‘API DAYS

A comparison of REST APIs

WHO IS PREET?

- Web stack eng. since '99, research in AI, beginner in Go
- Scalability architecture, ScalableML, algorithm design.
- @singhdaddy
- preet@randomcanary.com, github.com/randomcanary
- I freelance, mostly with qwinix.io

MOTIVATION

- REST APIs: building blocks of microservice arch, blah blah, REST APIs are important blah blah
- Potentially identify areas / libs for improvement
- Basically, I wanted to spin up REST APIs in a few different tech stacks, just play around and *observe*.

SPRING BOOT

- First choice for REST APIs if you're a Java engineer
- Full-fledged framework, all the bells and whistles, plenty of magic - autobinding ORM, implicit routes, etc.
- Hardly an apples-to-apples comparison with Go+net/http, more of a David-and Goliath comparison.
- However, similarities: both compiled langs, both single binaries in deployment.

TORNADO

- Scalable, nonblocking Python app engine
- Has a rep for high performance
- Used by **Quora, FriendFeed, hipmunk, bit.ly**

A RATHER SUBJECTIVE CODETIME ANALYSIS

Lines of code	nonDB ver.	DB ver.
Go	36	91
Java	47	70
Python	25	38

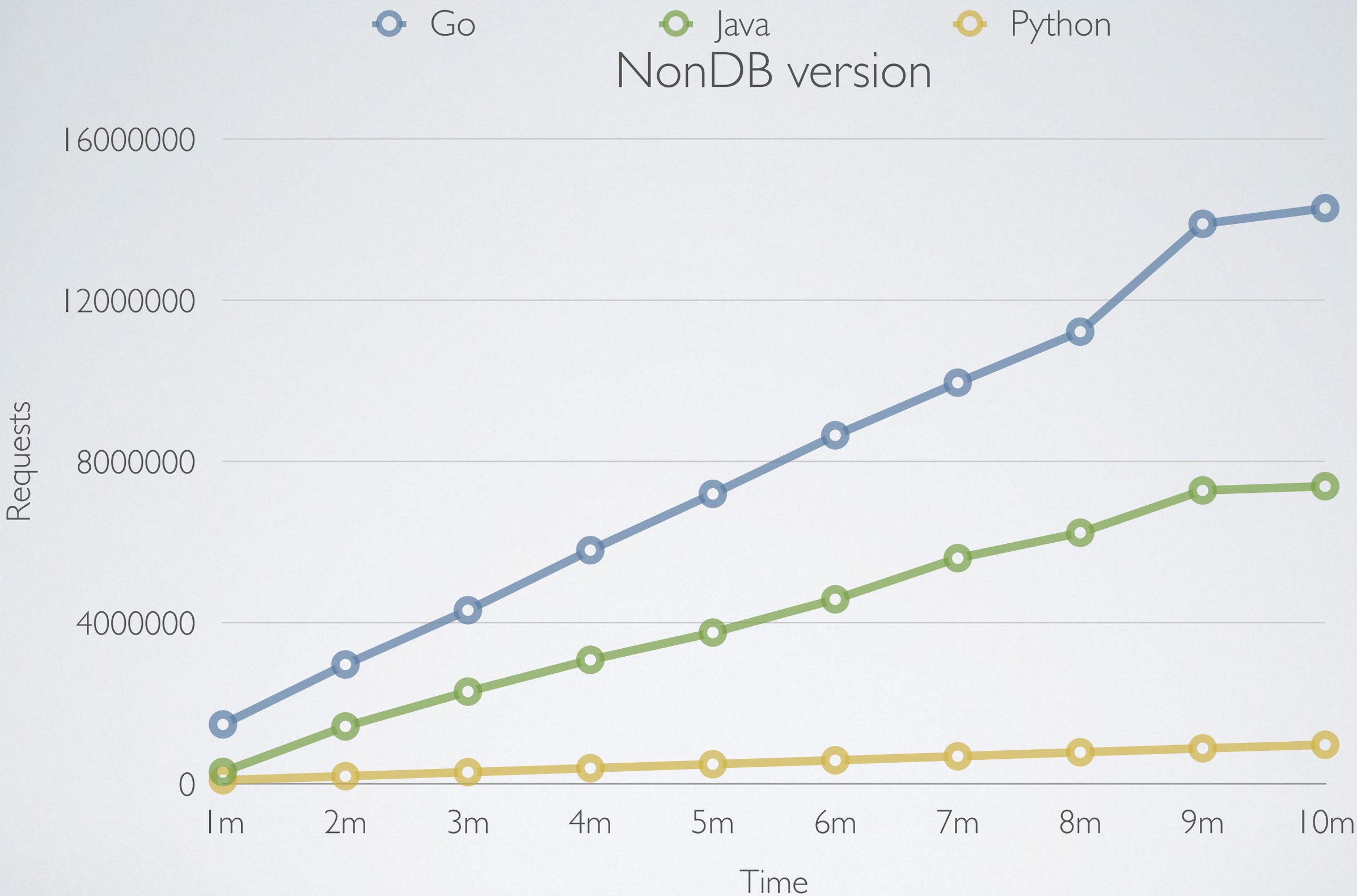
SETUP

- Go ver. 1.4.2, only net/http
- Python 2.7.11, Tornado ver. 4.3
- Java 1.8, Spring Boot ver. 1.3.2
- Ran on my Macbook Pro 2.4 GHz i5, 8 GB RAM
- Used ***wrk*** for benchmarking
- For code and benchmarking params, check github.com/randomcanary
- Ran two request types - one trivial, one nontrivial.

