Data Race Detection In Go From Beginners Eye

Vaibhav Gupta

@97vaibhav

- Indian 🇮🇳
- Software engineer
- A Two-year-old Gopher
- Anime,Culture And Travel

Data Race Detection

Outline

- *Fundamentals*
- *Internals*
- *Best Practice to avoid race bugs*
- *Evaluation*

What is a Data Race ?

What is a Data Race ?

“When two or more goroutines access shared memory data concurrently and one goroutine is a write ”

"2つ以上のGo routineが同時に共有メモリ・データにアクセスし、1つのゴルーチンが書き込み"

What is a Data Race ?

“When two or more goroutines access shared memory data concurrently and one goroutine is a write ”

"2つ以上のGo routineが同時に共有メモリ・データにアクセスし、1つのゴルーチンが書き込み"

Let’s Look at an example
例を見てみましょう

```go
package main

import (
	"fmt"
)

var counter = 0

func IncrementCounter() {
	if counter == 0 {
		counter++
	}
}

func main() {

	go func() {
		IncrementCounter()
	}()

	go func() {
		IncrementCounter()
	}()

	fmt.Println("Final Counter Value:", counter)
}
```

```go
package main

import (
    "fmt"
)

var counter = 0

func IncrementCounter() {
    if counter == 0 {
        counter++
    }
}

func main() {

    go func() {
        IncrementCounter()
    }()

    go func() {
        IncrementCounter()
    }()

    fmt.Println("Final Counter Value:", counter)
}
```

G1

```go
package main

import (
    "fmt"
)

var counter = 0

func IncrementCounter() {
    if counter == 0 {
        counter++
    }
}

func main() {

    go func() {
        IncrementCounter()
    }()

    go func() {
        IncrementCounter()
    }()

    fmt.Println("Final Counter Value:", counter)
}
```

G1

G2

```go
package main

import (
    "fmt"
)

var counter = 0

func IncrementCounter() {
    if counter == 0 {
        counter++
    }
}

func main() {

    go func() {
        IncrementCounter()
    }()

    go func() {
        IncrementCounter()
    }()

    fmt.Println("Final Counter Value:", counter)
}
```

G1

G2

counter =1

?

```go
package main

import (
    "fmt"
)

var counter = 0

func IncrementCounter() {
    if counter == 0 {
        counter++
    }
}

func main() {

    go func() {
        IncrementCounter()
    }()

    go func() {
        IncrementCounter()
    }()

    fmt.Println("Final Counter Value:", counter)
}
```

G1

G2

counter =2

?

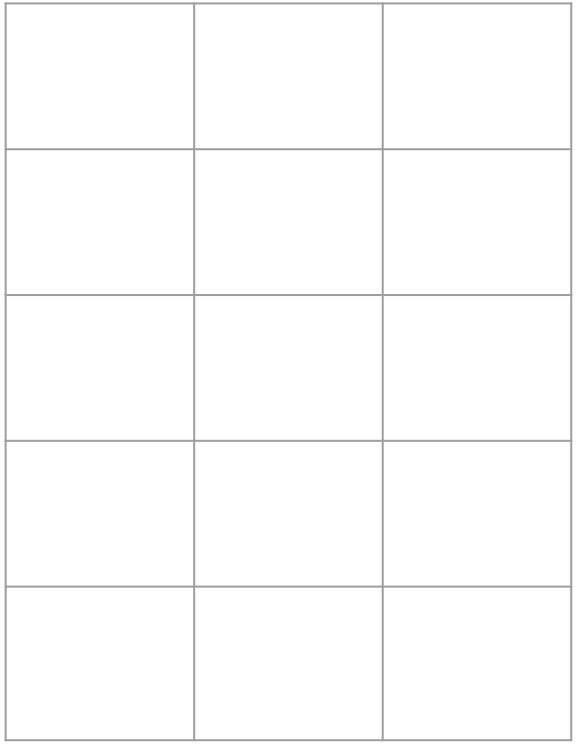
```go
package main

import (
    "fmt"
)

var counter = 0

func IncrementCounter() {
    if counter == 0 {
        counter++
    }
}

func main() {

    go func() {
        IncrementCounter()
    }()

    go func() {
        IncrementCounter()
    }()

    fmt.Println("Final Counter Value:", counter)
}
```

G1

G2

わからない🥲

```go
package main

import (
    "fmt"
)

var counter = 0

func IncrementCounter() {
    if counter == 0 {
        counter++
    }
}

func main() {

    go func() {
        IncrementCounter()
    }()

    go func() {
        IncrementCounter()
    }()

    fmt.Println("Final Counter Value:", counter)
}
```



```go
package main

import (
    "fmt"
)

var counter = 0

func IncrementCounter() {
    if counter == 0 {
        counter++
    }
}

func main() {

    go func() {
        IncrementCounter()
    }()

    go func() {
        IncrementCounter()
    }()

    fmt.Println("Final Counter Value:", counter)
}
```

G1

G2

G1R		

```go
package main

import (
    "fmt"
)

var counter = 0

func IncrementCounter() {
    if counter == 0 {
        counter++
    }
}

func main() {

    go func() {
        IncrementCounter()
    }()

    go func() {
        IncrementCounter()
    }()

    fmt.Println("Final Counter Value:", counter)
}
```

G1

G2

G1R		
G1W		

```go
package main

import (
    "fmt"
)

var counter = 0

func IncrementCounter() {
    if counter == 0 {
        counter++
    }
}

func main() {

    go func() {
        IncrementCounter()
    }()

    go func() {
        IncrementCounter()
    }()

    fmt.Println("Final Counter Value:", counter)
}
```

G1

G2

G1R		
G1W		
G2R		

```go
package main

import (
    "fmt"
)

var counter = 0

func IncrementCounter() {
    if counter == 0 {
        counter++
    }
}

func main() {

    go func() {
        IncrementCounter()
    }()

    go func() {
        IncrementCounter()
    }()

    fmt.Println("Final Counter Value:", counter)
}
```

G1

G2

G1R		
G1W		
G2R		
G2W		

```go
package main

import (
    "fmt"
)

var counter = 0

func IncrementCounter() {
    if counter == 0 {
        counter++
    }
}

func main() {

    go func() {
        IncrementCounter()
    }()

    go func() {
        IncrementCounter()
    }()

    fmt.Println("Final Counter Value:", counter)
}
```

counter =1

G1

G2

G1R		
G1W		
G2R		
G2W		
		

```go
package main

import (
    "fmt"
)

var counter = 0

func IncrementCounter() {
    if counter == 0 {
        counter++
    }
}

func main() {

    go func() {
        IncrementCounter()
    }()

    go func() {
        IncrementCounter()
    }()

    fmt.Println("Final Counter Value:", counter)
}
```

counter =2 🤔

G1

G2

G1R	G1R	G1R
G1W	G2R	G2R
G2R	G1W	G2W
G2W	G2W	G1W
✅	❌	❌

If we want to find Data Race in our CodeBase how can we do that in a manner which is reliable ??

CodeBaseのデータ・レース状態を追跡したい場合、どのようにすれば信頼できるのでしょうか？？

Go Race Detector

Go Race Detector

Go Race Detector

- Go v1.1

Go Race Detector

- Go v1.1
- Integrated with the Go tool chain
  go run -race main.go

Go Race Detector

- Go v1.1
- Integrated with the Go tool chain
  go run -race main.go
- It is Based on C/ C++ ThreadSanitizer

https://go.dev/blog/race-detector

Detect Data Race in Go and report it

```go
package main

import (
	"fmt"
)

var counter = 0

func IncrementCounter() {
	if counter == 0 {
		counter++
	}
}

func main() {

	go func() {
		IncrementCounter()
	}()

	go func() {
		IncrementCounter()
	}()

	fmt.Println("Final Counter Value:", counter)
}
```

```go
package main

import (
	"fmt"
)

var counter = 0

func IncrementCounter() {
	if counter == 0 {
		counter++
	}
}

func main() {

	go func() {
		IncrementCounter()
	}()

	go func() {
		IncrementCounter()
	}()

	fmt.Println("Final Counter Value:", counter)
}
```

go run -race main.go

```go
package main

import (
    "fmt"
)

var counter = 0

func IncrementCounter() {
    if counter == 0 {
        counter++
    }
}

func main() {

    go func() {
        IncrementCounter()
    }()

    go func() {
        IncrementCounter()
    }()

    fmt.Println("Final Counter Value:", counter)
}
```

go run -race main.go

```
WARNING: DATA RACE
Write at 0x000003205d90 by goroutine 6:
  main.IncrementCounter()
      /Users/vaibhavgupta/go/src/github.com/97vaibhav/race_detection/main.go:11 +0x52
  main.main.func1()
      /Users/vaibhavgupta/go/src/github.com/97vaibhav/race_detection/main.go:18 +0x18

Previous read at 0x000003205d90 by main goroutine:
  main.main()
      /Users/vaibhavgupta/go/src/github.com/97vaibhav/race_detection/main.go:25 +0x6d

Goroutine 6 (running) created at:
  main.main()
      /Users/vaibhavgupta/go/src/github.com/97vaibhav/race_detection/main.go:17 +0x27
==================
Found 1 data race(s)
exit status 66
```

Internals

Go Memory Model

Go memory model states that

- in a goroutine reads and writes are ordered
- when multiple goroutines are present shared data must be synchronized

Goのメモリ・モデルでは、次のようになる。

- ゴルーチンでは、読み書きの順序が決まっている。
- 複数のゴルーチンが存在する場合、共有データは同期されなければならない

Go Memory Model

Go memory model states that

- in a goroutine reads and writes are ordered
- when multiple goroutines are present shared data must be synchronized

Goのメモリ・モデルでは、次のようになる。

- ゴルーチンでは、読み書きの順序が決まっている。
- 複数のゴルーチンが存在する場合、共有データは同期されなければならない

Ref. https://go.dev/ref/mem#introduction

Go Memory Model

Go memory model states that

- in a goroutine reads and writes are ordered
- when multiple goroutines are present shared data must be synchronized

Goのメモリ・モデルでは、次のようになる。

- ゴルーチンでは、読み書きの順序が決まっている。
- 複数のゴルーチンが存在する場合、共有データは同期されなければならない

Ref. https://go.dev/ref/mem#introduction

Data Race 😭

The synchronization Types

Channels	Mychannel <- element
Mutex	• Lock() • Unlock() https://go.dev/tour/concurrency/9
Atomics	https://pkg.go.dev/sync/atomic

Happens Before Approach Using Vector Clocks

Happens Before

When two events lets say a and b are present , only one of the following three scenarios is possible

- a happens before b
  a < b
- b happens before a
  b < a
- If Neither of the above is true
  - In this case, we say a and b are concurrent

```go
package main

import (
	"fmt"
	"sync"
)

var counter = 0
var mutex sync.Mutex

func IncrementCounter() {
	mutex.Lock()
	if counter == 0 {
		counter++
	}
	mutex.Unlock()
}

func main() {
	go func() {
		IncrementCounter()
	}()

	go func() {
		IncrementCounter()
	}()

	fmt.Println("Final Counter Value:", counter)
}
```

```go
package main

import (
	"fmt"
	"sync"
)

var counter = 0
var mutex sync.Mutex

func IncrementCounter() {
	mutex.Lock()
	if counter == 0 {
		counter++
	}
	mutex.Unlock()
}

func main() {
	go func() {
		IncrementCounter()
	}()

	go func() {
		IncrementCounter()
	}()

	fmt.Println("Final Counter Value:", counter)
}
```

mutex.Lock()

```go
package main

import (
	"fmt"
	"sync"
)

var counter = 0
var mutex sync.Mutex

func IncrementCounter() {
	mutex.Lock()
	if counter == 0 {
		counter++
	}
	mutex.Unlock()
}

func main() {
	go func() {
		IncrementCounter()
	}()

	go func() {
		IncrementCounter()
	}()

	fmt.Println("Final Counter Value:", counter)
}
```

mutex.Lock()

mutex.Unlock()

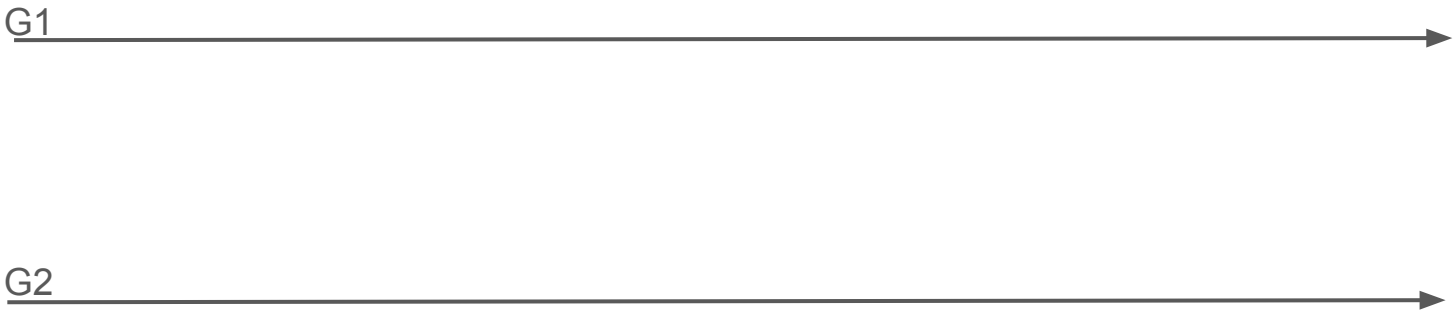
```go
package main

import (
	"fmt"
	"sync"
)

var counter = 0
var mutex sync.Mutex

func IncrementCounter() {
	mutex.Lock()
	if counter == 0 {
		counter++
	}
	mutex.Unlock()
}

func main() {
	go func() {
		IncrementCounter()
	}()

	go func() {
		IncrementCounter()
	}()

	fmt.Println("Final Counter Value:", counter)
}
```

mutex.Lock()

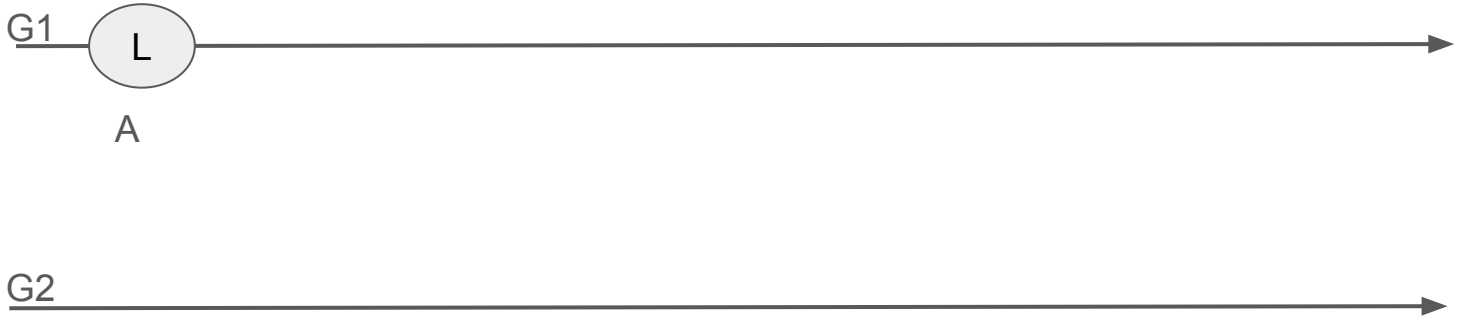
mutex.Unlock()

