Why It Matters

What Is Go

- Open Source
- Concurrent
- Garbage Collected
- Efficient
- Scalable
- Simple
- Native binary
- Boring (for some people….)

Go Journey

Born — **2007**

Open Source — **2009**

Go 1.0 — **2012**

Go 1.9 — **2017**

Robert Griesemer
Hotspot JVM

Rob Pike
Plan9, UTF-8

Kenneth Thompson
Unix, B, Belle, UTF-8

Why GO Is GO

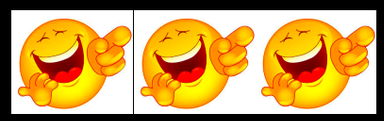
September 25, 2007

From: Rob 'Commander' Pike <r@google.com>
Date: Tue, Sep 25, 2007 at 3:12 PM
To: Robert Griesemer <gri@google.com>
Cc: ken@google.com

i had a couple of thoughts on the drive home.

1. name.

'go'. you can invent reasons for this name but it has nice properties. it's short, easy to type. tools: goc, gol, goa. if there's an interactive debugger/interpreter it could just be called 'go'. the suffix is .go.

When the three of us got started, it was pure research. The three of us got together and decided that we hated C++ 

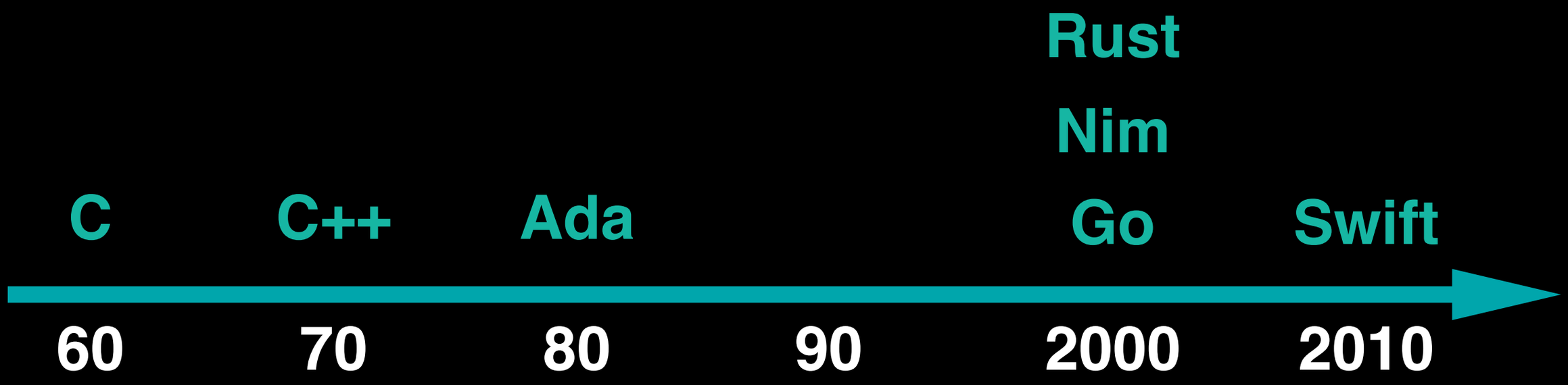
We started off with the idea that all three of us had to be talked into **every feature in the language**, so there was no **extraneous garbage** put into the language for any reason.

Ken Thompson

Go was designed by and for people who write, read, debug and maintain large software system. It’s purpose not research into programming language design but to make its designer’s programming lives better.