TESTING I: BASIC QUERY

- Motivation: mimic *constant-time op* for REST APIs
- generates a random number, returns as json
- Idea: To test **basics** of REST API functionality:
  - *Accepting requests*
  - *Routing*
  - *(serving json responses)*


```
{
    "RandomInt": 87
}
```


Go
Java
Python
NonDB version
16000000
12000000
8000000
4000000
0
Requests
1m
2m
3m
4m
5m
6m
7m
8m
9m
10m
Time

NONDB FIGURES

- Java : **7,390,908** requests in 10 minutes
- Python : **976,466** requests in 10 minutes
- Go : **14,290,975** requests in 10 minutes

GO IS INTRINSICALLY FASTER

NET/HTTP LIB IS REALLY AS GOOD AS WE THOUGHT IT WAS

TESTING II

- We try testing with a non-trivial request:
  - Should access a DB (MySQL, ***world*** db, City names)
  - Reasonably large number of rows (~4000)
  - Perform some >= Linear Op on DB data (sorting, n log(n) Average runtime)
  - Return result as JSON

- world db (https://dev.mysql.com/doc/index-other.html)
- City Names (~4000 rows)
- Sort rows, return as large JSON object


```
{
  "Citynames": [
    {
      "CityName": "A Coruña (La Coruña)"
    },
    {
      "CityName": "Aachen"
    },
    {
      "CityName": "Aalborg"
    },
    {
      "CityName": "Aba"
    },
    {
      "CityName": "Abadan"
    },
    {
      "CityName": "Abaetetuba"
    },
    {
      "CityName": "Abakan"
    },
    {
      "CityName": "Abbotsford"
    },
    {
      "CityName": "Abeokuta"
    },
    {
      "CityName": "Aberdeen"
    },
```

