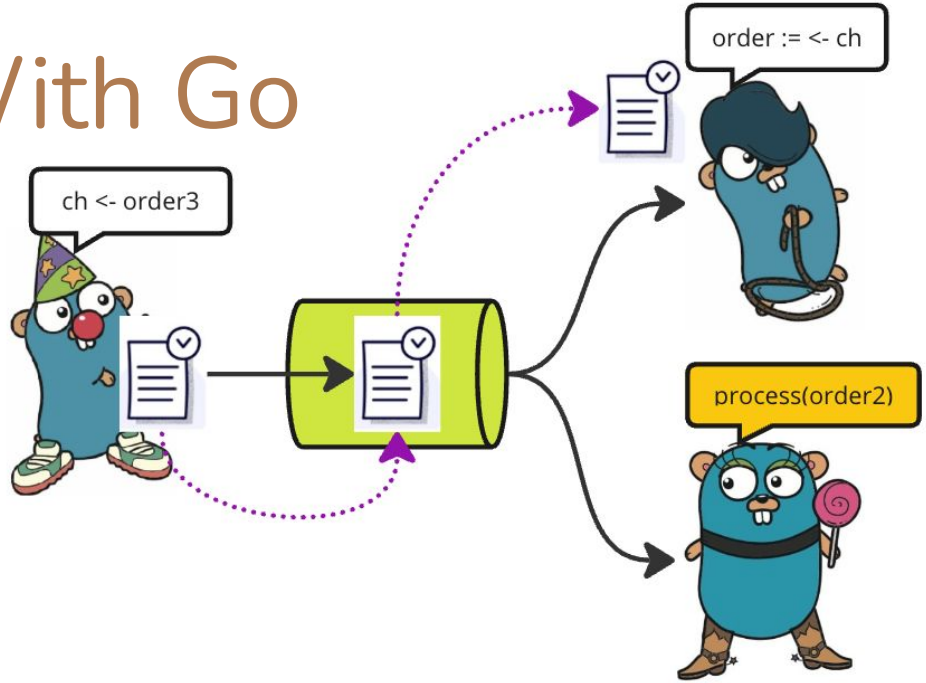
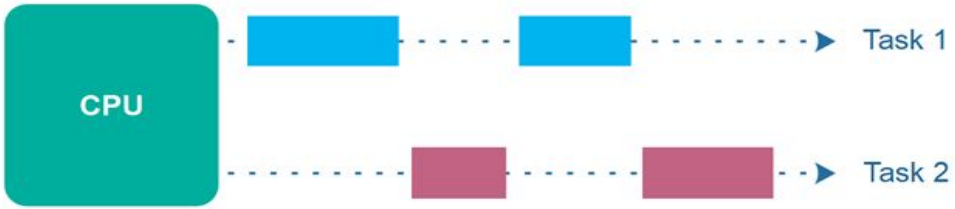
Concurrency With Go



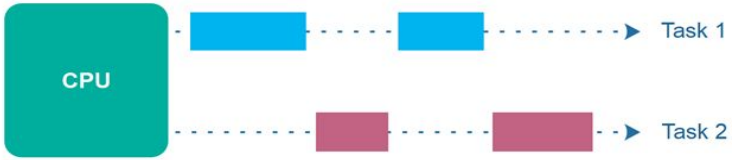
Old Problem

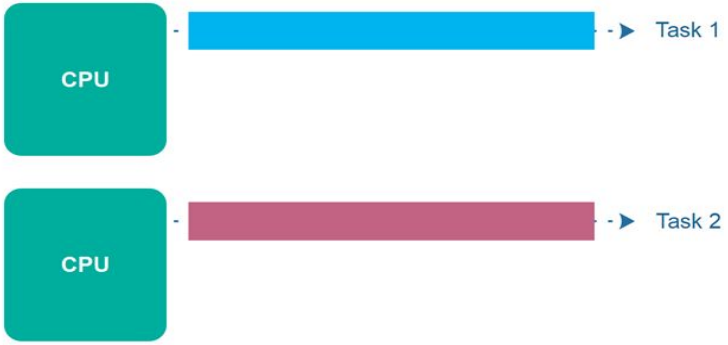
- Sync data/delete data profile list in Responsys but it took > 5 seconds per request because it’s in batch
- Rate limit
- Hundred of thousands data as fast as possible
- We use **concurrency** way to solve it

What is concurrency?



Concurrency vs Parallelism





Deadlock & race condition

- Deadlock is a situation where two or more threads or processes are unable to proceed because each is waiting for the other to release a resource.
- A race condition is a situation in which the behavior of a program depends on the relative timing of events, such as the order of execution of multiple threads or processes accessing shared resources without proper synchronization.