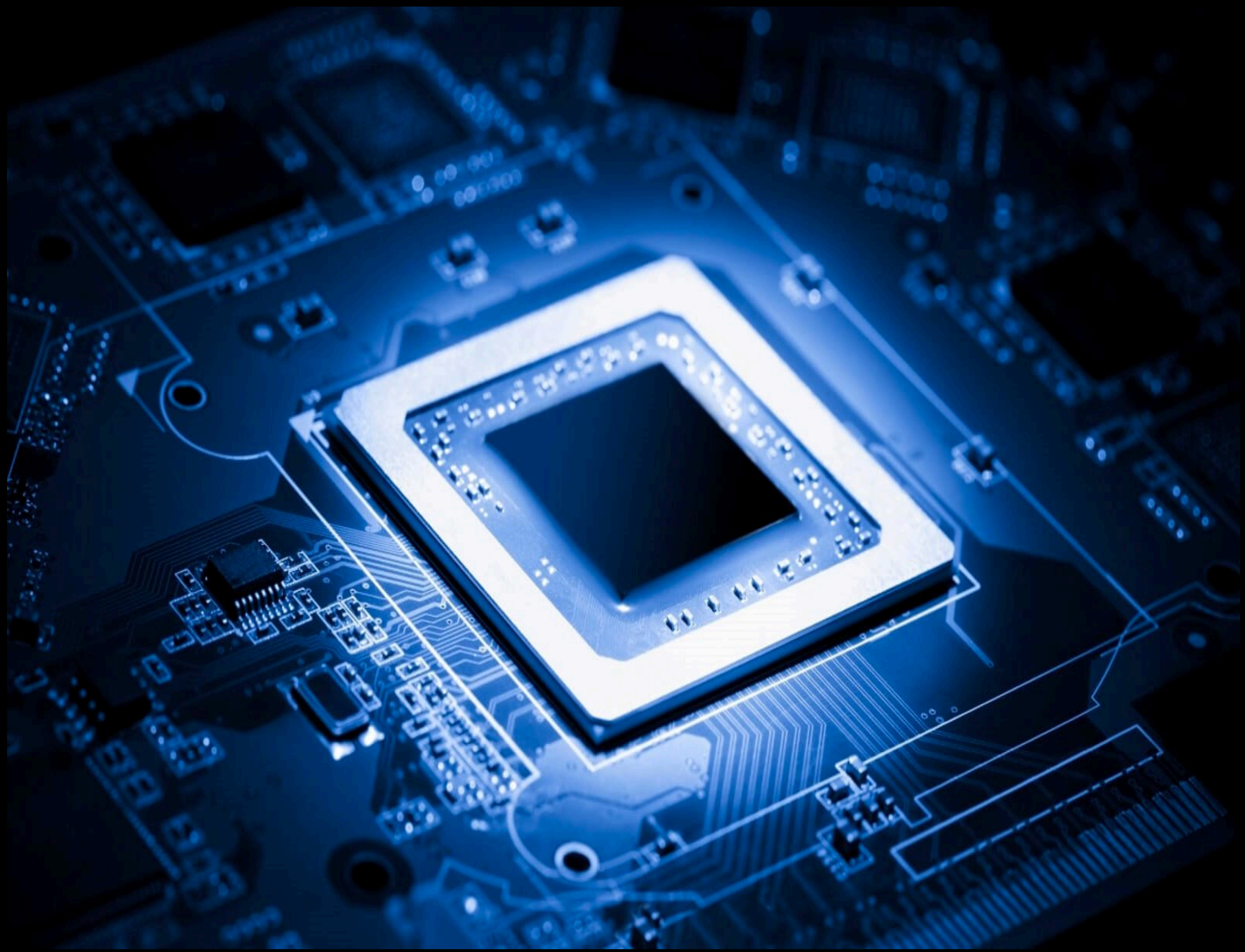
Rob Pike

System Programming Language Evolution

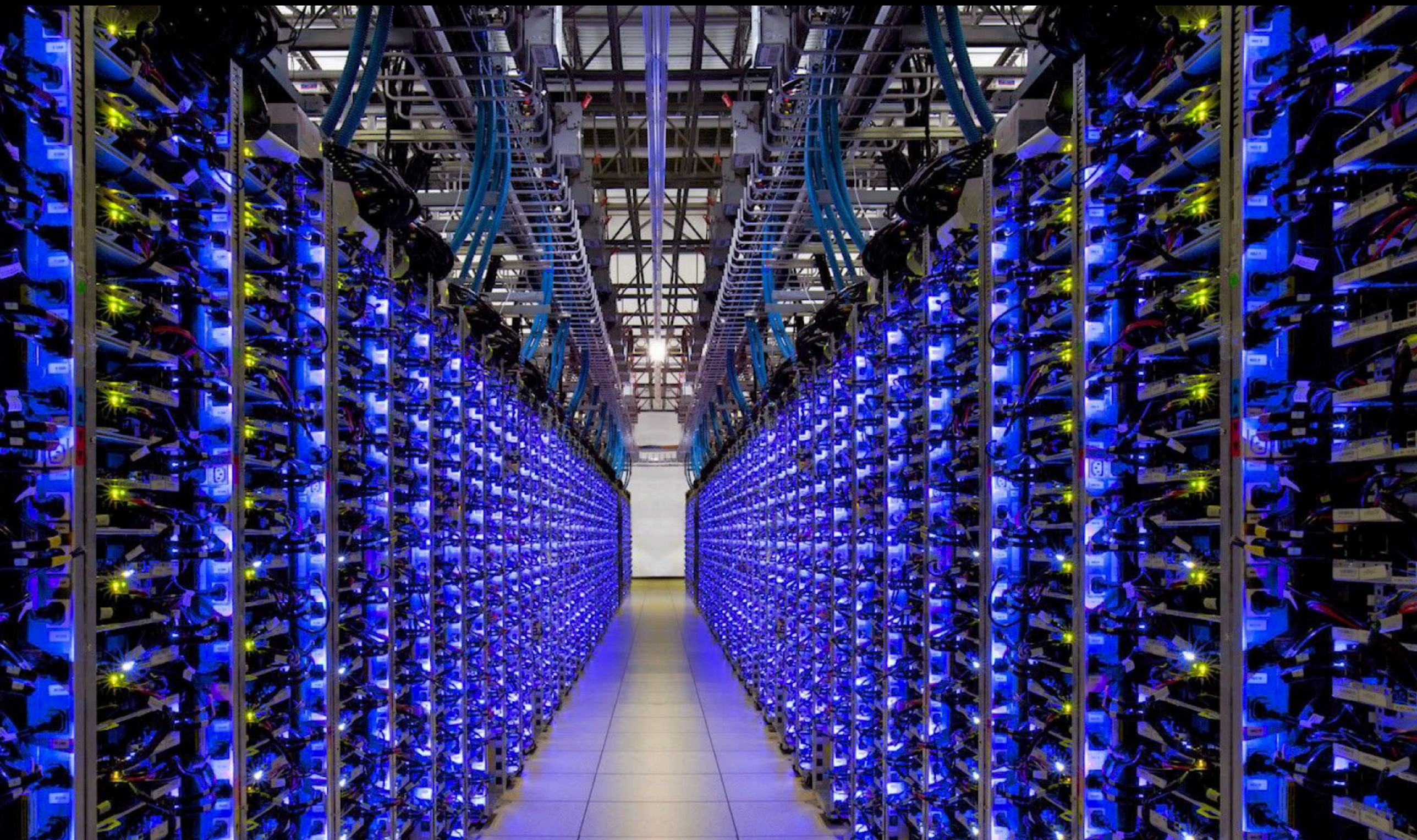


Evolution of Eco System

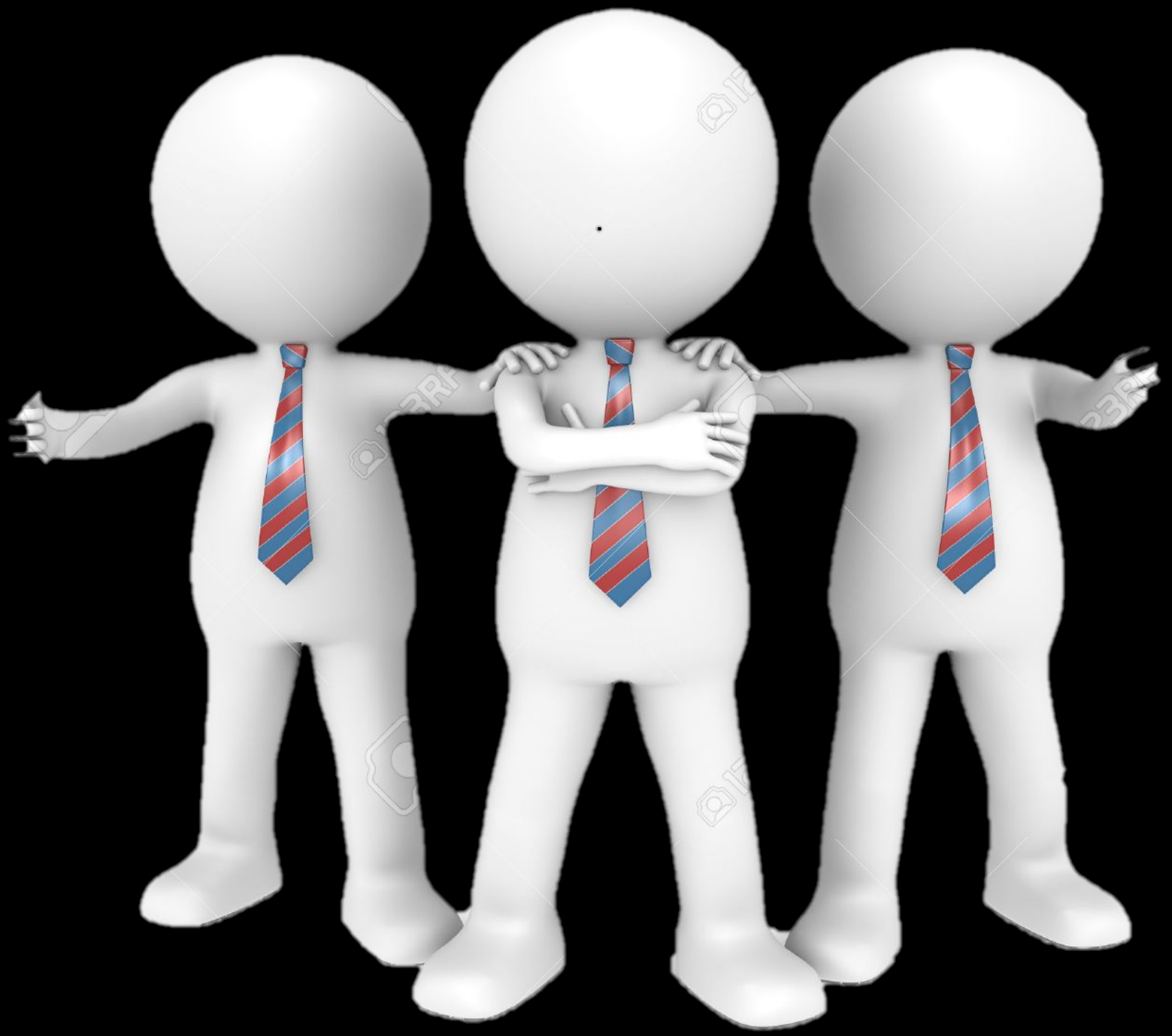
Small Machines



Big Clusters



Small Teams



Big Distributed Teams

Codebase Has Grown

Simple Is Better Than Complex

- **26 keywords**

- 26 keywords
- **One approach over many**

- 26 keywords
- One approach over many

- **struct over class**

Struct

```
1   package geekNight
2
3   type Attendee struct {
4       name         string
5       organization string
6   }
7
```

- 26 keywords
- One approach over many
- struct
- **functions over constructor**

```
using System;
namespace handsON
{
    public class Employee
    {
        public Employee(string firstName, string lastName )
        {

        }

        public Employee(string firstName, string lastName, string address)
        {

        }
    }
}

```

```
using System;
namespace handsON
{
    public class Employee
    {