Synchronize Pairs

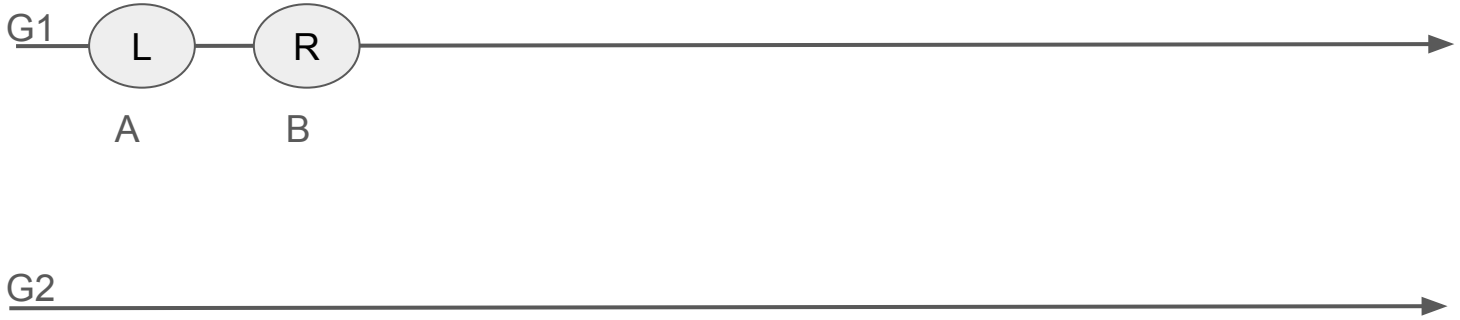
Example



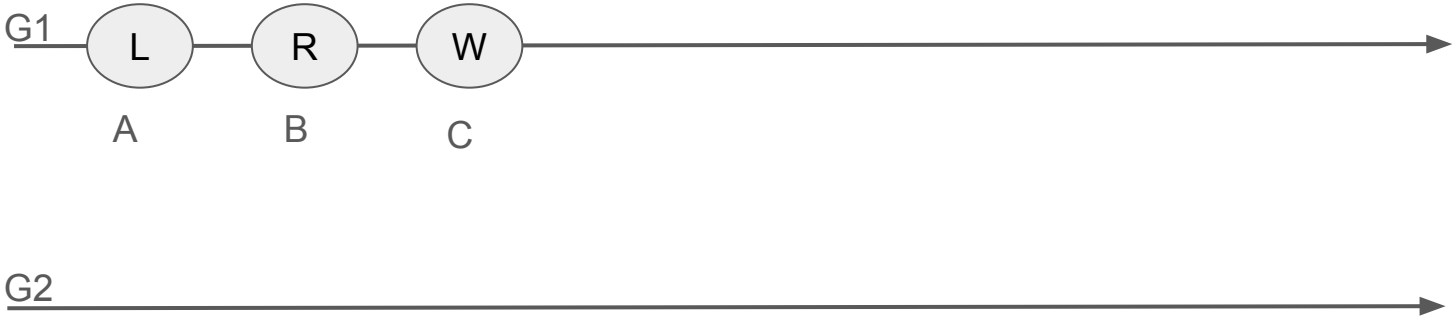
Example



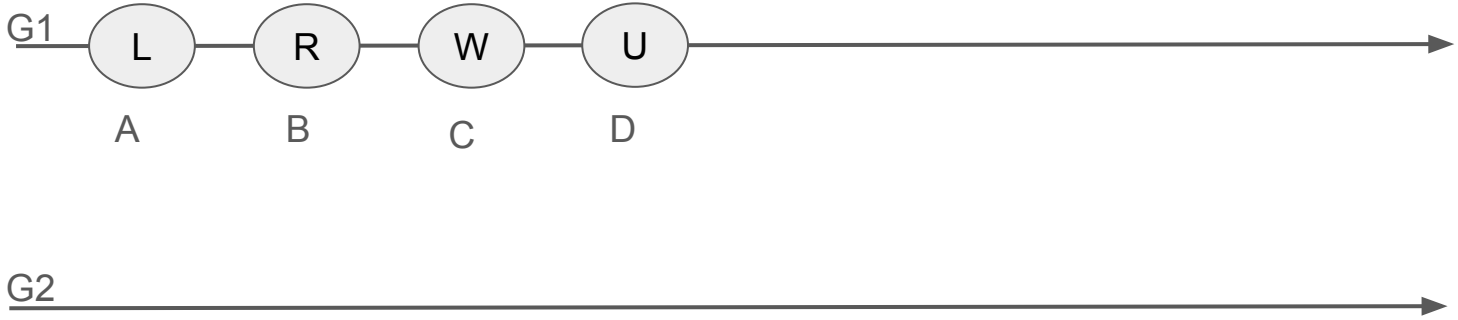
Example



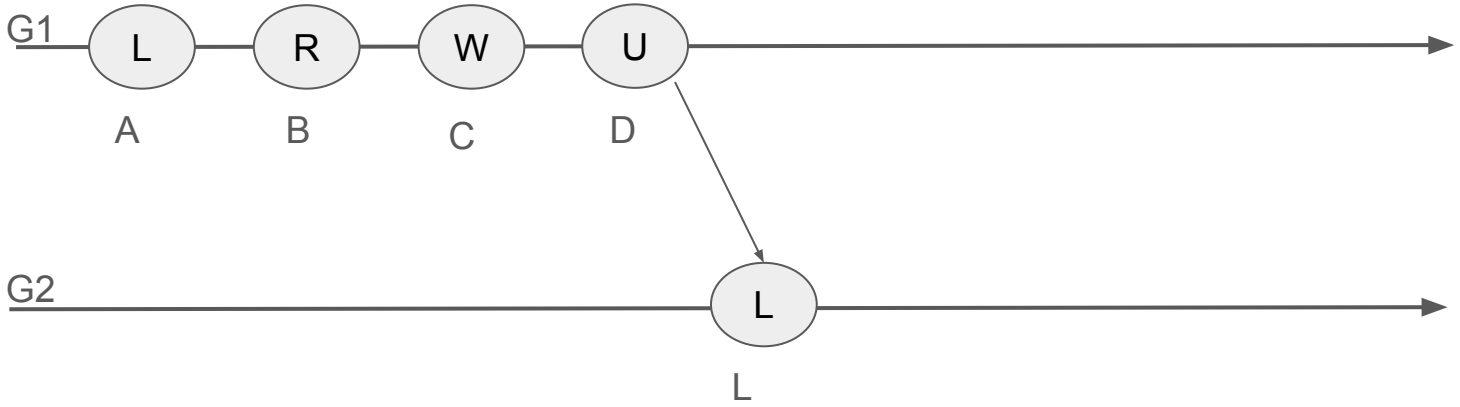
Example



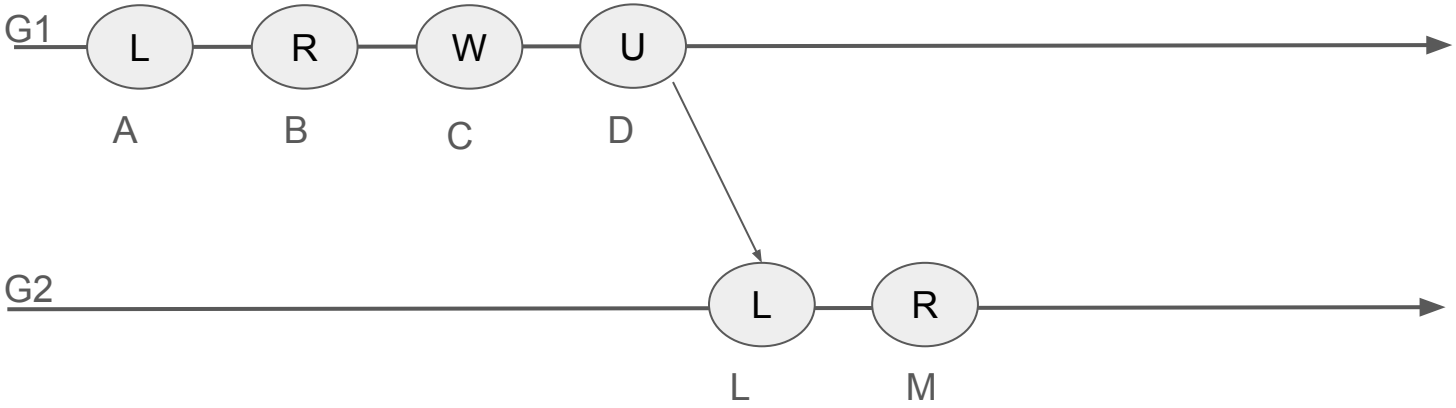
Example



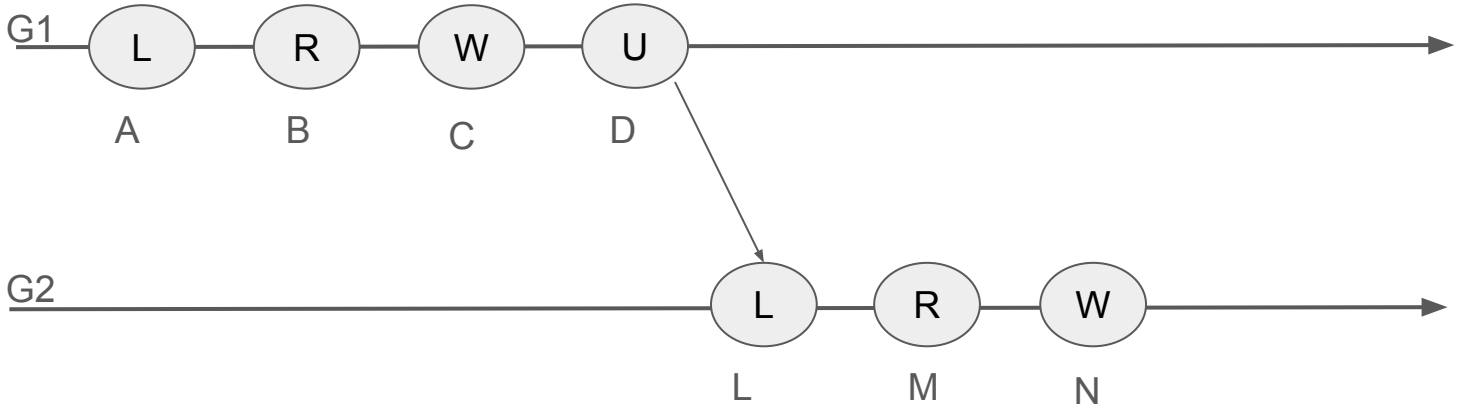
Example



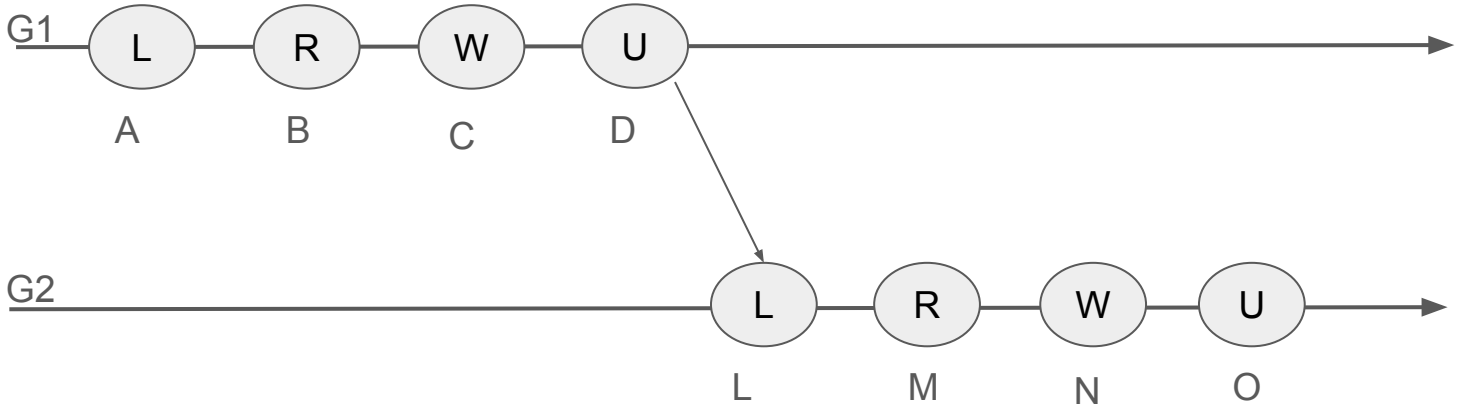
Example



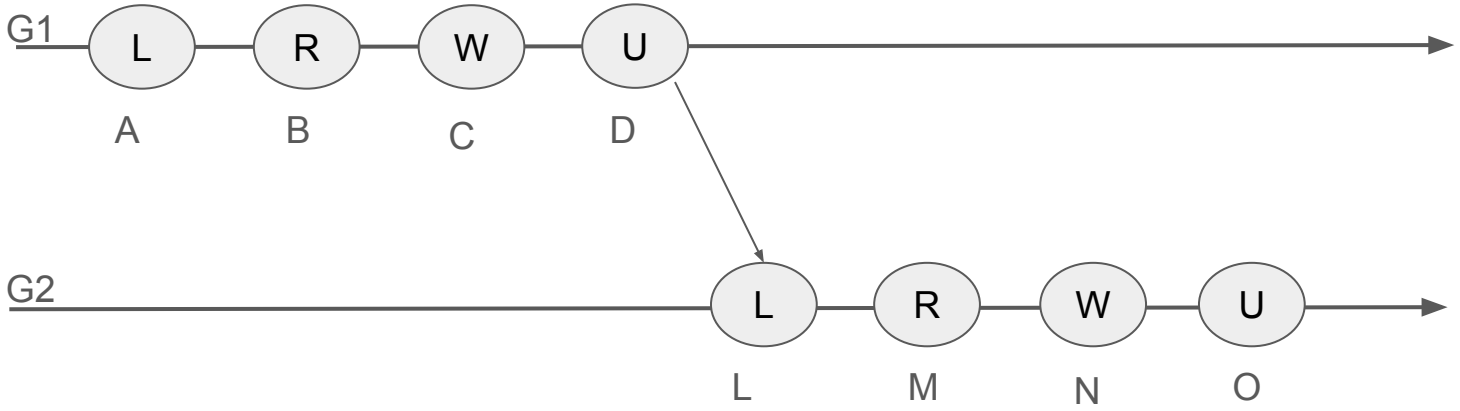
Example



Example

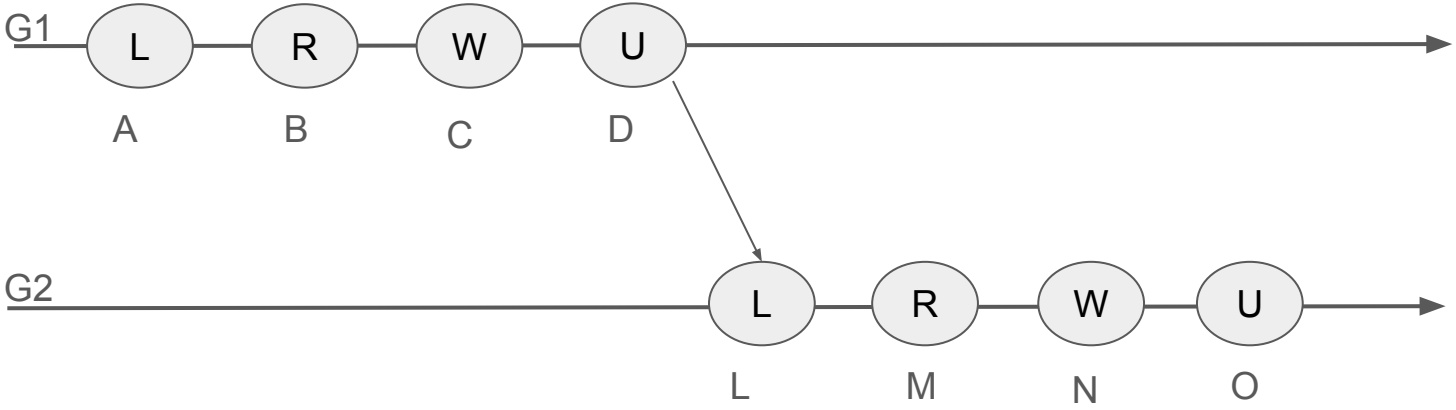


Example



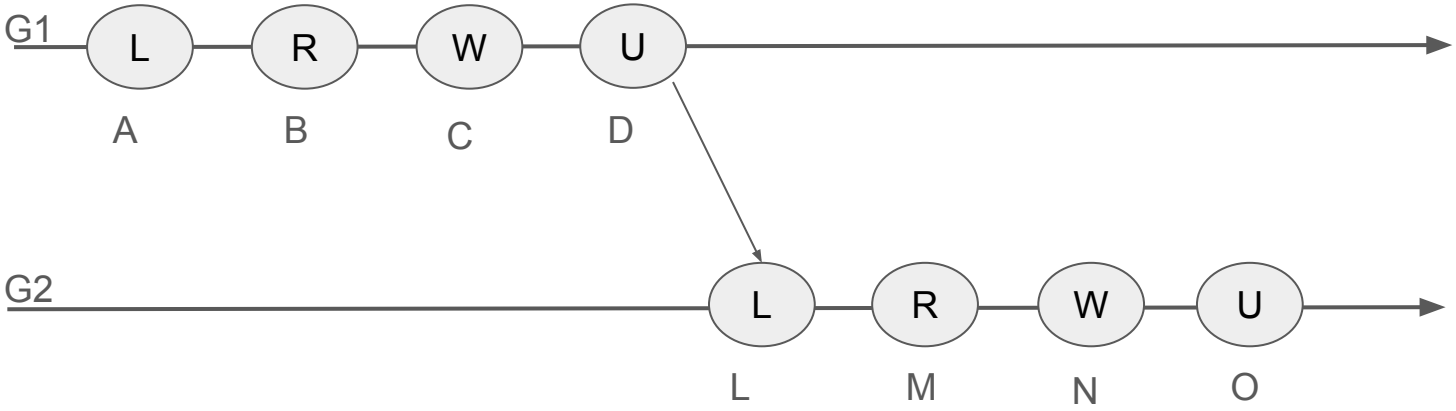
B<C

Example



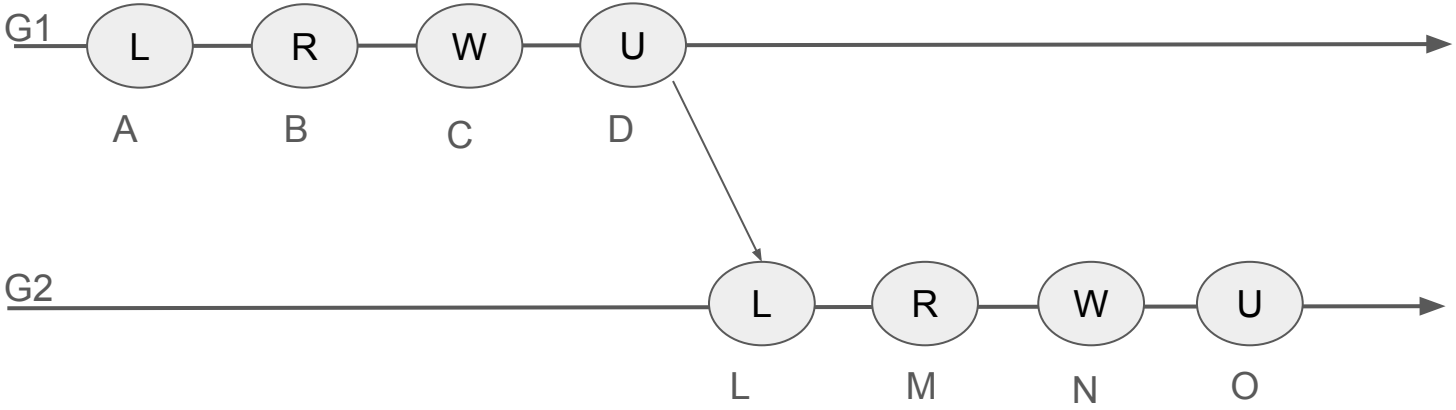
B<C

Example



B<C D<L

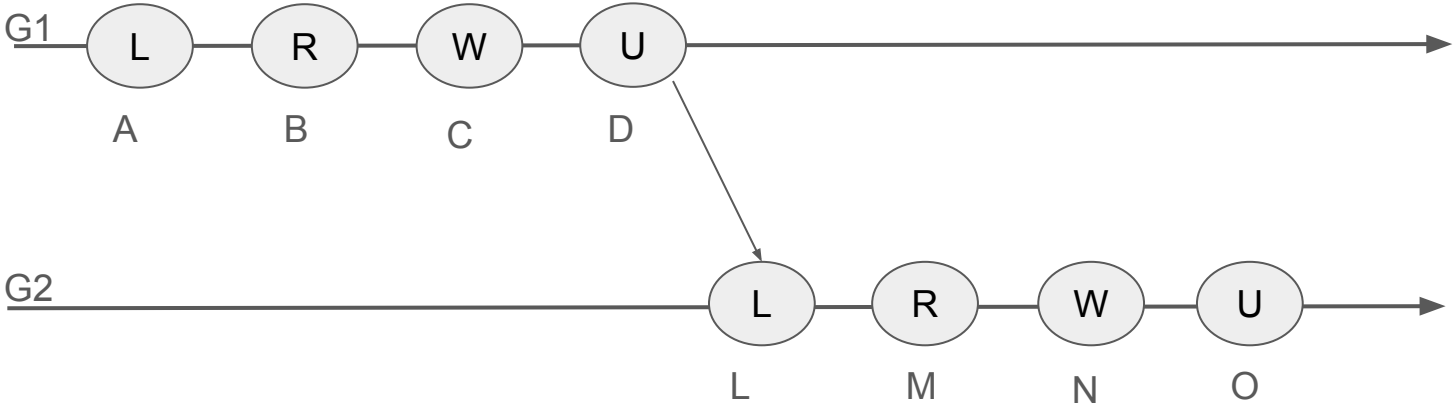
Example



B<C ✅

D<L ✅

Example

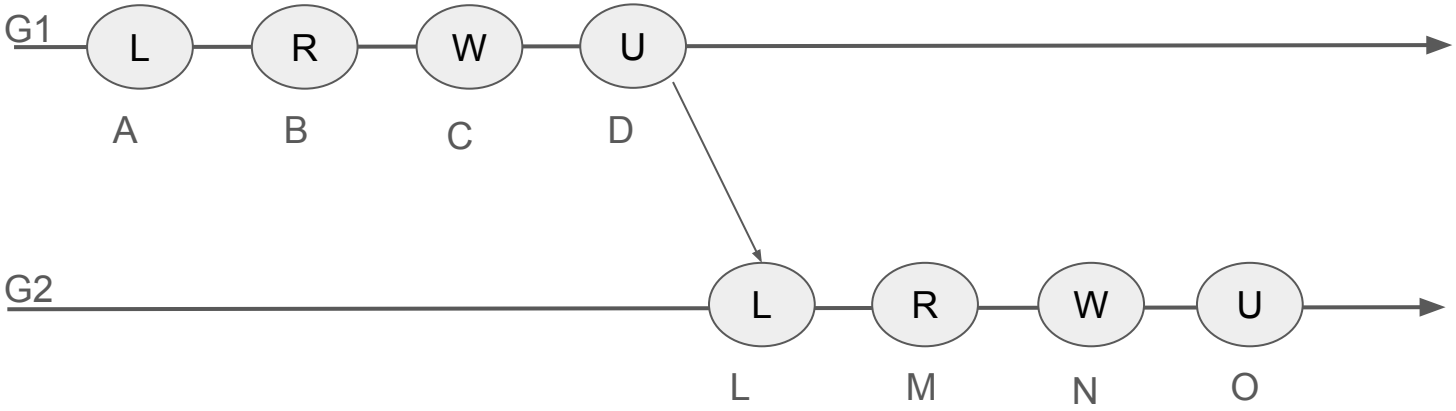


B<C ✅

D<L ✅

C<M ?

