Radoslav Rábara, Václav Hála, Jana Michálková

Glow

MapReduce implementation in GOlang

Go....what?

Go == Golang

by Google

in 2007

fast & simple programming language

Similarities:

C/C++

- compiled right to machine code
- aimed at speed

```go
package main

import "fmt"

func main() {
    fmt.Println("Hello, World")
}
```

Similarities:

Java

- garbage collector
- memory-safe

Similarities:

JavaScript

- anonymous functions
- closures

```
func intSeq() func() int {
    i := 0
    return func() int {
        i += 1
        return i
    }
}
```

Similarities:

Yoda

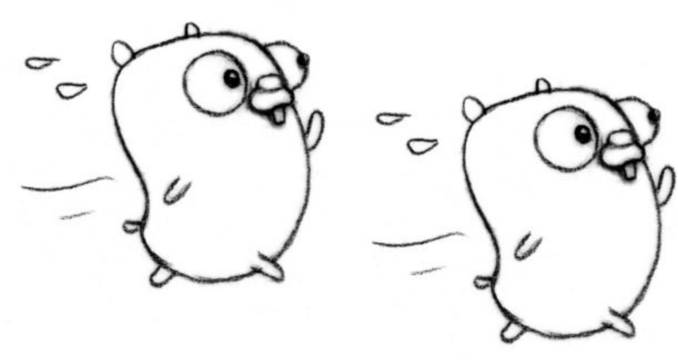
```go
func add(x int, y int) int {
    return x + y
}

func swap(x, y string) (string, string){
    return y, x
}
```

