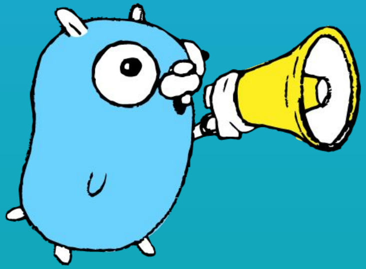
SHEFFIELD, UK - 7th FEB 2019

# CONCURRENCY VS PARALLELISM

**JACK ADAMS**

U Account

jack.adams@uaccount.uk



ABOUT ME 01

CONCURRENCY vs PARALLELISM 02

THINK CONCURRENCY 03

EXAMPLES 04

SUMMARY 05

FURTHER READING 06

## ABOUT ME

- Developer for ~10 years
- Using Go sporadically for ~6 months

- Get in touch: jack.adams@uaccount.uk

## CONCURRENCY VS PARALLELISM

- Concurrency is about dealing with lots of things at once
- Parallelism is about DOING lots of things at once

- Concurrency should be by design, it doesn’t require parallelism

**CONCURRENCY !== PARALLELISM**

## THINK CONCURRENCY

- Design your programs to be concurrent, do not assume parallelism

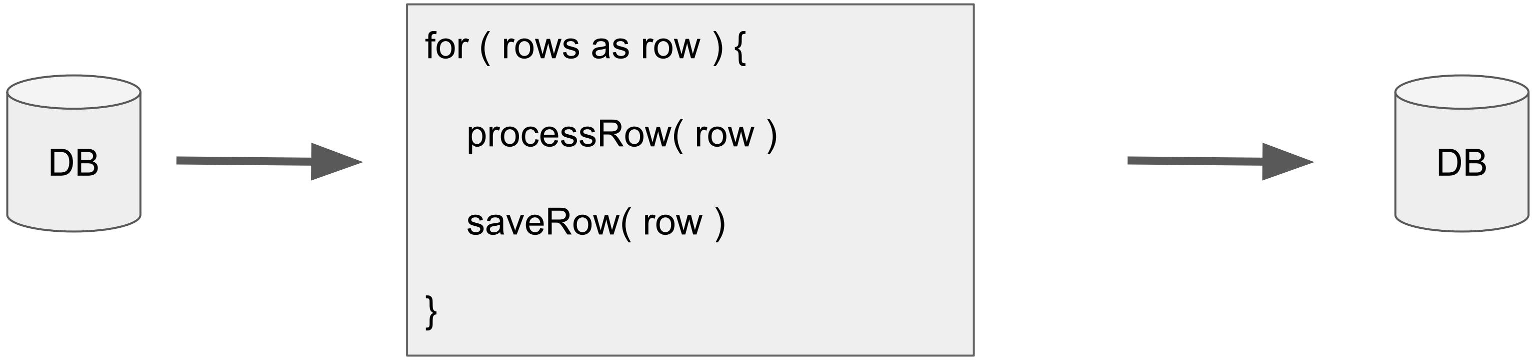
- Split tasks into smaller and smaller chunks
- Add workers to the chunks (for concurrency), not just the tasks (for parallelism)

## BY EXAMPLE - GOPHERS

- One worker moving books to the incinerator.

With thanks to Rob Pike

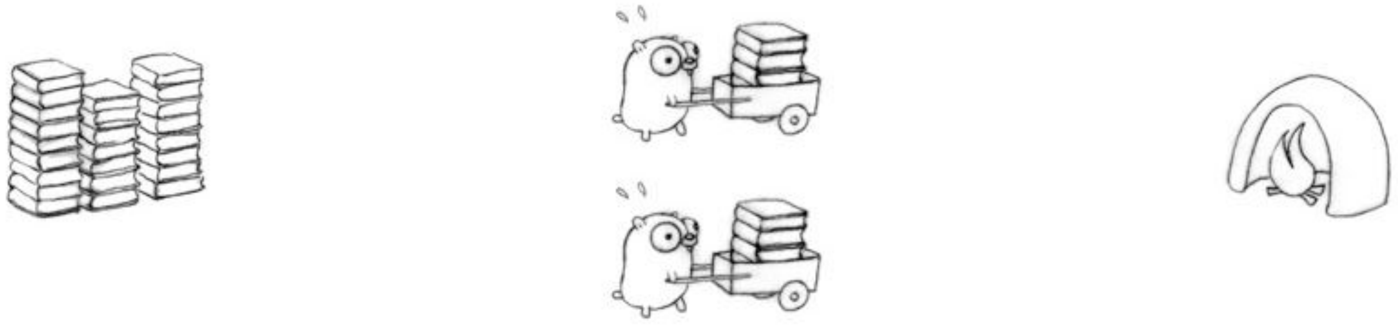
## BY EXAMPLE - NON-CONCURRENT



```
for ( rows as row ) {
    processRow( row )
    saveRow( row )
}
```