        private Employee()
        {

        }

        public static Employee CreateEmployee(string firstName, string lastName )
        {
            return new Employee() { };
        }

        public static Employee CreateEmployeeWithAddress(string firstName, string lastName, string address)
        {
            return new Employee() { };
        }
    }
}
```

```go
package geekNight

type Attendee struct {
    name         string
    organization string
}

func NewAttendee(name string, organization string) Attendee {
    return Attendee{name: name, organization: organization}
}

```

- 26 keywords
- One approach over many
  - struct
  - functions over constructor
  - **multiple returns over out**

```go
package geekNight

type Attendee struct {
    name         string
    organization string
}

func NewAttendee(name string, organization string) Attendee {
    return Attendee{name: name, organization: organization}
}

func (attendee Attendee) Details() (string, string) {
    return attendee.name, attendee.organization

}

```

- 26 keywords
- One approach over many
  - struct
  - functions over constructor
- multiple returns over out
- **compositions over inheritance**

Why Composition ?

In one of the java meets up, James Gosling (Java's inventor) was the featured speaker. During the Q&A session, someone asked him: "If you could do Java over again, what would you change?" "I'd leave out classes," he replied. After the laughter died down, he explained that the real problem wasn't classes per se, but rather implementation inheritance (the extends relationship). Interface inheritance (the implements relationship) is preferable. You should avoid implementation inheritance whenever possible.

Pseudo Inheritance

- Embedded Properties

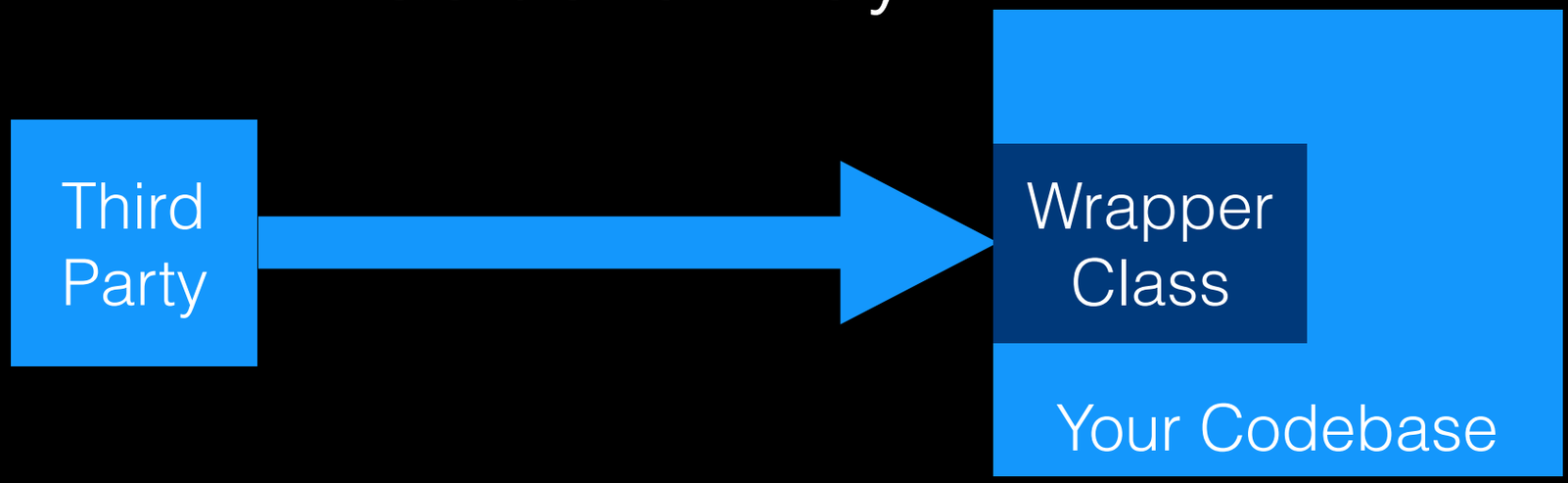
- 26 keywords
- One approach over many
  - Struct over class
  - Functions over constructor
- multiple returns over out
- compositions over inheritance
- **behaviour over type**