A QUICK LOOK AT MY CODE

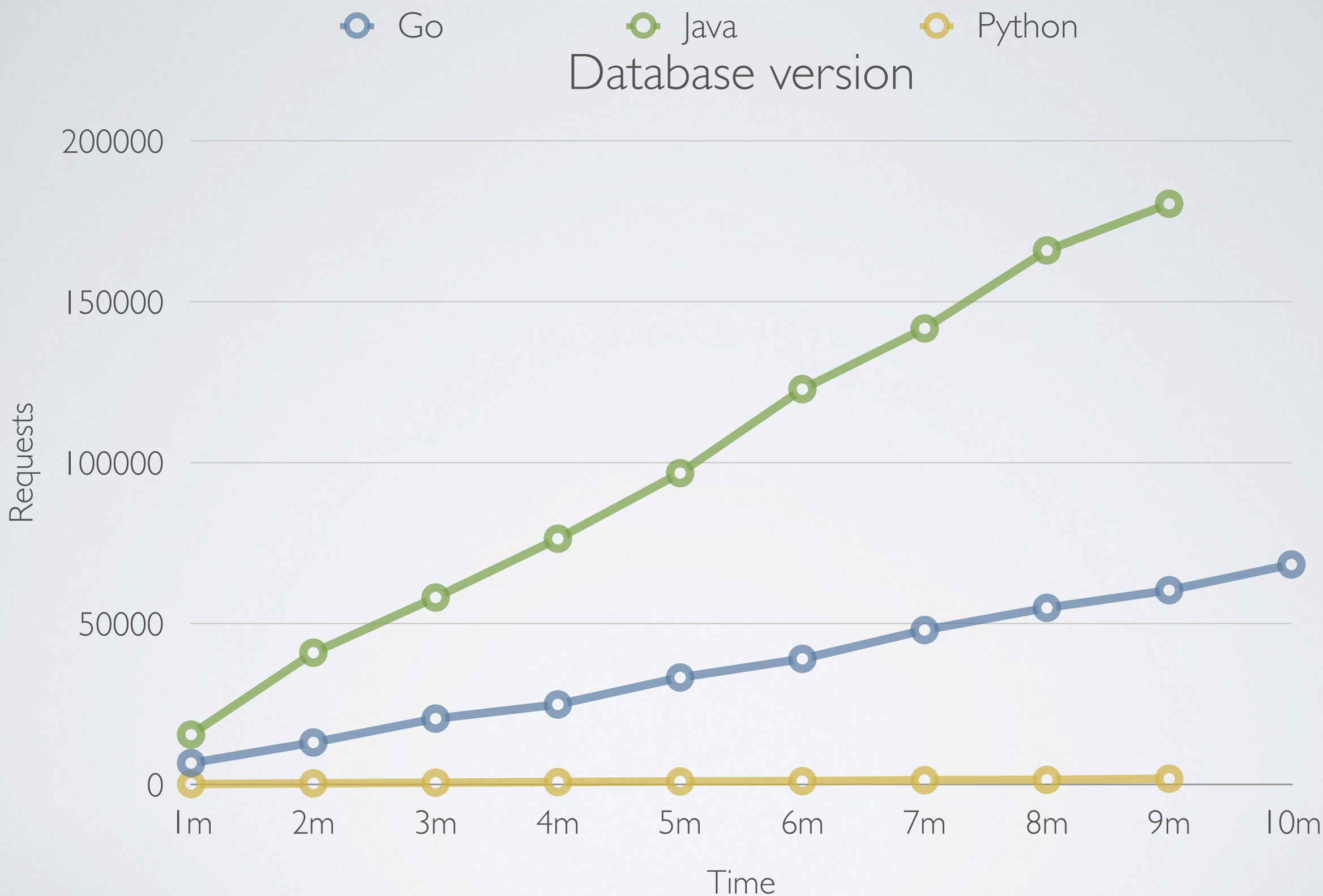
```go
import (
    "database/sql"
    "encoding/json"
    "fmt"
    _ "github.com/go-sql-driver/mysql"
    "log"
    "net/http"
    "sort"
)

var db *sql.DB
```

A QUICK LOOK AT MY CODE

```go
func main() {
    db, _ = sql.Open("mysql",
        "root:@tcp(127.0.0.1:3306)/world")
    db.SetMaxIdleConns(25)
    db.SetMaxOpenConns(25)
    defer db.Close()

    http.HandleFunc("/", pageHandler)
    http.ListenAndServe("localhost:8080", nil)
}
```

- {db.setMaxIdleConns : “Please don’t keep more than n connections when they’re not longer in use.”}
- {db.setMaxOpenConns : “Allow only max (n) connections to the db”}
- speed vs. footprint