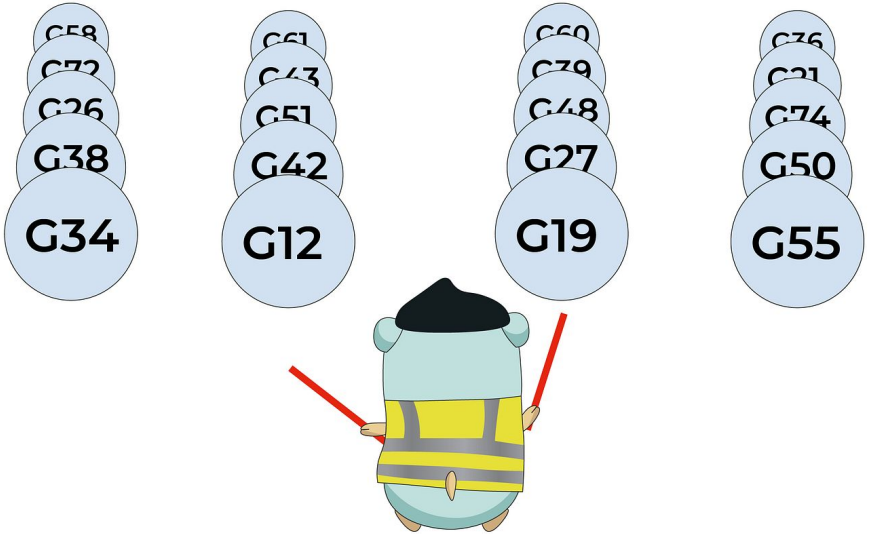
Why golang(for concurrency)

- Simple syntax, ease to learn
- Concurrency primitives(channel,goroutine,select)
- Performance(light, fast).
- Community and ecosystem
- Built-in Race Detector
- Scalability

What we will use & learn?

- Goroutines
- Channels
- Buffer Channels
- Sync package: Waitgroup & Mutex
- Select

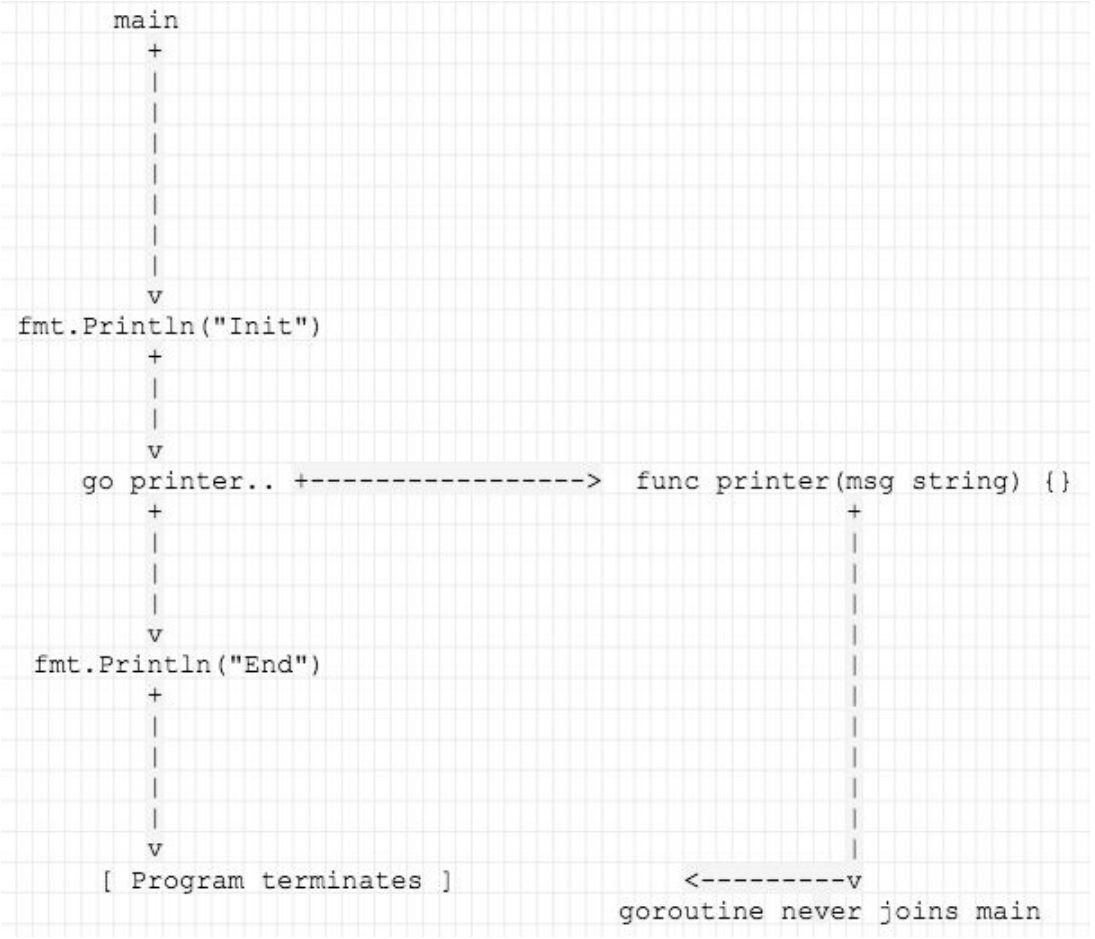
Goroutines



```go
func Goroutines() {
    go sayHello()

    fmt.Println("World")
}

func sayHello() {
    fmt.Println("Hello!")
}
```

```go
func GoroutinesAnonymousWithVariableDeclared() {
    sayHello := func() {
        fmt.Println("Hello")
    }

    go sayHello()

    fmt.Println("World")
}
```

```go
func GoroutinesAnonymous() {
    go func() {
        fmt.Println("Hello")
    }()

    fmt.Println("World")
}
```