Identify type not by its DNA but by its behaviour

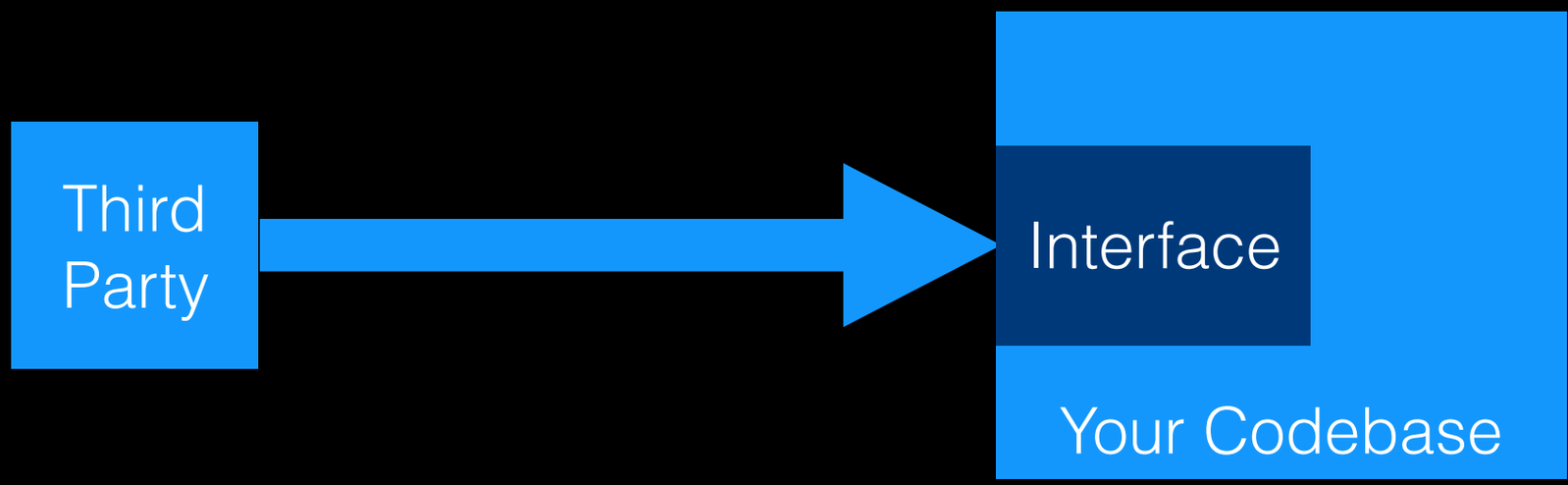
Interface

- No implements keyword
- Compile time “Duck Typing” support
- A contract to expose functionality ? Think again.
- A contact to consume functionality…….

Traditional Way



Go Way



- 26 keywords
- One approach over many
  - struct
  - functions over constructor
- multiple returns over out
- compositions over inheritance
- source control over package manager

Package Management

- Source control is your package manager
- Your commit hash is your version number
- Sub package access
- go get github.com/gin-gonic/gin
- go get gitHub.com/astaxie/beego/validation

Error Should Never

Pass Silently

- Forces you think about errors and deal with them
- Errors are normal, they are not exceptional
- Result is better, even if code is verbose