Concurrency

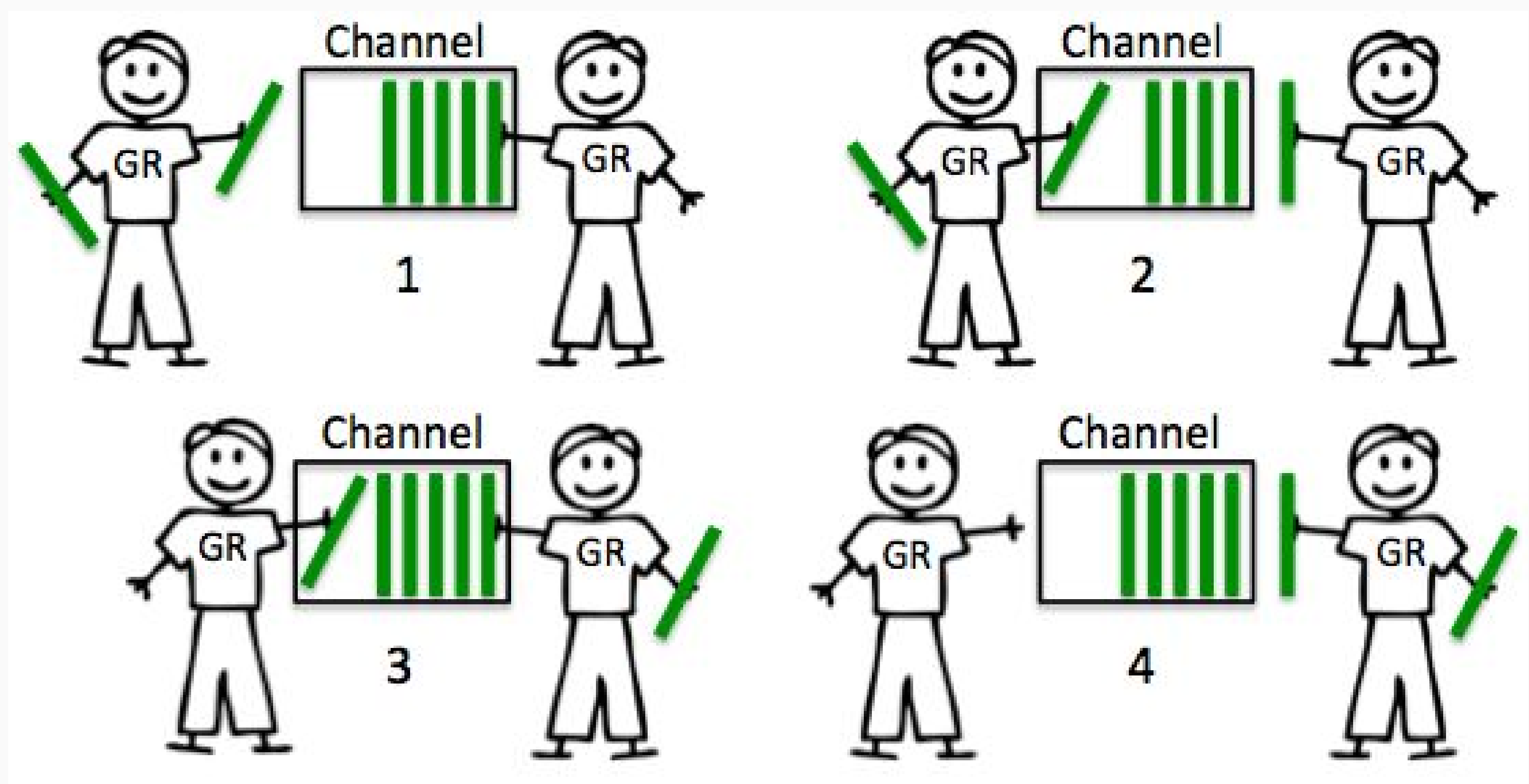
- takes advantage of multiple CPUs

goroutines

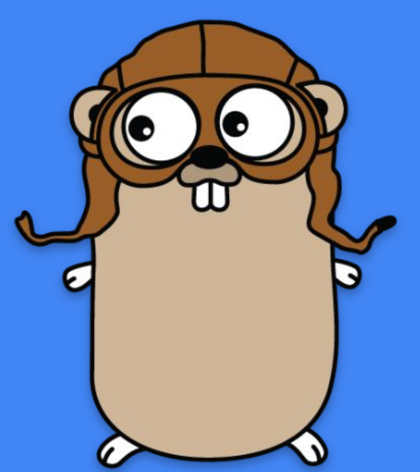
channels



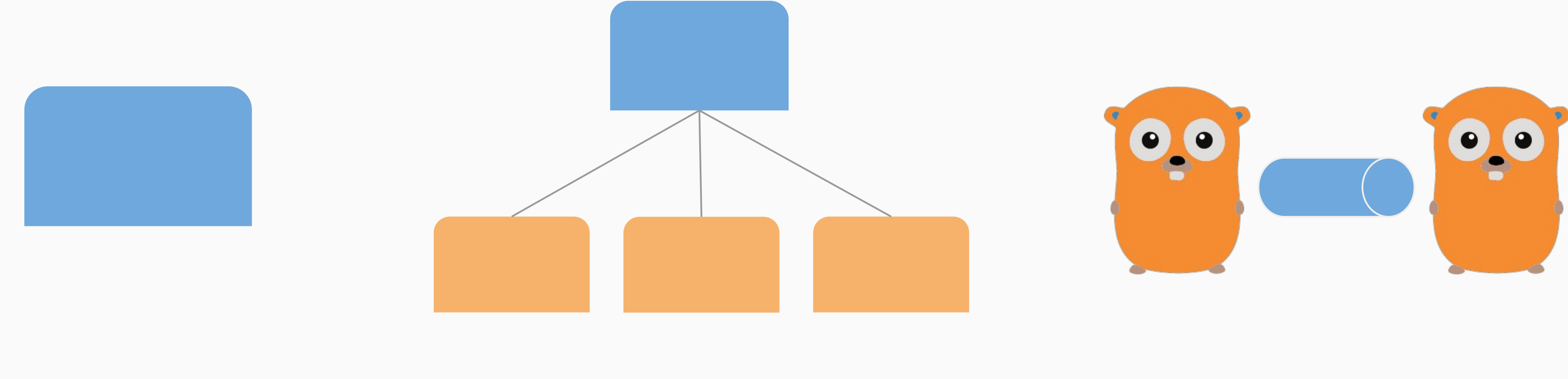
Goroutines and channels



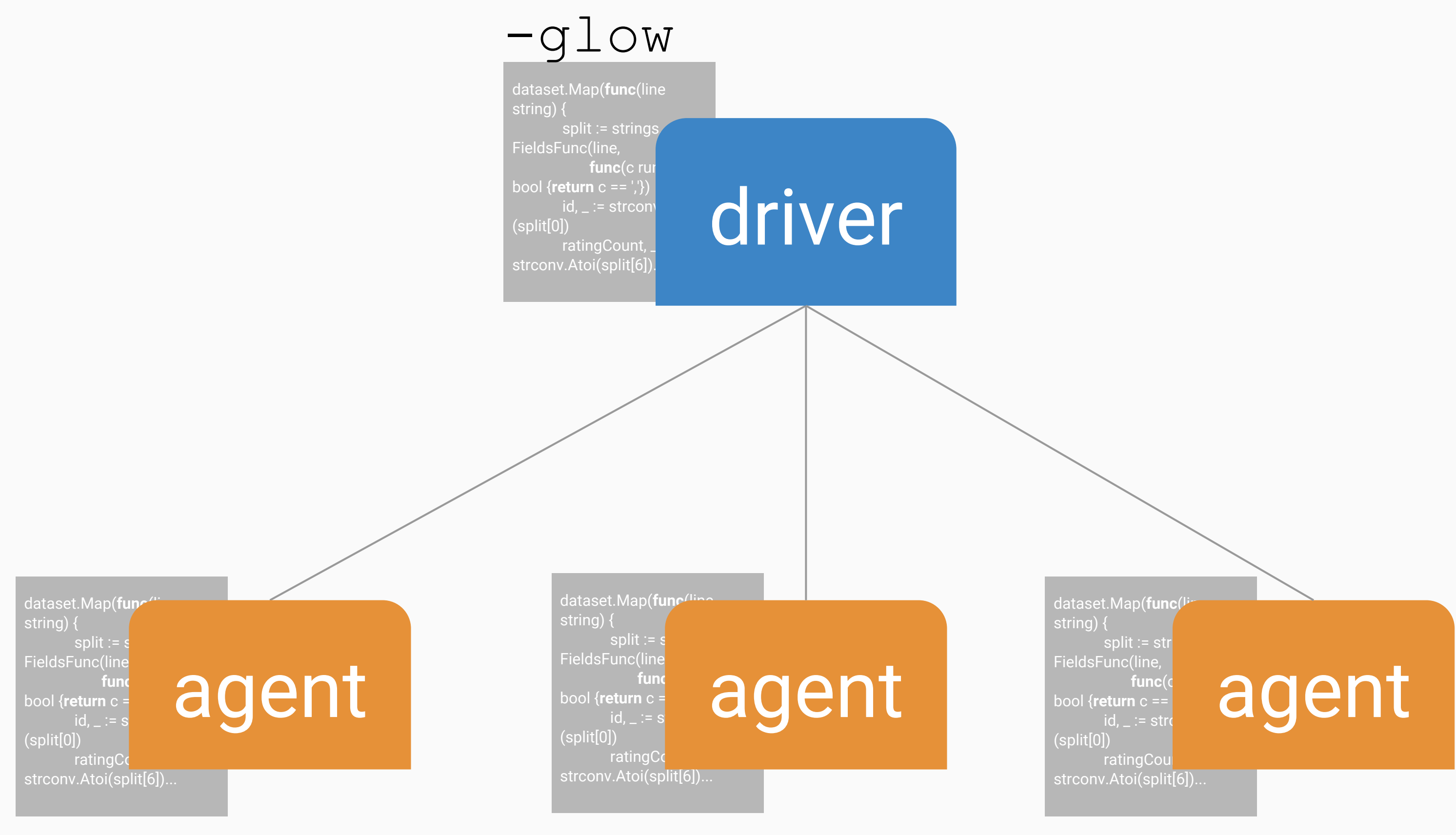
Glow



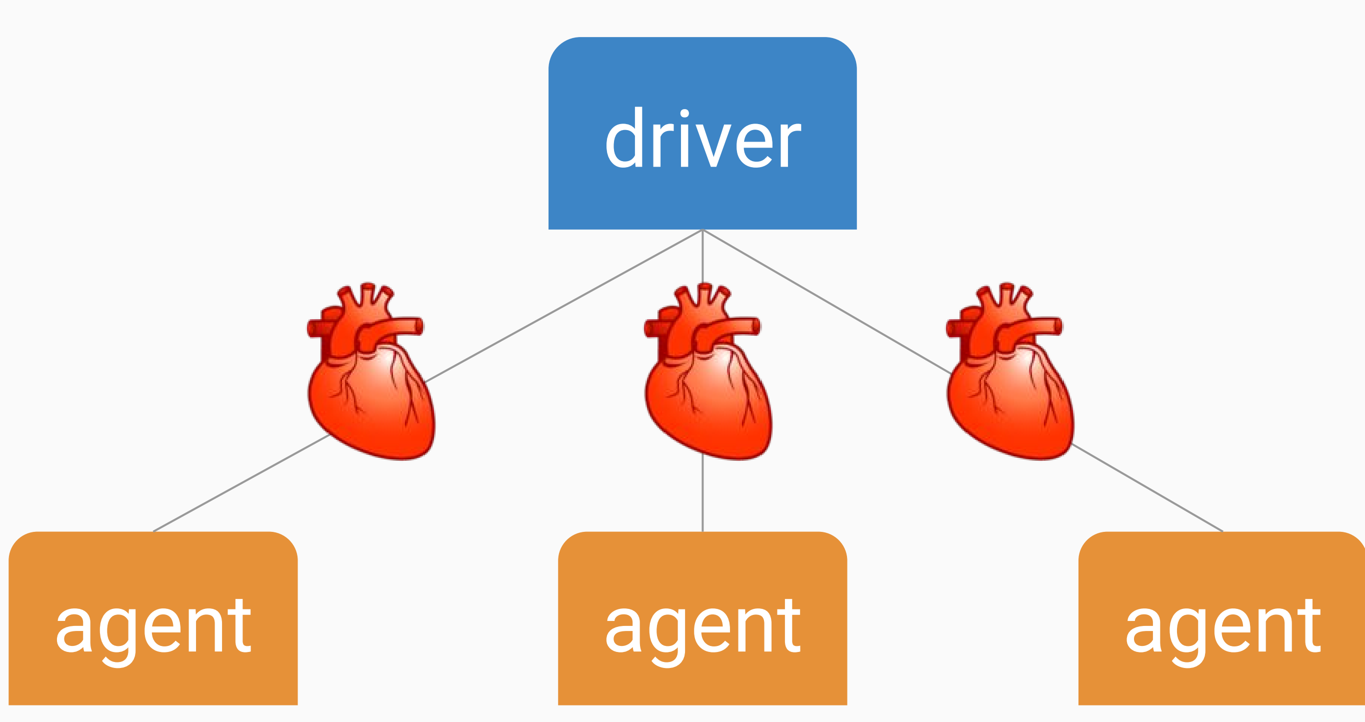
Architecture