Example



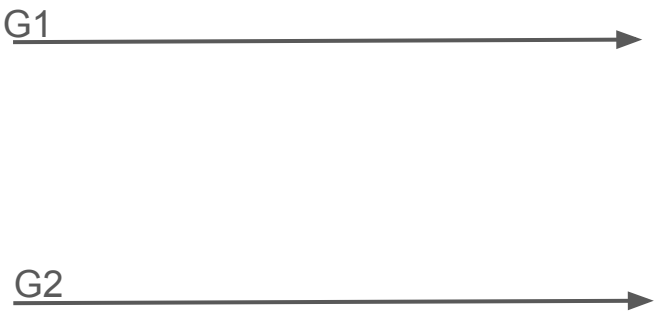
B<C ✅

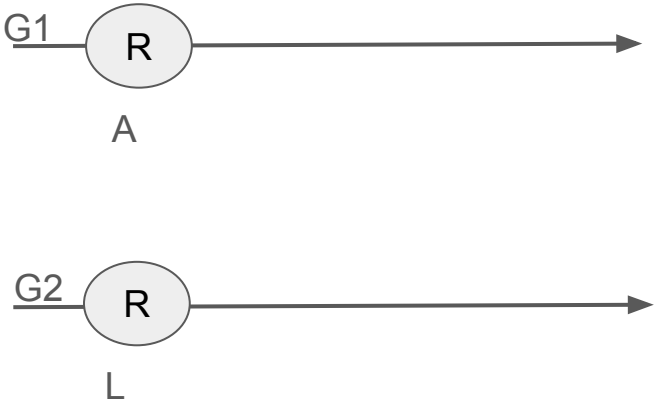
D<L ✅

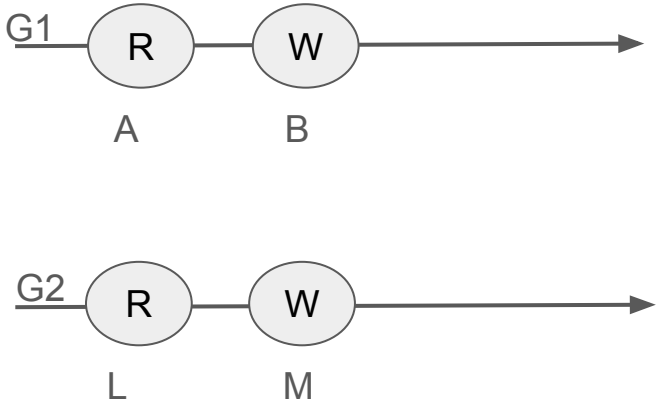
C<M ✅

```go
func IncrementCounter() {
    if counter == 0 {
        counter++
    }
}

func main() {
    go func() {
        IncrementCounter()
    }()

    go func() {
        IncrementCounter()
    }()

    fmt.Println("Final Counter Value:", counter)
}
```

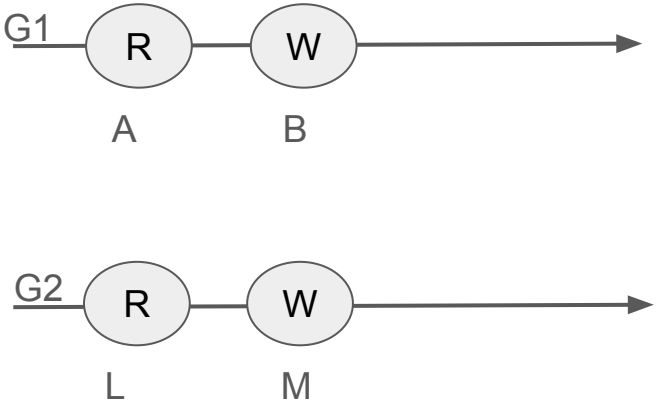
No Synchronize Pairs

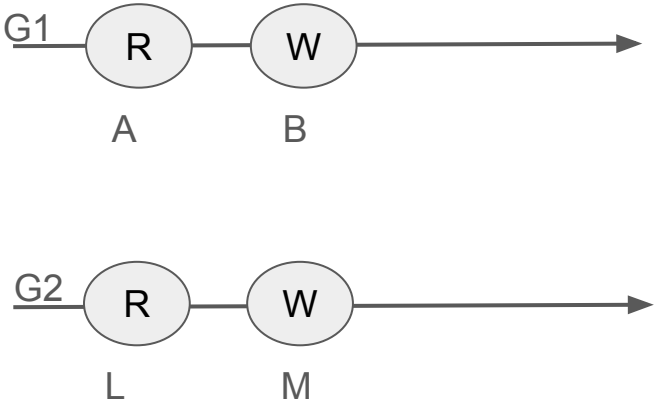

G1
G2


G1
R
A
G2
R
L


G1
R
W
A
B
G2
R
W
L
M

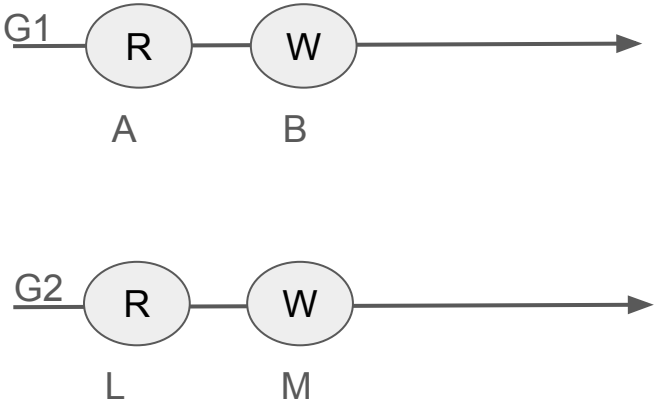
A<B





A<B

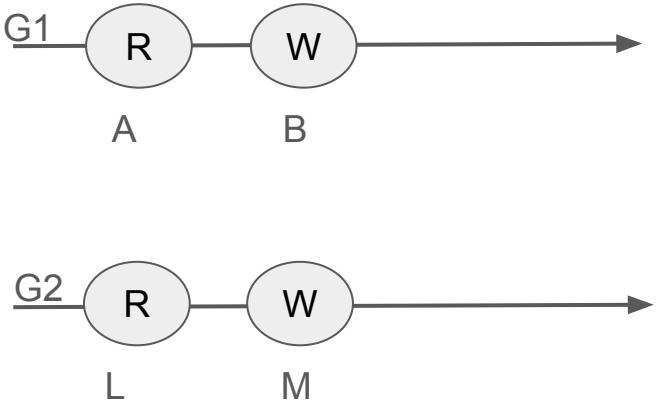
✅



A<B

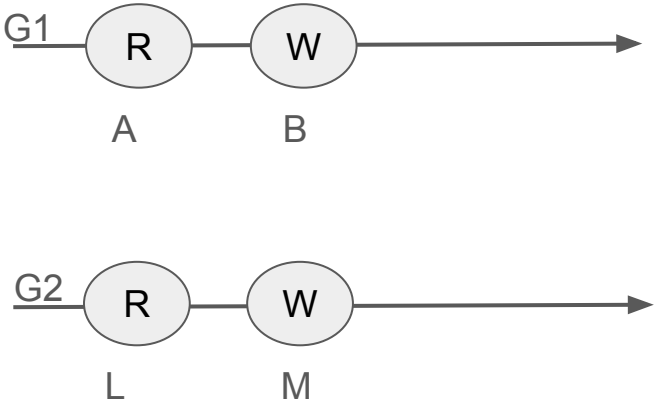
✅

L<M



A<B ✅

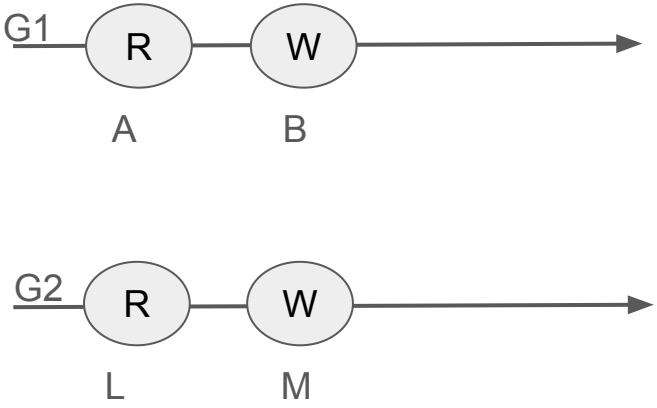
L<M ✅



A<B

L<M

A<L?

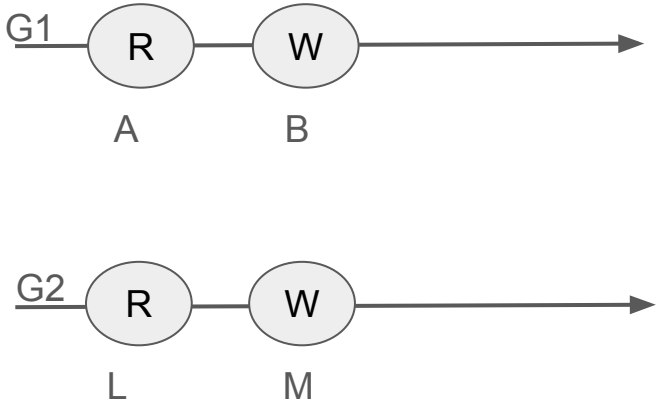


A<B ✅

L<M ✅

A<L?

B<M ?



A<B ✅

L<M ✅

A<L?

B<M ?

わからない🥲

How Happens-Before is Implemented ?

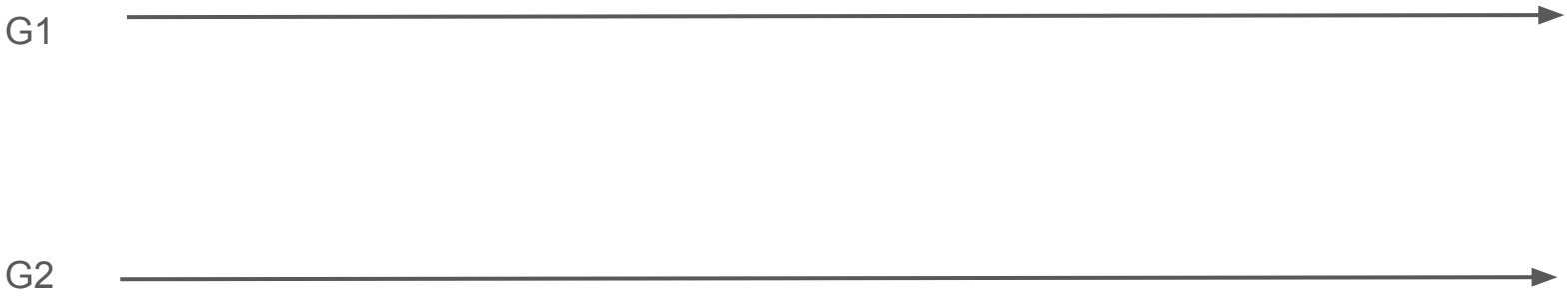
Vector Clocks

Vector Clocks

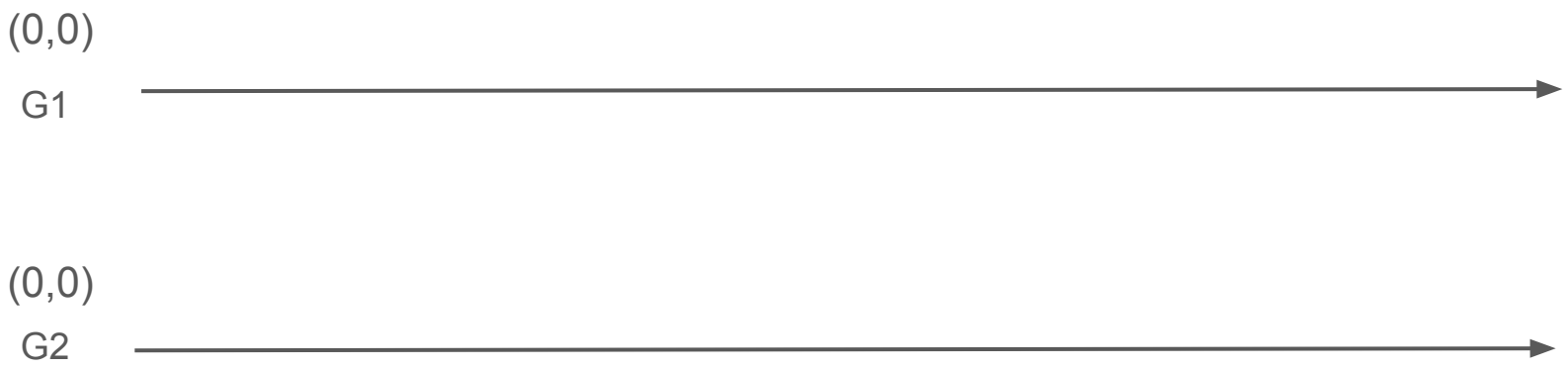
- Each process starts with a vector clock initialized to zero.
- Every time an event occurs at a process, it increments its own entry in the vector clock by 1.
- When a process sends a message, it includes its current vector clock in the message.
- Upon receiving a message, a process updates its own vector clock by taking the maximum of each entry in its vector clock and the corresponding entry in the received vector clock, then increments its own entry for its own process by 1.

- 各プロセスは、ゼロで初期化されたベクトルクロックを持って開始します。
- プロセスでイベントが発生するたびに、そのプロセスのベクトルクロックのエントリを1つ増やします。
- プロセスがメッセージを送信するときは、現在のベクトルクロックをメッセージに含めます。
- メッセージを受信すると、プロセスは、自分のベクトルクロックの各エントリと受信したベクトルクロックの対応するエントリの最大値を取り、それぞれのプロセスのエントリを1つ増やします。