Errors

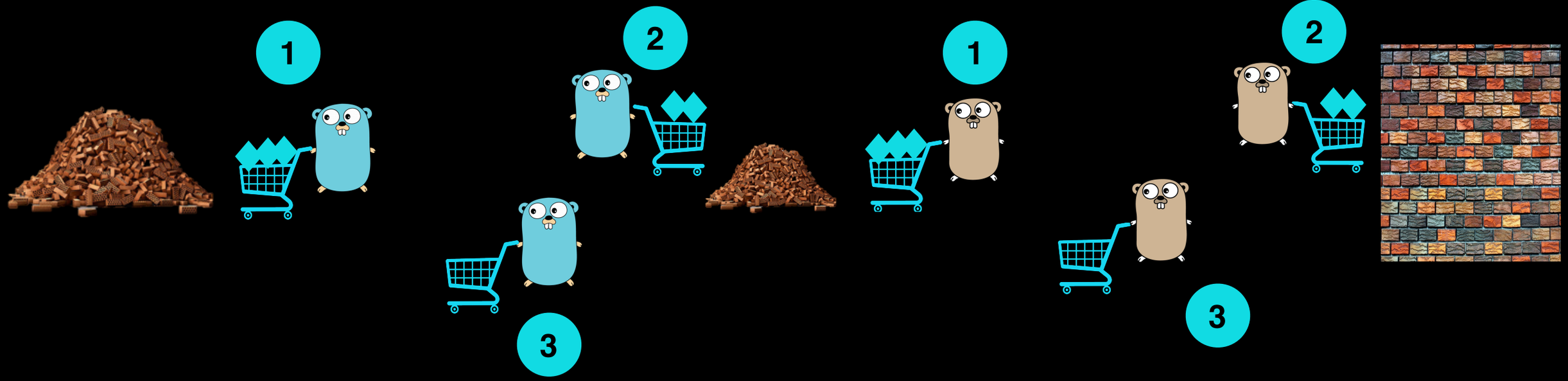
- Panic not exceptions
- Defer
- Recover

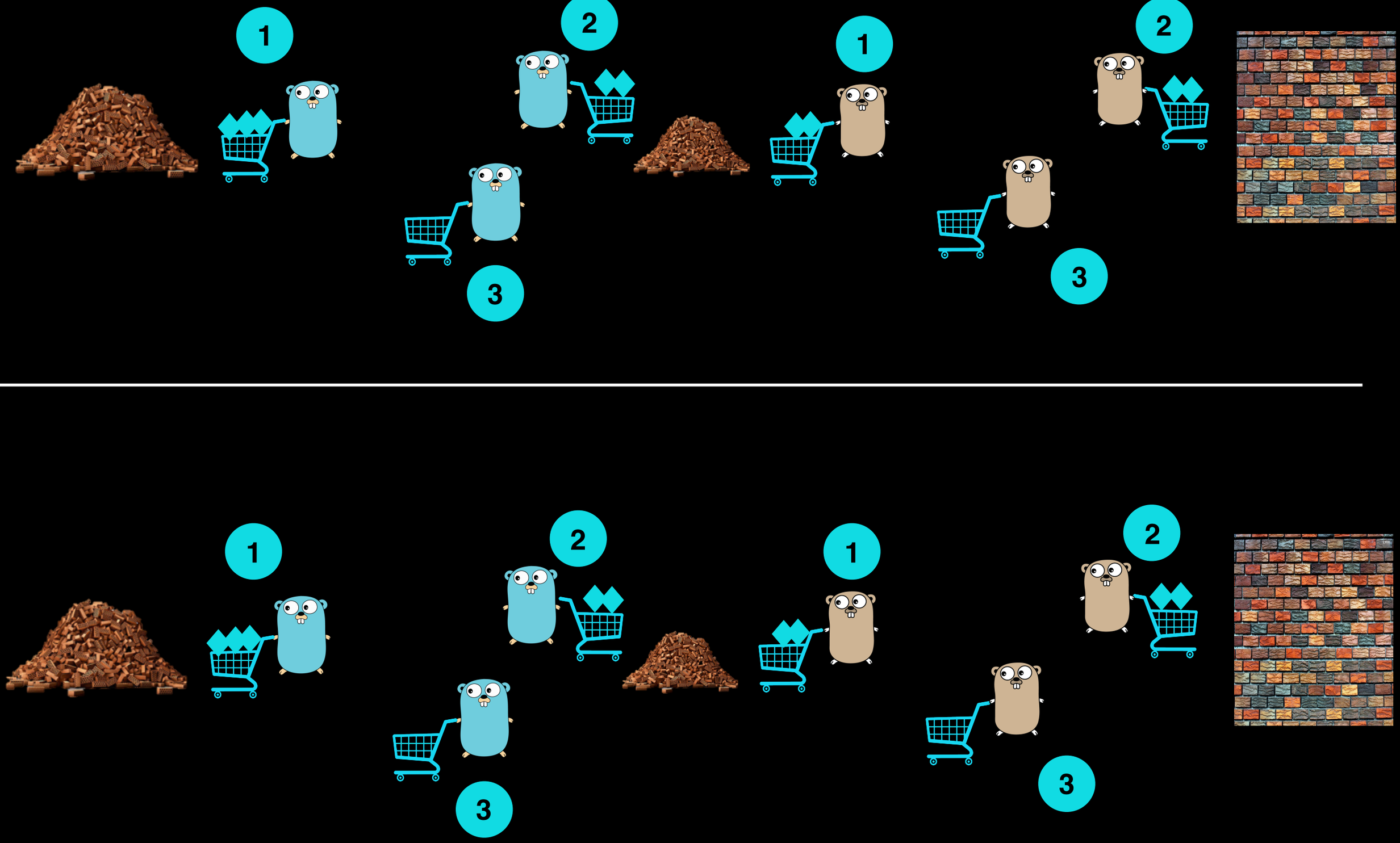
Concurrency

Concurrency Is Parallelism

Concurrency Is **Not**
Parallelism

Concurrency is about dealing with lots of things at once, while parallelism is about doing lots of things at once.