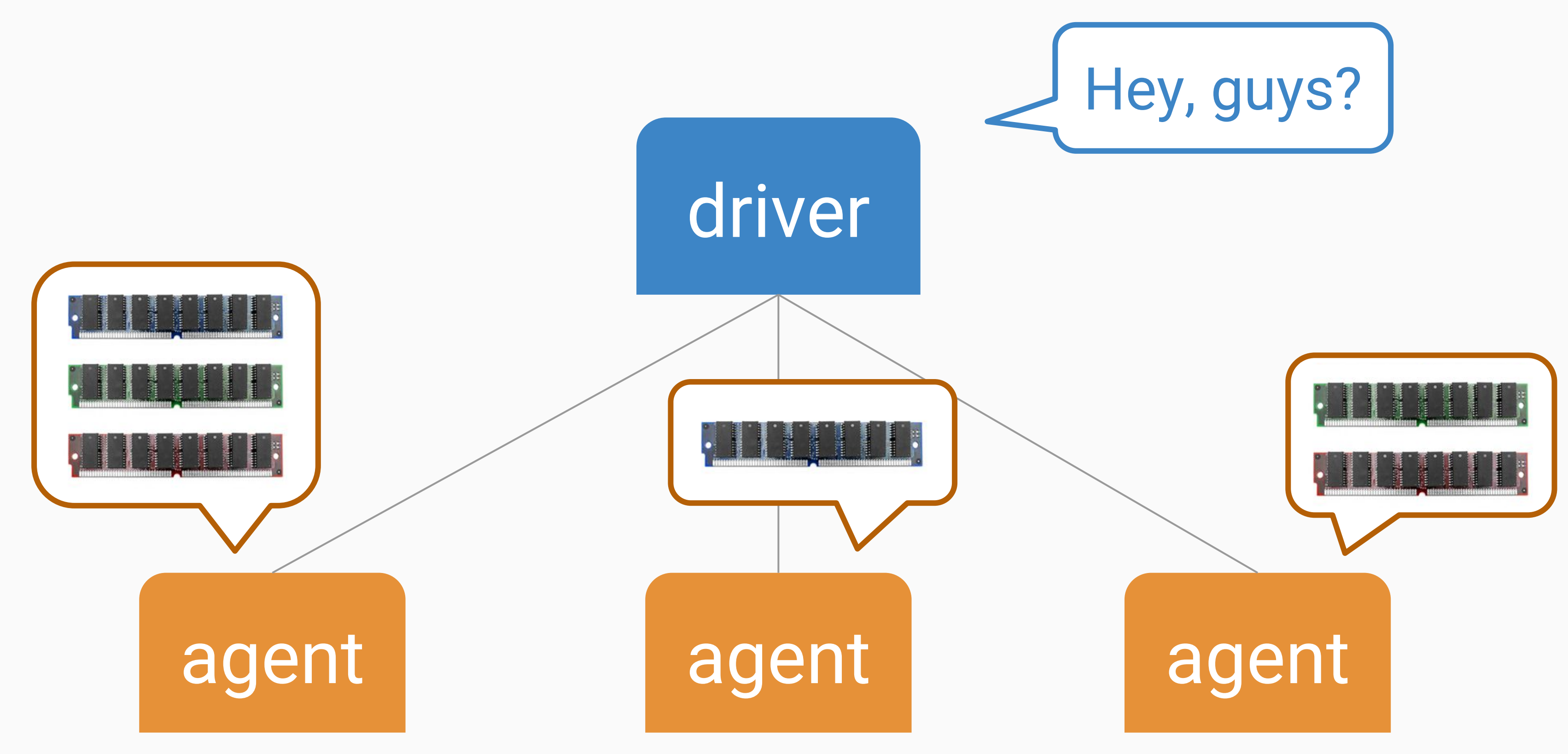
Distributed mode



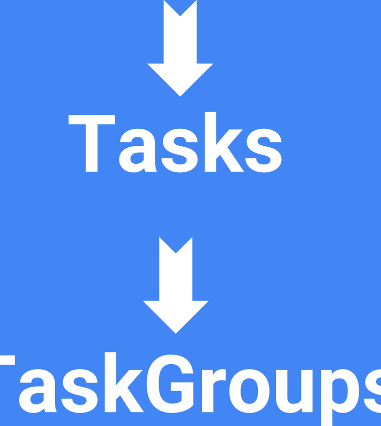
Heartbeats



Master collects memory and CPU information

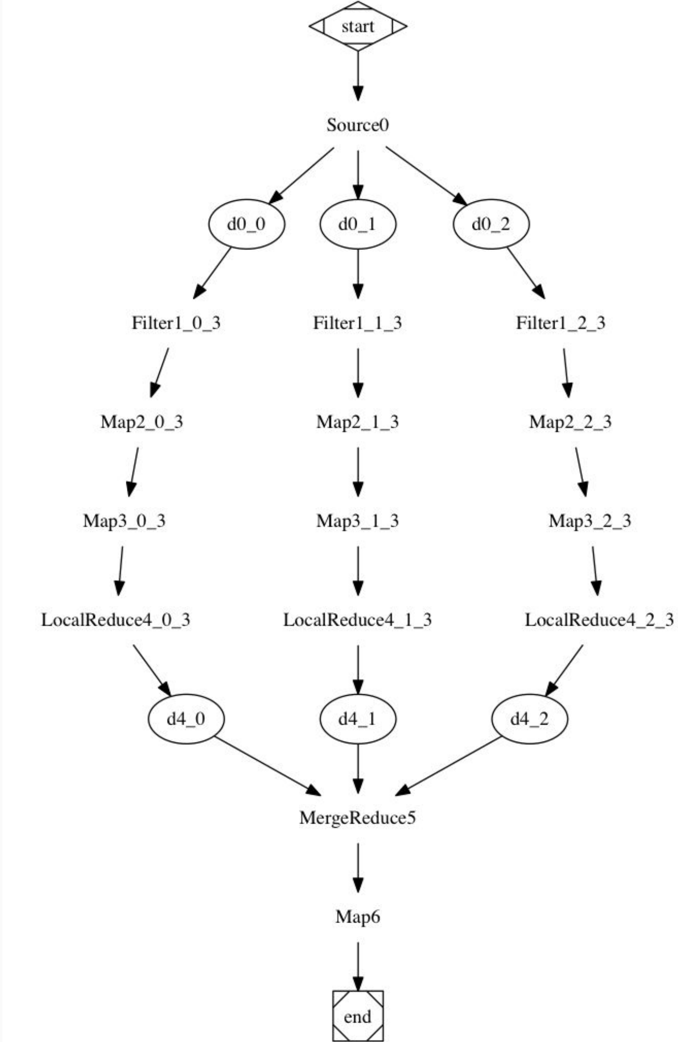


Flow diagram



Tasks

TaskGroups



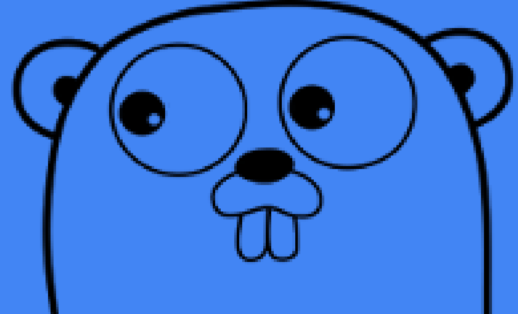
driver

Flow implementation

```go
package main
import (
  "flag"
  "github.com/chrislusf/glow/driver"
  "github.com/chrislusf/glow/flow"
)

var (
  f1 = flow.New()
  f2 = flow.New()

  // input or output channel if any
  inputChan = make(chan InputType)
  outputChan = make(chan OutputType)
)

func init() {
  // flow definitions

  f1.Channel(inputChan).Map(...).Reduce(...)
     .AddOutout(outputChan)

  f2.Slice(...).Map(...).Reduce(...)
}
```

```go
func main() {
  // these 2 lines are needed
  // to differentiate executor mode and driver mode.
  // Just always add them.
  flag.Parse()
  flow.Ready()

  // start the flow
  go f1.Run()

  // feed into input channel if any
  ...

  // wait for the output channel if any
  ...

  // possibly start other flow
  f2.Run()
}
```