- We retrieve N rows from a database
- We loop through each row, one at a time, and perform some task and save the result back to the database

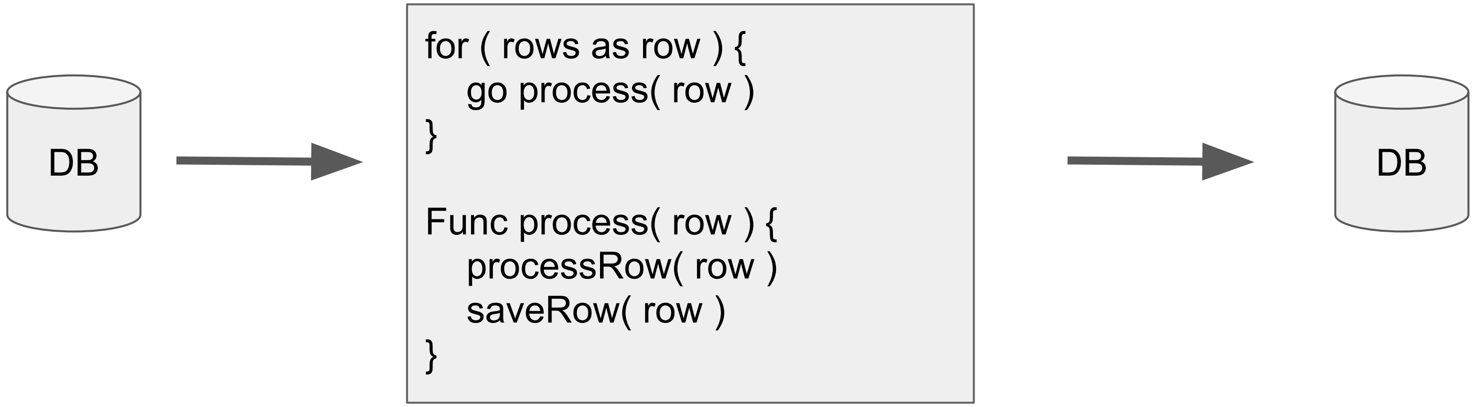
## BY EXAMPLE - GOPHERS - PARALLEL?



- Two gophers, undoubtedly faster
- BUT can get stuck at the pile of books or at the incinerator!

With thanks to Rob Pike

## BY EXAMPLE - PARALLELISM

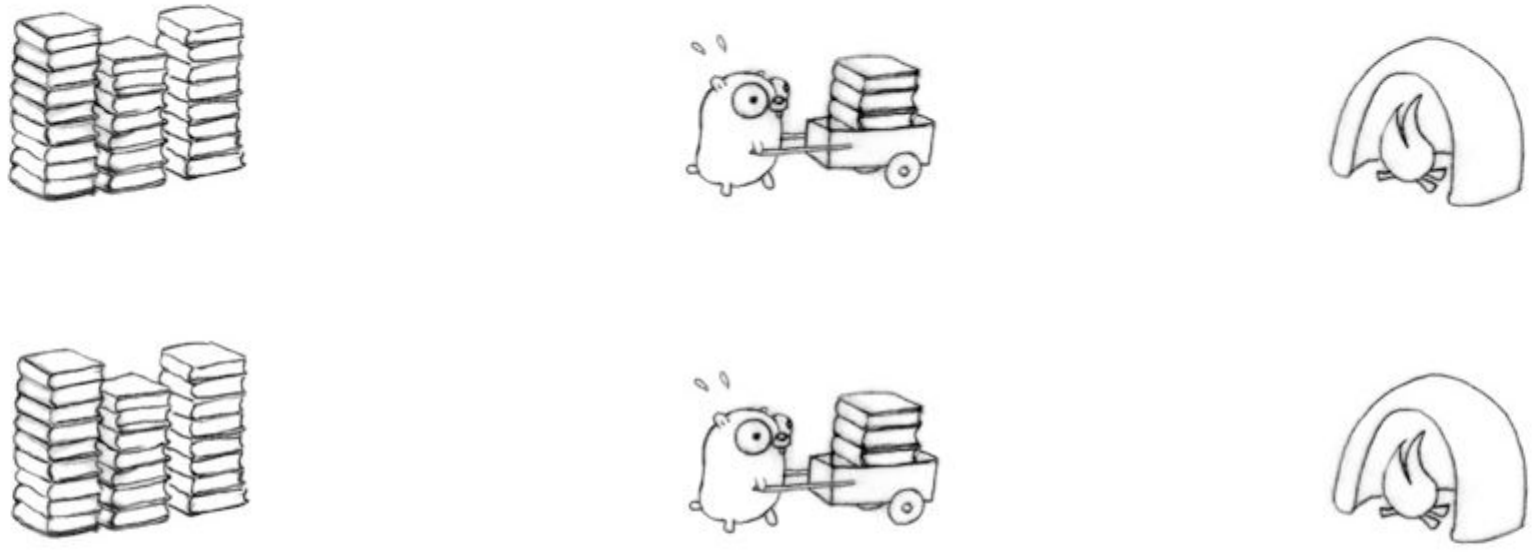


```
for ( rows as row ) {
    go process( row )
}

Func process( row ) {
    processRow( row )
    saveRow( row )
}
```