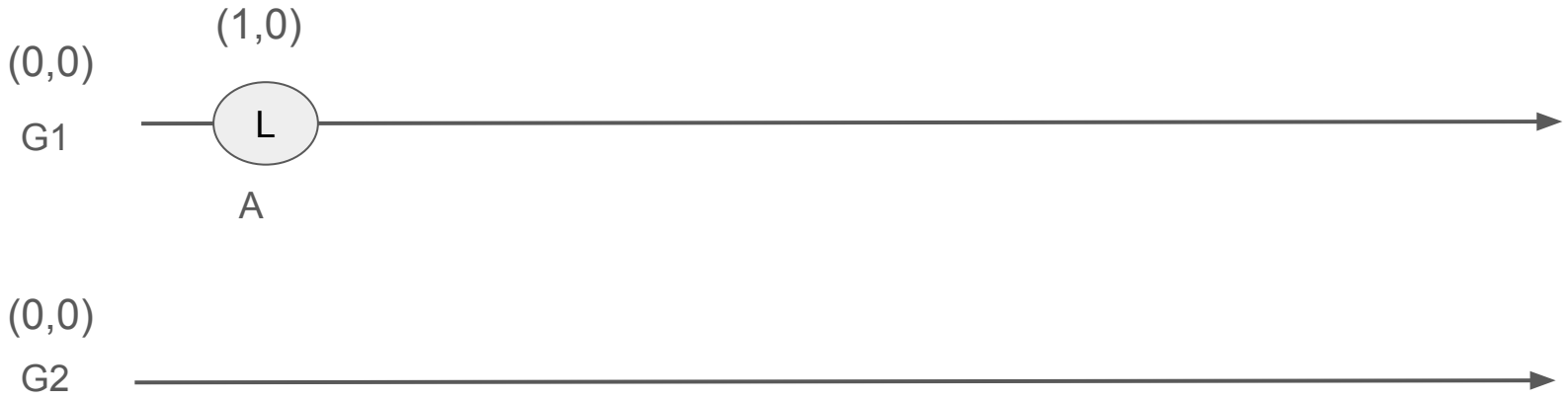
Vector Clocks



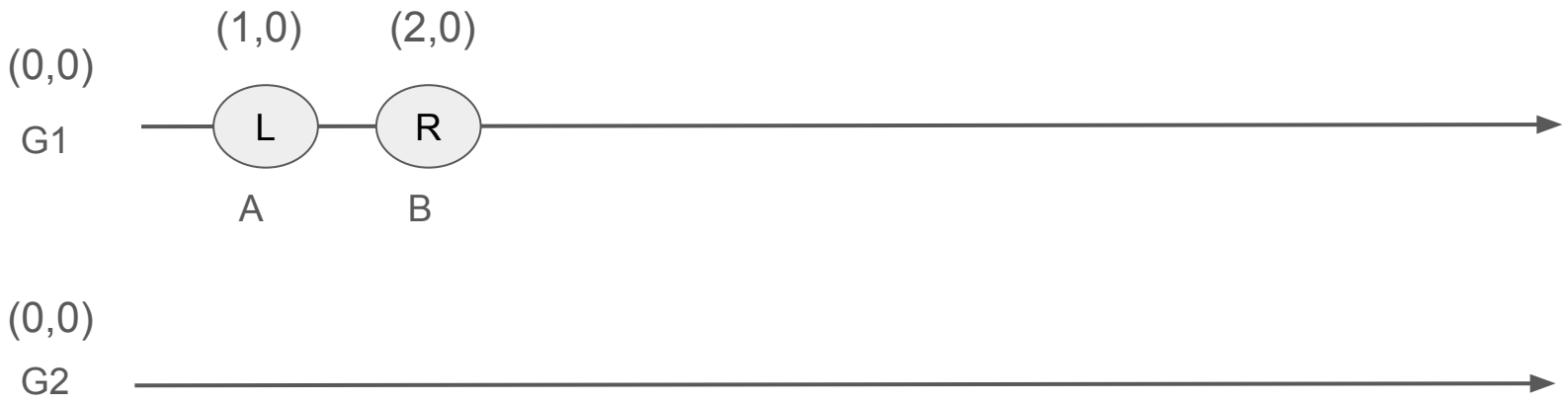
Vector Clocks



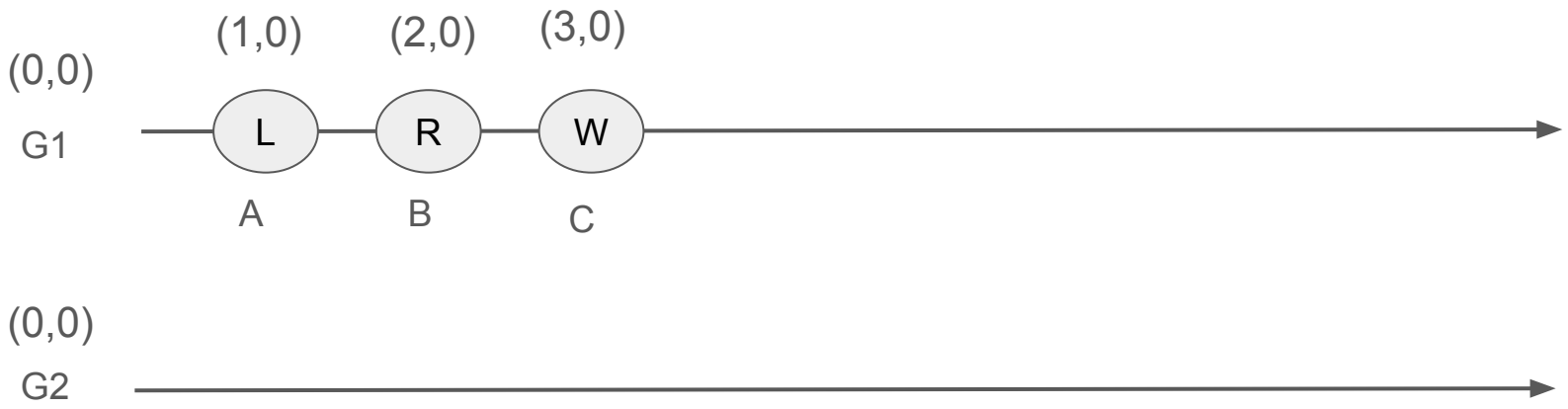
Vector Clocks



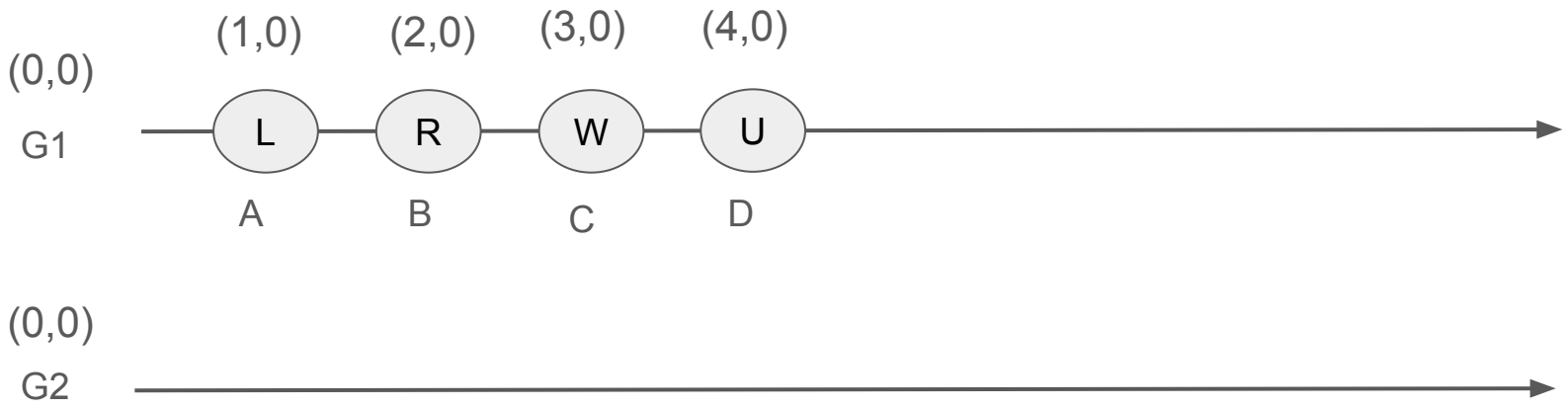
Vector Clocks



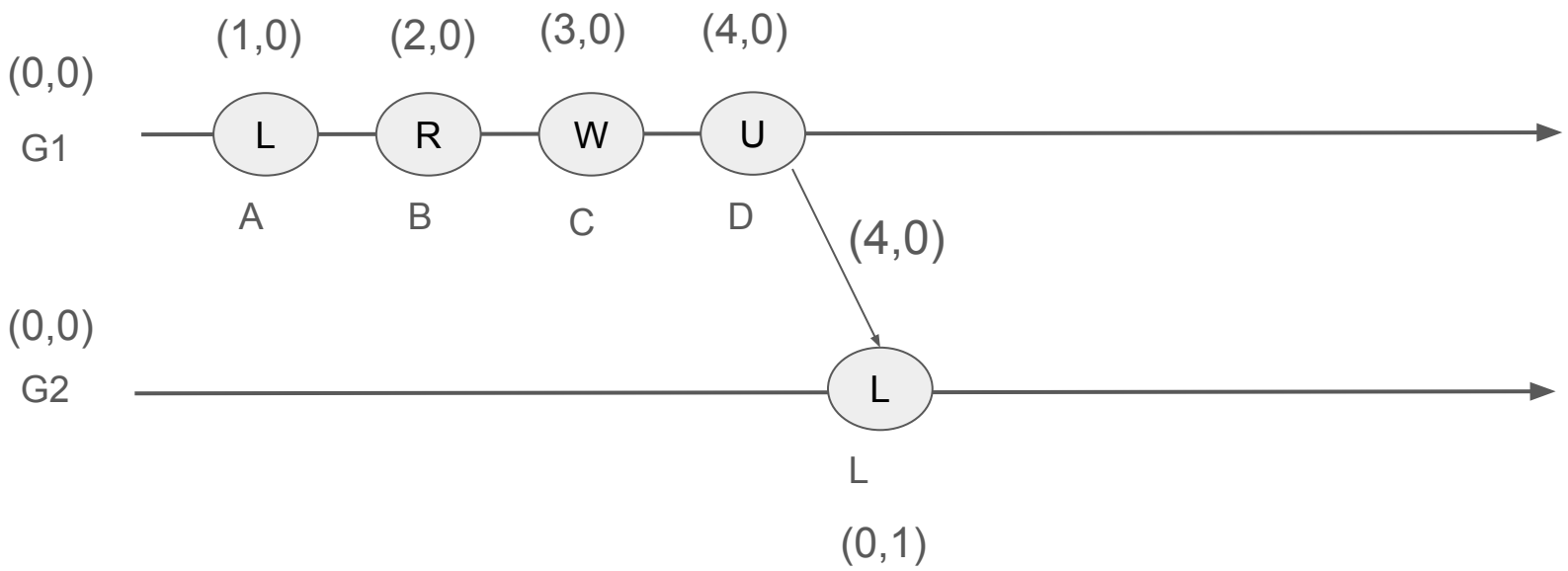
Vector Clocks



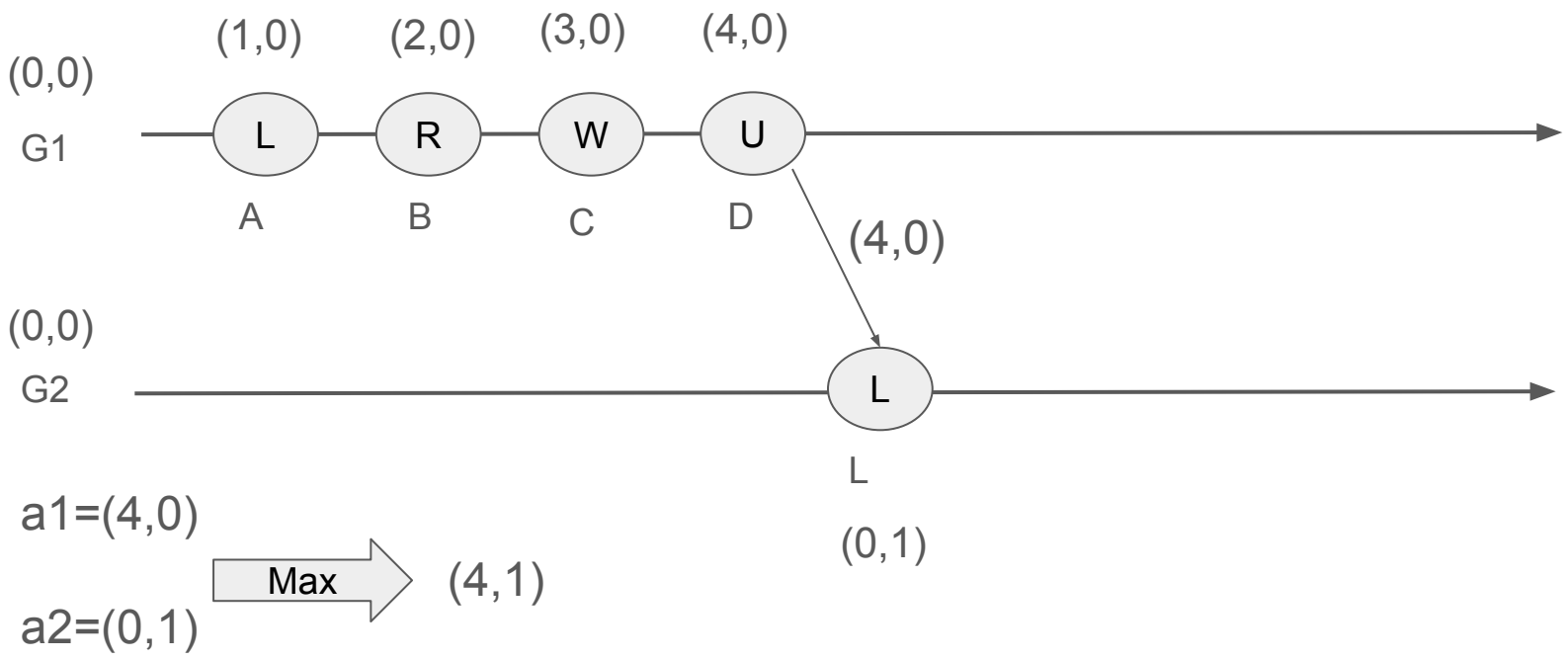
Vector Clocks



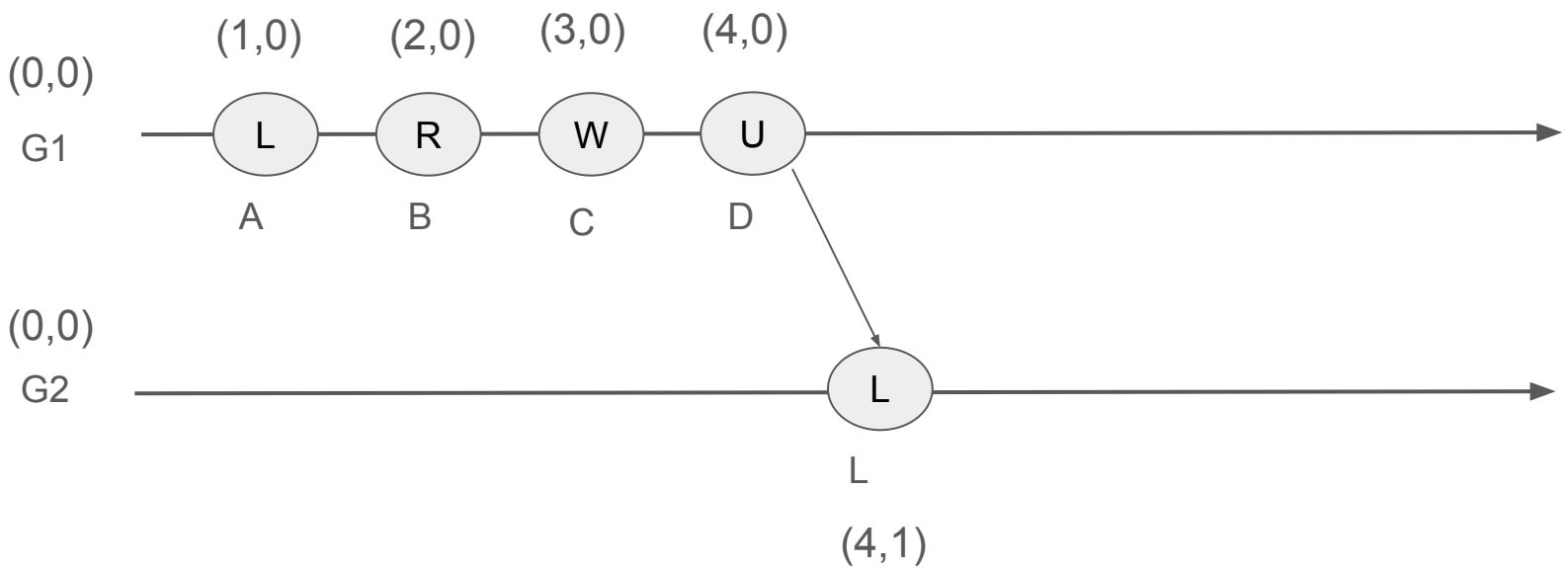
Vector Clocks



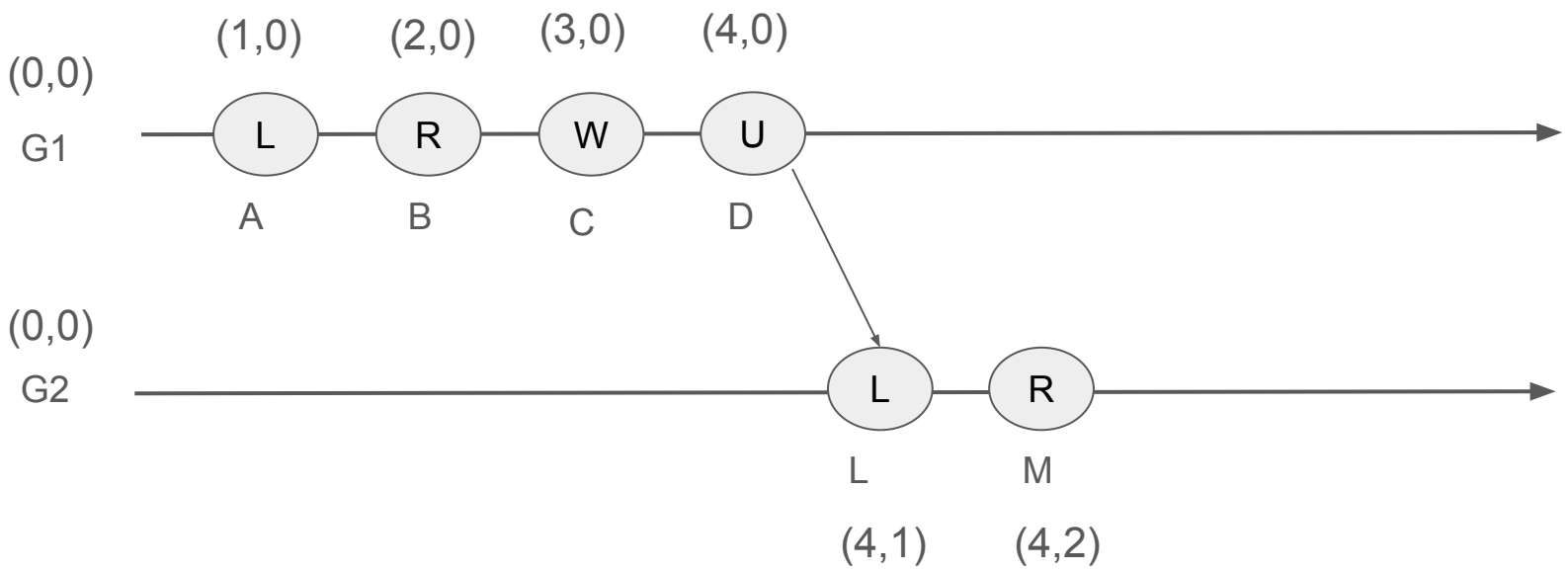
Vector Clocks



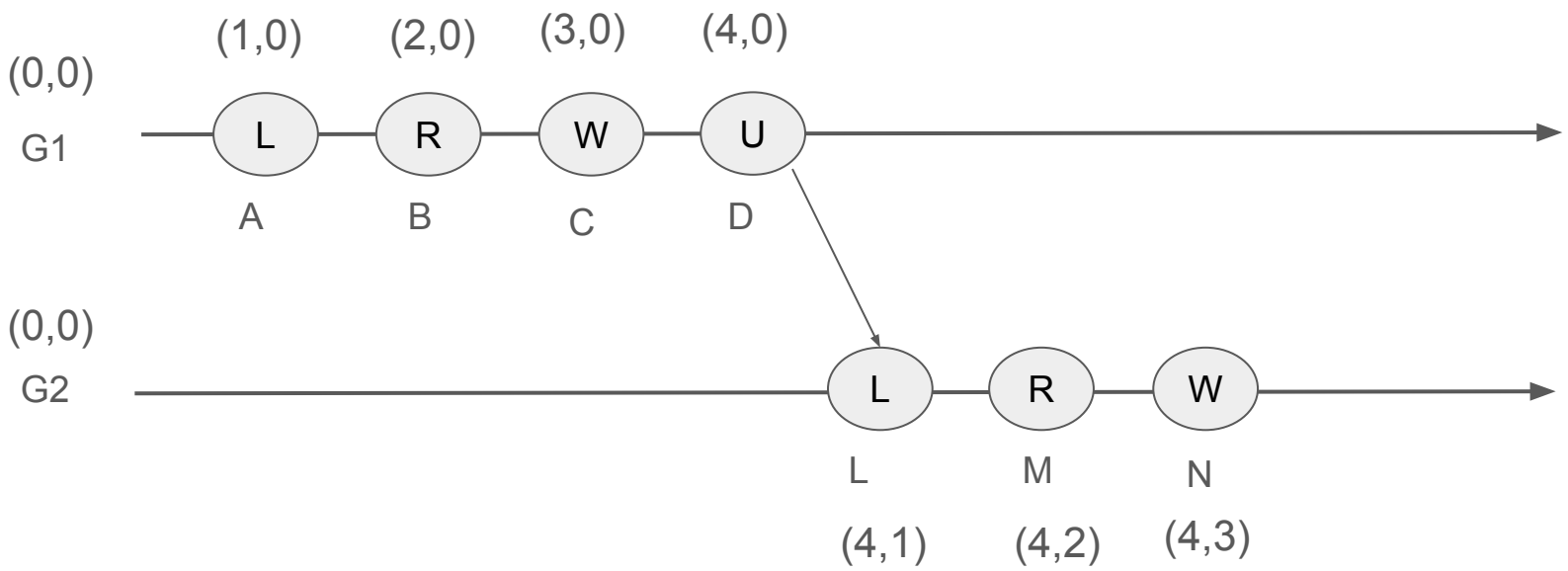
Vector Clocks



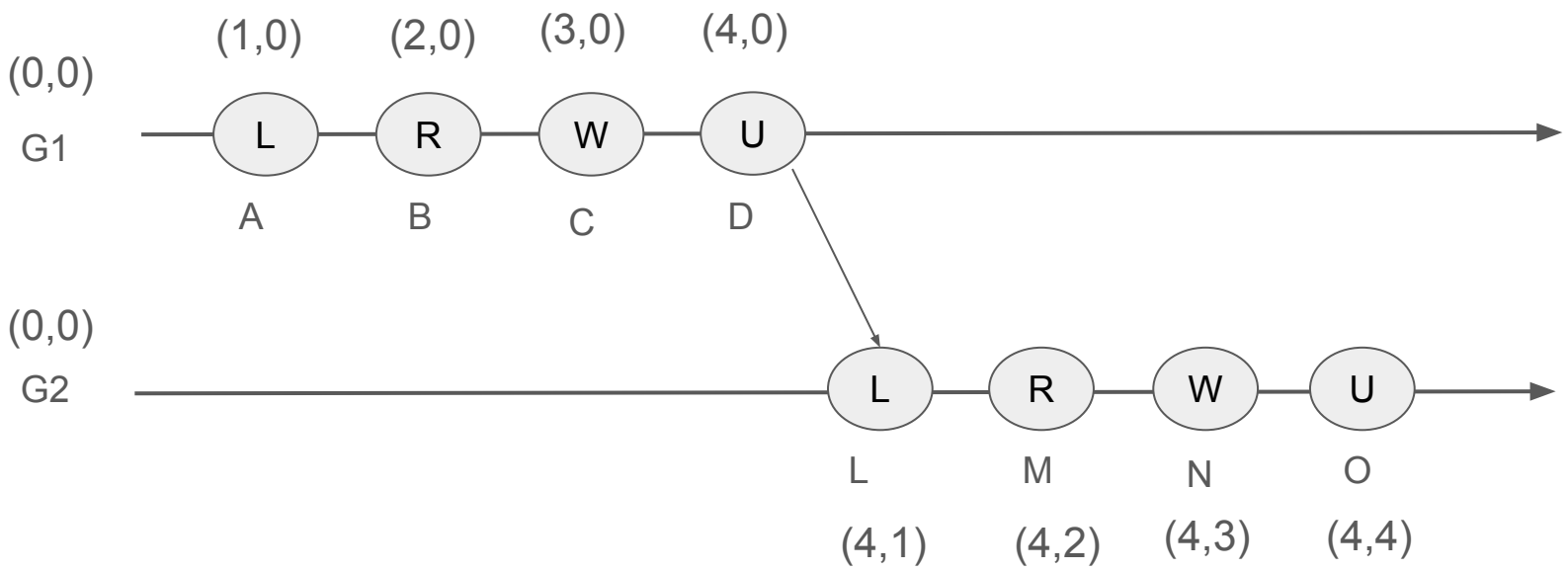
Vector Clocks



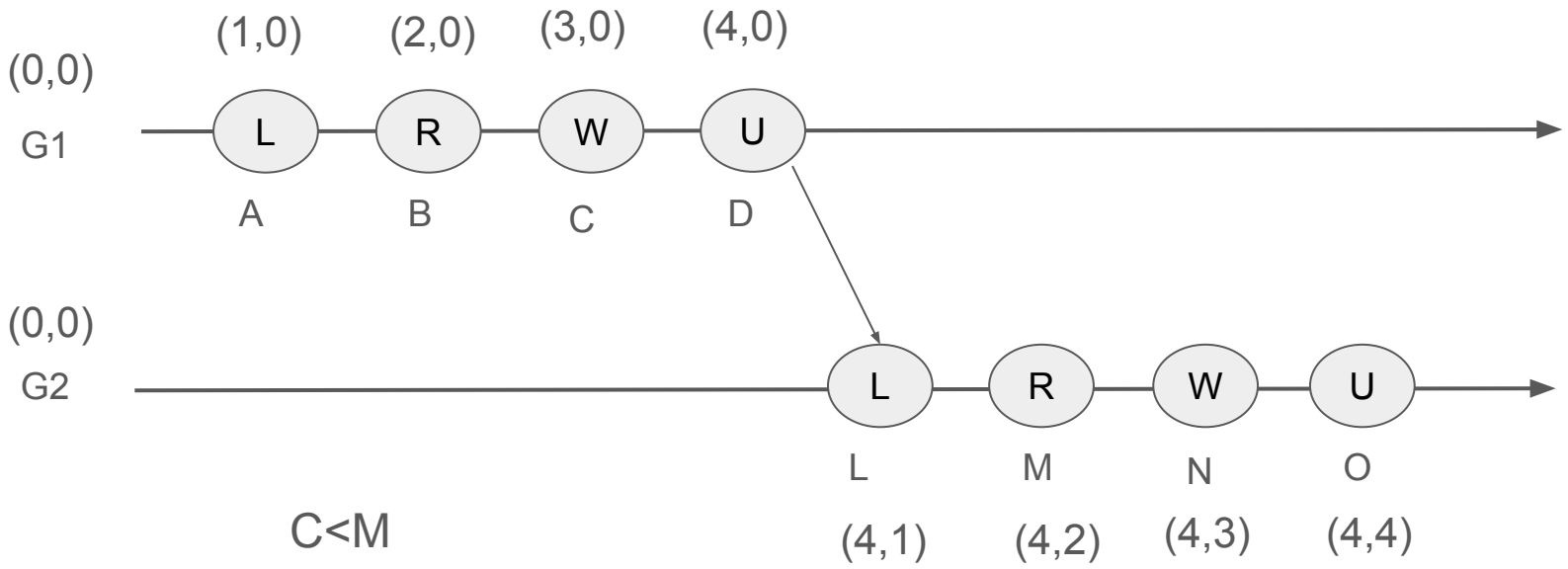
Vector Clocks



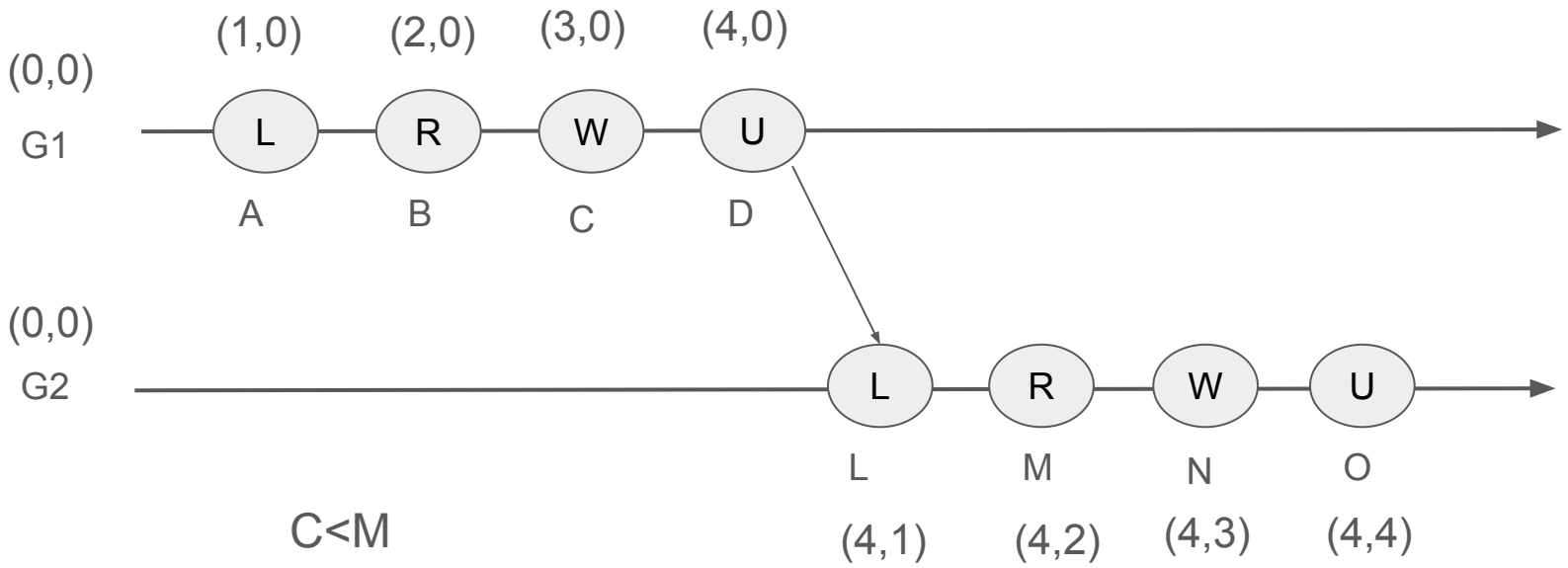
Vector Clocks



Vector Clocks



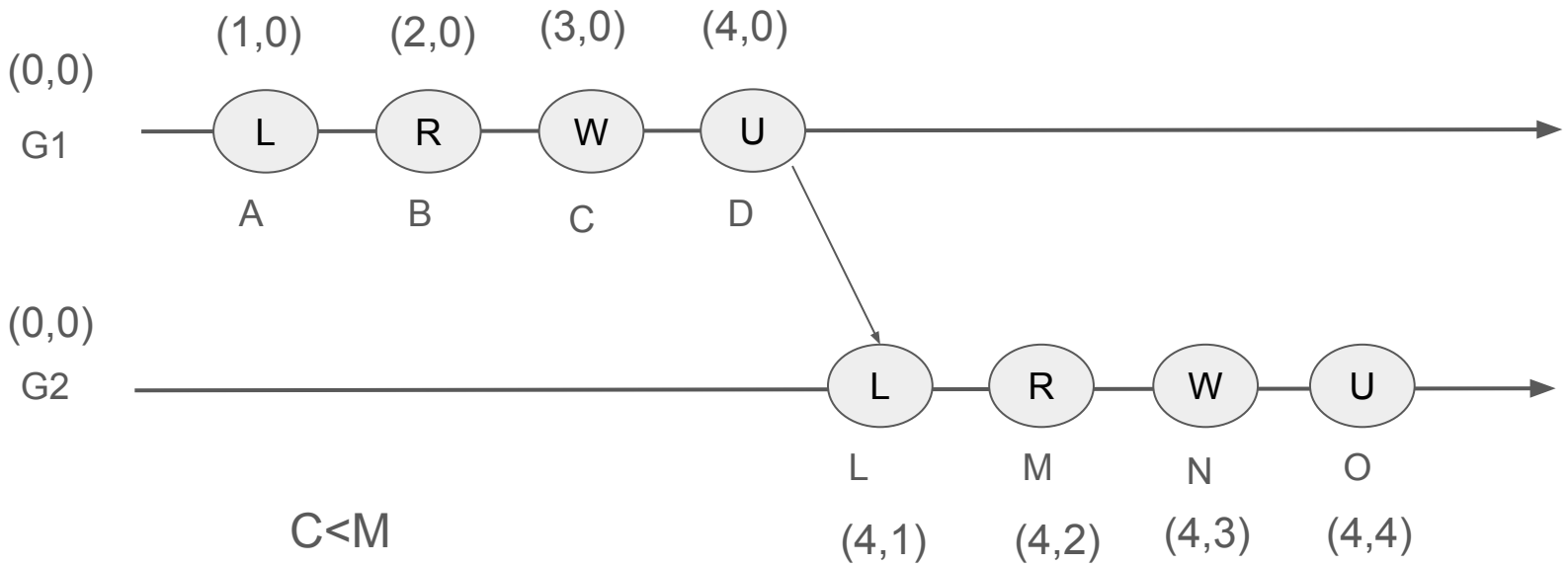
Vector Clocks



C<M