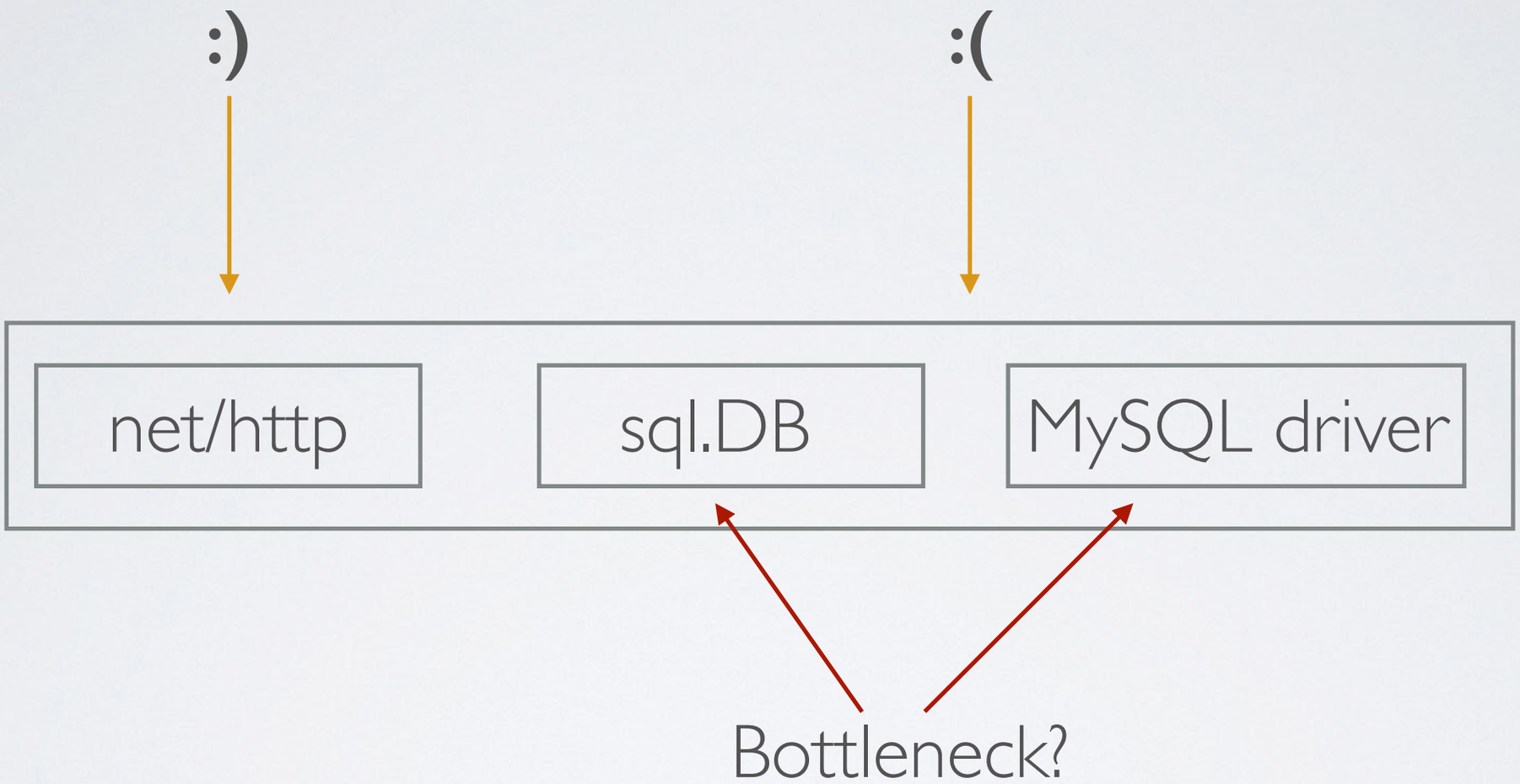
```go
func getPayloadComponents() []City {
    var (
        Name   string
        cities Cities
    )
    rows, err := db.Query("select Name from City")
    if err != nil {
        log.Fatal(err)
    }

    defer rows.Close()

    for rows.Next() {
        err := rows.Scan(&Name)
        someCity := City{Name}

        cities = append(cities, someCity)
        if err != nil {
            log.Fatal(err)
        }
        //return id, identifier, call_sign
    }
    err = rows.Err()
    if err != nil {
        log.Fatal(err)
    }
    sort.Sort(cities)
    return cities
}
```