main
fmt.Println("Init")
go printer..
func printer(msg string) {}
fmt.Println("End")
[Program terminates]
goroutine never joins main

Sync package

- **WaitGroup**
- **Mutex**
- RWMutex
- Cond
- Once
- Pool

```go
// now we can print inside the go routine function
func GoroutinesWithWaitGroup() {
    var wg sync.WaitGroup // digunakan untuk menunggu goroutine

    sayHello := func() {
        // Done() is called within each goroutine to signal that it has completed its work.
        // When a goroutine finishes its task, it should call Done() to decrement the internal counter in the WaitGroup.
        // When the counter reaches zero, any goroutine waiting on Wait() will unblock.
        defer wg.Done() // defer keyword is used to delay the execution of a function or a statement until the nearby function returns

        fmt.Println("Run go routines with GoroutinesWithWaitGroup")
    }

    // You call Add(n) to indicate that you expect to wait for n goroutines to finish their work.
    wg.Add(1) // is used to specify the number of goroutines you want to wait for.

    go sayHello()

    // blocks the execution of the current goroutine until the internal counter in the WaitGroup reaches zero.
    // It effectively waits for all the goroutines you've indicated with Add() to call Done() and signal that they have completed their work.
    // Once the counter reaches zero, Wait() unblocks, and your program can continue.
    wg.Wait()

    fmt.Println("Outside the function")
}
```

```go
type counter struct {
    val int
}

func (c *counter) Add(int) {
    c.val++
}

func (c *counter) Value() int {
    return c.val
}

func RaceCondition() {
    runtime.GOMAXPROCS(2)

    var wg sync.WaitGroup
    var meter counter

    for i := 0; i < 1000; i++ {
        wg.Add(1)

        go func() {
            for j := 0; j < 1000; j++ {
                meter.Add(1)
            }

            wg.Done()
        }()
    }

    wg.Wait()
    fmt.Println(meter.Value())
}
```

