            .flowOn(Dispatchers.Default)
            .collect { it: Page
                log( s: "Done! $it")
            }
    }
}
```

```
Run:  WebScraperKt
/Users/Gaplo917/Library/Java/JavaVirtualMachines/corretto-17.0.3/Contents/Home/bin/java ...
Elapsed 0ms - [getUrlsToProcess] before IO, url=http://domain, expected delay=100
Elapsed 109ms - [getUrlsToProcess] after IO, url=http://domain, expected delay=100
Elapsed 121ms - [getPageUrl] before IO, url=http://domain/0, expected delay=5000
Elapsed 121ms - [getPageUrl] before IO, url=http://domain/2, expected delay=3000
Elapsed 3123ms - [getPageUrl] after IO, url=http://domain/2 with delay 3000
Elapsed 3124ms - [getPageUrl] before IO, url=http://domain/1, expected delay=4000
Elapsed 3125ms - [getPageHTML] before IO, url=http://domain/2/3000.html, expected delay=100
Elapsed 3229ms - [getPageHTML] after IO, url=http://domain/2/3000.html with delay 100
Elapsed 3234ms - Done! Page(data=kotlin.Unit)
Elapsed 5121ms - [getPageUrl] after IO, url=http://domain/0 with delay 5000
Elapsed 5122ms - [getPageHTML] before IO, url=http://domain/0/5000.html, expected delay=100
Elapsed 5226ms - [getPageHTML] after IO, url=http://domain/0/5000.html with delay 100
Elapsed 5226ms - Done! Page(data=kotlin.Unit)
Elapsed 7128ms - [getPageUrl] after IO, url=http://domain/1 with delay 4000
Elapsed 7129ms - [getPageHTML] before IO, url=http://domain/1/4000.html, expected delay=100
Elapsed 7232ms - [getPageHTML] after IO, url=http://domain/1/4000.html with delay 100
Elapsed 7232ms - Done! Page(data=kotlin.Unit)
```

再加入兩句代碼限制 2 並發請求 (concurrent request)

Color Scheme GapStyle

Actor 並發模型場景

Actor 並發模型

利用 Kotlin Coroutine Channel 設計 Actors

```kotlin
import kotlinx.coroutines.*
import kotlinx.coroutines.channels.Channel
import kotlinx.coroutines.channels.SendChannel

// actor envelop
sealed class ActorEnvelop<T>(open val message: T)

object Reset : ActorEnvelop<Unit>(Unit)
data class Plus(val value: Int) : ActorEnvelop<Int>(value)
data class Minus(val value: Int) : ActorEnvelop<Int>(value)
data class TellManager(val manager: SendChannel<ManagerEnvelop<*>>) : ActorEnvelop<Unit>(Unit)

// manager envelop
sealed class ManagerEnvelop<T>(open val message: T)

data class AggregatedResult(
  val sender: SendChannel<ActorEnvelop<*>>,
  val result: Int,
  val count: Int
) : ManagerEnvelop<Int>(result)
```

定義 Actor 之間的通訊郵件

Actor 並發模型

利用 Kotlin Coroutine Channel 設計 Actors

1 Actor 只有 1 線程

不用擔心線程安全問題去使用 Atomic

Actor 被 Suspend 去等待郵件

等待發送郵件完成
才做下一個訊息

```
27  val actor: SendChannel<ActorEnvelop<*>> by lazy {
28    Channel<ActorEnvelop<*>>().also { self →
29      GlobalScope.launch(Dispatchers.Default) { this: CoroutineScope
30        log( s: "actor launched with context=${coroutineContext}, thread=${Thread.currentThread()}")
31        var aggregatedValue = 0
32        var operationCount = 0
33        for (envelop in self) {
34          log( s: "received ActorEnvelop=($envelop) with context=${coroutineContext}, thread=${Thread.currentThread()}")
35          when (envelop) {
36            is Plus → {
37              aggregatedValue += envelop.value
38              operationCount += 1
39            }
40
41            is Minus → {
42              aggregatedValue -= envelop.value
43              operationCount += 1
44            }
45
46            is Reset → {
47              aggregatedValue = 0
48              operationCount = 0
49            }
50
51            is TellManager → {
52              envelop.manager.send(AggregatedResult(self, aggregatedValue, operationCount))
53            }
54          }
55        }
56      }
57    }
58  }
```

Color Scheme GapStyle

Actor 並發模型

利用 Kotlin Coroutine Channel 設計 Actors

```
val allDone: Channel<Unit> by lazy { Channel() }

val manager: SendChannel<ManagerEnvelop<*>> by lazy {
  Channel<ManagerEnvelop<*>>().also { self →
    GlobalScope.launch(Dispatchers.Default) { this: CoroutineScope
      for (envelop in self) {
        log( s: "received ManagerEnvelop=$envelop with context=${coroutineContext}, thread=${Thread.currentThread()}")
        when (envelop) {
          is AggregatedResult → {
            if (envelop.count ⩾ 10) {
              log( s: "Manager is checking the result...")
              delay( timeMillis: 1000)
              envelop.sender.send(Reset)
              log( s: "Completed, sending done message")
              allDone.send(Unit)
            }
          }
        }
      }
    }
  }
}
```

經理實現

Color Scheme GapStyle

Actor 並發模型

利用 Kotlin Coroutine Channel 實測

```kotlin
suspend fun sendTaskToActorInParallel(size: Int) = coroutineScope {
  Array(size) {
    async(Dispatchers.IO) {
      Plus(1).let { envelop ->
        log("send ActorEnvelop=$envelop with context=${coroutineContext}, thread=${Thread.currentThread().id}")
        actor.send(envelop)
      }
    }
  }.toList()
   .awaitAll()
}

fun runActorExample() = runBlocking {
  sendTaskToActorInParallel(10)

  actor.send(TellManager(manager))