Go
Java
Python
Database version
200000
150000
100000
50000
0
Requests
1m
2m
3m
4m
5m
6m
7m
8m
9m
10m
Time

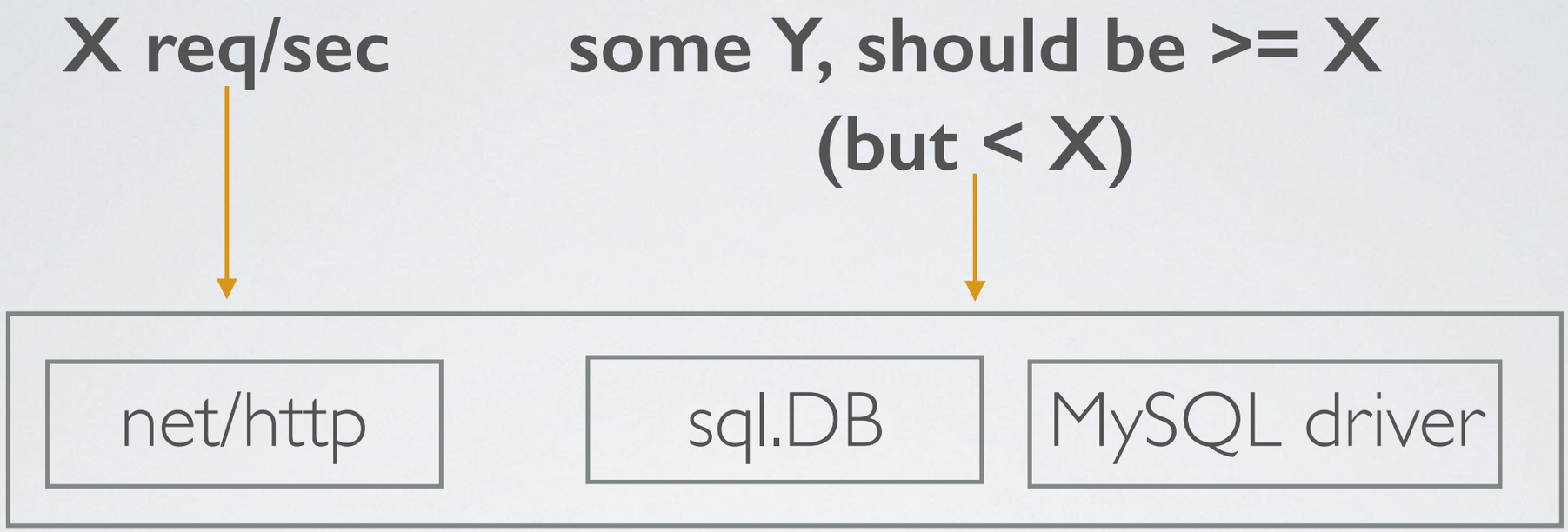
WAIT, SPRINGBOOT'S FASTER?

- I was expecting a David-and Goliath story, a bit disappointed.
- Is it because I used MySQL?
- Is is sql.DB? (conn pooling func)
- Is it because of the DB driver?

CONCLUSIONS



NEXT STEPS : QUESTIONS



- is Y (deliberately) variable? If sql.DB detects Y= say, X/2, does it signal net/http to slow down req. intake?

- If the idle conn pool works as a cache-style speed-up mechanism for Y, (how) can we dynamically compute **'good'** values for pool size?

NEXT STEPS II

- Look hard at the source code for the 3 pieces. Srsly. This is actually feasible in Go.
- MySQL / MySQL db driver is the issue? Test on diff databases.
- **Questions? Code to be uploaded on github.**