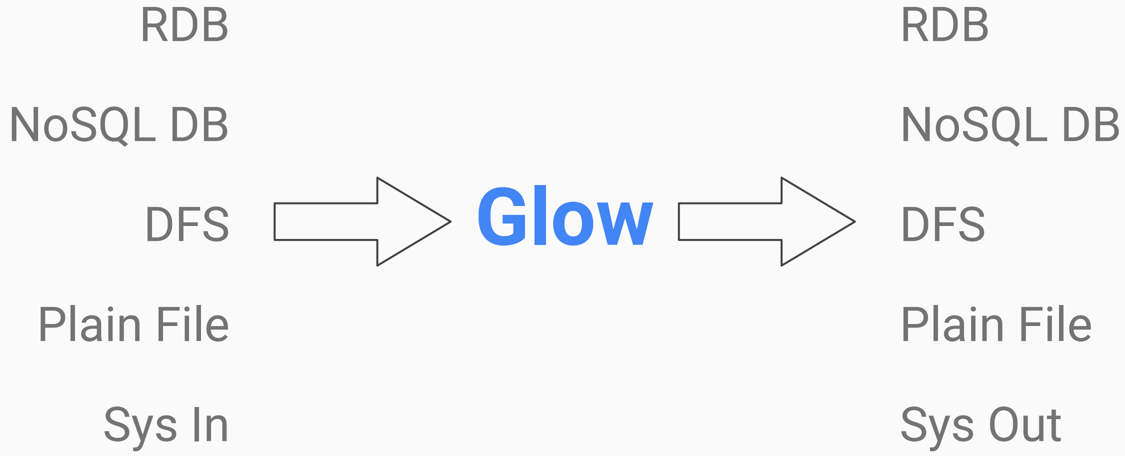
Data sources for Glow



Do One Thing and Do It Well

- Not opinionated about input origin or output destination
- Uses Go channels for data manipulation
- The computation part is always the same

Do One Thing and Do It Well



Strongly Typed Data

- Whole MapReduce execution graph is type checked prior to execution
- Type returned from step *n* must be accepted as input in step *n+1*
- Not checked by compiler but at runtime - IDE can't help :(

Strongly Typed Data

```go
type (
    Product struct {
        Id          int     `bson:"_id"`
        Rating      Rating  `bson:"aggregateRating"`
        Offer       Offer   `bson:"offers"`
        Description string  `bson:"description"`
        Name        string  `bson:"name"`
    }

    Rating struct {
        Value   float32 `bson:"ratingValue"`
        Count   int     `bson:"reviewCount"`
    }

    Offer struct {
        Availability    string  `bson:"availability"`
        Price           float32 `bson:"price"`
        Currency        string  `bson:"priceCurrency"`
    }
)
```

Obtaining Data Input

1. Connect to some storage, query raw data
2. Convert records to custom data structures
3. Send typed input to glow via channel

Hadoop Source

```go
func FetchHadoop(sink chan Product)
{
    f := flow.New()
    dataset := hdfs.Source(
        f,
        "hdfs://localhost:12300/data",
        3,
    )
```

```go
dataset.Map(func(line string) {
        split := strings.FieldsFunc(line,
            func(c rune) bool {return c == ','})
        id, _ := strconv.Atoi(split[0])
        ratingCount, _ := strconv.Atoi(split[6])
        price, _ := strconv.ParseFloat(split[3], 32)
        ratingVal, _ := strconv.ParseFloat(split[5], 32)

        p := Product{
            Id:id, Name:split[1], Description:split[7],
            Offer: Offer{Availability:split[2],
                Price:float32(price), Currency:split[4]},
            Rating: Rating{Value:float32(ratingVal),
              Count:ratingCount}}
        sink <- p
    }).Run()
```

MongoDB Source

```
func FetchMongo(sink chan Product) {
    session, _ := mgo.Dial("localhost")
    defer session.Close()
    session.SetMode(mgo.Monotonic, true)

    products := session.DB("eshop").C("products")
    iter := products.Find(nil).Iter()

    var p Product
    for iter.Next(&p) {
        sink <- p
    }
}
```

PostgreSQL Source

```
func FetchPostgres(sink chan Product) {
    db, _ := sql.Open("postgres",
        `host=localhost port=54321
        dbname=eshop user=postgres`)
    defer db.Close()

    rows, _ := db.Query(
        `SELECT id, name, description,
        price, availability, currency,
        rating, ratingCount FROM product;`)
    defer rows.Close()

    for rows.Next() {
        p := Product{}
        rows.Scan(&p.Id,
            &p.Name, &p.Description,
            &p.Offer.Price, &p.Offer.Availability,
            &p.Offer.Currency,
            &p.Rating.Value, &p.Rating.Count)

        sink <- p
    }
}
```