  // wait allDone received signal
  allDone.receive()
}
```

Run: ActorConcurrencyKt

```
/Users/Gaplo917/Library/Java/JavaVirtualMachines/corretto-17.0.3/Contents/Home/bin/java ...
Elapsed 0ms - send ActorEnvelop=Plus(value=1) with context=[DeferredCoroutine{Active}@73f44cd0, Dispatchers.IO], thread=31
Elapsed 0ms - send ActorEnvelop=Plus(value=1) with context=[DeferredCoroutine{Active}@4da81cb2, Dispatchers.IO], thread=26
Elapsed 0ms - send ActorEnvelop=Plus(value=1) with context=[DeferredCoroutine{Active}@73eab246, Dispatchers.IO], thread=28
Elapsed 0ms - send ActorEnvelop=Plus(value=1) with context=[DeferredCoroutine{Active}@54ed4b62, Dispatchers.IO], thread=33
Elapsed 0ms - send ActorEnvelop=Plus(value=1) with context=[DeferredCoroutine{Active}@23554c93, Dispatchers.IO], thread=27
Elapsed 0ms - send ActorEnvelop=Plus(value=1) with context=[DeferredCoroutine{Active}@1d84f779, Dispatchers.IO], thread=24
Elapsed 0ms - send ActorEnvelop=Plus(value=1) with context=[DeferredCoroutine{Active}@54bdf390, Dispatchers.IO], thread=29
Elapsed 0ms - send ActorEnvelop=Plus(value=1) with context=[DeferredCoroutine{Active}@49a67305, Dispatchers.IO], thread=23
Elapsed 0ms - send ActorEnvelop=Plus(value=1) with context=[DeferredCoroutine{Active}@65ee21f, Dispatchers.IO], thread=25
Elapsed 0ms - send ActorEnvelop=Plus(value=1) with context=[DeferredCoroutine{Active}@33457bd4, Dispatchers.IO], thread=30
Elapsed 11ms - actor launched with context=[StandaloneCoroutine{Active}@4add4f46, Dispatchers.Default], thread=Thread[DefaultDispatcher-worker-12,5,main]
Elapsed 15ms - received ActorEnvelop=(Plus(value=1)) with context=[StandaloneCoroutine{Active}@4add4f46, Dispatchers.Default],
 thread=Thread[DefaultDispatcher-worker-12,5,main]
Elapsed 15ms - received ActorEnvelop=(Plus(value=1)) with context=[StandaloneCoroutine{Active}@4add4f46, Dispatchers.Default],
 thread=Thread[DefaultDispatcher-worker-12,5,main]
Elapsed 15ms - received ActorEnvelop=(Plus(value=1)) with context=[StandaloneCoroutine{Active}@4add4f46, Dispatchers.Default],
 thread=Thread[DefaultDispatcher-worker-12,5,main]
Elapsed 15ms - received ActorEnvelop=(Plus(value=1)) with context=[StandaloneCoroutine{Active}@4add4f46, Dispatchers.Default],
 thread=Thread[DefaultDispatcher-worker-12,5,main]
Elapsed 15ms - received ActorEnvelop=(Plus(value=1)) with context=[StandaloneCoroutine{Active}@4add4f46, Dispatchers.Default],
 thread=Thread[DefaultDispatcher-worker-12,5,main]
Elapsed 15ms - received ActorEnvelop=(Plus(value=1)) with context=[StandaloneCoroutine{Active}@4add4f46, Dispatchers.Default],
 thread=Thread[DefaultDispatcher-worker-12,5,main]
Elapsed 15ms - received ActorEnvelop=(Plus(value=1)) with context=[StandaloneCoroutine{Active}@4add4f46, Dispatchers.Default],
 thread=Thread[DefaultDispatcher-worker-12,5,main]
Elapsed 15ms - received ActorEnvelop=(Plus(value=1)) with context=[StandaloneCoroutine{Active}@4add4f46, Dispatchers.Default],
 thread=Thread[DefaultDispatcher-worker-12,5,main]
Elapsed 16ms - received ActorEnvelop=(Plus(value=1)) with context=[StandaloneCoroutine{Active}@4add4f46, Dispatchers.Default],
 thread=Thread[DefaultDispatcher-worker-12,5,main]
Elapsed 16ms - received ActorEnvelop=(Plus(value=1)) with context=[StandaloneCoroutine{Active}@4add4f46, Dispatchers.Default],
 thread=Thread[DefaultDispatcher-worker-12,5,main]
Elapsed 18ms - received ActorEnvelop=(TellManager(manager=RendezvousChannel@1bc5fd2f{ReceiveQueued})) with context=[StandaloneCoroutine{Active}@4add4f46, Dispatchers
 .Default], thread=Thread[DefaultDispatcher-worker-14,5,main]
Elapsed 22ms - received ManagerEnvelop=AggregatedResult(sender=RendezvousChannel@6c951fb7{ReceiveQueued}, result=10, count=10) with
 context=[StandaloneCoroutine{Active}@1a956fb0, Dispatchers.Default], thread=Thread[DefaultDispatcher-worker-14,5,main]
Elapsed 22ms - Manager is checking the result...
Elapsed 1023ms - Completed, sending done message
Elapsed 1024ms - received ActorEnvelop=(kotlin2022.chinese.talk.Reset@4708b059) with context=[StandaloneCoroutine{Active}@4add4f46, Dispatchers.Default],
 thread=Thread[DefaultDispatcher-worker-12,5,main]
```

Color Scheme GapStyle

“Java 19 也支援了虛擬線程(Virtual Thread)及結構化並發(Structured Concurrency)。

為什麼我要投資到 *Kotlin* ？

一起看看 Java 19 的結構化並發

現場代碼展示

Java 19 的結構化並發

那一年才能使用 Java 19？ 三年後？