1
2
1
2
3
3


1
2
3
1
2
3

- Concurrency is about structure, parallelism is about execution
- Concurrency is not parallelism but it enable parallelism

Goroutine

- Goroutine are independently executing function in the same address space
- They are green filed threads, managed by runtime
- Set up and tear down is very fast as no new OS thread initiate
- Switching cost is very light as no need to save and restore all CPU register

Channels

Channels are type safe message queue that have the intelligence to control the behaviour of any goroutine attempting to receive or send information.

Simple is simple because someone has done a lot of complicated work to make if simple for you !!

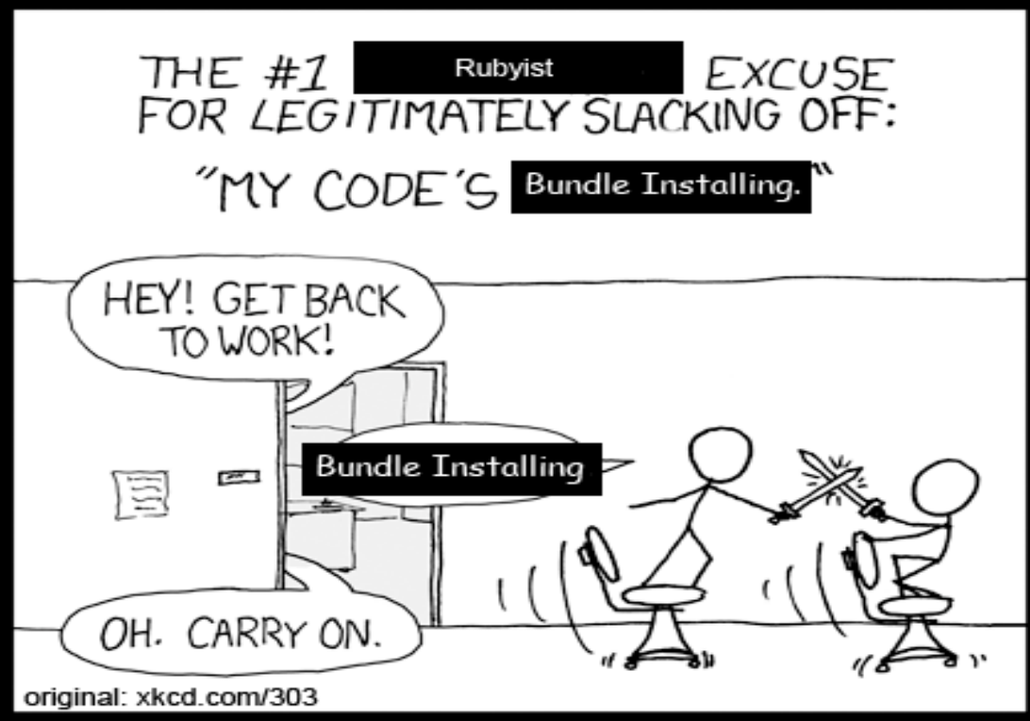
Rob Pike

Speed Thrills

But

Sometime It Doesn’t Kill

- Learning Curve
- Common Consensus
- Build
- Deployment
- Execution



GO Adoption

Cloud


openstack.


CLOUDFLARE

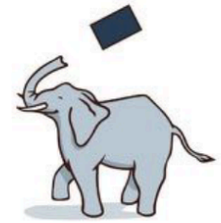

Dropbox


OPENSHIFT


IBM Bluemix


P

SAP




HUAWEI


DEIS


juju


HEROKU


docker


NATS


Google Cloud Platform
Microsoft
Azure


cybozu


七牛云


exoscale


HASHICORP


rackspace

Global Companies

SAP

IBM

Google

GE

Walmart

DELL EMC

intel

Hewlett Packard Enterprise

redhat

Disney

Microsoft

Technology


Moz://a
SPACEX
Couchbase
mongoDB
toggl
exoscale
Yandex
JET BRAINS
trisoft
Universal Mind
diffeo
HUAWEI
Bai du 百度
UCLOUD
专业云计算服务商
cybozu
DELL
http://gopherchina.org/en/#sponsors
七牛云
PingCAP
CoreOS
pilosa
Mendelics
CANONICAL
Atlassian
Dgraph
Dropbox
fastly

Gaming/ Media

Success Stories

Uber With GO

- **High developer productivity**. As per UBER experience, "*Go typically took just a few days for a C++, Java or Node.js developer to learn, and the code was easy to maintain. (Thanks to static typing, no more guessing and unpleasant surprises)*".