(3,0)< (4,2)

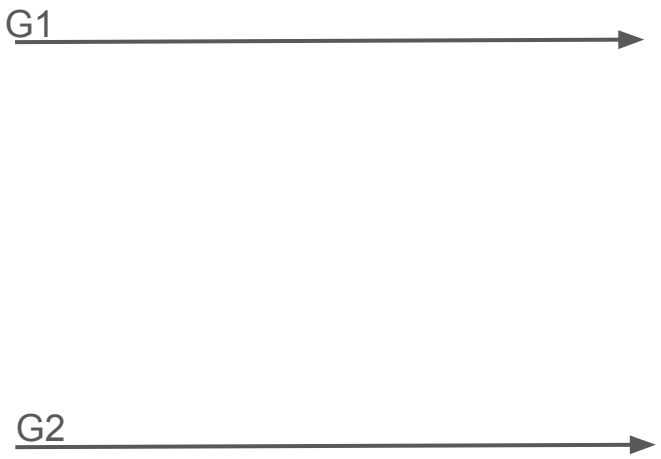
Vector Clocks

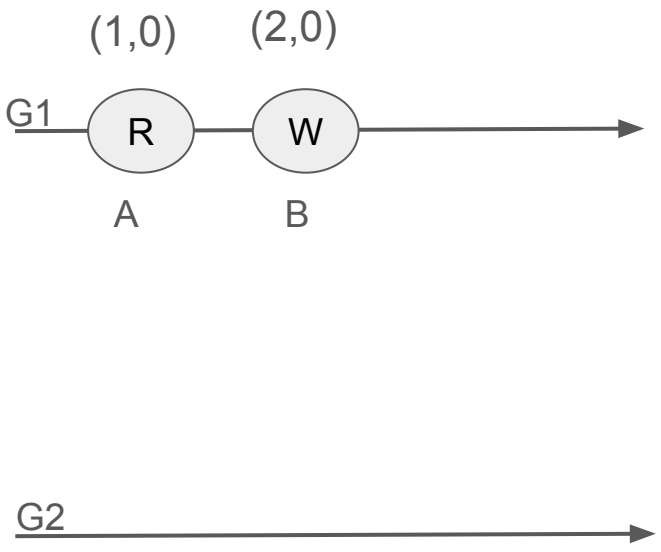


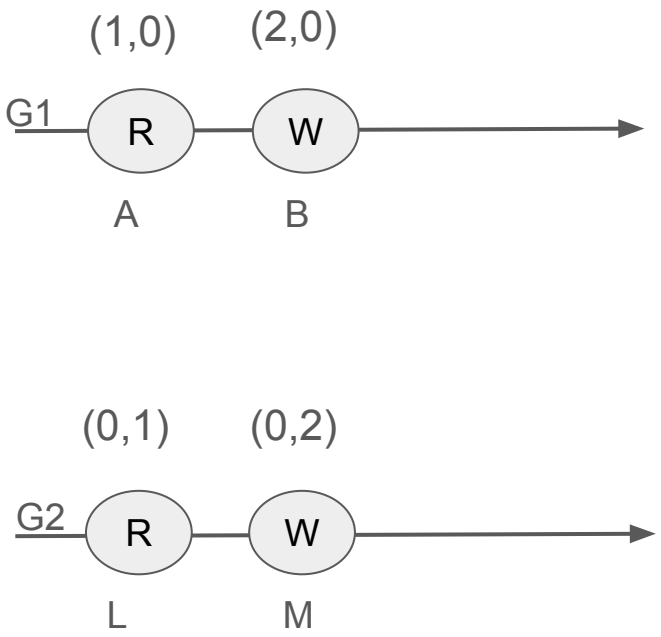
C<M

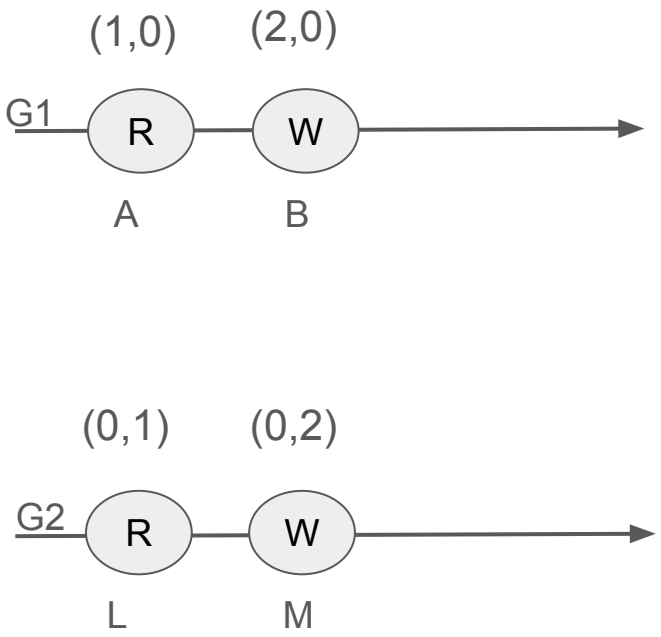
(3,0)< (4,2)

✅


G1
G2

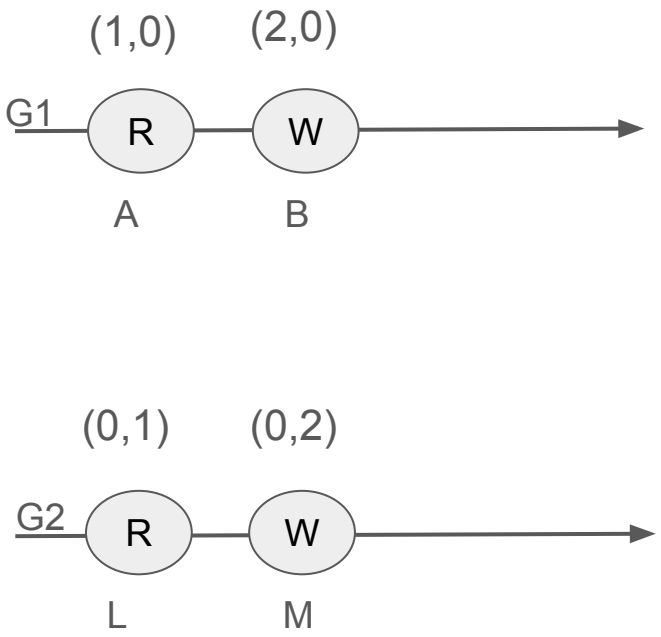

(1,0)
(2,0)
G1
R
W
A
B
G2


(1,0)
(2,0)
G1
R
W
A
B
(0,1)
(0,2)
G2
R
W
L
M



A<B

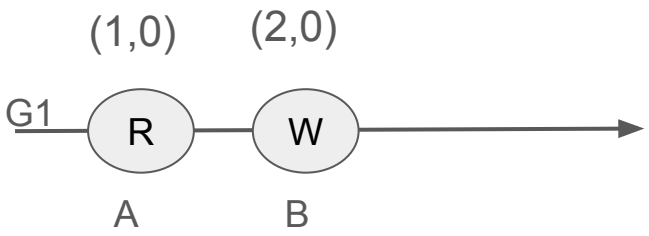
(1,0)<(2,0)

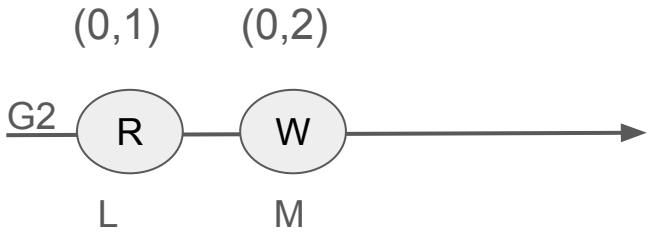


A<B

(1,0)<(2,0)

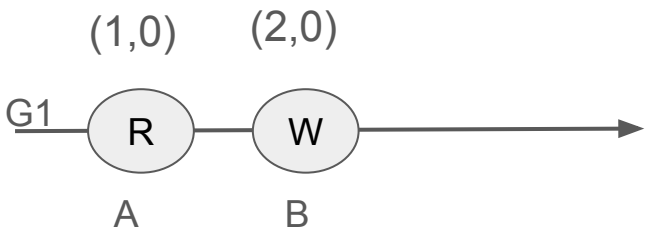
✅

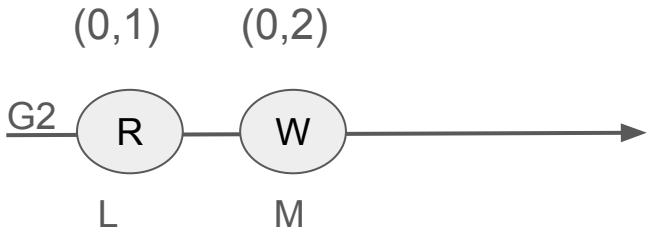




A<B
(1,0)<(2,0)
✅

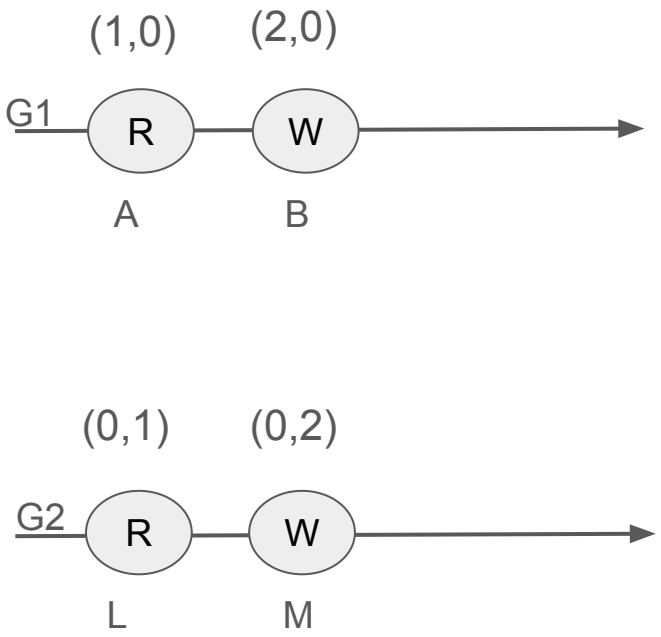
L<M
(0,1)<(0,2)





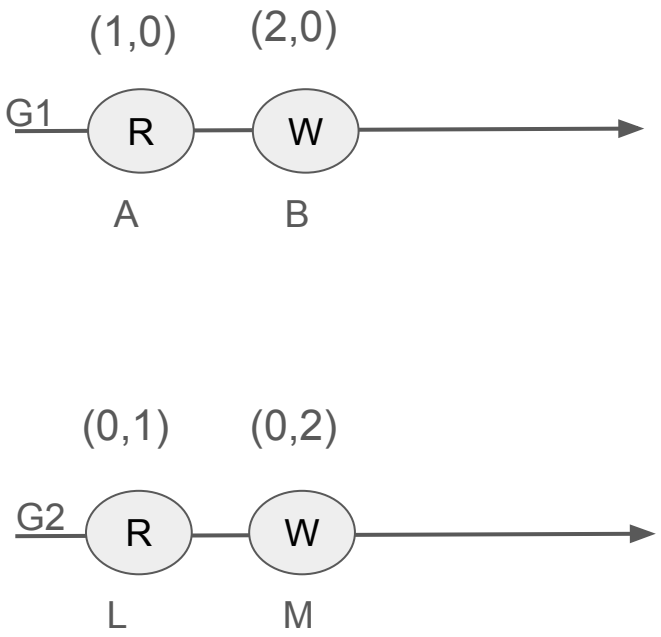
A<B
(1,0)<(2,0)
✅

L<M
(0,1)<(0,2)
✅



A<M

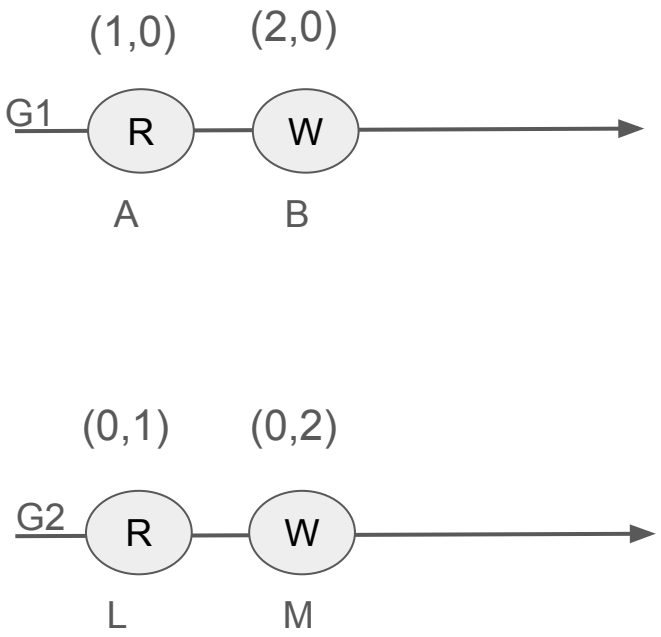
(1,0)<(0,2)