```java
// required Spring MVC to work in virtual threads pool
Gary Lo
@RequestMapping("/mvc-bio-structured-concurrency-parallel/{ioDelay}")
public ResponseEntity<List<DummyResponse>> blockingNativeJavaStructuredConcurrencyApi(
    @PathVariable Long ioDelay
) throws InterruptedException, ExecutionException {
  try (var scope = new StructuredTaskScope.ShutdownOnFailure()) {
    final var future1 : Future<DummyResponse> = scope.fork(() -> ioService.blockingIO(ioDelay));
    final var future2 : Future<DummyResponse> = scope.fork(() -> ioService.blockingIO(ioDelay));
    // Wait for all threads to finish or the task scope to shut down.
    // This method waits until all threads started in the task scope finish execution
    scope.join();
    scope.throwIfFailed();
    return ResponseEntity.ok(List.of(future1.resultNow(), future2.resultNow()));
  }
}
}
```

現有的 StructuredTaskScope 不能直接支援異步 I/O，也沒有 Suspend 標誌

Kotlin Coroutine 的結構化並發

現在就可以用

```
Gary Lo
@RequestMapping("/mvc-nio-coroutine-parallel/{ioDelay}")
suspend fun nonBlockingCoroutineParallelIOApi(@PathVariable ioDelay: Long): ResponseEntity<List<DummyResponse>> {
    // kotlinx-coroutines-jdk8 extension
    // `suspend fun <T> CompletionStage<T>.await(): T`
    return coroutineScope { this: CoroutineScope
        val resp1 = async { ioService.nonBlockingIO(ioDelay).await() }
        val resp2 = async { ioService.nonBlockingIO(ioDelay).await() }
        ResponseEntity.ok(listOf(resp1.await(), resp2.await())) ^coroutineScope
    }
}
```

平行結構化並發，IDE 有 suspend 符號標誌令代碼更易閱讀

Kotlin Coroutine 使用 Java 19 Virtual Thread

現在就可以用

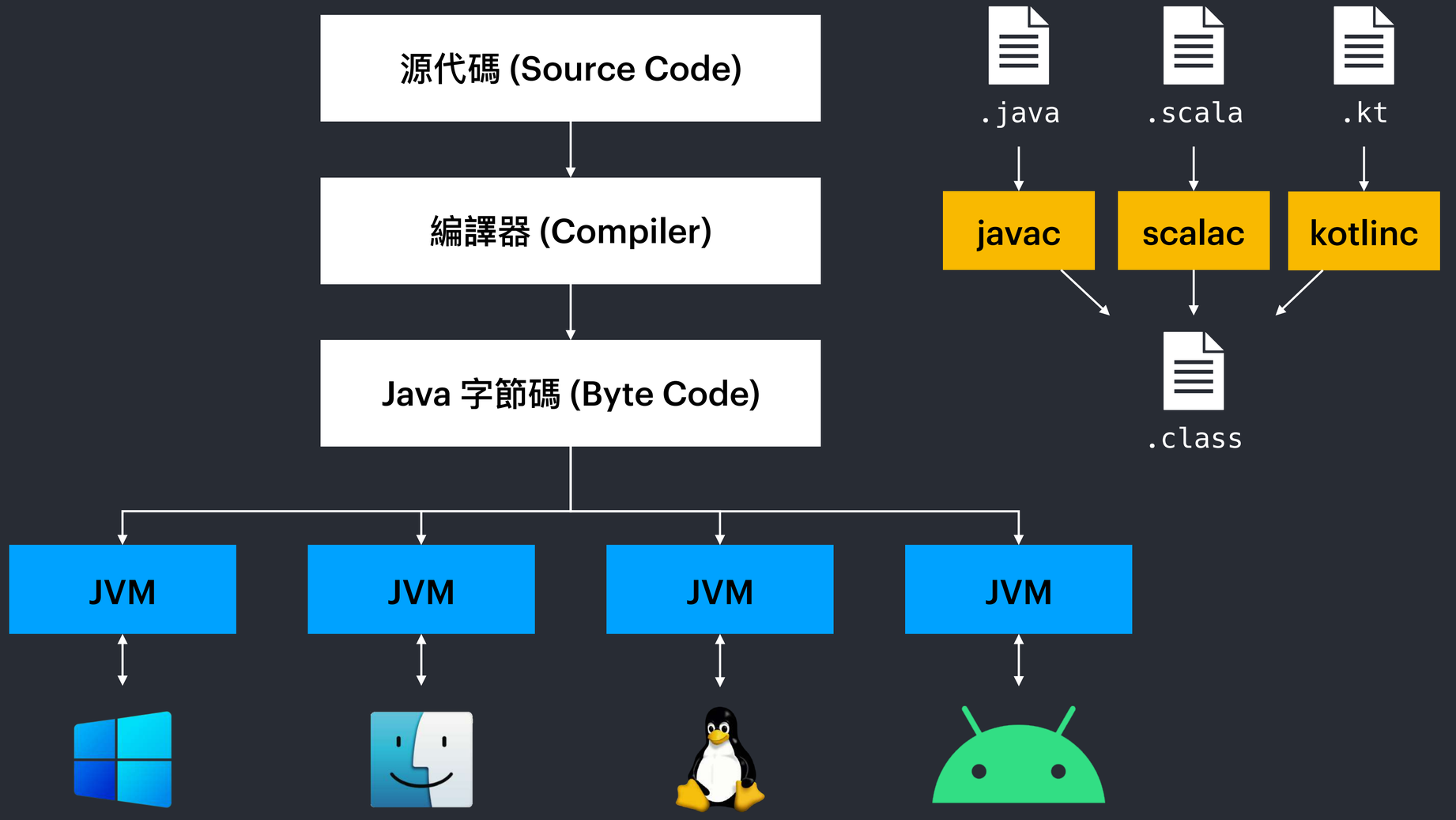
```
private val virtualThreadExecutor = Executors.newVirtualThreadPerTaskExecutor()
private val virtualThreadCoroutineDispatcher = virtualThreadExecutor.asCoroutineDispatcher()