- **High performance in throughput and latency**. In Uber's main data centre a single service handled a peak load of 170k QPS with 40 machines running at 35% CPU usage on NYE with response time < 50 ms at the 99th percentile.

- **Super reliable**. The service was 99.99% uptime since inception.

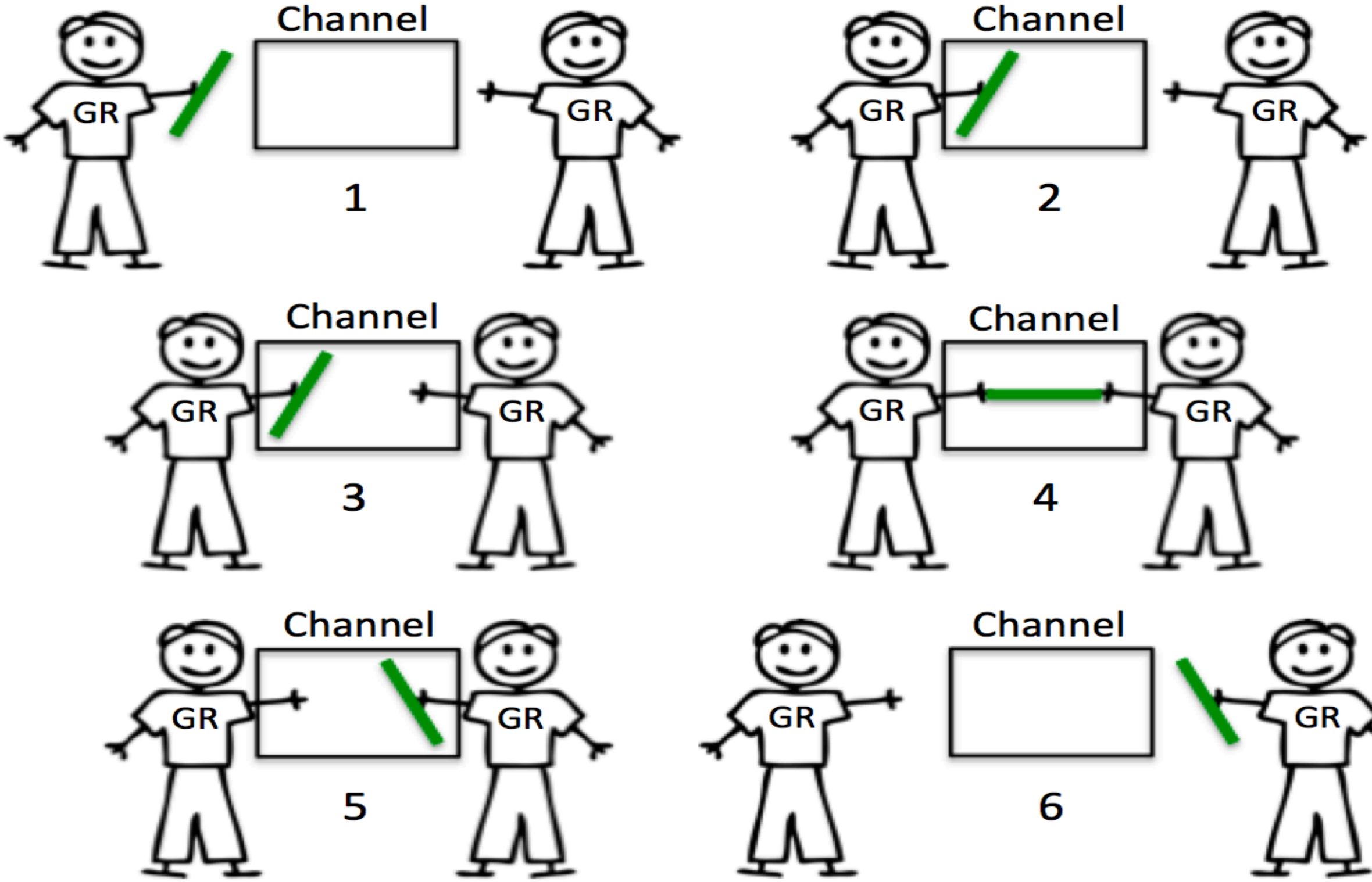
iron.io

- Server count
- CPU Utalization

Thanks !!

Question ?

Unbuffered Channels
c:= make(chan type)



Buffered Channels
c:= make(chan int 5)