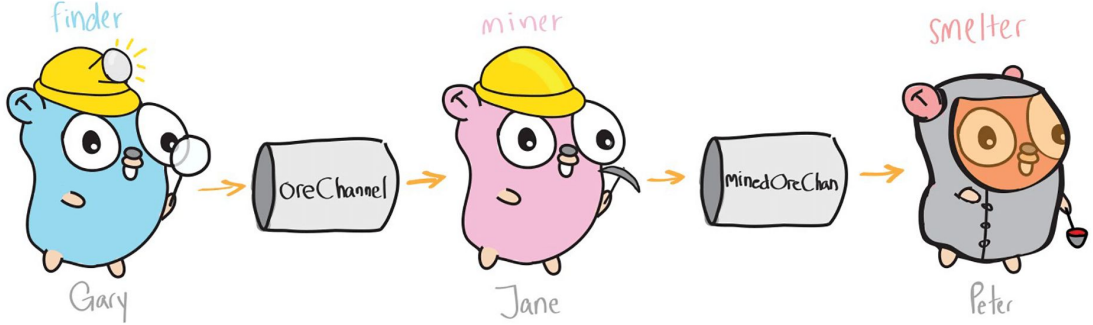
```
→  go-concurrency go run -race main.go
==================
WARNING: DATA RACE
Read at 0x00c0000ba028 by goroutine 8:
  go-concurrency/goroutines.(*counter).Add()
      /Users/wicak/Documents/guild/go-concurrency/goroutines/mutex.go:77 +0x49
  go-concurrency/goroutines.RaceCondition.func1()
      /Users/wicak/Documents/guild/go-concurrency/goroutines/mutex.go:95 +0x44

Previous write at 0x00c0000ba028 by goroutine 7:
  go-concurrency/goroutines.(*counter).Add()
      /Users/wicak/Documents/guild/go-concurrency/goroutines/mutex.go:77 +0x5b
  go-concurrency/goroutines.RaceCondition.func1()
      /Users/wicak/Documents/guild/go-concurrency/goroutines/mutex.go:95 +0x44

Goroutine 8 (running) created at:
  go-concurrency/goroutines.RaceCondition()
      /Users/wicak/Documents/guild/go-concurrency/goroutines/mutex.go:93 +0x87
  main.main()
      /Users/wicak/Documents/guild/go-concurrency/main.go:19 +0x24

Goroutine 7 (finished) created at:
  go-concurrency/goroutines.RaceCondition()
      /Users/wicak/Documents/guild/go-concurrency/goroutines/mutex.go:93 +0x87
  main.main()
      /Users/wicak/Documents/guild/go-concurrency/main.go:19 +0x24
==================
904916
Found 1 data race(s)
exit status 66
```

```
→  go-concurrency go run -race main.go
==================
WARNING: DATA RACE
Read at 0x00c0000c4028 by goroutine 17:
  go-concurrency/goroutines.(*counter).Add()
      /Users/wicak/Documents/guild/go-concurrency/goroutines/mutex.go:77 +0x49
  go-concurrency/goroutines.RaceCondition.func1()
      /Users/wicak/Documents/guild/go-concurrency/goroutines/mutex.go:95 +0x44

Previous write at 0x00c0000c4028 by goroutine 7:
  go-concurrency/goroutines.(*counter).Add()
      /Users/wicak/Documents/guild/go-concurrency/goroutines/mutex.go:77 +0x5b
  go-concurrency/goroutines.RaceCondition.func1()
      /Users/wicak/Documents/guild/go-concurrency/goroutines/mutex.go:95 +0x44

Goroutine 17 (running) created at:
  go-concurrency/goroutines.RaceCondition()
      /Users/wicak/Documents/guild/go-concurrency/goroutines/mutex.go:93 +0x87
  main.main()
      /Users/wicak/Documents/guild/go-concurrency/main.go:19 +0x24

Goroutine 7 (finished) created at:
  go-concurrency/goroutines.RaceCondition()
      /Users/wicak/Documents/guild/go-concurrency/goroutines/mutex.go:93 +0x87
  main.main()
      /Users/wicak/Documents/guild/go-concurrency/main.go:19 +0x24
==================
867272
Found 1 data race(s)
exit status 66
```

```go
// A Mutex provides a concurrent-safe way to express exclusive
// access to these shared resources.
func MutexBasic() {
    var mu sync.Mutex
    sharedResource := 0

    // Simulate some work
    for i := 0; i < 5; i++ {
        // Acquire the lock to access the shared resource
        mu.Lock()

        // Modify the shared resource
        sharedResource++
        fmt.Printf("Iteration %d: sharedResource = %d\n", i+1, sharedResource)

        // Release the lock
        mu.Unlock()
    }

    fmt.Printf("All iterations have finished. Final sharedResource = %d\n", sharedResource)
}
```

```go
type counter struct {
    sync.Mutex
    val int
}

func (c *counter) Add(int) {
    c.Lock()
    c.val++
    c.Unlock()
}

func (c *counter) Value() int {
    return c.val
}

func RaceConditionFix() {
    runtime.GOMAXPROCS(2)

    var wg sync.WaitGroup
    var meter counter

    for i := 0; i < 1000; i++ {
        wg.Add(1)

        go func() {
            for j := 0; j < 1000; j++ {
                meter.Add(1)
            }

            wg.Done()
        }()
    }

    wg.Wait()
    fmt.Println(meter.Value())
}
```