A<M

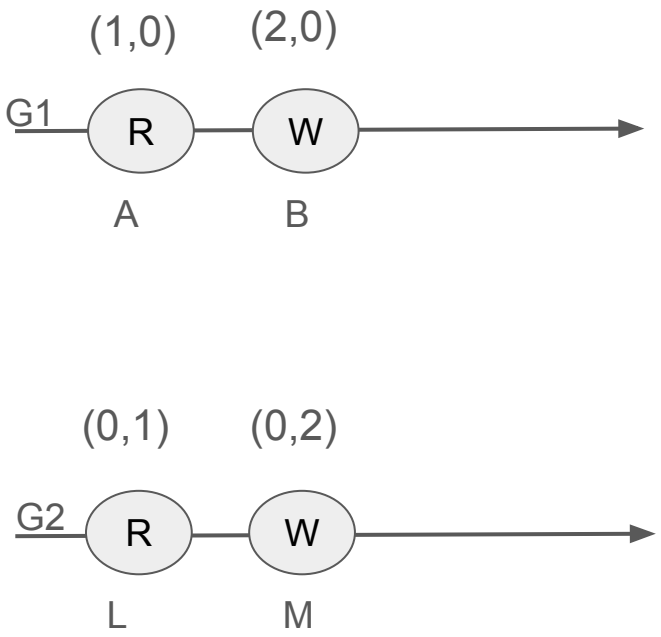
(1,0)<(0,2)

❌



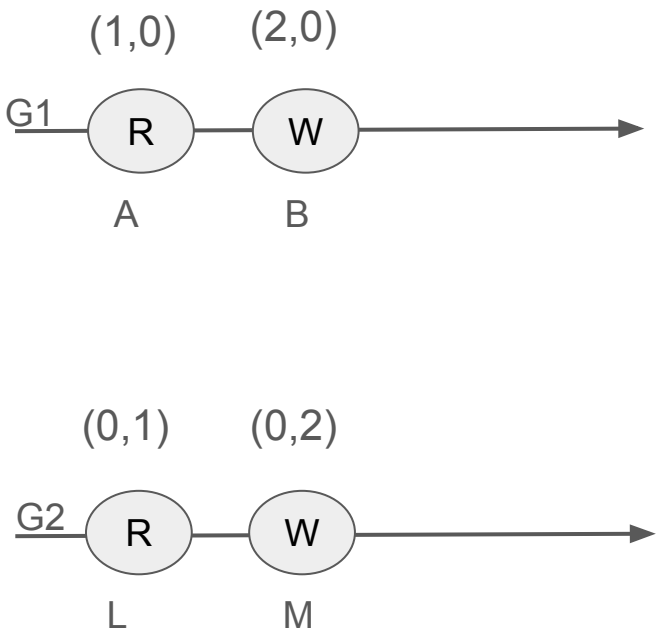
A<M
(1,0)<(0,2)
❌

M<B
(0,2)<(2,0)