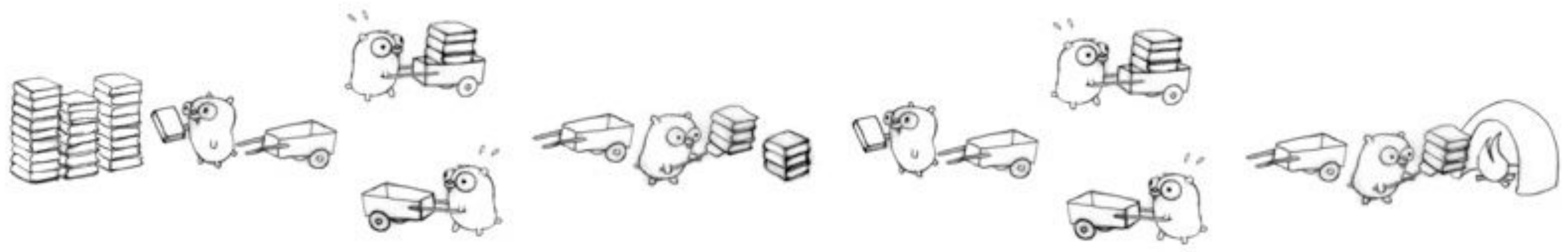
- We retrieve N rows from a database
- We loop through each row and start a sub-routine to process and save the data in parallel

## BY EXAMPLE - GOPHERS - CONCURRENT/PARALLEL?



With thanks to Rob Pike

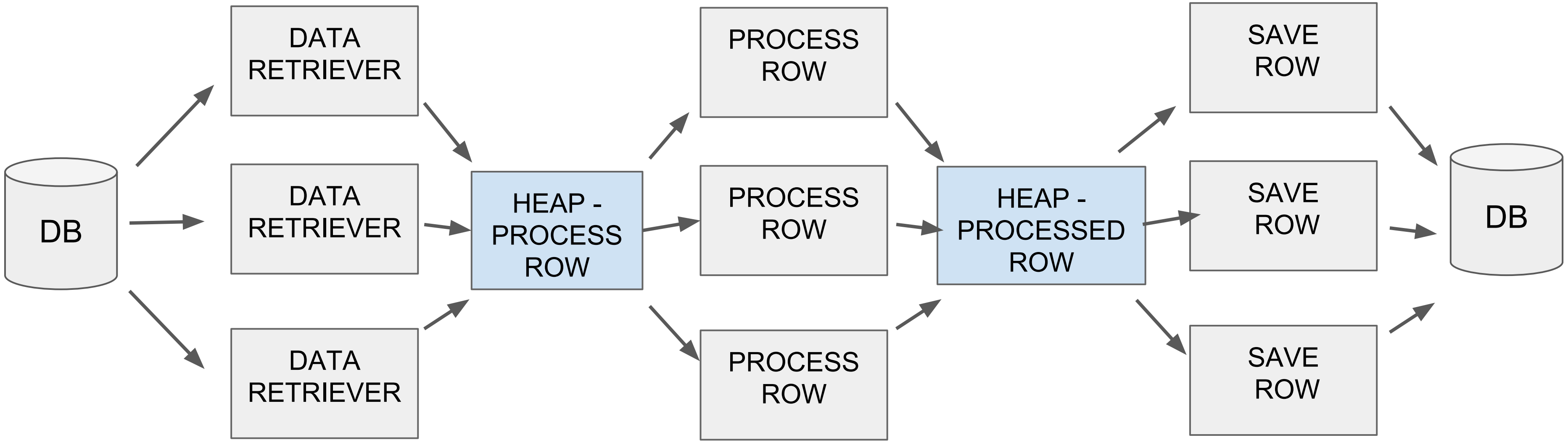
## BY EXAMPLE - GOPHERS - CONCURRENCY!