Gary Lo
@RequestMapping("/mvc-bio-coroutine-parallel/{ioDelay}")
suspend fun blockingCoroutineInVTParallelIOApi(@PathVariable ioDelay: Long): ResponseEntity<List<DummyResponse>> =
    withContext(virtualThreadCoroutineDispatcher) { this: CoroutineScope
        val resp1 = async { ioService.blockingIO(ioDelay) }
        val resp2 = async { ioService.blockingIO(ioDelay) }
        ResponseEntity.ok(listOf(resp1.await(), resp2.await())) ^withContext
    }
```

將同步I/O 封裝在 Java 19 Virtual Thread 內，不佔用現有線程池

Color Scheme GapStyle

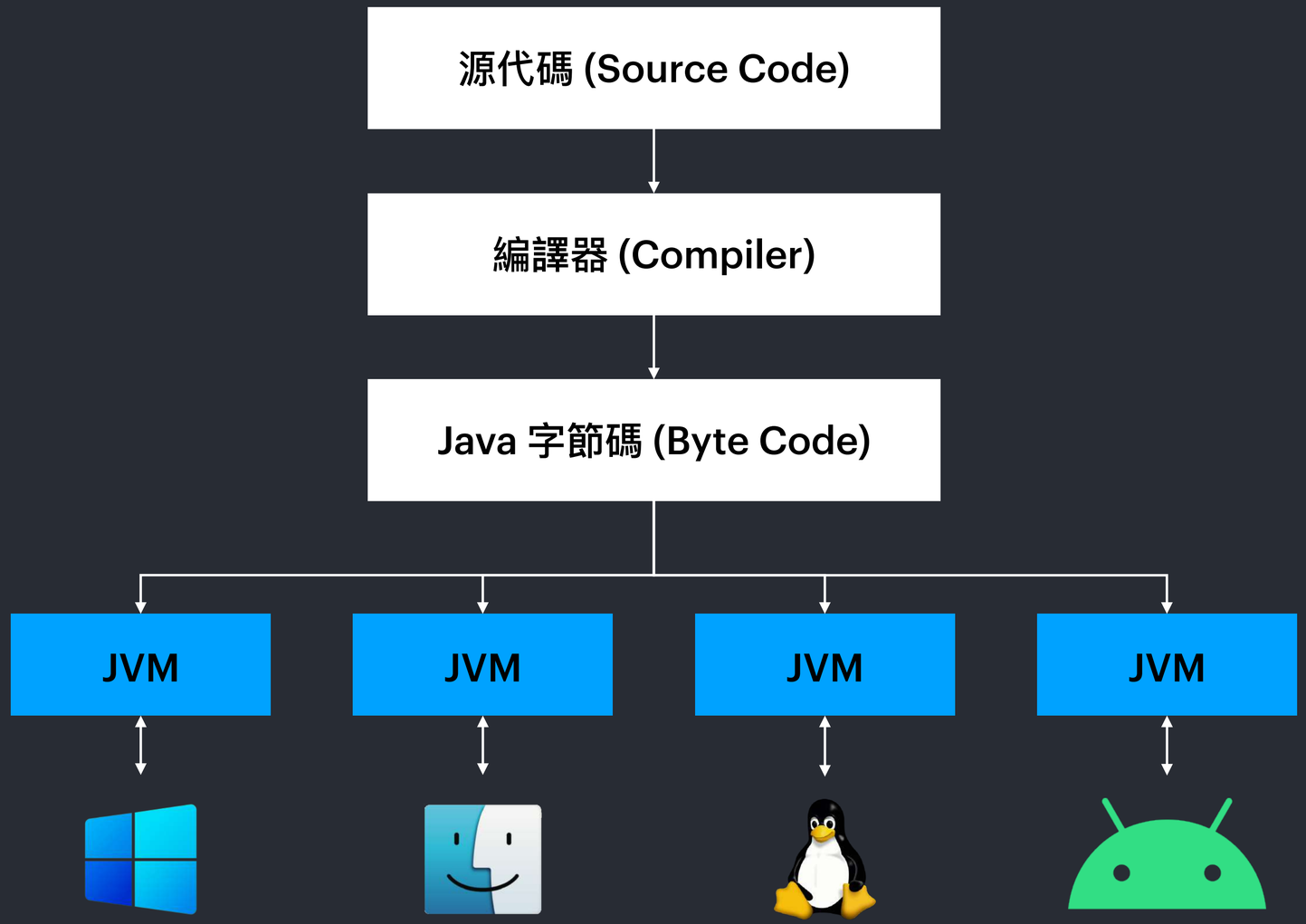
JVM

由開發到執行的過程



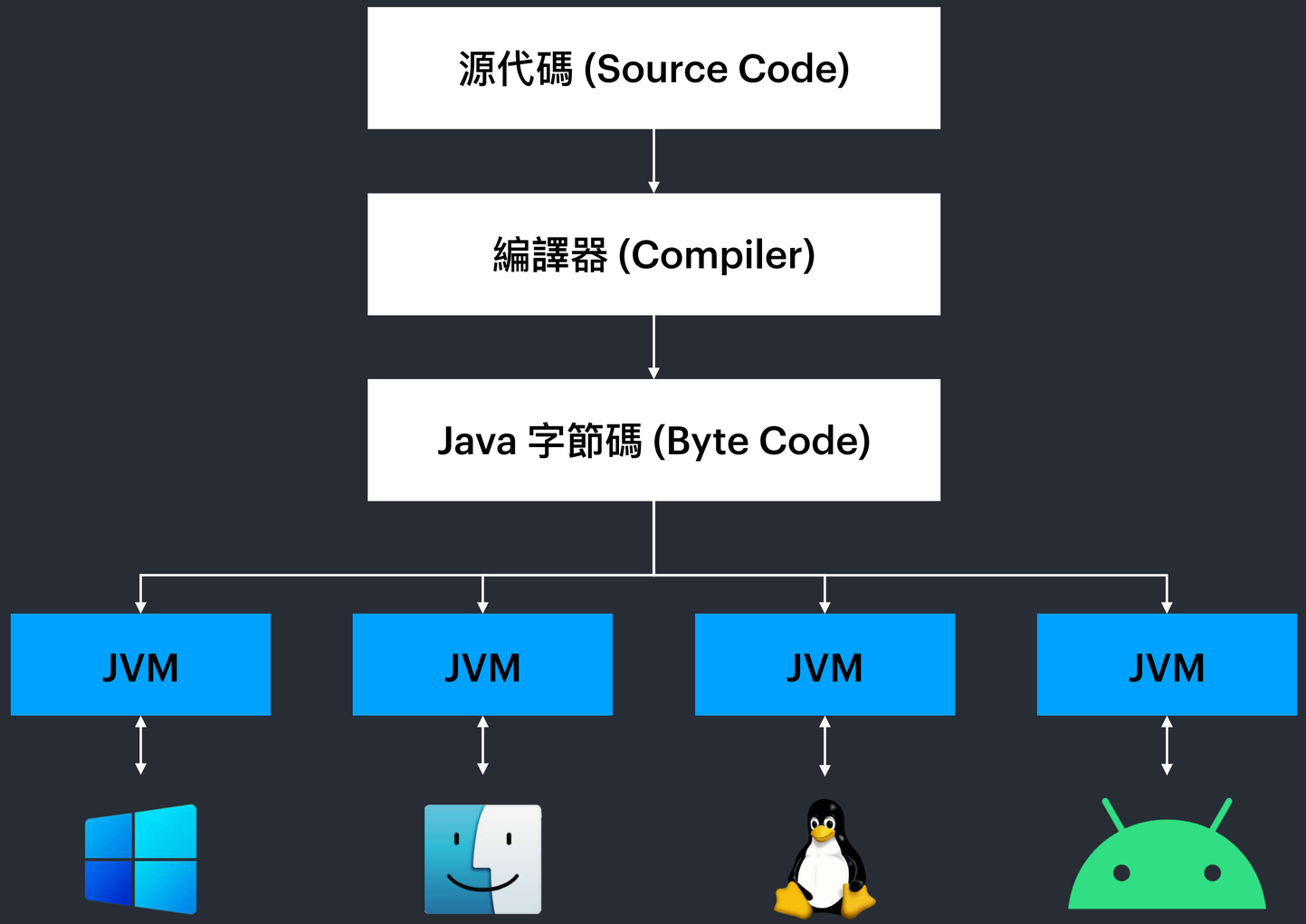
JVM

由開發到執行的過程



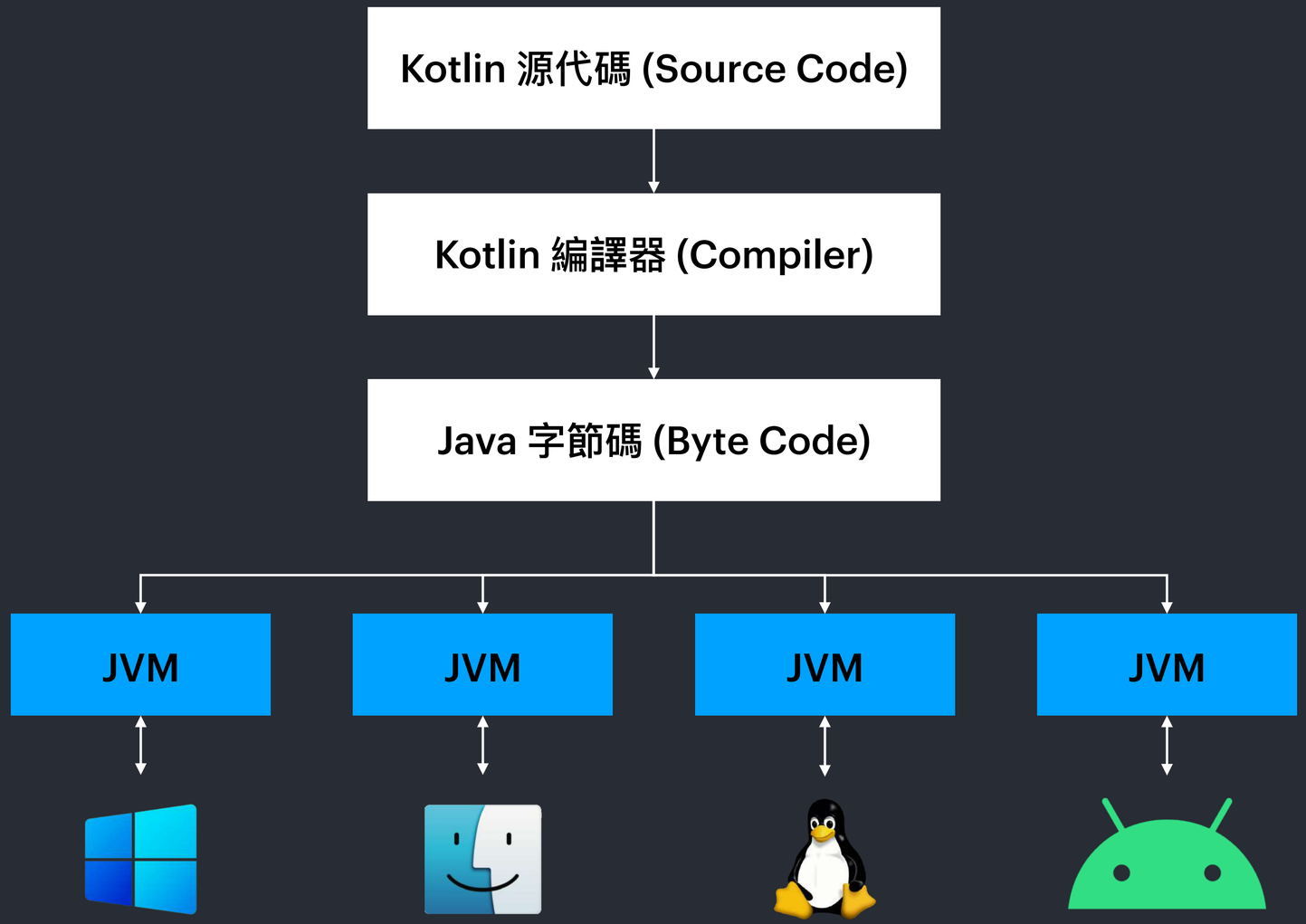
Kotlin Multiple Platform

由開發到執行的過程