Socket Output

```
func SendOverNet(host string, port int) chan string {
    addr, _ :=net.ResolveTCPAddr("tcp",
        fmt.Sprintf("%s:%d", host, port))
    netOutput, _ := net.DialTCP("tcp", nil,addr)
    glowSink := make(chan string)
    go write(glowSink, netOutput)
    return glowSink;
}

func write(source chan string, output *net.TCPConn){
    for part := range source{
        output.Write([]byte(part))
    }
    output.Close()
}
```

And Many More

- plain file (server logs)
- named pipe (pcap packet dump)
- another program running in Go
- ...

Putting it All Together

```
func main() {
    sink := make(chan Product)
    go FetchHadoop(sink)
    go FetchMongo(sink)
    go FetchPostgres(sink)
    f := flow.New().Channel(sink)
    flow.Ready()

    f.Map(func(p Product) {
        ...
    }).AddOutput(SendOverNet(host, port))

    f.Run()
}
```

Demo

Glow and data in the cluster

Manatee use case

Motivation

- Parallelization of the Manatee’s operation

Motivation

- Parallelization of the Manatee’s operation
- Mantee is a corpus manager system

Motivation

- Parallelization of the Manatee’s operation
- Mantee is a corpus manager system
    - Thousands of users, including Oxford University Press or Cambridge University Press

Motivation

- Parallelization of the Manatee's operation
- Mantee is a corpus manager system
    - Thousands of users, including Oxford University Press or Cambridge University Press
    - New version is written in Go

Motivation

- Parallelization of the Manatee’s operation
- Mantee is a corpus manager system
  - Thousands of users, including Oxford University Press or Cambridge University Press
  - New version is written in Go
  - Corpus is a collection of texts

Motivation

- Parallelization of the Manatee’s operation
- Mantee is a corpus manager system
  - Thousands of users, including Oxford University Press or Cambridge University Press
  - New version is written in Go
  - Corpus is a collection of texts
  - Corpora can be huge -- billions of words

Motivation

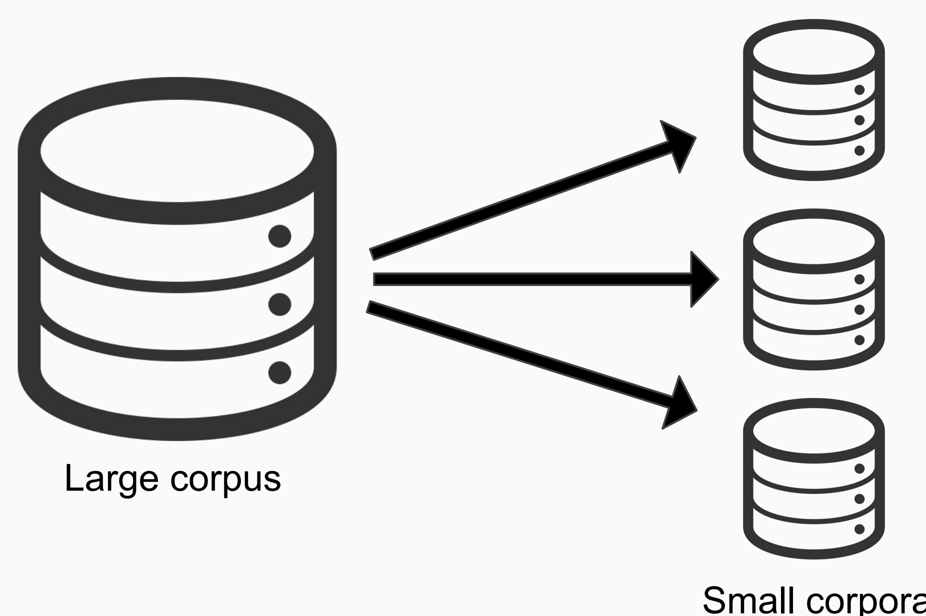
- Parallelization of the Manatee's operation
- Mantee is a corpus manager system
  - Thousands of users, including Oxford University Press or Cambridge University Press
  - New version is written in Go
  - Corpus is a collection of texts
  - Corpora can be huge -- billions of words

Language	Name	Words	
English	enTenTen [2013]	19 billions	
Russian	ruTenTen [2011]	14 billions	
English	enTenTen [2012]	11 billions	
French	frTenTen [2012]	10 billions	