A<M
(1,0)<(0,2)
❌

M<B
(0,2)<(2,0)
❌



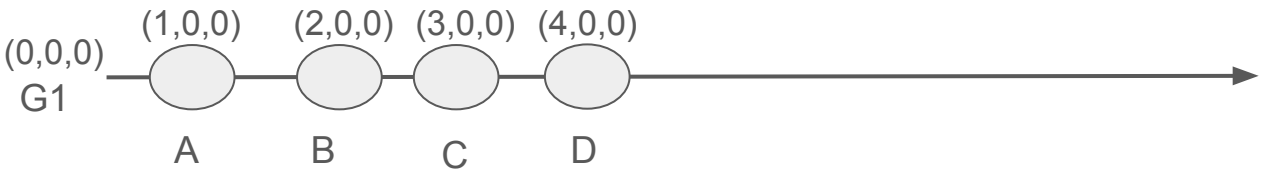
A<M
(1,0)<(0,2)
❌

M<B
(0,2)<(2,0)
❌

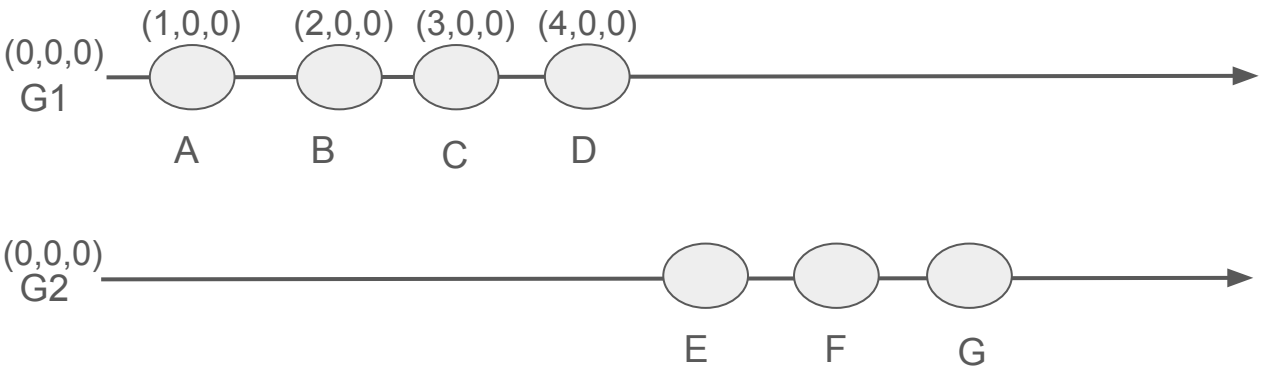
Concurrent
☹️

Vector Clocks

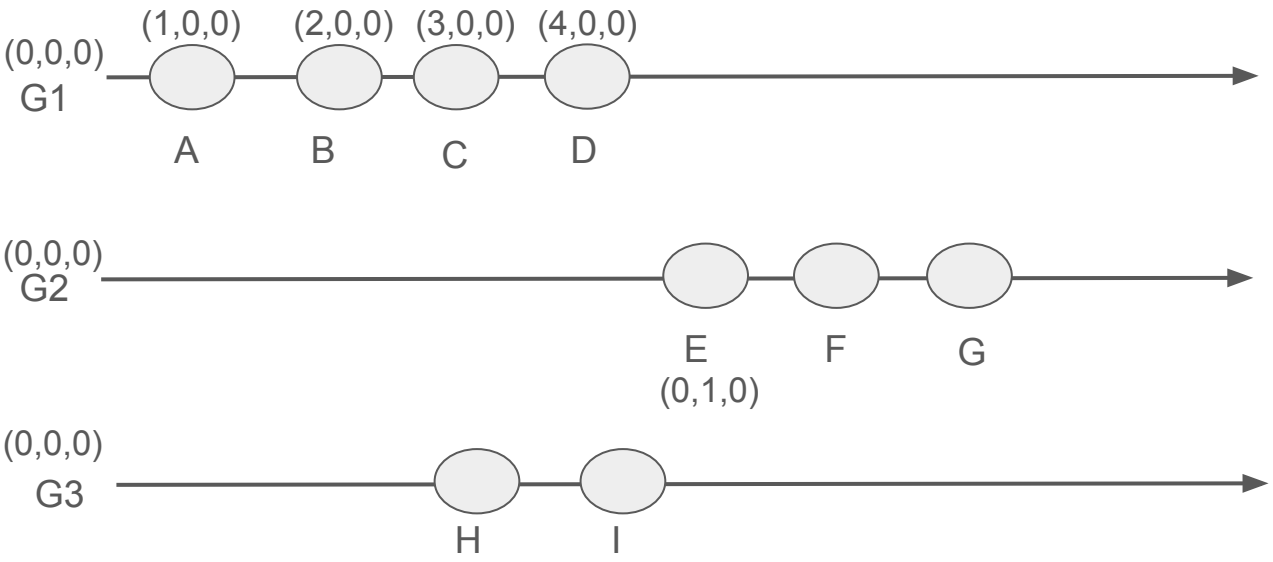
Vector Clocks



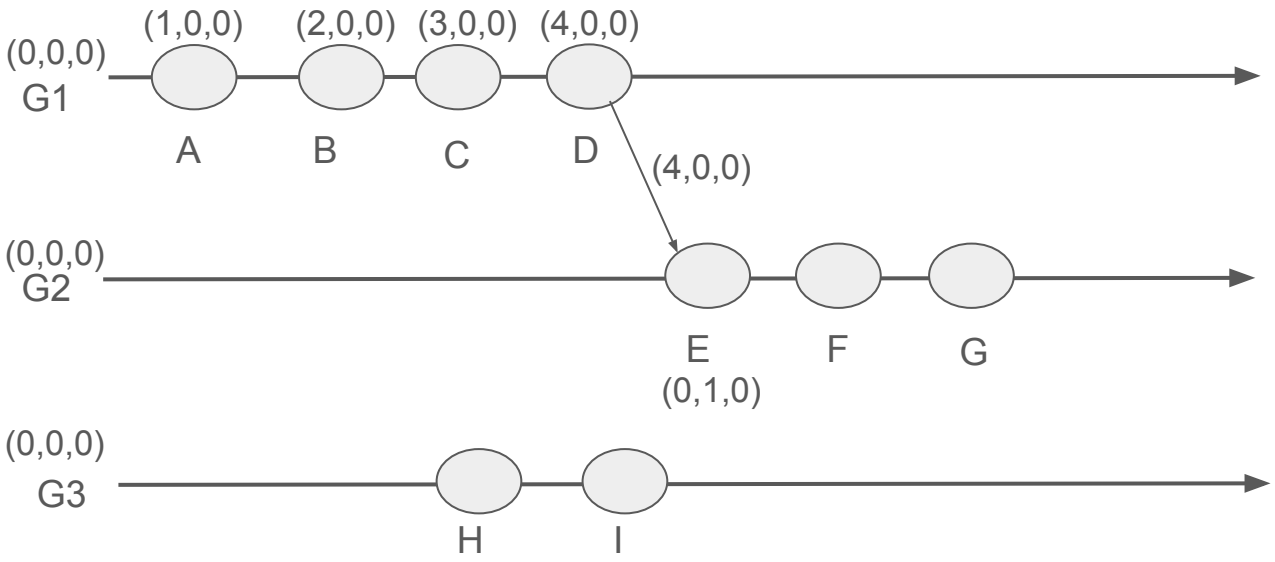
Vector Clocks



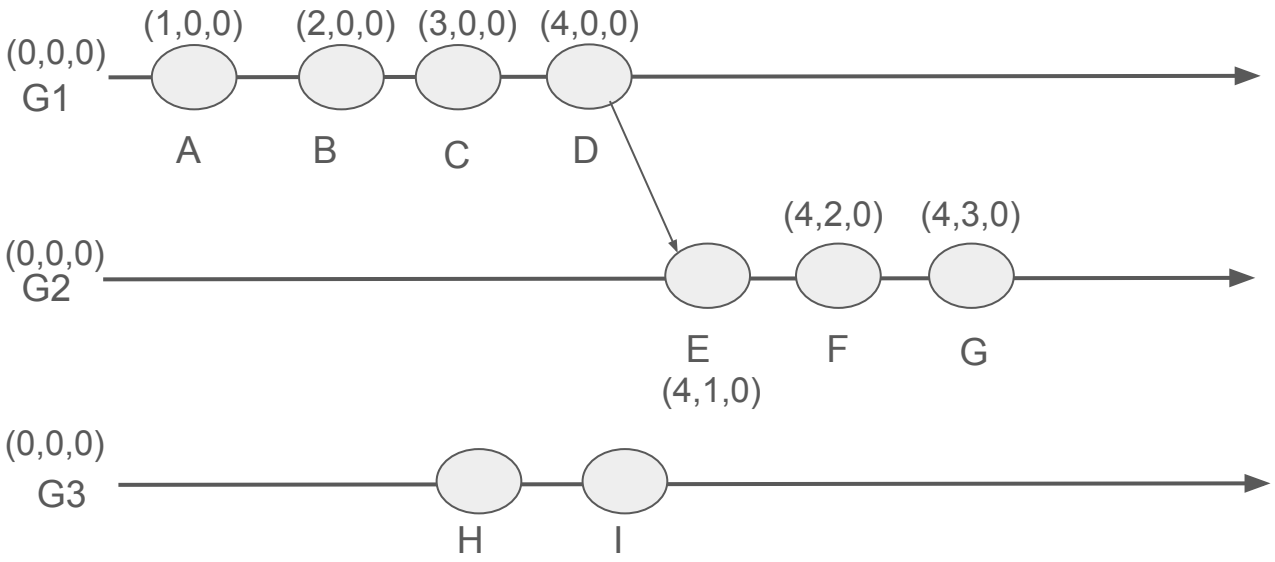
Vector Clocks



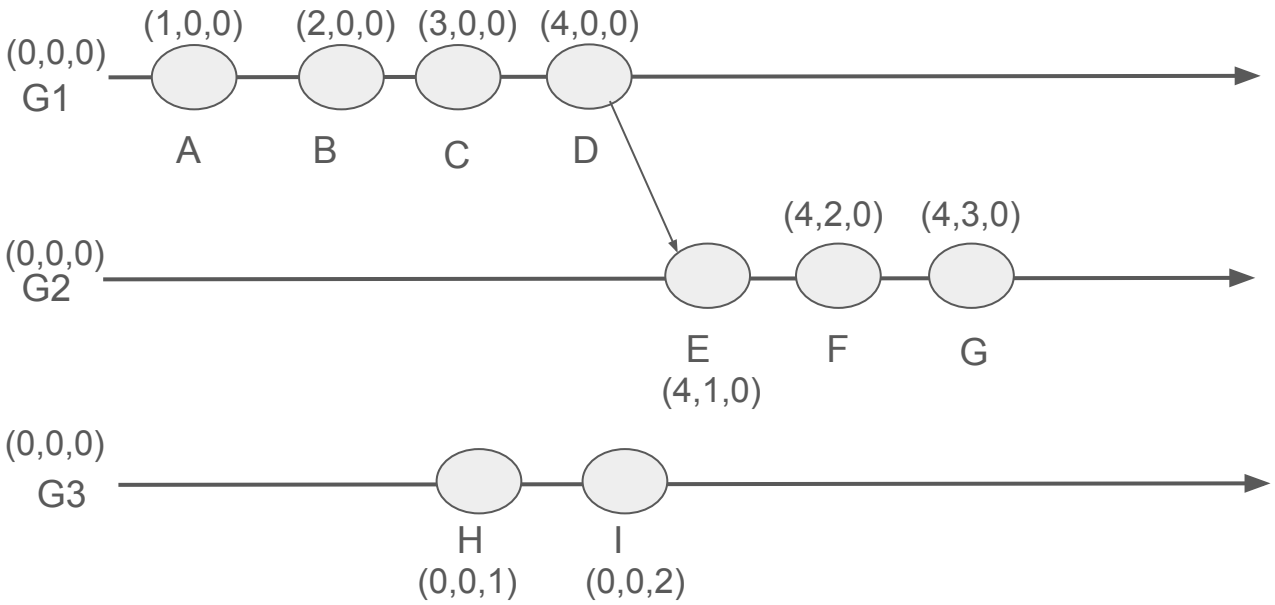
Vector Clocks



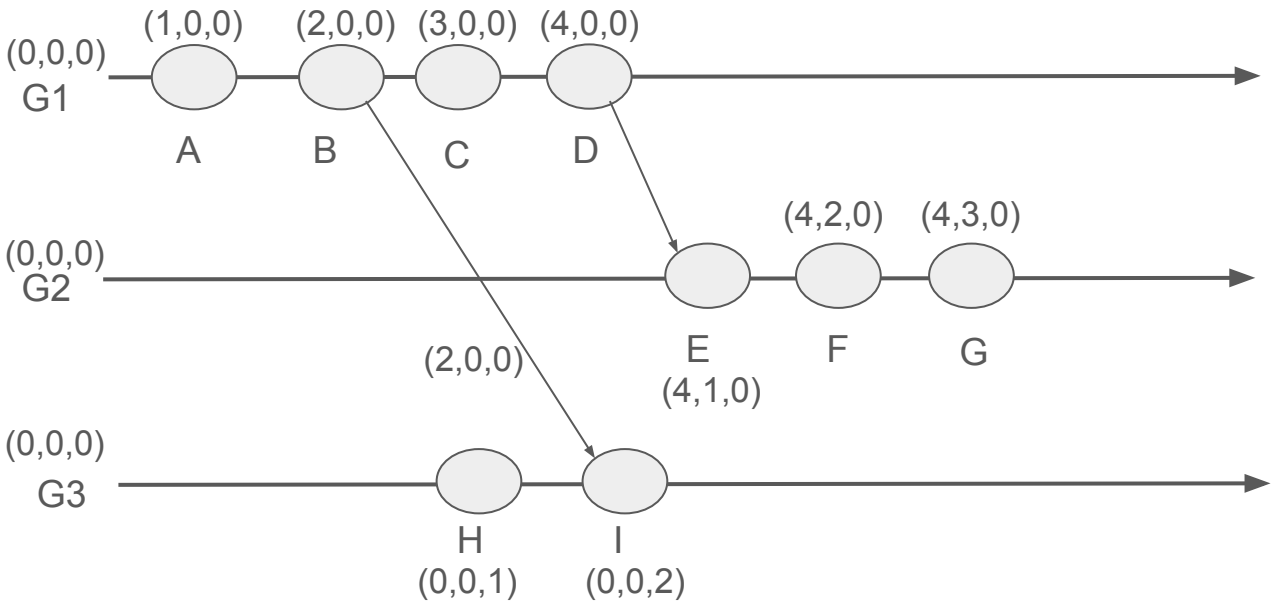
Vector Clocks



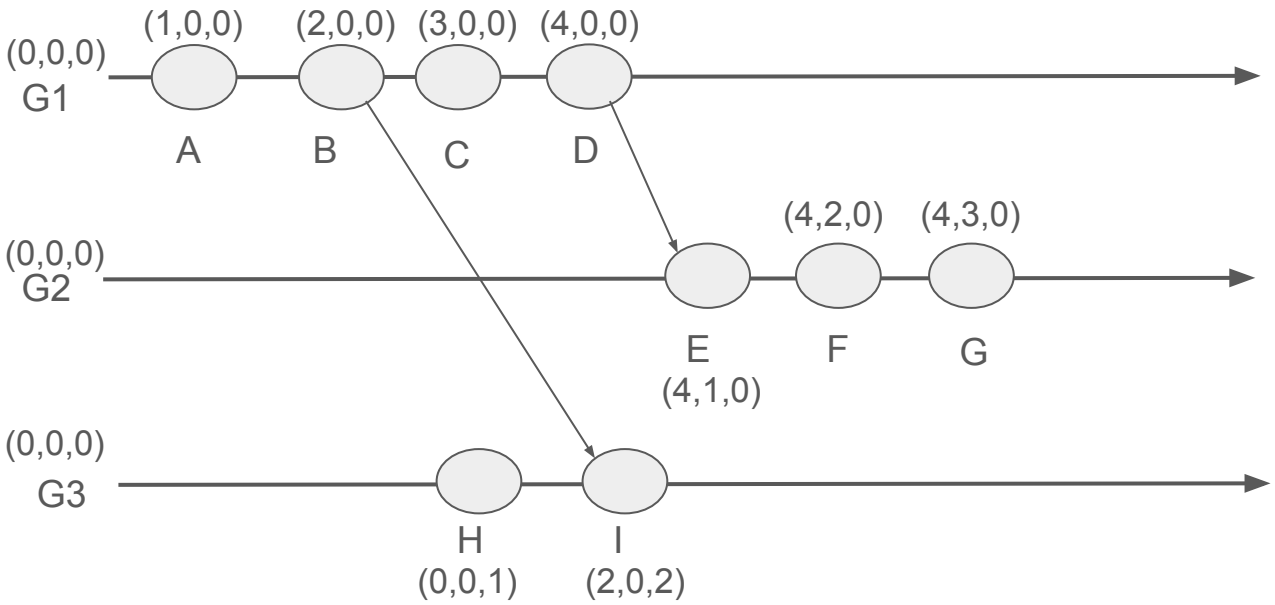
Vector Clocks



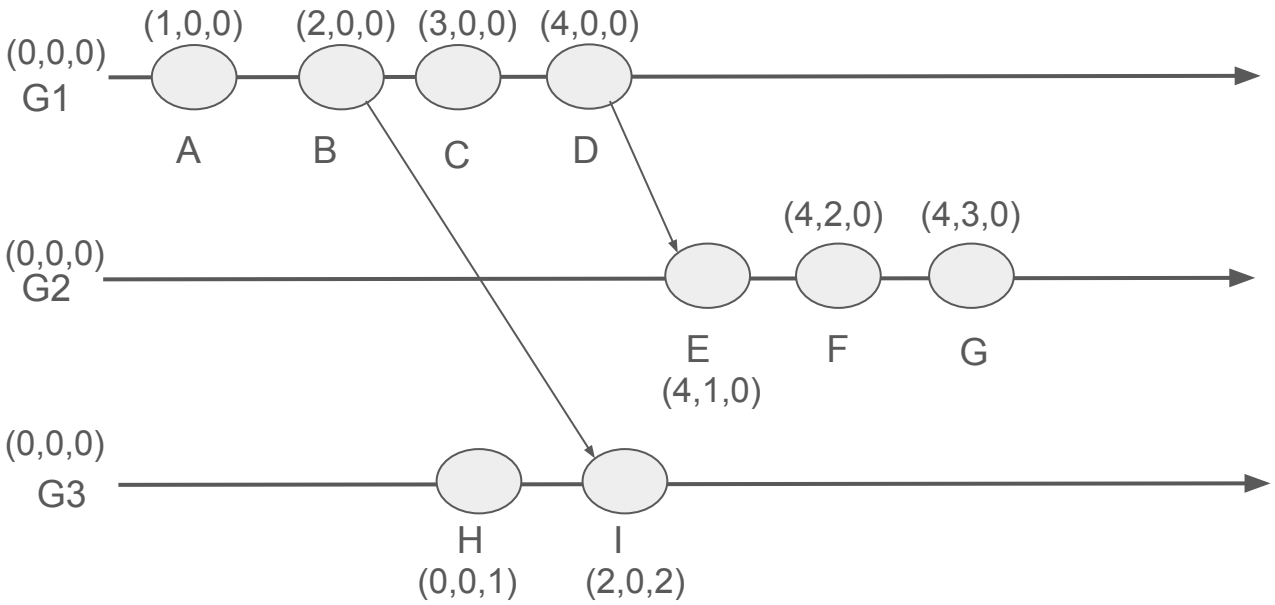
Vector Clocks



Vector Clocks

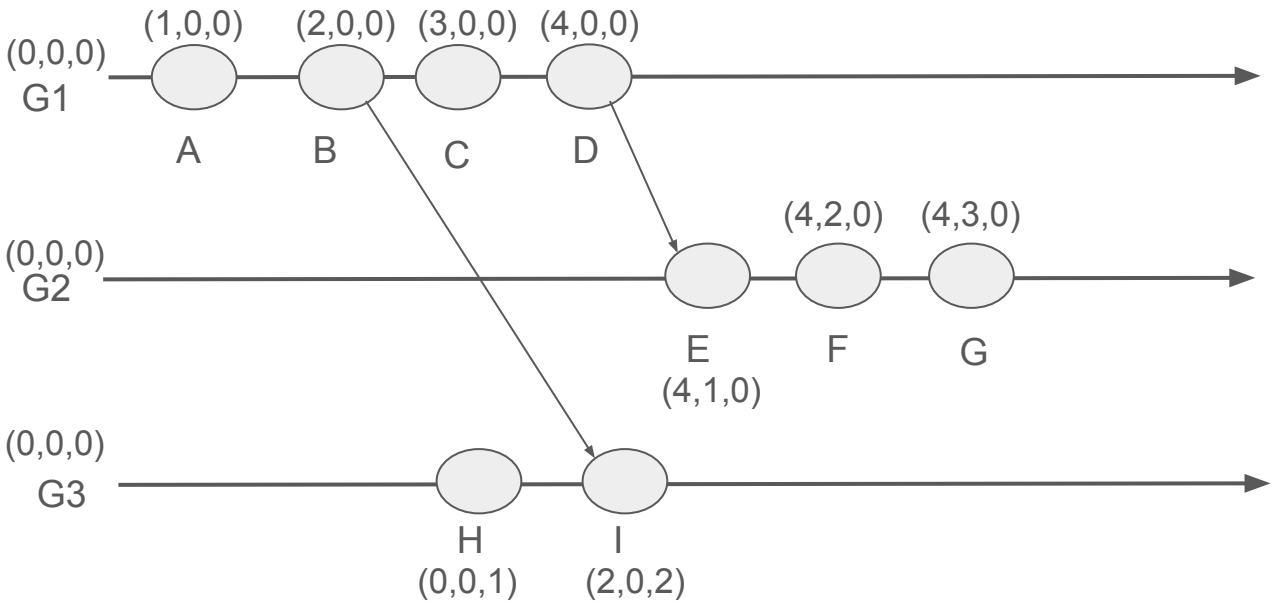


Vector Clocks



G<I

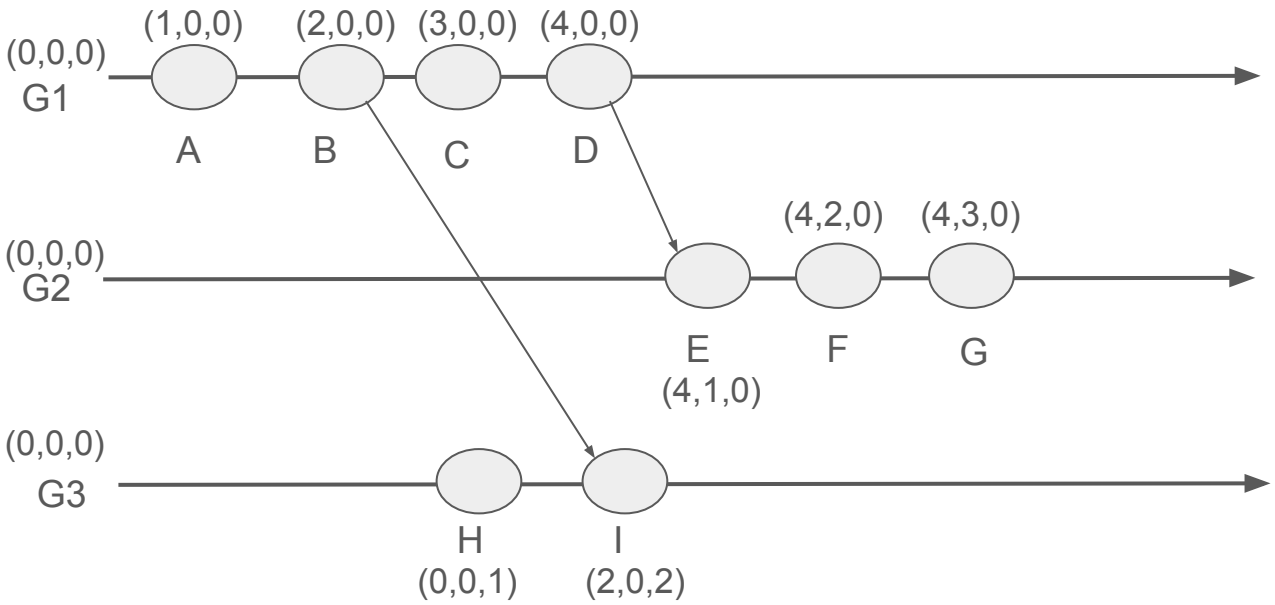
Vector Clocks



G<I

(4,3,0) < (2,0,2)

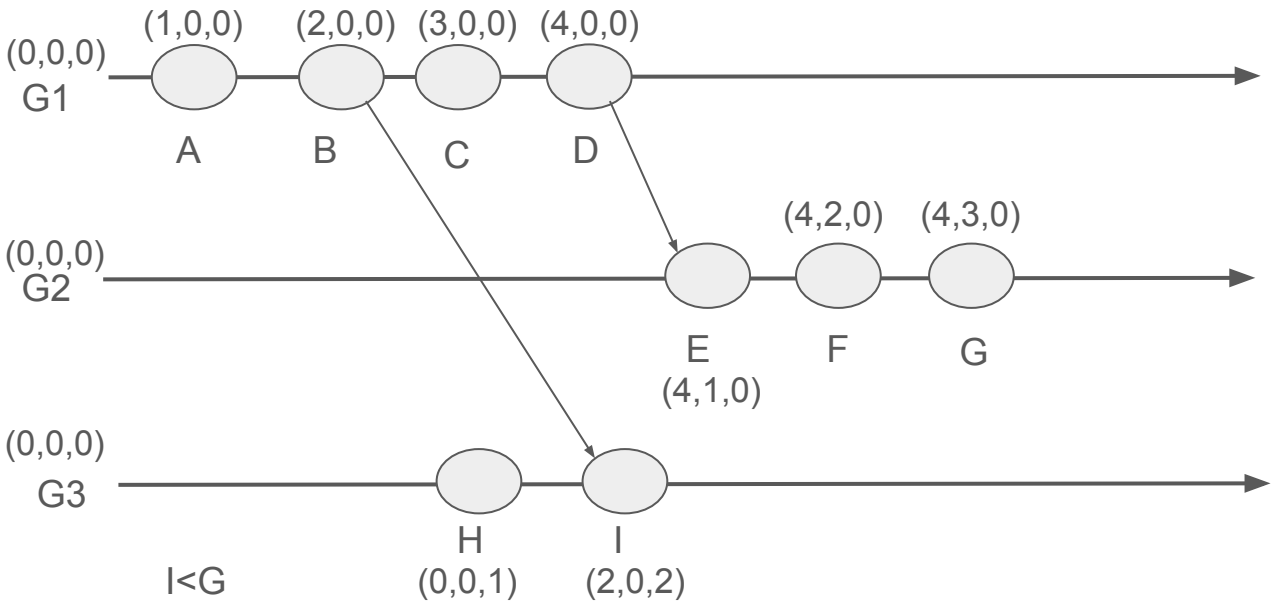
Vector Clocks



G<I

(4,3,0) < (2,0,2)

Vector Clocks



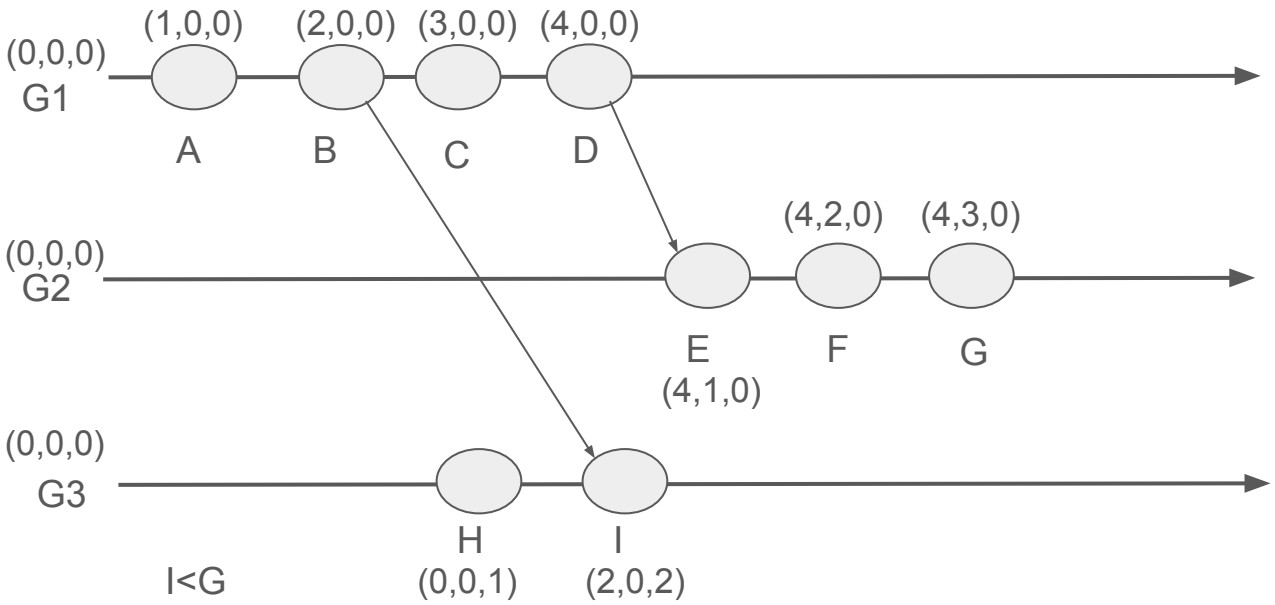
G<I

(4,3,0) < (2,0,2)

I<G

(2,0,2) < (4,3,0)

Vector Clocks



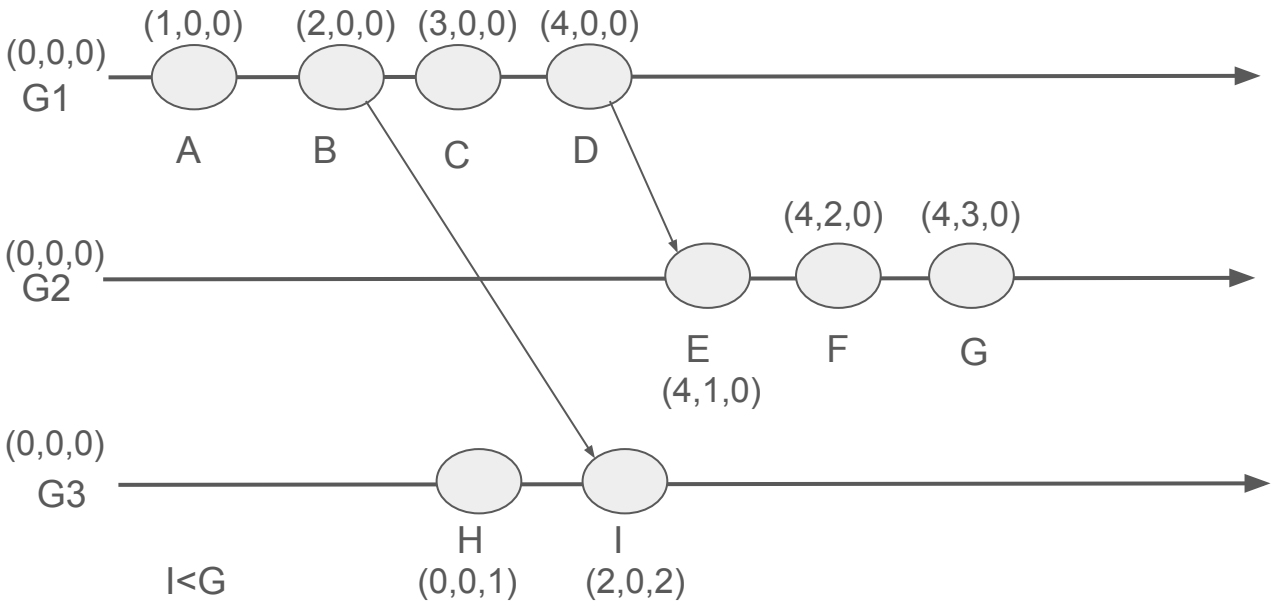
G<F

(4,3,0) < (2,0,2) ❌

I<G

(2,0,2) < (4,3,0) ❌

Vector Clocks



G<F

(4,3,0) < (2,0,2) ❌

I<G

(2,0,2) < (4,3,0) ❌

Concurrent

Go Race Detector follows pure happens-before Approach

Goレースディテクタはpure happens-beforeアプローチに従う

Go Race Detector follows pure happens-before Approach

Goレースディテクタはpure happens-beforeアプローチに従う

Go Race Detector Determines if the accesses to a memory location can be ordered by happens-before, using vector clocks.

Goレースディテクタは、ベクトルクロックを使用して、メモリ位置へのアクセスがhappens-beforeによって順序付けられるかどうかを判断する。

How ??

```
go run -race main.go
```

```
go run -race main.go
```

```
go run -race main.go
```

- Memory Access
- Creating Vector clocks For goroutines
- Updating vector clocks on synchronization events
- It compares Vector clocks to detect happens-before relation

```go
package main

import (
    "fmt"
)

var counter = 0

func IncrementCounter() {
    if counter == 0 {
        counter++
    }
}

func main() {

    go func() {
        IncrementCounter()
    }()

    go func() {
        IncrementCounter()
    }()

    fmt.Println("Final Counter Value:", counter)
}
```

go run -race main.go

```
WARNING: DATA RACE
Write at 0x000003205d90 by goroutine 6:
  main.IncrementCounter()
      /Users/vaibhavgupta/go/src/github.com/97vaibhav/race_detection/main.go:11 +0x52
  main.main.func1()
      /Users/vaibhavgupta/go/src/github.com/97vaibhav/race_detection/main.go:18 +0x18

Previous read at 0x000003205d90 by main goroutine:
  main.main()
      /Users/vaibhavgupta/go/src/github.com/97vaibhav/race_detection/main.go:25 +0x6d

Goroutine 6 (running) created at:
  main.main()
      /Users/vaibhavgupta/go/src/github.com/97vaibhav/race_detection/main.go:17 +0x27
==================
Found 1 data race(s)
exit status 66
```

```
GOOS=linux GOARCH=amd64 go tool compile -S main.go
```

GOOS=linux GOARCH=amd64 go tool compile -S main.go

```
TEXT    main.IncrementCounter(SB), NOSPLIT|NOFRAME|ABIInternal, $0-0
FUNCDATA        $0, gclocals·g2BeySu+wFnoycgXfElmcg==(SB)
FUNCDATA        $1, gclocals·g2BeySu+wFnoycgXfElmcg==(SB)
CMPQ    main.counter(SB), $0
JNE     21
MOVQ    $1, main.counter(SB)
RET
```

```
TEXT    main.IncrementCounter(SB), ABIInternal, $24-0
CMPQ    SP, 16(R14)
PCDATA  $0, $-2
JLS     109
PCDATA  $0, $-1
PUSHQ   BP
MOVQ    SP, BP
SUBQ    $16, SP
FUNCDATA        $0, gclocals·g2BeySu+wFnoycgXfElmcg==(SB)
FUNCDATA        $1, gclocals·g2BeySu+wFnoycgXfElmcg==(SB)
MOVQ    +24(FP), AX
PCDATA  $1, $0
CALL    runtime.racefuncenter(SB)
LEAQ    main.counter(SB), AX
NOP
CALL    runtime.raceread(SB)
CMPQ    main.counter(SB), $0
JNE     98
LEAQ    main.counter(SB), AX
CALL    runtime.raceread(SB)
MOVQ    main.counter(SB), CX
MOVQ    CX, main..autotmp_1+8(SP)
LEAQ    main.counter(SB), AX
CALL    runtime.racewrite(SB)
MOVQ    main..autotmp_1+8(SP), CX
INCQ    CX
MOVQ    CX, main.counter(SB)
CALL    runtime.racefuncexit(SB)
ADDQ    $16, SP
POPQ    BP
RET
NOP
PCDATA  $1, $-1
PCDATA  $0, $-2
CALL    runtime.morestack_noctxt(SB)
PCDATA  $0, $-1
JMP     0
```