Channels & Select



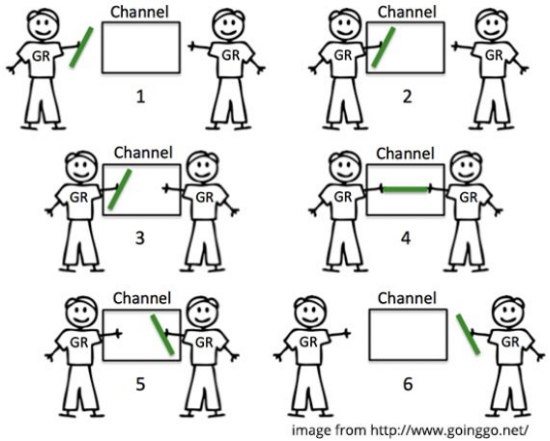
```go
func BidirectionalChannel() {
    var dataStream chan interface{}
    dataStream = make(chan interface{})
    dataInt := make(chan int16)

    go func() {
        dataStream <- "Hello channels"
        dataInt <- 1
    }()

    fmt.Println(<-dataStream)
    fmt.Println(<-dataInt)
}
```

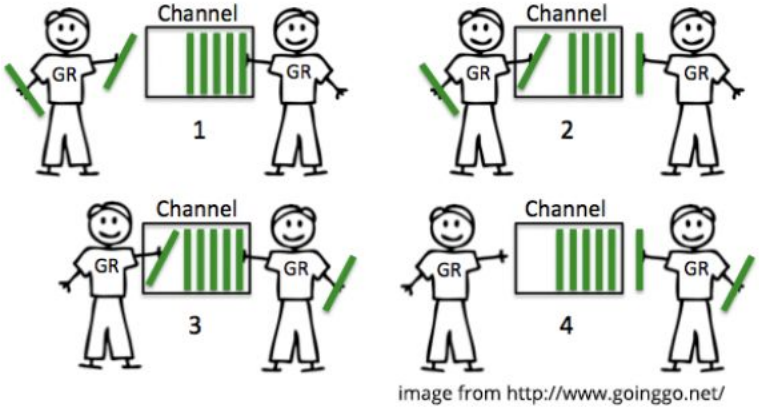
```go
func BasicUnindirectionalSendOnly() {
    var dataStream chan<- interface{}
    dataStream = make(chan<- interface{})
}
```

```go
func BasicUnindirectionalReceiveOrReadOnly() {
    var dataStream <-chan interface{}
    dataStream = make(<-chan interface{})
}
```

```go
func Channel() {
    stringStream := make(chan string)
    go func() {
    }()

    fmt.Println(<-stringStream)
}
```

Channel(Unbuffered) vs Buffered





```go
func BufferBasic() {
    // Create a buffered channel with a capacity of 3
    ch := make(chan int, 3)

    // Start a goroutine to send data to the channel
    go func() {
        for i := 1; i <= 5; i++ {
            fmt.Printf("Sending %d to the channel\n", i)
            ch <- i
        }
        // Close the channel when done sending
        close(ch)
    }()

    // Allow some time for the goroutine to start
    time.Sleep(time.Second)

    // Receive data from the channel
    for num := range ch {
        fmt.Printf("Received %d from the channel\n", num)
    }

    // Channel is closed, and all values have been received
    fmt.Println("Done receiving")
}
```

```go
func SelectBasic() {
    var c1, c2 <-chan interface{}
    var c3 chan<- interface{}

    select {
    case <-c1:
        //
    case <-c2:
        //
    case c3 <- struct{}{}:
        fmt.Println("haha")
    default:
        //
    }
}
```

```go
func SelectWithChannelClosed() {
    c1 := make(chan interface{})
    close(c1)
    c2 := make(chan interface{})
    close(c2)

    var c1Count, c2Count int
    for i := 10; i >= 0; i-- {
        fmt.Println(i)
        select {
        // c1 and c2 have equal chance of being selected. Go runtime will
        // perform pseudo-random uniform selection over the set of case statements
        case <-c1:
            fmt.Println("Signal received from c1")
            fmt.Println(<-c1)
            c1Count++
        case <-c2:
            fmt.Println("Signal received from c2")
            fmt.Println(<-c2)
            c2Count++
        }
    }

    fmt.Printf("c1Count: %d\nc2Count: %d\n", c1Count, c2Count)
}
```

```go
func SelectWithGoRoutine() {
    done := make(chan interface{})
    go func() {
        time.Sleep(5 * time.Second)
        close(done)
    }()

    workCounter := 0

loop:
    for {
        select {
        case <-done: // receive signal channel closed, let's break the loop
            break loop
        default:
        }

        // Simulate work
        workCounter++
        time.Sleep(1 * time.Second)
    }

    fmt.Printf("Achieved %v cycles of work before signalled to stop.\n", workCounter)
}

```

What next?

- Concurrency pattern
- Avoiding Data Races
- Best Practices and Pitfalls
- Behind Goroutines and go Runtime
- Other Go feature(context, error handling, etc)
- …

THANK YOU !!!