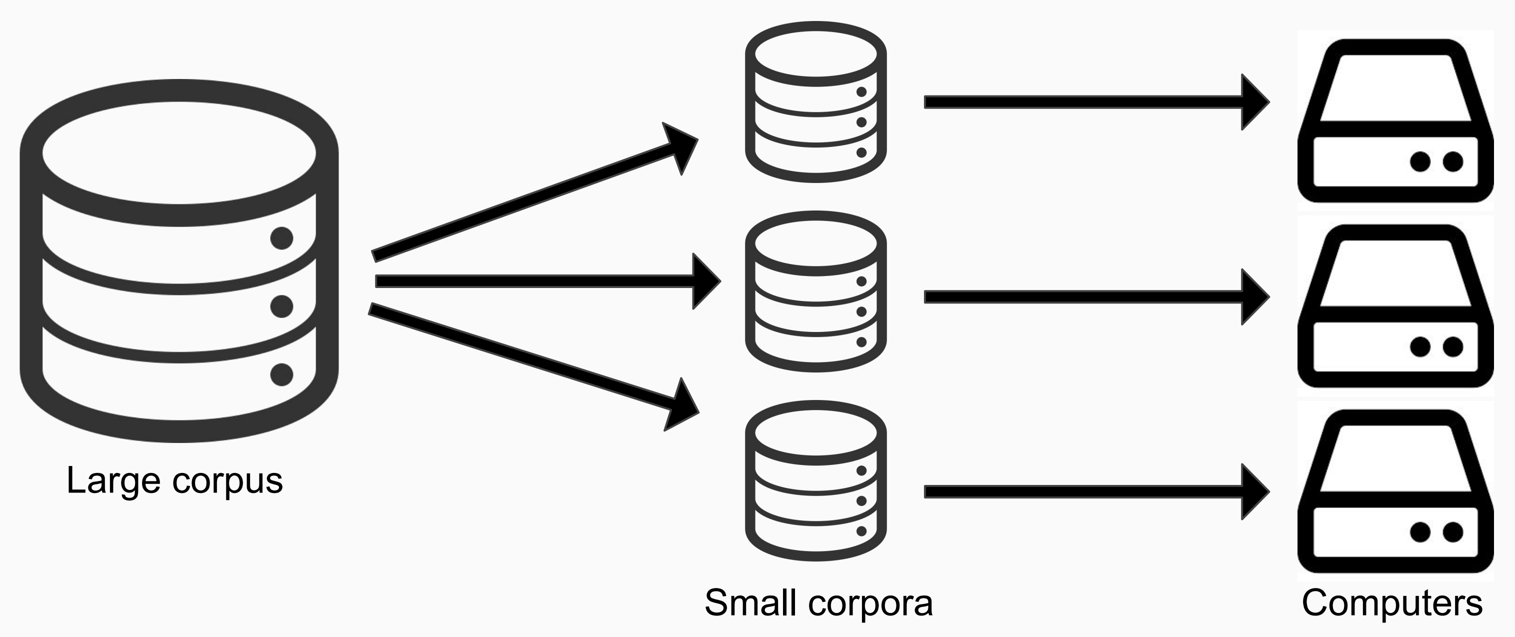
How to store data in the cluster?

Large corpus

How to store data in the cluster?



How to store data in the cluster?



Problem: Glow does not know about data in the cluster

Data source flow:

Solution: Enhance Glow

```
type ComputeResource struct {
    CPUCount    int     `json:"cpuCount,omitempty"`
    CPULevel    int     `json:"cpuLevel,omitempty"` // higher number means higher compute power
    MemoryMB    int64   `json:"memoryMB,omitempty"`
}
```

Solution: Enhance Glow

```
type ComputeResource struct {
    CPUCount         int    `json:"cpuCount,omitempty"`
    CPULevel         int    `json:"cpuLevel,omitempty"` // higher number means higher compute power
    MemoryMB         int64  `json:"memoryMB,omitempty"`
    RequiredResource string `json:"resources,omitempty"`
}
```

Agent → Master

Solution: Enhance Glow

- Agent: cmd flag

```
./glow agent --resources="/corpora/ententen001"
```

Solution: Enhance Glow

- Driver: data source

```
f.WithResources(func(corpusName string) mapInputData {
    return mapInputData{corpusName, query, criteria}
}, listOfRequiredCorpora).Map(func(data mapInputData) concord.FreqDistData {
    return frequencyDistribution(data)
}).Reduce(func(a, b concord.FreqDistData) concord.FreqDistData {
    return MergeFrequencyDistributions(a, b)
}).AddOutput(resultChan)
```

Demo: Glow & Manatee & Frequency distribution

- enTenTen corpus (12 billions of tokens)
  - Divided to 130 parts
- 65 computers/agents
  - Each computer/agent holds 2 parts