GOOS=linux GOARCH=amd64 go tool compile -S -race main.go

```
TEXT    main.IncrementCounter(SB), ABIInternal, $24-0
CMPQ    SP, 16(R14)
PCDATA  $0, $-2
JLS     109
PCDATA  $0, $-1
PUSHQ   BP
MOVQ    SP, BP
SUBQ    $16, SP
FUNCDATA        $0, gclocals·g2BeySu+wFnoycgXfElmcg==(SB)
FUNCDATA        $1, gclocals·g2BeySu+wFnoycgXfElmcg==(SB)
MOVQ    +24(FP), AX
PCDATA  $1, $0
CALL    runtime.racefuncenter(SB)
LEAQ    main.counter(SB), AX
NOP
CALL    runtime.raceread(SB)
CMPQ    main.counter(SB), $0
JNE     98
LEAQ    main.counter(SB), AX
CALL    runtime.raceread(SB)
MOVQ    main.counter(SB), CX
MOVQ    CX, main..autotmp_1+8(SP)
LEAQ    main.counter(SB), AX
CALL    runtime.racewrite(SB)
MOVQ    main..autotmp_1+8(SP), CX
INCQ    CX
MOVQ    CX, main.counter(SB)
CALL    runtime.racefuncexit(SB)
ADDQ    $16, SP
POPQ    BP
RET
NOP
PCDATA  $1, $-1
PCDATA  $0, $-2
CALL    runtime.morestack_noctxt(SB)
PCDATA  $0, $-1
JMP     0
```

GOOS=linux GOARCH=amd64 go tool compile -S -race main.go

raceread()

GOOS=linux GOARCH=amd64 go tool compile -S -race main.go

```
TEXT    main.IncrementCounter(SB), ABIInternal, $24-0
CMPQ    SP, 16(R14)
PCDATA  $0, $-2
JLS     109
PCDATA  $0, $-1
PUSHQ   BP
MOVQ    SP, BP
SUBQ    $16, SP
FUNCDATA        $0, gclocals·g2BeySu+wFnoycgXfElmcg==(SB)
FUNCDATA        $1, gclocals·g2BeySu+wFnoycgXfElmcg==(SB)
MOVQ    +24(FP), AX
PCDATA  $1, $0
CALL    runtime.racefuncenter(SB)
LEAQ    main.counter(SB), AX
NOP
CALL    runtime.raceread(SB)
CMPQ    main.counter(SB), $0
JNE     98
LEAQ    main.counter(SB), AX
CALL    runtime.raceread(SB)
MOVQ    main.counter(SB), CX
MOVQ    CX, main..autotmp_1+8(SP)
LEAQ    main.counter(SB), AX
CALL    runtime.racewrite(SB)
MOVQ    main..autotmp_1+8(SP), CX
INCQ    CX
MOVQ    CX, main.counter(SB)
CALL    runtime.racefuncexit(SB)
ADDQ    $16, SP
POPQ    BP
RET
NOP
PCDATA  $1, $-1
PCDATA  $0, $-2
CALL    runtime.morestack_noctxt(SB)
PCDATA  $0, $-1
JMP     0
```

raceread()

racewrite()

GOOS=linux GOARCH=amd64 go tool compile -S -race main.go

```
TEXT    main.IncrementCounter(SB), ABIInternal, $24-0
CMPQ    SP, 16(R14)
PCDATA  $0, $-2
JLS     109
PCDATA  $0, $-1
PUSHQ   BP
MOVQ    SP, BP
SUBQ    $16, SP
FUNCDATA        $0, gclocals·g2BeySu+wFnoycgXfElmcg==(SB)
FUNCDATA        $1, gclocals·g2BeySu+wFnoycgXfElmcg==(SB)
MOVQ    +24(FP), AX
PCDATA  $1, $0
CALL    runtime.racefuncenter(SB)
LEAQ    main.counter(SB), AX
NOP
CALL    runtime.raceread(SB)
CMPQ    main.counter(SB), $0
JNE     98
LEAQ    main.counter(SB), AX
CALL    runtime.raceread(SB)
MOVQ    main.counter(SB), CX
MOVQ    CX, main..autotmp_1+8(SP)
LEAQ    main.counter(SB), AX
CALL    runtime.racewrite(SB)
MOVQ    main..autotmp_1+8(SP), CX
INCQ    CX
MOVQ    CX, main.counter(SB)
CALL    runtime.racefuncexit(SB)
ADDQ    $16, SP
POPQ    BP
RET
NOP
PCDATA  $1, $-1
PCDATA  $0, $-2
CALL    runtime.morestack_noctxt(SB)
PCDATA  $0, $-1
JMP     0
```

raceread()

racewrite()

racefuncexit()

```go
func IncrementCounter() {
    if counter == 0 {
        counter++
    }
}
```

```
TEXT    main.IncrementCounter(SB), ABIInternal, $24-0
CMPQ    SP, 16(R14)
PCDATA  $0, $-2
JLS     109
PCDATA  $0, $-1
PUSHQ   BP
MOVQ    SP, BP
SUBQ    $16, SP
FUNCDATA        $0, gclocals·g2BeySu+wFnoycgXfElmcg==(SB)
FUNCDATA        $1, gclocals·g2BeySu+wFnoycgXfElmcg==(SB)
MOVQ    +24(FP), AX
PCDATA  $1, $0
CALL    runtime.racefuncenter(SB)
LEAQ    main.counter(SB), AX
NOP
CALL    runtime.raceread(SB)
CMPQ    main.counter(SB), $0
JNE     98
LEAQ    main.counter(SB), AX
CALL    runtime.raceread(SB)
MOVQ    main.counter(SB), CX
MOVQ    CX, main..autotmp_1+8(SP)
LEAQ    main.counter(SB), AX
CALL    runtime.racewrite(SB)
MOVQ    main..autotmp_1+8(SP), CX
INCQ    CX
MOVQ    CX, main.counter(SB)
CALL    runtime.racefuncexit(SB)
ADDQ    $16, SP
POPQ    BP
RET
NOP
PCDATA  $1, $-1
PCDATA  $0, $-2
CALL    runtime.morestack_noctxt(SB)
PCDATA  $0, $-1
JMP     0
```

```go
func IncrementCounter() {
	raceread()
	if counter == 0 {

		counter++
	}

}
```

```
TEXT    main.IncrementCounter(SB), ABIInternal, $24-0
CMPQ    SP, 16(R14)
PCDATA  $0, $-2
JLS     109
PCDATA  $0, $-1
PUSHQ   BP
MOVQ    SP, BP
SUBQ    $16, SP
FUNCDATA        $0, gclocals·g2BeySu+wFnoycgXfElmcg==(SB)
FUNCDATA        $1, gclocals·g2BeySu+wFnoycgXfElmcg==(SB)
MOVQ    +24(FP), AX
PCDATA  $1, $0
CALL    runtime.racefuncenter(SB)
LEAQ    main.counter(SB), AX
NOP
CALL    runtime.raceread(SB)
CMPQ    main.counter(SB), $0
JNE     98
LEAQ    main.counter(SB), AX
CALL    runtime.raceread(SB)
MOVQ    main.counter(SB), CX
MOVQ    CX, main..autotmp_1+8(SP)
LEAQ    main.counter(SB), AX
CALL    runtime.racewrite(SB)
MOVQ    main..autotmp_1+8(SP), CX
INCQ    CX
MOVQ    CX, main.counter(SB)
CALL    runtime.racefuncexit(SB)
ADDQ    $16, SP
POPQ    BP
RET
NOP
PCDATA  $1, $-1
PCDATA  $0, $-2
CALL    runtime.morestack_noctxt(SB)
PCDATA  $0, $-1
JMP     0
```

```go
func IncrementCounter() {
	raceread()
	if counter == 0 {
	racewrite()
		counter++
	}

}
```

```
TEXT    main.IncrementCounter(SB), ABIInternal, $24-0
CMPQ    SP, 16(R14)
PCDATA  $0, $-2
JLS     109
PCDATA  $0, $-1
PUSHQ   BP
MOVQ    SP, BP
SUBQ    $16, SP
FUNCDATA        $0, gclocals·g2BeySu+wFnoycgXfElmcg==(SB)
FUNCDATA        $1, gclocals·g2BeySu+wFnoycgXfElmcg==(SB)
MOVQ    +24(FP), AX
PCDATA  $1, $0
CALL    runtime.racefuncenter(SB)
LEAQ    main.counter(SB), AX
NOP
CALL    runtime.raceread(SB)
CMPQ    main.counter(SB), $0
JNE     98
LEAQ    main.counter(SB), AX
CALL    runtime.raceread(SB)
MOVQ    main.counter(SB), CX
MOVQ    CX, main..autotmp_1+8(SP)
LEAQ    main.counter(SB), AX
CALL    runtime.racewrite(SB)
MOVQ    main..autotmp_1+8(SP), CX
INCQ    CX
MOVQ    CX, main.counter(SB)
CALL    runtime.racefuncexit(SB)
ADDQ    $16, SP
POPQ    BP
RET
NOP
PCDATA  $1, $-1
PCDATA  $0, $-2
CALL    runtime.morestack_noctxt(SB)
PCDATA  $0, $-1
JMP     0
```

```go
func IncrementCounter() {
	raceread()
	if counter == 0 {
	racewrite()
		counter++
	}
	racefuncexit()
}
```

```
TEXT    main.IncrementCounter(SB), ABIInternal, $24-0
CMPQ    SP, 16(R14)
PCDATA  $0, $-2
JLS     109
PCDATA  $0, $-1
PUSHQ   BP
MOVQ    SP, BP
SUBQ    $16, SP
FUNCDATA        $0, gclocals·g2BeySu+wFnoycgXfElmcg==(SB)
FUNCDATA        $1, gclocals·g2BeySu+wFnoycgXfElmcg==(SB)
MOVQ    +24(FP), AX
PCDATA  $1, $0
CALL    runtime.racefuncenter(SB)
LEAQ    main.counter(SB), AX
NOP
CALL    runtime.raceread(SB)
CMPQ    main.counter(SB), $0
JNE     98
LEAQ    main.counter(SB), AX
CALL    runtime.raceread(SB)
MOVQ    main.counter(SB), CX
MOVQ    CX, main..autotmp_1+8(SP)
LEAQ    main.counter(SB), AX
CALL    runtime.racewrite(SB)
MOVQ    main..autotmp_1+8(SP), CX
INCQ    CX
MOVQ    CX, main.counter(SB)
CALL    runtime.racefuncexit(SB)
ADDQ    $16, SP
POPQ    BP
RET
NOP
PCDATA  $1, $-1
PCDATA  $0, $-2
CALL    runtime.morestack_noctxt(SB)
PCDATA  $0, $-1
JMP     0
```

https://github.com/golang/go/blob/960fa9bf66139e535d89934f56ae20a0e679e203/src/sync/mutex.go#L81C17-L81C21

```go
func (m *Mutex) Lock() {
    // Fast path: grab unlocked mutex.
    if atomic.CompareAndSwapInt32(&m.state, 0, mutexLocked) {
        if race.Enabled {
            race.Acquire(unsafe.Pointer(m))
        }
        return
    }
    // Slow path (outlined so that the fast path can be inlined)
    m.lockSlow()
}
```

```go
func Acquire(addr unsafe.Pointer) {

}
```

ThreadSanitizer(Tsan)

- Go race detector is built on Tsan
- TSan implements happens-before race detection .
- Tsan creates, updates vector clocks for goroutines - ThreadState
- Tsan keeps track of memory access, synchronization events such as lock, unlock - Shadow State
- Tsan compares vector clocks to detect data races in our code .

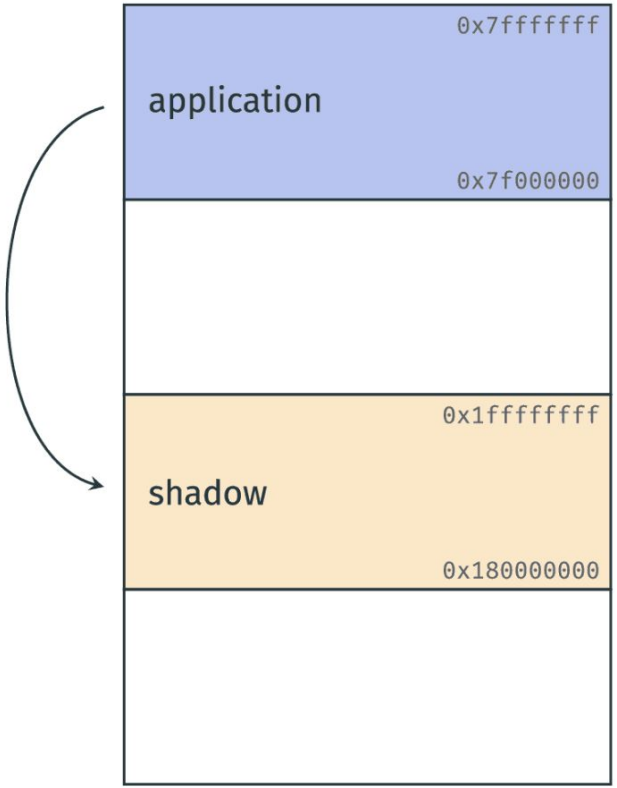
Shadow State

Stores information about memory accesses.

8-byte shadow word for an access

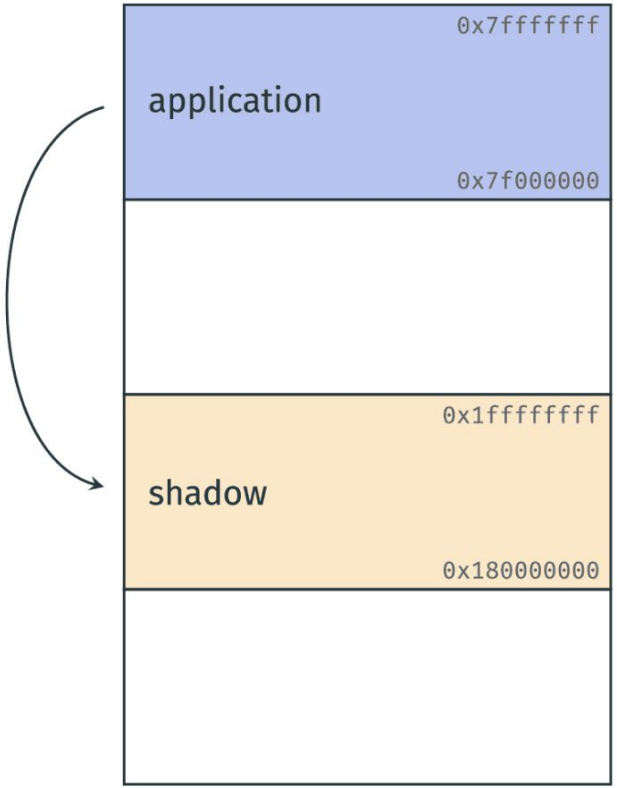
TID	epoch	addr	rd/wr

```
Struct Shadow {
 tid // thread id
 epoch // thread's clock time
 addr // memory access address
 write // read/write
}
```



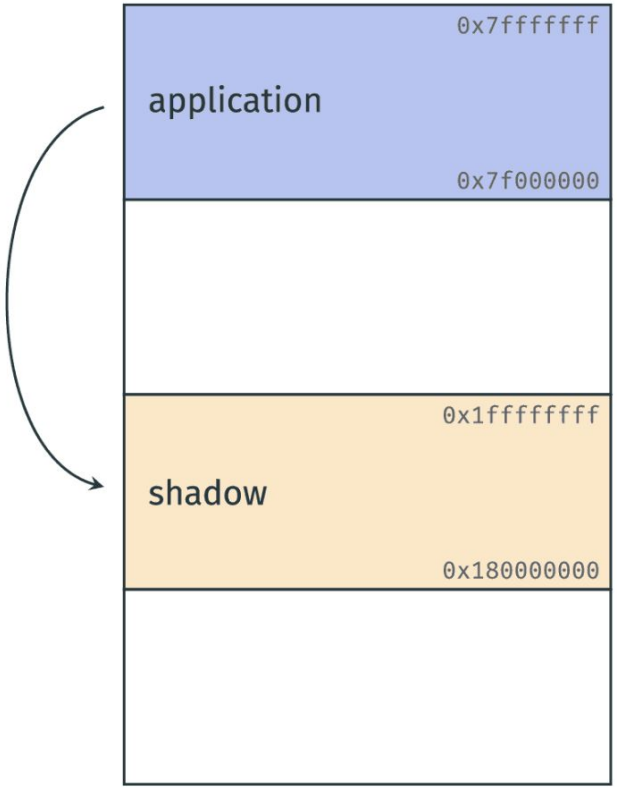
Shadow State

Do the accesses overlap?
Is one of them a write?
Are the thread IDs different?
Are they unordered by happens-before?



Shadow State

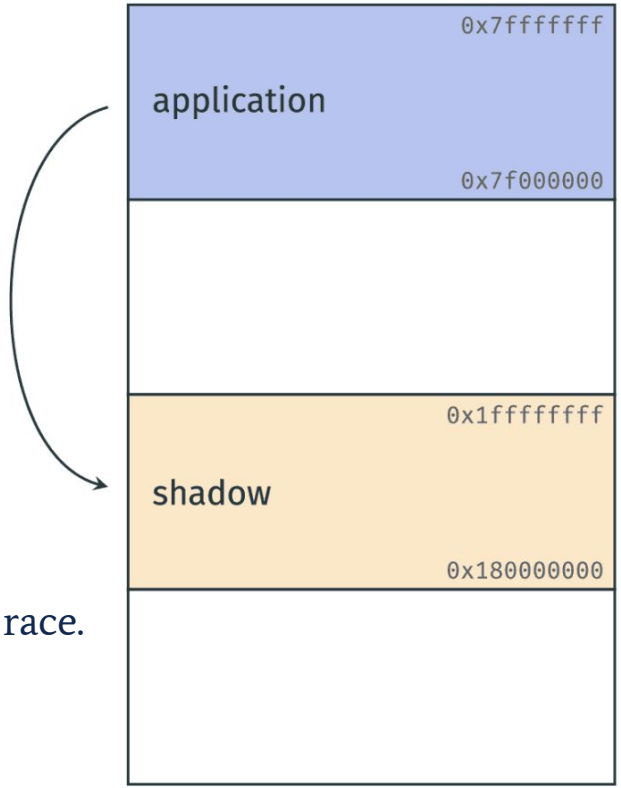
Do the accesses overlap ✅
Is one of them a write ✅
Are the thread IDs different ✅
Are they unordered by happens-before ✅



Shadow State

Do the accesses overlap ✅
Is one of them a write ✅
Are the thread IDs different ✅
Are they unordered by happens-before ✅

If these conditions are met, TSan detects and reports a data race.



Best Practices to avoid Data Race
Race Race を回避するためのベストプラクティス

1. Use Synchronization Primitives
2. Avoid Unprotected Access
3. Practice Goroutine Communication
4. Test Concurrency
5. Monitor and Profile

Lets Evaluate Race Detector

Reliable

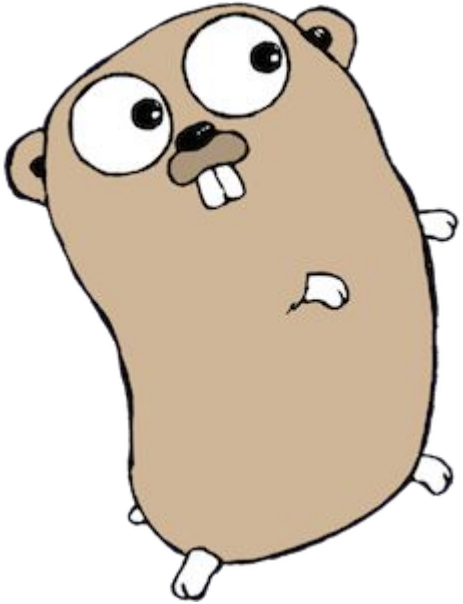
- Won't give you unnecessary results
- The Race detector does not give false positives
- Can miss races condition https://medium.com/@val_deleplace/does-the-race-detector-catch-all-data-races-1afed51d57fb

Scalable

- Execution time = 2x-20x
- memory usage = 5x-10x https://go.dev/doc/articles/race_detector

References :

- https://speakerdeck.com/kkc/go-race-detector-under-the-hood?slide=11
- https://go.dev/doc/articles/race_detector
- https://medium.com/@val_deleplace/does-the-race-detector-catch-all-data-races-1afed51d57fb
- https://go.dev/doc/articles/race_detector#Typical_Data_Races
- https://tarides.com/blog/2023-10-18-off-to-the-races-using-threadsanitizer-in-ocaml/
- https://www.youtube.com/watch?v=4wq-YlaI_vk&t=1312s
- https://www.infoq.com/presentations/go-race-detector/



聞いてくれて
ありがとうございます

Q/A