- Job is broken in to many smaller tasks, allowing more gophers to work at once.
- None of these gophers are waiting on each other

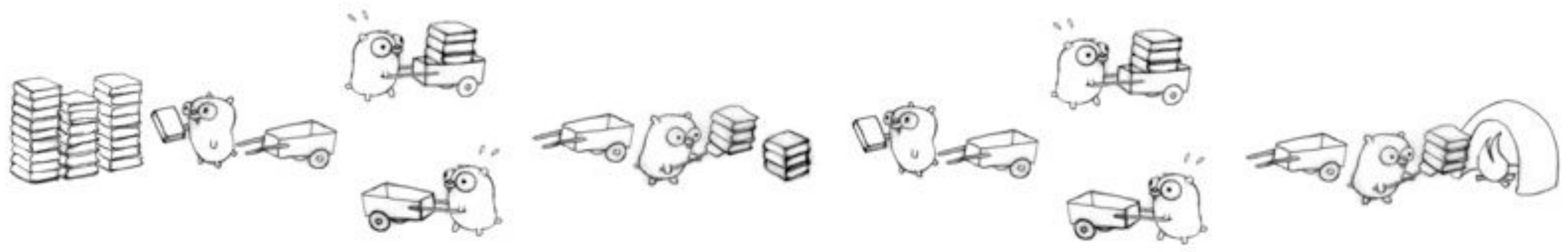
With thanks to Rob Pike

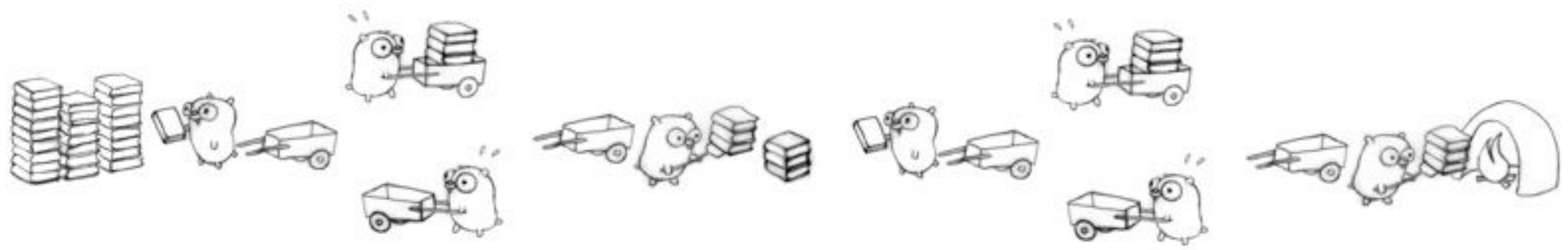
# BY EXAMPLE - CONCURRENCY



- GET, PROCESS and SAVE can all run concurrently
- Bottlenecks are now an infrastructure issue, the program is scalable by design

# EXAMPLE – GOPHER CONCURRENCY & PARALLELISM





## CONCURRENCY IN GO

### GOROUTINES

```
go func() {
     time.Sleep(deltaT)
      fmt.Println(“DONE!”)
}()

// don’t run this!
for {
    go func() {
        fmt.Println(“HELLO!”)
    }
}
```

NOT THREADS!

But think of them like much cheaper threads

Like using & in shell - go do this, but let me carry on

## CONCURRENCY IN GO

### CHANNELS

```
timerChan := make(chan time.Time)

go func() {
    time.Sleep(deltaT)
    timerChan <- time.Now()
}()

go func() {
    Time := <-timerChan
}
```

Communication between goroutines

Allow data to be passed between workers

# CONCURRENCY IN GO

## SELECT

```
for {
    select {

    case v := <-ch1:
        fmt.Println("channel 1 sends", v)

    case v := <-ch2:
        fmt.Println("channel 2 sends", v)

    default: // optional
        fmt.Println("neither ready")
    }
}
```

Select an appropriate channel based on its ability to communicate

Think ‘load balancer’ when coupled with a heap of ‘channels’?

## SUMMARY

LET THE SYSTEM HANDLE THE PARALLELISM

THINK SMALL

GOROUTINES ARE CHEAP, MAKE MORE OF THEM

## FURTHER READING

1. Rob Pike - Concurrency Not Parallelism
   https://www.youtube.com/watch?v=cN_DpYBzKso
2. Google - Go Concurrency Patterns
   https://www.youtube.com/watch?v=f6kdp27TYZs

If you'd like to see a working version of the concept in GO give me an email at: jack.adams@uaccount.uk