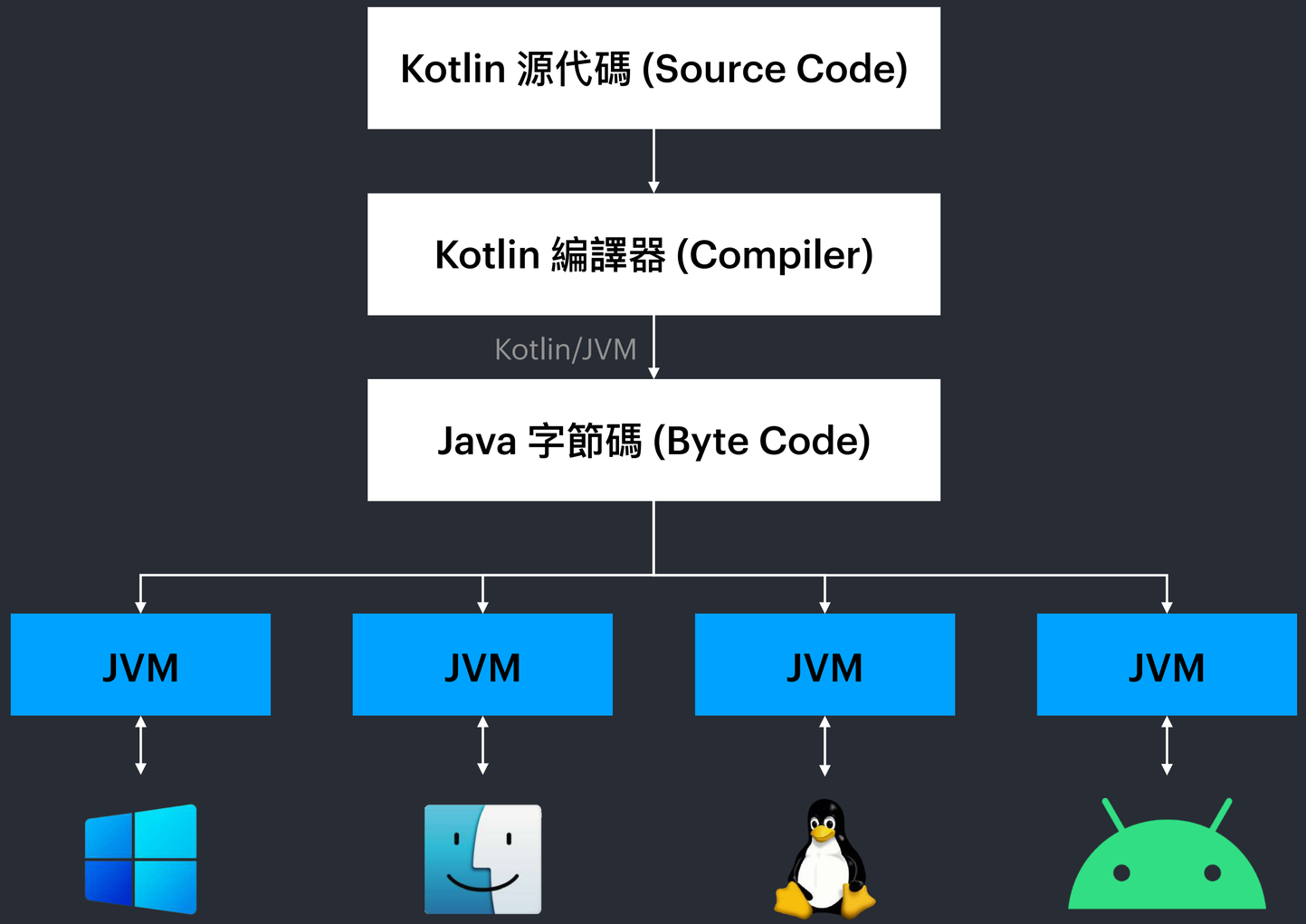
Kotlin Multiplatform

由開發到執行的過程



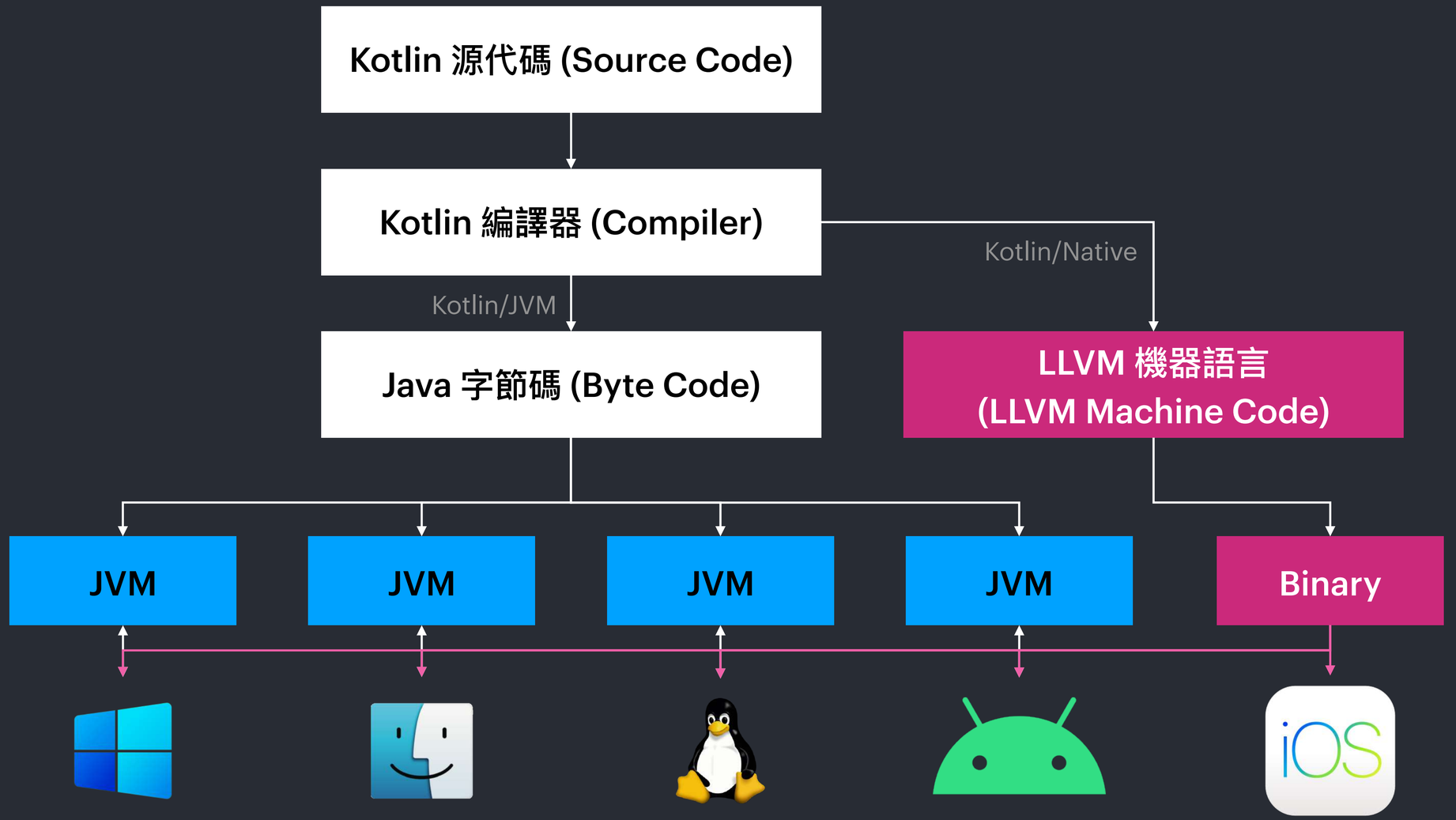
Kotlin Multiplatform

由開發到執行的過程



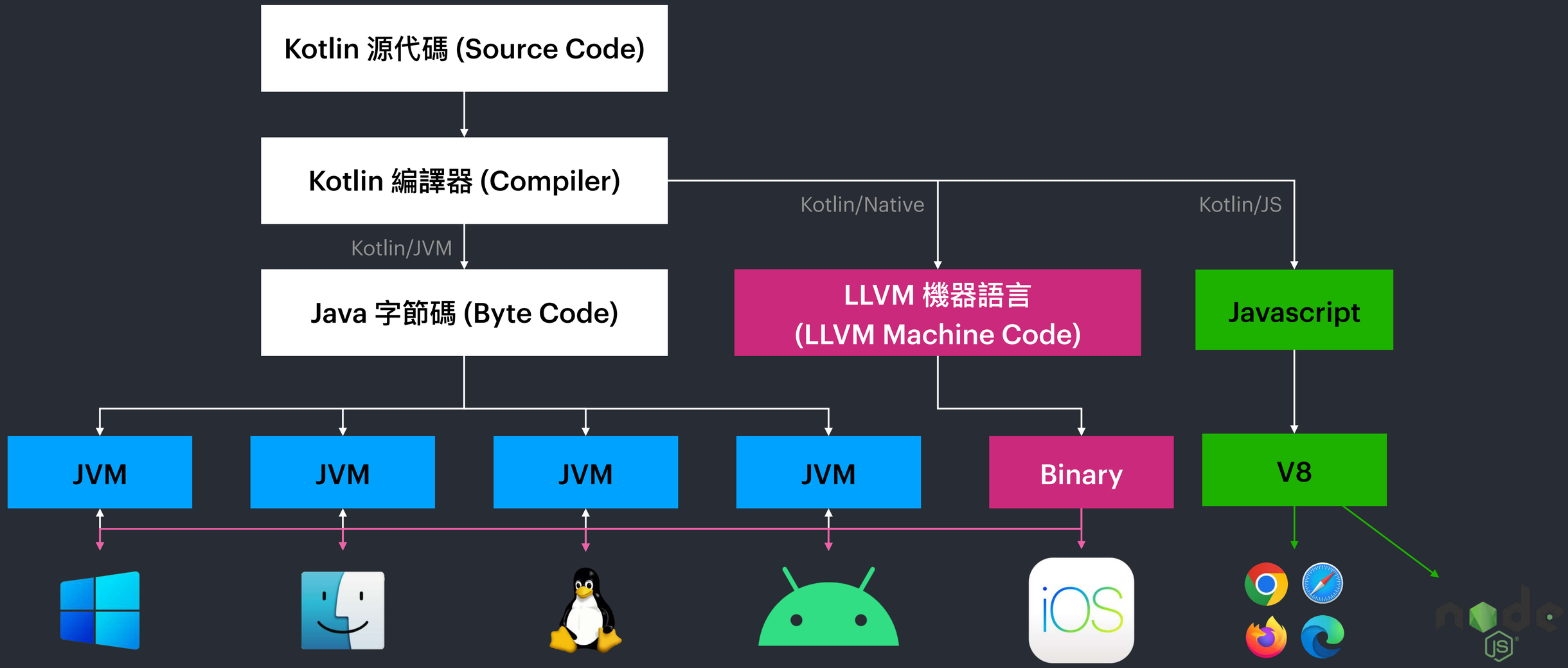
Kotlin Multiplatform

由開發到執行的過程



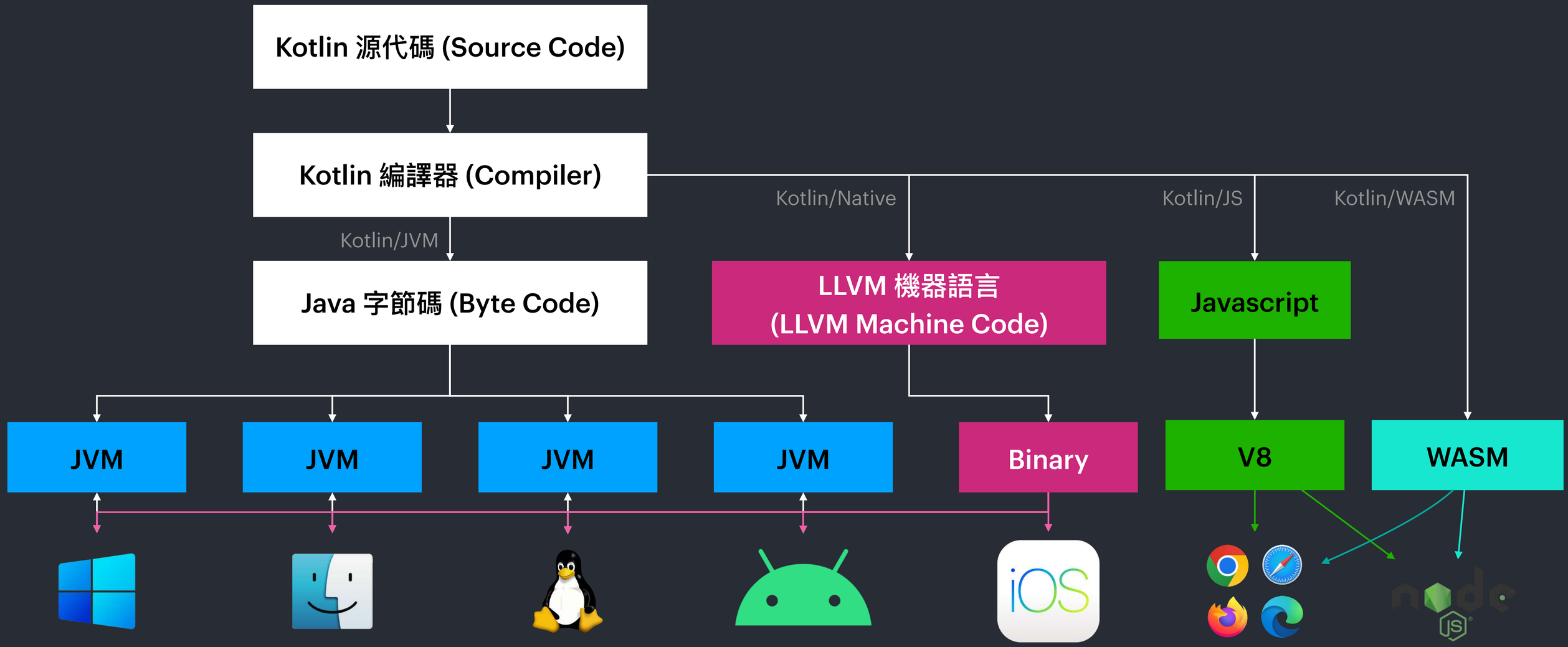
Kotlin Multiplatform

由開發到執行的過程



Kotlin Multiplatform

由開發到執行的過程



Kotlin Multiplatform

由開發到執行的過程

Kotlin 源代碼 (Source Code)

Kotlin 編譯器 (Compiler)

Kotlin/JVM

Java 字節碼 (Byte Code)

Kotlin/Native

LLVM 機器語言
(LLVM Machine Code)

Kotlin/JS

Javascript

Kotlin/WASM

JVM

JVM

JVM

JVM

Binary

V8

WASM

支援 Coroutine

總結

總結

- **同步 I/O 不利開發高並發後台服務**
- **異步 I/O 在 Java 的處理方案 (Completable Future、Reactor、RxJava) 學習曲線很高**
- **Coroutine 結構化並發 (Structured Concurrency) 有效提升代碼可讀性**
- **Coroutine 封裝靈活**
  - 簡單封裝現有的 Async IO Java 庫
  - 實踐 Actor Concurrency
  - 實踐 Concurrent Stream Processing 如
- **Coroutine 開在各大平台也有開發(Kotlin Multiplatform) 支援同樣概念及寫法，值得投資**

Thanks!
Have a nice Kotlin

Gap撈Tech

Blogs (Cantonese)
https://gaplo.tech

Blogs (English)
https://intl.gaplo.tech

Facebook Page
https://facebook.com/gaplotech